

Copyright © 1978, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.


COVERT COMMUNICATION CHANNELS
IN TIMESHARING SYSTEMS

by

J. C. Huskamp

Memorandum No. UCB/ERL M78/37

May 1978



COMPUTER SCIENCE

UNIVERSITY OF CALIFORNIA

BERKELEY

Covert Communication Channels in Timesharing Systems

by

Jeffrey Craig Huskamp
May 1978

TECHNICAL REPORT

No. UCB—CS—78—02

and

Electronics Research Laboratory
Memorandum No. ERL—M78/37

ABSTRACT

In many timesharing systems, the subsystems executed to perform a job are written by several different people. For example, the system I/O routines written by the system group are frequently invoked by other users. This type of usage is found in computer utilities in which "building on the work of others" is commonplace. To provide security in this case, the operating system must not only prevent unauthorized direct access to data, but must also provide confinement for subsystems in a job (senders) that might leak information through covert channels. In covert channels, information is transmitted from the sender subsystem to a cooperating spy subsystem by the spy observing the performance of the system. One typical covert channel that is present in virtually all timesharing systems makes use of the CPU scheduling algorithm. The purpose of this thesis is to study the CPU scheduling covert channel to determine whether the amount of covert information leakage is significant, to explore the factors that influence the amount of leakage, and to determine the feasibility of partial confinement, which trades off leakage for computation cost.

To analyze the scheduling covert channel, a simulator of the CTSS system developed at the Massachusetts Institute of Technology is used. By obtaining the user's response time and CPU time used from the simulator, the cost of confinement can be computed. The amount of information leakage through the covert channel is the usual information theoretic definition of channel capacity. The computation of the channel capacity is broken into three steps. The first step derives a semi-Markov model of the timesharing system from the simulator measurements. The states of the model represent the state of the scheduler and the transition times represent the channel outputs seen by the spy process. The second step constructs a channel transition probability matrix from the semi-Markov model which gives the probability of each channel output for each channel input. The third step is to compute the channel capacity from the channel transition probability matrix using computer iteration. This analysis technique allows different confinement techniques to be compared based on cost and the amount of leakage.

The results of this investigation show that the amount of leakage that must be assumed for a sender can be very large in some cases. Thus covert channel leakage can not be ignored. The major factor in the capacity value is the number of users on the system. This factor is much more significant than even the scheduling algorithm used. For timesharing systems with a heavy workload, the difference in cost between zero-leakage scheduling and non-zero leakage scheduling is small which indicates that non-zero leakage scheduling would not be used for this workload. For light workloads, there is a large difference in the cost of zero leakage and non-zero leakage schedulers so that a trade-off of capacity and cost does exist. Even though the non-zero leakage case costs less than the zero leakage case, the cost of providing confinement can be as large as a factor of six over unconfined execution. Thus using confinement can be very costly, which is in agreement with earlier conjectures. These results are useful to the system designer and the potential confinement users in making the decision of whether confinement (and if so what type of confinement) should be implemented in a particular system.

ACKNOWLEDGMENTS

I wish to thank Robert S. Fabry for his patience in sorting through the many iterations of the ideas in this thesis, for his guidance in navigating the many pitfalls encountered along the way, and for his careful scrutiny of the final product. The ideas presented here are also greatly influenced by Ronald W. Wolff and Aram J. Thomasian who gave generously of their time and expertise. A large measure of appreciation goes to my wife, Beth, who provided much needed encouragement when it was needed the most. I also wish to thank Janet S. Chin, Robert M. Long, and Mary E. Zosel for their help and encouragement. This research was partially supported by the Electronics Research Laboratory of the University of California, Berkeley (under the Joint Services Electronics Program contract number F44620-76-C-0100), by Lawrence Livermore Laboratory (under the U.S. Department of Energy contract number W-7405-Eng-48), and by the Institute for Defense Analyses.

This is a reproduction of a PhD thesis submitted to the Computer Science Division, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, May 1978. Reproduction was partially supported by NSF grant number (NSF) MCS75-23739.

Copyright © 1978 by Jeffrey Craig Huskamp.

Author's current address: Institute for Defense Analysis, Princeton, New Jersey, 08510.

Covert Communication Channels in Timesharing Systems

Jeffrey Craig Huskamp

1. INTRODUCTION

1.1 Confinement

Within the past decade, an increasing amount of information has been stored in computerized data banks. The types of information available in these data systems range in sensitivity from classified military intelligence to company payroll information. For these systems, a major concern is the theft of the information from the data bases. In spite of the protection problems involved, data base systems are becoming more widespread, even in smaller companies, to do routine data management tasks in a more cost-effective manner. With the cost of the hardware investment needed for data systems decreasing and with the increasing availability of software packages tailored for specific business applications, the future outlook for an increasing number of data processing systems in business applications is very bright [Lettieri76].

One of the common environments for data base systems (in particular large data base systems) is the computer utility [Corbato72]. In a computer utility, many different types of users and applications can be supported in a general manner by one computer system. This type of system support encourages many applications to share the same computer facility. The consolidation of the computer needs of many users is, in many cases, more cost effective than providing an independent computer facility for each, since the fixed system cost is shared.

One advantage of using a computer utility is that the protection mechanism facilitates sharing information between users. The protection mechanism allows one user, A, to express his amount of trust in another user, B, by the access A gives B to A's objects. This regulation of the degree of access makes A more willing to share objects with B (for example a program) by knowing that B can only modify the program if given the appropriate access by A. For example, users of a computer system routinely use standard subroutine libraries provided by the systems group. It is rare that a higher-level language user would write his own set of these routines. This "building on the work of others" is a good method for decreasing the cost of program development for new application subsystems, and for decreasing the length of time needed to get a new application running. Thus, each user comes to rely on software subsystems written by (possibly) unknown authors which execute on the user's behalf. Since the interaction between a user (or his process) and a foreign subsystem is common in a computer utility, special names are given to the participants. The user that invokes the foreign subsystem is called the *customer*, since he invokes the subsystem to perform a task. The foreign subsystem is called the *service*.

As an example of a typical customer-service interaction, suppose user B, a very successful electronics manufacturer, decides that one of his circuit analysis subsystems is sufficiently well designed that other users might want to use it. Thus, user B makes the subsystem available to any user willing to pay a usage fee. User A, a competitor of B whose company has a very small programming staff, decides to use B's subsystem rather than expend the manpower to develop a similar package. To initiate the interaction, user A provides a description of the circuit to be analyzed (along with an IOU for payment) to the analysis subsystem. On completion of the service, A may expect the voltages between every pair of nodes and the current in each wire to be returned as shown

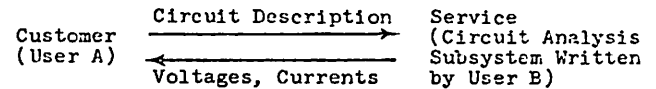


Figure 1.1
Circuit Analysis Service

in Figure 1.1. Since A and B are both competitors in the electronics market, it may be important to A that his circuit designs will not be leaked to B by invoking the circuit analysis service. One possible way this could be done is for A or a trusted intermediary to examine the program listing of B's subsystem to see if it will attempt to injure A. The problem with this approach is that it is difficult to prove that a subsystem performs correctly with no unacceptable side effects, such as leaking information to B. This problem is similar to the operating system verification problem in which the presence of protection flaws can be demonstrated but the absence can not. This is a consequence of the current state of program verification.

The same problems inherent in a customer-service relationship are also present in military applications where jobs of different security classifications are to be executed. The current practice is to execute jobs of only one classification level at a time on a machine. Before jobs of another classification level are introduced, the machine must be "scrubbed" to eliminate any residue from previous executions—a process requiring about an hour of real time. A more cost effective approach is to be able to execute jobs of different security classifications at the same time on the same machine. Recent efforts by Belady [Belady74] and Weissman [Weissman75] have concentrated on "hardening" the security mechanisms of a virtual machine implementation to permit simultaneous execution of jobs with different security levels. To provide a complete solution to this problem, not only must processes be denied direct access to unauthorized data, but also indirect access through any available means. A typical type of indirect access occurs when an unclassified subsystem is executed on classified inputs. Steps must be taken to insure the unclassified subsystem does not leak any of the classified inputs. One example is the "copy file" utility that is present on most timesharing systems.

1.2 Previous Research

An early development in protection mechanisms designed to protect more privileged subsystems from less privileged subsystems was the implementation of the ring architecture in the MULTICS system. The MULTICS protection architecture described by Schroeder in [Schroeder72B] is composed of eight concentric rings numbered from 0 for the innermost (most privileged) ring to 7 for the outermost (least privileged) ring. The access rights to segments that a process can exercise are a function of the user that owns the process (the principal) and of the ring in which the process is executing. The access list mechanism [Saltzer74] enumerates for each principal the type of access allowed and the rings in which the access can be exercised (denoted by the ring brackets). There is one ring bracket for read access, one for write access and one for execute access. For this discussion, only the read and write brackets need be considered. The lowest bracket numbers for read access and write access are fixed at

zero and the highest bracket numbers may be any number from zero to seven. In MULTICS, by convention, rings 0 through 3 contain supervisor and administrative subsystems while rings 4 through 7 are reserved for user subsystems. Proprietary subsystems are considered to be more trustworthy than user subsystems but less trustworthy than the supervisor and are executed in ring 3.

The ring architecture outlined above can partially solve the problem of preventing the called subsystem from reading or writing the caller subsystem and vice-versa (the *mutually suspicious subsystem problem*). To see this, suppose a MULTICS customer executing in the most privileged user ring (ring 4) invokes the proprietary circuit analysis subsystem which executes in ring 3. When the incarnation of the subsystem is created and the ring change (from ring 4 to ring 3) occurs, all the read/write privileges enjoyed by the customer can be exercised by the service due to the lower ring number of the service (remember that the read and write ring brackets have a fixed lower bound of zero). However, the ring mechanism prevents the customer from reading or modifying any of the proprietary subsystem's files. The ring mechanism can only solve one half of the mutually suspicious subsystem problem. That is, either the customer can be protected from the service (the customer executes in a more privileged ring) or the service can be protected from the customer (the service executes in a more privileged ring), but not both at the same time.

The MULTICS ring architecture can be generalized to solve the problem of protecting both the customer and the service. The solution is to provide a separate *domain* for each party as described by Lampson [Lampson69] for the BCC Model I. Each domain provides a complete protection environment that is independent of any other domain in the process. Thus the customer can regulate the type of access the service can exercise on the passed parameters, and the service can regulate the access permitted by the customer by restricting his access to execute-only. A typical customer-service interaction would be initiated by the customer invoking the circuit analysis service which causes a domain change to occur. When the service is incarnated, the only access the service has to the customer's objects is through the passed parameters. Thus the customer is protected from the service. Likewise, the service is protected from the customer. In addition, the customer can not access any of the service's objects unless access to the objects is passed back to the customer as one of the outputs from the service. Thus the protection aspect of the mutually suspicious subsystem problem that could not be adequately solved by MULTICS (see [Schroeder72A] for an extension of MULTICS to provide domains) is solved. But the domain mechanism still does not solve all the problems inherent in a customer-service interaction. The remaining problem is that the service may either retain the input parameters passed by the customer after ending execution, or transmit the parameters to another user during execution. A simple realization of this would be for the service to copy the inputs to a new file and to save this file in the file system under another user's name. The problem of insuring that the service does not leak any information to unauthorized subsystems or users while executing on behalf of the customer is termed the *confinement problem*.

The general problem of information confinement is not peculiar to computer systems. A real-world example in which information confinement is necessary is during bidding in an international bridge tournament. Information is legitimately passed among the players by each player in turn announcing a legitimate bid describing his hand to the other players. When one player bids, the process may be described as the player encoding his hand and the previous bids (inputs) into

a new legitimate bid (output). Measures must be taken to insure that the announced bid is the only method by which partners communicate. One illegal method of communication is for the bidder to announce his bid in either a loud voice or a quiet voice to convey some extra information about his hand. Another (illegal) method for transmitting information is for one partner to write information on a scrap of paper and to pass the paper to his partner without attracting the attention of the opponents. Another technique would be for the partners to engage in toe-nudging under the table, or to encode information in facial expressions, coughing, or chair scraping. The number and implementation of these illegal information channels is limited only by the imagination.

In a computer system, the same types of information leakage as in the bidding example are present. These illegal methods for leaking information can be classified into three categories [Lampson73]: storage channels, legitimate channels, and covert channels. These channels are discussed next.

1.2.1 Storage Channels

Storage channels are communication channels implemented by the service writing information into some form of memory which a cooperating confederate (*spy*) can read. Here the spy is assumed to be written by the service author and to receive full cooperation from the service - i.e. the service is actively trying to pass information to the spy. The operating system is assumed to be trusted in the sense that it does not knowingly aid in the leakage of information. In the bidding example, passing scraps of paper with written information is an example of this type of channel. The strategy used during bridge tournaments to block this information leakage is to employ officials to watch the players and to insure no written information is passed. A computer system implementation of this channel would be for the service to write the information in a file shared with the spy, or for the service to send the spy a message through the interprocess communication facility.

Three methods have been explored for blocking storage channels. The least general method is to impose stringent restrictions that can be tested with a simple static check on how the service is written as is implemented for "message confinement" in COPS [Andrews74]. Here storage channel confinement is implemented by insuring that the service can not modify any object inherited when incarnation of the subsystem occurs and does not call another procedure. This insures that the service will be memoryless so that no confidential information is retained. There are two problems with this approach. The first problem is that certification of message confinement involves auditing the service. As noted before, auditing can only prove the existence of errors and not their absence. The second problem with message confinement is that denying the service the ability to call another subsystem nullifies the major advantage of a computer utility, which is the ability to build on the work of others [Dennis68]. This is a high price to pay for confinement since each subsystem must utilize only absolutely trusted programs. This could mean that some functions would have to be reimplemented rather than using existing versions of the functions which could involve an appreciable software cost.

The second method for closing storage channels is the all-or-nothing strategy implemented by HYDRA [Cohen75]. In this mechanism, a confined call causes all capabilities contained in the service's LNS (capability list) and its transitive closure to lose the right to modify the object named, and all capabilities which may be used to invoke another subsystem add the condition that the call must be made confined. The only objects that can be modified by the service are newly

created objects and objects passed by the customer with modify rights enabled. Transitivity of confinement when the service invokes another procedure [Lampson73] is guaranteed since capabilities from the incarnation segment can only be used to make confined calls. Unconfined calls may be made only by using capabilities passed by the customer as arguments, in which case the customer has vouched for their safety. This mechanism provides total confinement for storage channels in the sense that leakage of both confidential and non-confidential information is confined. However, in some applications, all of the information processed by the service does not need to be suppressed. Allowing the unclassified information in the arguments to be distributed by the service while confining sensitive information is called the *selective confinement problem*, which is addressed by the third method.

The third method for storage channel confinement provides selective confinement of the information given to the service. Two mechanisms for implementing this have been investigated by Lipner [Lipner75] and Denning [Denning76] [Denning77]. Lipner's approach is a static method of enumerating all storage channels by analyzing the effects section of a Parnas function specification [Parnas72] of an operating system as in [Schiller75]. A Parnas function specification details the implementation of a program using two types of functions: *o*-functions (operation functions that can change the state of the operating system) and *v*-functions (viewing functions that only can observe the state variables of the operating system). The method is to demonstrate that there does not exist an *o*-function that uses a *v*-function of a certain classification (e.g. top secret) and modifies a *v*-function of a lower classification (e.g. unclassified). For example in the military classification system, this analysis would detect an instance where a top secret file is copied to another top secret file with a side effect that some of the information is also written in an unclassified file. Here, the declassification of information must be prohibited. In this case the unclassified file read *v*-function is altered by the top secret file write *o*-function. Since the unclassified *v*-function is altered by an *o*-function which uses a top secret *v*-function, a storage channel is present.

Denning's approach is to continuously monitor (e.g. by hardware) or analyze the information flow in a subsystem to insure that only permissible flows are allowed. This monitoring prevents confidential information from being disclosed but permits non-sensitive information to be distributed. See [Fenton74] and [Gat76] for other implementations. To make the monitoring possible, each word of information is identified with its information classification. In a static compile-time implementation, the identification is done by marking each variable with its information classification in an internal compiler table. In a dynamic run-time implementation, the identification is done with a tagged architecture. The classification of a result computed from information of different classifications is given by a rule in the model. The set of rules for combining information in different security classes can provide different selective confinement policies including the high-water-mark policy [Weissman69] and the military security classifications with need-to-know categories [Bell73].

The disadvantage of both selective confinement approaches is that when the number of security classes becomes large, the model and implementation become unwieldy. For example, in the military classification scheme, each word of information must be tagged with the Cartesian product of the security category (e.g. unclassified, confidential, secret, top secret) and the need-to-know category. For a small number of need-to-know categories (e.g. 15), the Cartesian product

can be represented by a few bits. But in a general computer utility where each user needs to protect his information from all other users, the number of classifications may rise dramatically. In the mutually suspicious electronics firm example, the customer wants to insure that no information can be leaked to his competitor, the service author, even if the author has been cleared to the highest classification level available in the system. This amounts to creating a new need-to-know category for the customer which will govern the flow of information for his interaction with the service. It is also conceivable that the customer would want to selectively share this information with other users. This amounts to creating additional need-to-know categories for each sharing arrangement. Since these considerations are common in a computer utility, the number of information classifications needed is potentially very large. This rules out a dynamic mechanism for selective confinement implemented by a tagged architecture. For this reason, the most practical method for providing storage channel confinement in a computer utility appears to be the total confinement approach taken by HYDRA. In HYDRA, the sharing requirement is easily implemented by the capability mechanism.

1.2.2 Legitimate Channels

Another type of information channel through which leakage occurs is the legitimate channel [Lampson73]. For the bidding example, this channel is represented by the bid announced to all players. In order to make the bridge game more competitive, the legal bids at any time are tightly controlled. If there were no restrictions, each partnership could devise their own system of bids which could convey a large amount of information to the bidder's partner. One such system would be to encode every card in the bidder's hand into a single number (e.g. bidding 1.7329849 hearts). The computer system counterpart for this channel is the bill for services which is specified by the service and passed through the accounting system to the service author and the customer. The amount charged for the resources used could be an encoding used to transmit a large amount of information from the service to the service author as in the bidding example. These channels can be enumerated and blocked by using either Lipner's approach [Lipner75] or Denning's approach [Denning76] since the bill can be viewed as a storage channel when the resource usage information is passed to the accounting system. In HYDRA, because selective confinement is not available, the bill must be handled separately from the other system objects. Rotenberg [Rotenberg74] investigated blocking the billing channel by charging a fixed sum (e.g. per invocation, per month) for the service. If this approach is too restrictive, Rotenberg suggests limiting the amount of information contained in the bill to a fixed number of digits or rounding the amount charged (throttling the channel) which is analogous to restricting the permissible bids in the bidding example. In this case the amount of information passed is not zero but is some "small" amount. Thus the legitimate channels can be blocked either totally or partially depending on the system policy for handling the bill.

Another scheme, which is due to Lampson, involves publishing the charging algorithm, and having the customer compute the charge in advance. The service can then accept or reject the customer's amount of payment if it is incorrect. If, however, the service makes invalid rejections after observing the input parameters, then another covert channel is possible which is composed of acceptances (the service contributes to the workload) and rejections (the service does not contribute to the workload). Under this scheme, all rejections must be valid.

1.2.3 Covert Channels

The last type of information channel used to transmit information, and the most difficult to analyze, is the covert channel [Lampson73]. Covert channels can be thought of as sending information from the service to the spy through the computing environment rather than directly through some type of storage. For the covert channels analyzed here, the supervisor is trusted to not knowingly leak information from the service to the spy. This is a different perspective than in the Morse-code problem [Popek74] in which the supervisor actively aids in the leakage. In the bidding example, covert channels are implemented by chair scraping, facial expressions, toe-touching, or any noise made by one partner. For example, a cough could mean "I hold three aces" and a sneeze could mean "I have no trump support." To help combat this type of information leakage, bridge tournaments employ bidding screens reaching to the floor to insure that partners don't see or touch each other during bidding. However, noises can still be used to transmit information.

In computer systems, covert channels are implemented by the service requesting resources in a pattern that could be noticed by the spy. For example, consider a user program that calls an untrusted output program to write data into a file. If the output program wants to leak the data contained in the program arguments to a cooperating spy program by using the CPU resource, the output program would first encode the arguments into a sequence of CPU times that would be used to satisfy the calling program's request. For example, a typical encoding of the number 10 might be for the output program to use Q , $Q/2$ and Q CPU seconds for the next three CPU allocations (Q is the maximum quantum length that can be allocated). The output program would then perform its normal function of writing the data into the file during the non-zero quantum allocations. The spy process, which is executing at the same time as the output program, would have its queueing time affected by the amount of CPU time used by the output program. A larger CPU usage by the output program would mean a longer queueing delay experienced by the spy process and vice-versa. Since the output program and the spy process are cooperating, the spy process knows the encoding algorithm used by the output program. With this information, the queueing delays experienced by the spy would then give information about the sequence of CPU times used by the output program, which in turn would provide information about the data that the output program is trying to transmit. By this method, the argument value given to the output program could be leaked to another process. The channels implemented by the spy process observing system performance variations do not always send perfect information due to the possibility of other processes making resource requests during the channel operation. However, as observed by Lampson [Lampson73], by using information theory, a non-zero amount of information can be reliably sent through these channels.

To block these channels, Lampson proposed the concept of *masking* in which the inputs to all covert channels must be specified by the customer. This solution will completely close the channel but at a possibly large cost due to the ignorance of the customer concerning the dynamic resource needs of the service. In general, the resource needs will vary over the execution of the service and the customer has no method for specifying the optimal resource allocation. Another proposal for blocking covert channels [Popek74] [Lipner75] is to eliminate the perception of real time by the service and the spy in order to prevent the spy from evaluating the performance of the system. The suggestion is to associate a virtual clock with each process which runs only while the associated process is executing. Thus, for example, page fault delays can

not be measured directly by the spy. This solution would work in a completely batch system where the turnaround time can be made constant. However, in a timesharing environment, this solution would not eliminate all covert information leakage. As observed in [Lipner75], the spy could correlate virtual time with real time by the service author constantly entering the time-of-day through the terminal. The grain-of-time is usually much larger than that of the hardware real-time clock with this method but is still small enough for non-zero information leakage to occur.

1.3 Thesis Content and Result Summary

Of the three types of information channels, covert channels appear to be the hardest to analyze and the most costly to block. The problem with analyzing covert channels is that a suitable model of the timesharing system must be found that allows the effect of one process (the service) to be measured. The cost results from the constraints that must be placed on the operating system to make the amount of leakage analyzable. The objective of this research is to explore the trade-off between the cost of implementing different confinement mechanisms and the rate of information leakage for the CPU scheduling covert channel in an interactive timesharing system. Mechanisms that permit zero information leakage as well as mechanisms that allow some (measurable) positive amount of leakage are studied. The consequences of providing a confinement option on the operating system design will also be discussed.

Chapter 2 defines the terminology from information theory that will be needed to discuss information leakage. The concepts of a discrete memoryless information channel, an information channel with memory, and channel capacity are introduced. The specific channel models used in the later analysis are also introduced. Chapter 3 explains in detail what a covert channel is in a timesharing system. The concepts of absolute confinement and partial confinement are introduced and their effects on the operating system design are discussed. Chapter 4 discusses the covert channel implemented by the system scheduler which is to be analyzed in the remaining chapters. The implementation of the scheduling covert channel in each of the scheduling algorithms to be analyzed is explained. Chapter 5 contains the semi-Markov model to be used in the analysis. The technique used for the model validation and the validation results are given. Chapter 6 has the analysis results for the scheduling covert channel using both the discrete memoryless channel model and the channel with memory model. Chapter 7 summarizes the results in the previous chapters.

The major accomplishments of this thesis are: 1) An analyzable model has been developed from which the capacity of covert channels can be determined. This model is applicable to all covert channels that can be characterized by a semi-Markov model. 2) The analysis results show that a significant amount of information can be leaked through the system scheduler. Since virtually all timesharing systems have a system scheduler, covert channels are a widespread problem. 3) The simulations of an interactive timesharing system with a scheduler that limits the amount of information leakage shows that allowing no information leakage is indeed expensive, but that by allowing some small known amount of leakage, the cost can be reduced in some cases. 4) The development of a model to analyze the information leakage provides a firm theoretical foundation for timesharing system designers to state an upper bound on the average amount of information leakage resulting from use of the system. This is an important consideration when comparing timesharing systems.

2. INFORMATION THEORY PRELIMINARIES

2.1 Introduction

The purpose of this chapter is to introduce the information theory-related terms that will be used extensively in the remainder of the thesis. The intent is to provide only a brief discussion here since there are several good texts on the subject (e.g. [Gallager68], [Ash65], and [Wolfowitz64]) that can be consulted for the necessary proofs and background details.

This chapter covers the concept of information, the distinction between a memoryless channel and a channel with memory, and the calculation of channel capacity. These concepts permit an upper bound to be calculated on the amount of information that can be sent through a covert channel. Also, three types of synthesized channels (i.e. cascaded, compound, and cascaded compound channels) are discussed for modifying the characteristics of a given channel. These synthesized channels are used to modify covert channels so that the channel capacity is less, or the cost of confinement is less, or both. The reader who is familiar with these ideas can skip this chapter without loss of continuity.

2.2 The Concept of Information

One favorite party game for small children is the "telephone game." This game is started by assembling a group of children in a straight line between two game coordinators. The first game coordinator whispers a message to the child next to him. Each child in turn whispers the message to the next child until the message reaches the second game coordinator. The original message and the received message are then compared to determine how well the communication line worked. Usually, the two messages differ greatly due to transmission errors. Parts of this human communication link have special names which describe their function in the communication process. The first game coordinator is called the sender, since the information to be transmitted originates with him. The children represent the channel, or the method by which the information is moved from one place to another. The second game coordinator is the receiver, since he is the destination for the information. This simple description can be complicated by adding a message encoder between the sender and the channel and a decoder between the channel and the receiver. The function of the encoder and decoder is to transmit over the channel only information well-suited to the channel. For example, if the channel can transmit only the letters 0 and 1 and the sender generates messages containing the letters 0, 1 and 2, the message must be encoded into 0's and 1's before transmission. The decoder will then take the 0's and 1's received and produce the original message with the 0's, 1's and 2's. In this thesis, the sender specifies the message that is to be sent. The message is then encoded into a series of letters which are sent through the channel, one at a time. The letters are decoded by the receiver to form the original input message. Thus the actual channel inputs and outputs are letters.

A major problem of real channels is that the transmission may not be error-free, as in the telephone game example. In this case, when the receiver obtains the channel output, he is uncertain of the input sent. More formally, uncertainty occurs when two or more distinct channel inputs can cause the same channel output to be received with non-zero probability. A measure of this uncertainty is the average entropy, H , computed by the function:

$$H(X) = \sum_i [-P(x_i) \cdot \log_2(P(x_i))]$$

where X is an ensemble, the x_i are the members of X , P stands for probability, and the logarithm is base 2 for the

result to be in units of bits. (An ensemble is composed of a set of events and the associated probability of occurrence for each event). This uncertainty function has the intuitively appealing result that if the input letter sent is known (the probability of x_i for some value of i is one), then the average entropy is zero (assuming that $0 \log 0$ is zero). Otherwise the entropy value is greater than zero.

The concept of entropy can be extended to define the notion of mutual information between two ensembles. Mutual information can be thought of as the amount of uncertainty resolved about the element transmitted from one ensemble, given an observation in the other ensemble. In an information channel, the ensemble representing the channel outputs, Y , does give information about the element from the channel input ensemble, X , that was transmitted. In terms of uncertainty, the mutual information of the input and output ensembles, $I(X;Y)$, is computed by the formula:

$$I(X;Y) = H(X) - H(X|Y)$$

where $H(X)$ is the original uncertainty about the channel input and $H(X|Y)$ (read the entropy of the X ensemble given the Y ensemble) is the remaining uncertainty about the channel input after the channel output is observed:

$$H(X|Y) = \sum_{i,j} [P(x_i, y_j) \cdot \log_2(P(x_i|y_j))]$$

Mutual information is then interpreted as the amount of uncertainty resolved about the channel input given the channel output observed. Note that the mutual information function is an average as is the entropy function. Thus when certain elements of Y are observed, more or less information than $I(X;Y)$ may be obtained from the observation. Another interesting feature of the mutual information function is that the value 0 occurs only when $H(X)$ equals $H(X|Y)$. This occurs if the X and Y ensembles are independent. Thus zero information is transmitted through a channel only if the channel outputs are independent of the channel inputs.

The concept of mutual information is used to determine how much information can be transmitted through a channel. The maximum of the mutual information function (obtained by varying the input letter probabilities) gives the channel capacity which is the largest average amount of information that is able to be transmitted through a channel. The notion of channel capacity will be used extensively in the remainder of this thesis for analyzing covert channels in scheduling algorithms. For covert channels, the channel capacity gives the largest average amount of information that can be leaked from a confined subsystem.

Channel capacity is more rigorously defined in the Noisy Channel Coding Theorem and its converse. The Noisy Channel Coding Theorem states that if binary data is to be transmitted at rate $R < C$, for C the channel capacity, then by appropriate encoding and decoding strategies, the error at the decoder output can be made as small as desired. In other words, all information rates up to capacity can be transmitted over the channel with zero error. The converse to the theorem states that if the entropy of the data to be transmitted is greater than C , then with even the best encoder and decoder, the error probability at the decoder output can not be less than some positive number which depends on the information being sent and on C . This says that information rates greater than the channel capacity can not be sent with zero error. A more formal statement and proof of these two theorems can be found in [Gallager68]. Since this thesis will be concerned only with transmission rates which have a zero error probability, the channel capacity is the highest transmission rate with this property. This does not mean that this

rate is achievable in practice since the cost of the encoder and decoder needed to achieve capacity may be prohibitively expensive.

In the above context, the term channel does not refer to any specific type of channel but rather to any method of information transmission that can be formally defined in terms of probabilities. In this thesis, two general types of channels will be used for the covert channel analysis: the discrete memoryless channel (DMC) and the discrete channel with memory (DCWM). Both of these channel types will be used to model interactions between two processes in a computer system through a covert channel. However, the channel characteristics and the method for computing the channel capacity are different. The details are presented in the next two sections.

2.3 The Discrete Memoryless Channel

The discrete memoryless channel (DMC) is characterized by both the input ensemble, X , and the output ensemble, Y , containing discrete values, and by the channel being in the same state before each input letter (element of the X ensemble) is transmitted (i.e. the channel is memoryless). The discrete channel inputs and outputs mean that channel capacity is calculated by a summation procedure involving the inputs and outputs rather than by the integration procedure which would be necessary if the inputs and outputs were continuous. The memoryless characteristic means that the channel operation can be described by a single channel transition probability matrix (CTPM) which gives, for each input letter, the probability of each output letter being received by the receiver. If the channel characteristics change in response to the inputs transmitted through the channel, the channel is said to have memory and more than one CTPM is necessary to describe the channel as discussed in the next section. A

	y_1	y_2	y_3	y_4
x_1	a_1	a_2	a_3	a_4
x_2	b_1	b_2	b_3	b_4
x_3	c_1	c_2	c_3	c_4

Figure 2.1
A Sample CTPM

sample CTPM is shown in Figure 2.1 in which each row corresponds to the probability density of the output letters given that the input letter for the row is transmitted. Due to the definition of the CTPM, all rows must sum to 1.0. For all the CTPM's in this thesis, the channel encoder and the channel decoder, if present, are not included in the CTPM. Thus the CTPM is not necessarily square since the channel input and output ensembles are not generally the same size.

Given the CTPM for a DMC, the channel capacity is computed by adjusting the input letter probabilities in the messages to be transmitted to give the largest value of mutual information. The usual channel capacity computation is done by a computer iteration on the input probabilities which terminates when the channel capacity is computed to within a certain tolerance. The specific algorithm used in this analysis to control the iteration is the one developed by R. Blahut and explained in [Blahut72]. The algorithm starts with an initial guess of the input letter probabilities supplied by the user. This is usually the uniform probability weighting of all the inputs. The algorithm then computes the mutual information for each row of the CTPM. On the next iteration, a row hav-

ing a larger mutual information than the average is assigned a larger probability, while a row having a smaller mutual information than the average is assigned a smaller probability. The input letter probabilities are then adjusted so that the average mutual information for the channel is always increasing. When the difference between the upper bound capacity (the largest mutual information for any row) and the lower bound capacity (the average mutual information for all rows) becomes smaller than the allowable error tolerance, the algorithm terminates.

2.4 The Discrete Channel with Memory

As for the DMC, the discrete channel with memory (DCWM) also has a discrete ensemble of channel inputs, X , and a discrete ensemble of channel outputs, Y . However, the two channels differ because the DMC is in the same state before each input letter is sent (only one CTPM is required), while the DCWM may be in one of several states when an input letter is transmitted. Each state is represented by one CTPM. The CTPM that governs the transmission of any input letter is dependent on the current state of the channel. An example of a channel with memory would be a burst noise channel which is typical of telephone line transmissions. The operation of the burst noise channel is error-free a majority of the time. However, when errors do occur, they occur in clusters. This type of channel can be modeled by

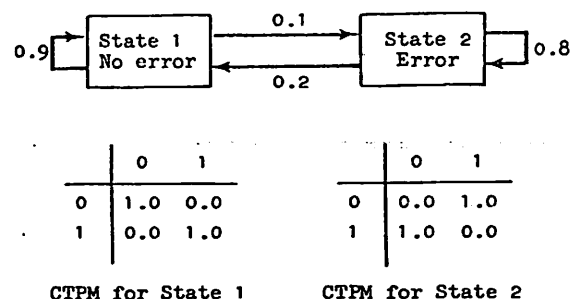


Figure 2.2
A Burst Noise Channel Model

two states [Gilbert60] as in Figure 2.2. One state (state 1) represents errorless transmission and the other state (state 2) represents totally erroneous transmission. The burst noise channel and the DCWM modeled in this thesis are represented by a Markov model. This means that the next channel state is dependent only on the current channel state and the input letter transmitted. From the state transition probabilities in Figure 2.2, the channel tends to persist in its current state with a high probability, which is characteristic of the burst noise effect.

The capacity of the general DCWM with many states is dependent on the number of consecutive channel inputs transmitted. The usual interpretation is that the number of input letters goes to infinity, but this may yield a capacity that is greater than or less than the capacity when only a finite number of letters are sent. For example, the first ten channel inputs should have one probability distribution, the next ten another, etc. However, although the simple two state model for the burst noise (or burst error) channel has been successfully solved [Gilbert60], the method can not be applied to models with more than two states, since the two state assumption is used in the analysis. Attempts have been made to develop algorithms to compute the capacity of the general channel with memory models (e.g. [Wolfowitz63]),

but no practical methods have yet been developed. Since the models in this thesis consist of a minimum of six states to provide sufficient accuracy, the channel capacity of these models can not be computed. Further restrictions on the channel must be made to permit computation of the DCWM capacity.

To circumvent the problem of computing the capacity of the DCWM, the channel operation is restricted to transmitting n input letters, then letting the channel return to some equilibrium state before transmitting the next n input letters. The equilibrium state attained before transmission of the next n input letters must always be the same. Of course, as n approaches infinity, the capacity for this method approaches the capacity for the generalized DCWM model. For the analysis in this thesis, n will be less than or equal to three. The computation time needed for the $n=4$ case is prohibitive and is measured in tens of hours of computer time.

This restricted channel operation can be modeled as a DMC with all possible channel input letter n -tuples composing the X ensemble as input letters and all possible channel output letter n -tuples composing the Y ensemble as output letters. The CTPM is obtained by iterating through all possible state sequences in the Markov channel model to obtain the necessary probabilities. A statement of the algorithm used to construct the CTPM is postponed until Chapter 5, at which time it is discussed in connection with the semi-Markov channel model used in this thesis. Once the CTPM for the restricted DCWM is found, the channel capacity can be found by using the DMC capacity algorithm in [Blahut72]. This algorithm was discussed in the previous section for the discrete memoryless channel.

Since the DCWM will be modeled as a DMC, the following sections on channel synthesis will discuss only the DMC case. The discussions are equally applicable to the restricted DCWM as defined above.

2.5 Cascaded Channels

When a channel does not have suitable characteristics for a certain application, one method of changing the characteristics is to add a channel in cascade with the original channel. The CTPM's for both the channels can be combined into one CTPM for the equivalent channel by using straightforward probability operations. For example, the cascaded channels

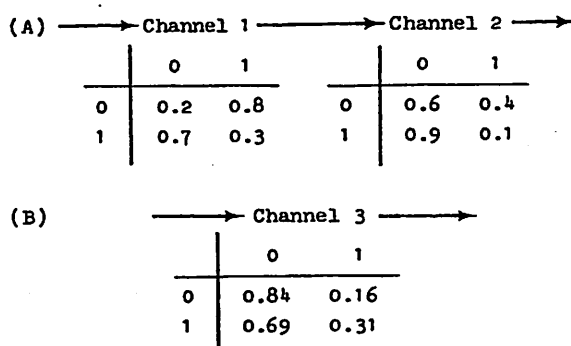


Figure 2.3

Cascading Channel 1 and Channel 2 to Form Channel 3

in Figure 2.3A can be represented by the single channel in Figure 2.3B. If $P_i(y|x)$ is the probability of the output y being observed given that the input is x in the i^{th} channel, then the CTPM for the equivalent channel (channel 3) is computed by the formulas:

$$P_3(0|0) = (P_1(0|0) P_2(0|0)) + (P_1(1|0) P_2(0|1))$$

$$P_3(1|0) = (P_1(0|0) P_2(1|0)) + (P_1(1|0) P_2(1|1))$$

$$P_3(0|1) = (P_1(0|1) P_2(0|0)) + (P_1(1|1) P_2(0|1))$$

$$P_3(1|1) = (P_1(0|1) P_2(1|0)) + (P_1(1|1) P_2(1|1))$$

For the study of covert communication channels, cascaded channels have the property that the capacity of the two channels together is less than or equal to the capacity of either one separately. This is formally proved in the Data Processing Theorem in [Gallager68]. Briefly stated, the Data Processing Theorem says that post-processing the channel outputs can only decrease the channel capacity. In the cascaded channel example of Figure 2.3A, channel 2 can be viewed as post-processing the outputs of channel 1. Thus the equivalent channel capacity is less than or equal to the capacity of channel 1. If channel 1 and channel 2 are switched, the equivalent channel is the same but the above argument says that the equivalent channel capacity is less than or equal to the capacity of channel 2. Since the objective in computer systems is to limit the capacity of covert channels, cascading channels may produce a more desirable channel.

2.6 Compound Channels

A compound channel, as defined in [Wolfowitz64], is a DMC with more than one state and with one CTPM per state in which the state governing each input letter transmission is chosen according to a probability density function. For each input letter sent, the state of the channel is independent of any previous inputs or chosen states. For this thesis, only 2-state compound channels will be used. The 2-state compound channel formulation is an alternative to cascading for combining two channels into one channel. The resulting channel characteristics are a combination of the two channels

	p_1			p_2	
	↓			↓	
	State 1			State 2	
		0 1			0 1
0		0.7 0.3		0	0.6 0.4
1		0.2 0.8		1	0.6 0.4

Figure 2.4
Compound Channel Model

used. Figure 2.4 shows an example of the compound channel considered here. When an input letter is to be sent, a p_1 biased coin is flipped to determine the channel state and hence the CTPM governing the transmission. Notice that the rows of the CTPM are different for state 1 (capacity greater than zero) while the rows of the CTPM are the same for state 2 (capacity equals zero). The capacity of the compound channel would then be expected to be less than or equal to state 1, with the amount of decrease depending on the value of p_1 . In the analysis of covert channels, state 1 represents sending a channel input through the normal covert channel and state 2 represents refusing to transmit an input letter when one can be transmitted. Since nothing is actually transmitted in state 2, the rows for state 2 are the same regardless of the input letter to be sent. This concept of channel synthesis will be explained in more detail in Chapter 6.

The DMC with the same capacity as the compound channel depends on whether the sender, the receiver, both the sender and the receiver, or neither the sender nor the receiver know which state governs the transmission of the input letter, as outlined in [Wolfowitz64]. For reasons that will

be made clear in later chapters, the sender knows which state governs the transmission but the receiver does not. However, due to the special form of state 2 (i.e. the state has all rows equal), knowing which state governs the transmission does not enable the sender to increase the capacity above the case where neither the sender nor the receiver know the state governing transmission. This can be proved by straightforward mathematical computation using the equations for both cases in [Wolfowitz64] and is not done here. To compute the CTPM for the DMC with the same capacity as the compound channel for the case in which neither the sender nor the receiver know the state of the channel, the CTPM of each state is weighted by the appropriate state probability (i.e. p_1 or p_2). For the compound channel in Figure 2.4, the capacity-equivalent DMC CTPM is given by:

$$P(0|0) = (0.7)p_1 + (0.6)(1-p_1)$$

$$P(0|1) = (0.2)p_1 + (0.6)(1-p_1)$$

$$P(1|0) = (0.3)p_1 + (0.4)(1-p_1)$$

$$P(1|1) = (0.8)p_1 + (0.4)(1-p_1)$$

The channel capacity of this DMC is then calculated using the algorithm in [Blahut74].

2.7 Cascaded Compound Channels

The last method for altering the characteristics of a channel to be considered here is to combine the cascaded channel with the compound channel. This involves first finding the capacity-equivalent CTPM for the cascaded channel representation, then using that channel as input to the compound channel synthesis. The final CTPM for the equivalent DMC channel is the equivalent CTPM from the compound channel synthesis operation. An example of a cascaded compound

		p_1		p_2	
		State 1 CTPM for the Cascaded Channels		State 2 CTPM for Zero CPU Time Allocated	
		0	1	0	1
0		0.5	0.5	0.6	0.4
1		0.1	0.9	0.6	0.4

Figure 2.5
Cascaded Compound Channel Model

channel is shown in Figure 2.5. The channel capacity for the cascaded compound channel can be found from the final CTPM by using the algorithm in [Blahut74]. The capacity for the cascaded compound channel can not be greater than the original channel and thus may be useful in partial confinement.

2.8 Summary

This section introduced the concepts of information, channels, and channel capacity which will be used often in the remaining chapters. The two different types of channels to be analyzed, the DMC and the DCWM, are introduced. The computation of the channel capacity for the DMC is done by using the algorithm in [Blahut74]. However, the capacity of the DCWM is not computable in general. Consequently a restricted form of the DCWM is used in this thesis. The particular restricted form chosen can be analyzed as a DMC from the standpoint of channel capacity and the algo-

rithm in [Blahut72] can be used.

The three channel models that will be used extensively in Chapter 6 (the cascaded model, the compound model, and the cascaded compound model) are presented and the method used to find the capacity equivalent DMC CTPM is explained. All three of these channel models can be used with either the DMC or the DCWM channel types since both types can be reduced to a single CTPM. The difference between these channels is that in the case of the DMC, the channel inputs are 1-tuples of channel input letters while for the DCWM they are n-tuples (n less than or equal to three for this analysis). The results of this chapter will be used extensively in Chapters 5 and 6 when the semi-Markov model is explained and the analysis of covert channel capacity is done.

3. SYSTEM DESIGN CONSIDERATIONS FOR COVERT CHANNEL CONFINEMENT

3.1 Characterization of Covert Channels

An executing process can usually detect more information about the state of the system than is volunteered by status-type system calls. Information about various system resources can be inferred by the process making a request for part of the resource and observing the system's response. The elapsed real time until the request is honored may be indicative of the present request load for the resource, the allocated portion may be indicative of previous requests, and the error responses (e.g. resource reserved by another user) may be indicative of the status. Thus from the perspective of an executing process, the computer system can be described by an n -tuple (n large) with components describing the current resource allocations and the state of the system resources. In this discussion, resources include both hardware entities (e.g. I/O controllers) and software entities (e.g. allocation of system table entries) that can be detected by an executing process through any means available. Typical components of this n -tuple include the process currently assigned the CPU, the resources reserved by active and inactive users, the amount and location of unassigned secondary storage, the head position of each movable head disk, and the system overlay currently resident in primary memory (assuming the operating system is arranged in overlays). All of these state components are potentially observable by the spy, though perhaps not with absolute certainty, and perhaps not all the time. For example, an observation of the disk head position may be taken at any time by timing an I/O request for a file on a known cylinder, but the number of unused file index entries may be observed only when the file index table is full and requests to create a new file are denied.

During the execution of a process, most components of this state description can be altered. For example, if the confined subsystem makes a system call to create a secondary storage file, the currently resident system overlay may be changed, a previously unassigned space on secondary storage is reserved, and an empty file index table entry is assigned to contain the description of the file. If the spy also issues the file create system call when he becomes eligible for execution, the presence of the file create overlay of the operating system may be detected by observing the system call processing time. By observing the placement of the spy's disk file and knowing the file placement policy, an educated guess about the length or other attributes of the secondary storage file created by the confined subsystem may sometimes be obtained. A file placement policy that tries to spread the disk files among the available disks by allocating each newly created file in a cyclic manner among the n physical disks is one example. That is, the first file would be created on disk 1, the second on disk 2, ..., the n^{th} on disk n , the $(n+1)^{\text{st}}$ on disk 1, etc. The information channel would then be the disk number on which the spy's newly created file resides. Subtracting this number from the disk number for the spy's previous newly created file is the channel output received by the spy. As another example, if the system has only a very few file index table entries left, the confined subsystem and the spy may also try to communicate by creating and deleting files much like the reader/writer channel in [Lampson73].

As a generalization of the above examples, a covert channel occurs when the spy is able to observe a change in the system resource state caused by the confined subsystem. But in some cases, the distinction between covert channels and storage channels is not always clear. For example, if the confined subsystem and spy are using the physical head location on a movable-head disk to send information, the

confined subsystem positions the head at cylinder 1 (by reading or writing a disk file on the first cylinder) to send a zero bit, and positions the head at cylinder 20 to send a one bit. Assuming the spy also has a disk file on cylinder 1, then by reading or writing the spy's file and observing how long the I/O operation takes, the spy can receive a bit stream of information. In this instance, the disk head channel can be classified as a storage channel [Lampson73] since the head position could be viewed as a variable that is written by the operating system (at the request of the confined subsystem) and is read by the spy. However, other users also share the disk with the confined subsystem and the spy and may cause erroneous information to be received by the spy. The erroneous information is due to this channel not being intended for information transfer, which is Lampson's definition of a covert channel.

To avoid this ambiguity, covert channels will be defined as those channels which are a result of resource allocation policies and the resource management implementation. This is different from the usual storage channels which are implemented by a process observing the contents of a resource (e.g. a shared file). It is characteristic of such channels that if a process other than the confined subsystem and spy is executing, then perfect transmission through the channel may not be possible due to the resource requests of the other process. Using this definition, the disk head channel is a covert channel since it is a result of the implementation of resource management (i.e. disk reads or writes). This interpretation is consistent with the interpretation of storage channels in [Denning76] and [Lipner75], and classifies all the usual interpretations of covert channels correctly. This definition of a covert channel is used throughout the rest of this thesis.

It is worthwhile to note at this point that the outputs from covert channels do not always involve time as the parameter observed by the spy for information transmission. In the case of the disk file placement channel above, the spy only needs to know on which disk his newly created file resides for information to be passed. This information may be readily available from the operating system to enable optimized programs to better utilize the I/O configuration when multiple files are being read or written simultaneously. Since real time is not the only method for sending information through covert channels, any solution for the covert channel problem must do more than prevent the spy from observing real time as suggested in [Lipner75].

3.2 Absolute vs. Partial Confinement

The major drawback in providing confinement of covert channels in a system design is the cost to both the confiner and the other users of the system. The cost to the confiner is the amount of extra resources necessary for execution and the cost to the other users is the system degradation caused by the confined subsystem. There are two types of confinement which can be provided. The most expensive type is *absolute confinement* which insures that no information is leaked through any covert channel. The other type is *partial confinement* which permits some non-zero amount of information leakage at a reduced confined subsystem execution cost to the confiner. The more leakage allowed, the lower the resultant cost.

3.2.1 Consequences of Absolute Confinement

The reason for the large cost of absolute confinement is that system decisions for an absolutely confined subsystem must not be based on the confined subsystem's status or requests. In the case of memory management in a paged environment, the working set size assigned to the confined subsystem can not depend on the page fault history of the

confined subsystem. This usually causes either too large a working set size to be allocated, which denies other processes the memory pages, or too small a working set size to be allocated, which causes the confined subsystem to thrash. Another consideration is that even read-only pages brought into memory may be utilized as a covert channel if memory pages are shared between processes. This means that the confined subsystem must not affect the page sharing mechanism used by the unconfined processes. Another aspect of absolute confinement is that the maximum amount of resources needed for the confined subsystem execution must be made available to the confined subsystem for its entire execution. The reason is that once execution begins, additional resources can not be requested by the confined subsystem without potential information leakage. Of course, additional resources can be allocated during execution but based only on an algorithm that is independent of the input parameters and the confined subsystem's status. This requires the confiner (or the confined subsystem before the input parameters are read) to know what the resource requirements for the invocation will be in order to execute the confined subsystem in an efficient manner. However, this violates the concept of programming generality [Dennis68], since it requires the confiner to have more detailed information about the confined subsystem than just its calling sequence.

In addition to enforcing confined subsystem-independent resource allocation, the operating system must be sure that there is no flow of information from the confined subsystem to any subsystem other than the confiner. This includes prohibiting information flow from one instance (or incarnation as defined in [Spier73]) of the confined subsystem to another. However, information flow from any unconfined subsystem to the confined subsystem does not need to be regulated. As observed in [Lampson73], these restrictions also apply to any subsystem in the subsystem-call subtree with the confined subsystem as the root. This is called *transitivity of confinement*.

One consequence of the information flow restrictions is that the confined subsystem can not synchronize with any subsystem that is not invoked directly or indirectly by the confined subsystem unless specifically authorized to do so by the confiner. The ability to synchronize can leak information through a storage channel as shown in [Lampson73]. However, a confined proprietary subsystem might need to read a proprietary data base in order to perform its task. If the data base may be updated during the confined subsystem execution so as to render the entire data base inconsistent during the update, there is no general method for allowing a confined subsystem to always read a consistent data base representation. A scheme that time-stamps each record (or the entire data base) before and after it is updated so the confined subsystem knows that the record (or data base) has been updated can be used for cases in which the updates occur infrequently. The confined subsystem would be required to verify that the record (or data base) was not modified while the information was read. If the record was modified, the operation should be repeated. If the data base is updated frequently, this method of reading may not be a viable approach. In this case, the data base can be reserved at times and for durations that are independent of the confined subsystem execution. When the confined subsystem must read the data base, no more useful CPU time is allocated until one of the reservation times occurs.

The resource allocation and synchronization problems are typical of the restrictions inherent in absolute confinement. To permit synchronization or resource allocation in response to a request from the confined subsystem, some information may be leaked. In certain circumstances,

the trade-off of information leakage versus increased program efficiency might be acceptable. This issue will be discussed in great detail in Chapter 6 at which time the possible trade-offs will be discussed. It is interesting to note that even though synchronization represents a storage channel, it can be analyzed by using the methods developed for covert channels, as will be seen in the next section.

There are several implementations of absolute confinement for covert channels. The simplest implementation is to deny the confined subsystem the use of the resource on which the channel is based. However, in the case of the CPU resource, denial of usage will prevent execution and can not be used. Another implementation is to provide the confined subsystem access to the covert channel based on some probability density function that is independent of the confined subsystem's execution. If it is known that the confined subsystem needs more secondary storage as execution proceeds, an additional unit could be allocated with probability x every y real seconds until the confined subsystem completes. This scheme does allow more resources to be allocated, but at a potential cost of allocated but not used resources.

Lampson [Lampson73] proposes implementing confinement by having the covert channel input be confiner specified (masking). This is permissible as long as the confiner realizes that information may be inadvertently leaked through the specification. For example, if the confiner follows the advice of the confined subsystem author as to how resources should be allocated for certain input parameter combinations, then some inputs may be leaked by the spy observing the confiner-specified resource allocations. An example is a tax program where the specifications state that a memory size of 100,000 words should be requested if the confiner's income is less than \$20,000; and 200,000 words otherwise. The extra space is purported to be necessary for the tax computation to handle an expected larger number of deductions. However, following these specifications may leak one bit of information on the confiner's income.

Another implementation of absolute confinement consists of partitioning the resources so that two subsystems which must not communicate (e.g. the confined subsystem and its confederate spy) do not draw resources from the same partition. This strategy prevents the spy from observing the resource allocation patterns of the confined subsystem. For example, the resources for the confined subsystem would be in one partition and the resources for unconfined users in another. The confined subsystem can vary his actual resource usage only up to the maximum amount in the partition. This implementation is also a form of masking if the partition is created by the confiner before invocation of the confined subsystem. One system implemented with this resource allocation policy is Hansen's RC4000 multiprogramming system [Hansen70] in which the confined subsystem's resources are a subset of those allocated to the confiner. The partitioning method violates the objectives of programming generality, as noted before.

Another implementation of absolute confinement is to allocate resources to the confined subsystem based on a function of the resource usage by unconfined subsystems. This does not violate confinement since the unconfined processes can transmit information freely to the confined subsystem. This implementation permits the system to discriminate against confined subsystems in order to provide better service to the unconfined subsystems. Such a policy would be useful in CPU scheduling where the amount of CPU time allocated to confined subsystems could be decreased as the percentage of unconfined subsystems increases. When the workload is light, the system can afford CPU time to be possibly wasted

by confined subsystems and still provide an adequate response time for unconfined processes.

The last implementation of absolute confinement is the standard state approach. In this method, all system state variables potentially used by the confined subsystem must be reset to a standard state after the confined subsystem executes and before the next process is executed. For movable head disks, the head must be positioned at whatever cylinder is designated as the standard cylinder. The contents and allocation of the cache memory, if any, must be reset (e.g. purged). If this implementation is used, the spy may be able to detect when the confined subsystem executes by observing that some of the state variables are in the standard state. However, this does not convey any information since the standard state is independent of the service execution and the times at which the confined subsystem executes are chosen independently of the requirements of the confined subsystem. The detection of the confined subsystem execution will be a factor under partial confinement and a discussion of this factor is deferred until then.

3.2.2 Consequences of Partial Confinement

The most significant component of the extra cost for providing absolute confinement results from the inability of the operating system to dynamically respond to the resource requests of the confined subsystem without leaking information. To decrease this cost, partial confinement allows the resource allocation decisions for the confined subsystem to be partially dependent on the needs of the confined subsystem. As the dependence increases, the amount of information transmitted reliably through the covert channel (channel capacity) increases. This leads to a trade-off between channel capacity and cost.

Partial confinement can be usefully applied in two situations as long as the amount of information leakage can be measured. The first occurs when the confidential information given to the confined subsystem will become public knowledge after a certain period of time. Confinement is used in this case to ensure that no significant information is leaked in the interim. One example is a stock market appli-

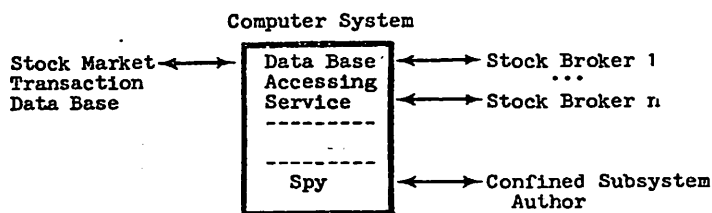


Figure 3.1

Stock Market Application for Partial Confinement

cation (Figure 3.1) in which stock brokers enter their pending stock transfer requests into a transaction data base by interacting with a partially confined data base access subsystem. The data base is used to record the transfers, to keep track of the requests as they are being filled, and to perform the billing for handling the transaction. However, the in-house author of the data base accessing subsystem quickly realizes that the information in the transaction data base can be used to enlarge the profit from his own portfolio of stocks. Advance information on large stock transfers enables the author to sell a stock at a higher price just before a client sells a large block of the same stock, and to buy the stock at a lower price after the large block is offered for sale. The price change is due to supply and demand. A similar scenario ap-

plies to a large sell request by a client. If the subsystem author can execute programs on the same machine as the stock market application when the stock market is open (i.e. the machine has too much capacity for just the stock market application and the excess is used for debugging new software), leakage may result from the stock market subsystem to the author's other program since they are executed concurrently. In this case, if the covert channel capacity is small enough, the information can not be transmitted quickly enough for the confined subsystem author to take advantage of pending stock transfers.

The second situation where partial confinement is useful occurs when the total amount of leakage by the confined subsystem is "small" for the particular application. As the sensitivity of the application increases, the number of bits that can be leaked decreases. In the most sensitive applications, no bits can be leaked. For example, the cost to the confiner of the information leaked by partial confinement may be balanced by the system performance gain. In this approach a value is placed on the information provided as input to the confined subsystem [Turn72]. If the lowered computing cost from partial confinement compares favorably with this value, then partial confinement is advantageous.

Partial confinement is implemented by adding noise to covert channels to reduce the capacity to a permissible level. A typical representation of a covert channel is shown in Fig-

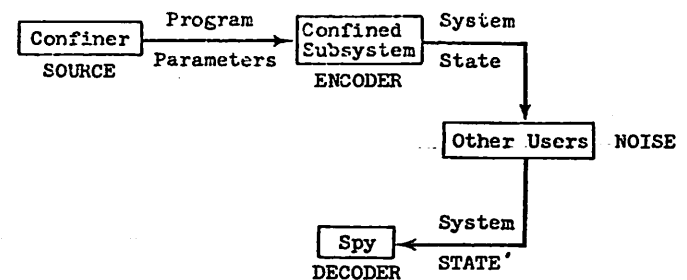


Figure 3.2

Typical Representation of a Covert Channel

ure 3.2. In covert channels, the confiner (information source) provides inputs to the confined subsystem (encoder) which encodes them into a sequence of system states (of length n for example) that will be transmitted by the confined subsystem during his next n executions. The spy will attempt to deduce the system state left by the confined subsystem. However, when the spy executes next, the system state seen by the spy may be modified by other users of the system which executed between the confined subsystem and the spy. Thus the system state actually seen by the spy is $STATE'$ in Figure 3.2 which may or may not be the system state left by the confined subsystem. The spy observes $STATE'$ and decodes the observation into a guess of the system state immediately after the confined subsystem execution. If there are no other users and the operating system does not try to limit the information passed between the confined subsystem and the spy, then a transmission stream containing no decoding errors may be received by the spy. As the number of other users in the system increases, the noise injected into the channel increases and the channel capacity should decrease. Under partial confinement, it is desirable for the operating system to insure that as the number of users fluctuates, the channel capacity becomes no greater than the value specified as acceptable by the confiner. This is accomplished by the operating system effectively adding another

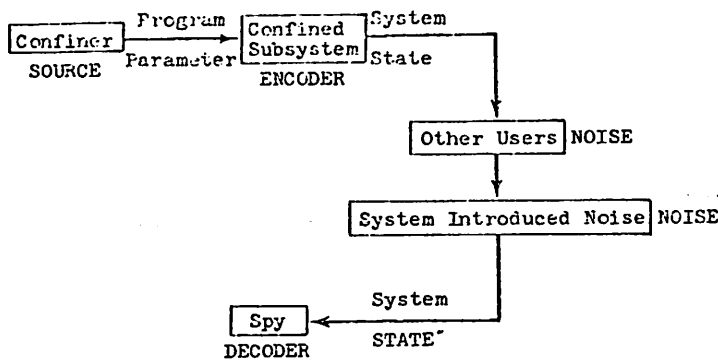


Figure 3.3
Covert Channel Representation for a
System Offering Partial Confinement

stage to the channel as shown in Figure 3.3. Here, the operating system-generated noise (NOISE2) is cascaded with the noise generated by other users (NOISE1). By the Data Processing Theorem given in Chapter 2, this new channel (Figure 3.3) has a capacity less than or equal to the channel with either NOISE1 or NOISE2 alone. As the capacity of the system noise component ranges from zero to one, the entire channel capacity can be adjusted from zero to a maximum of the capacity of the channel when the given number of users is present.

The cascaded channel representation of partial confinement in Figure 3.3 is the basis for the method to be used to allocate the confined subsystem more resources during execution and to determine the maximum amount of information leakage possible. As an example of the analysis that can be done, assume that the working set size of the confined subsystem varies over a large range during execution. This characteristic would be a motivation for the confiner to optimize the working set size allocated to the confined subsystem to permit efficient execution. In this situation, the channel transition probability matrix for the system noise component of the covert channel can be represented

Inputs \ Outputs	Outputs		
	Increase working set	Decrease working set	No change
Larger working set size needed	A1	A2	A3
Smaller working set size needed	B1	B2	B3
No change needed	C1	C2	C3

Figure 3.4
Channel Transition Probability Matrix for the
System Noise in the Working Set Size Channel

ed as in Figure 3.4. The inputs to the matrix are obtained from the executing confined subsystem. If excessive thrashing occurs, the input to the matrix is that a larger working set size is needed. If very few page faults occur, the input is that the working set size can be decreased. Otherwise the input is that no change necessary. The channel operates by the system questioning the confined subsystem at specific times (e.g. after every n seconds of allocated CPU time) and determin-

ing the confined subsystem's input to the working set size channel. A random number generator is used to choose the channel output based on the probabilities in the channel transition probability matrix and the input. The memory manager will then modify the working set size of the service according to the channel output. The channel capacity is obtained from the channel transition probability matrix (CTPM) in Figure 3.4 (the system noise channel component) and an experimentally observed CTPM which models the effects of the other users. As the values of A1, B2, and C3 in Figure 3.4 approach 1.0 (perfect response of the working set size to the wishes of the confined subsystem), errorless transmission through the system noise channel component is approached and the covert channel as a whole approaches the capacity of the CTPM representing the other system users. As the condition $A1 = B1 = C1$, $A2 = B2 = C2$, and $A3 = B3 = C3$ is approached, the capacity of the system noise channel approaches zero (as noted in Chapter 2) as does the covert channel as a whole. This demonstrates the approach to be used for limiting the capacity of the CPU scheduling channel in Chapter 6.

3.3 Designing a System Offering Confinement

A typical operating system has an extremely large number of possible covert channels. The large number is due to the number of components in the n -tuple describing the state of the system resources. For each component, there are generally many different ways of encoding information in the resource state. It is also a simple matter for the confined subsystem to manipulate many different channels at once. For example, in some systems when a process issues an I/O request then blocks, the I/O subsystem is notified of the operation to be performed, the memory manager is told not to swap the process out while the I/O is in progress, and the CPU scheduler is told not to allocate any more CPU time until the I/O completes. Thus one system call may change the state of more than one operating system subsystem. Each of these changes may be independently observed by the spy. In this example, the spy may be prevented from increasing his working set size during the I/O operation since some primary memory is reserved by the confined subsystem for the duration of the I/O request, the spy is able to receive more CPU cycles since the confined subsystem has blocked, and the I/O channel servicing the confined subsystem's request can not honor another request until the confined subsystem's request completes.

One approach to the system design problem is to limit the number and types of covert channels by enforcing rigid interaction rules between the confined subsystem and the operating system. This limits the amount of information originating from the confined subsystem that is used in operating system decisions. This "confined subsystem interface" is much like the interface for a virtual machine implementation [Buzen73] since all but the most important confined subsystem inputs have been omitted. This interface is the only method by which the confined subsystem can communicate with the operating system. The specific interface to be implemented depends on which covert channels are the most important in optimizing the system performance, or in lowering the cost of confined execution. One important covert channel is implemented by the CPU scheduling algorithm, since the CPU is essential for processes to make execution headway. If the CPU is allocated to the confined subsystem frequently when the confined subsystem can not use it, the response time of all other processes is adversely affected, with no increase in the amount of total useful work done. Other covert channel choices might be the memory management channel or the disk I/O scheduling strategy. The exact

choice depends on the type of workload being processed and the resources available. Any information about the confined subsystem that is not included in the interface can not be used by the operating system in making resource allocation decisions. This necessitates using one of the absolute confinement resource allocation methods (e.g. masking or partitioning) to allocate resources for which insufficient information is available in the interface.

The methods used to absolutely confine the non-critical channels can make a difference in the capacity of the partially confined channels since different channel models represent different absolute confinement strategies. If the confined subsystem and spy can synchronize (i.e. if the spy knows whether the confined subsystem executed between two successive spy quantum allocations), then the channel capacity is generally greater than if no synchronization occurs. An example of a channel model where synchronization is not guaranteed will be discussed in Chapter 6 and its capacity contrasted with the synchronized case. Synchronization observations by themselves do not leak information if the time between confined subsystem executions is absolutely confined, but they do enable the spy to better decode the observed outputs. Of the absolute confinement strategies previously discussed, only partitioning and masking (depending on the algorithm specified by the confiner) do not provide synchronization. The reason is that the state of the objects manipulated by the confined subsystem in these cases can not be observed by the spy since the spy's resource allocations do not change. In the other strategies, there is a possibility that the absolute confinement implementation would tell the spy whether the confined subsystem executed. For example, suppose the standard state method is used for absolutely confining the cache memory covert channel on a system having a cache memory. Here it is assumed that several different processes can hold parts of the cache at one time based on their storage references, and that the cache is not purged after a process has finished its quantum. Instead, the cache memory pages slowly age out of the cache using a least-recently-used policy. Information is transmitted over this channel by the confined subsystem reserving as much of the cache as possible during execution (large program locality) to send a one bit and as little of the cache as possible (small program locality) to send a zero bit. One standard state confining strategy would be to zero the cache after every confined subsystem execution so that the amount of the cache used by the confined subsystem can not be observed by the spy. But by observing its headway, the spy process would be able to detect a difference in execution speed depending on whether or not any of his memory pages are in the cache when his quantum begins. When the spy notices that none of his pages are in the cache at the beginning of a quantum, then the spy assumes that the confined subsystem has executed since the spy's last quantum allocation. This method of synchronization may not be completely accurate since the number of spy pages in the cache is affected by other user executions, but it may be unlikely that all of the spy's memory pages would age out of a large cache between two successive spy executions on a particular system; on such a system, synchronization would be possible.

3.4 Covert Channels Outside the System

With the above mechanism for partial confinement, there are still channels through which information may be covertly passed. These channels occur at the boundary of the machine and its outside environment and may not be able to be blocked by the operating system. The existence of these channels is due to the confined subsystem affecting the outside environment during execution. The most common ex-

ample of these channels is covert leakage through electromagnetic radiation [Ware67]. Instead of the spy being a concurrently executing process, the spy is a radiation scanner which records the emanations from the computer. Information is transmitted by the confined subsystem performing a set of operations that have a low probability in the normal workload as a start signal, with the message immediately following. This channel can be blocked by using screens to block all radiations from the machine.

A more subtle channel is implemented by the spy taking advantage of some computer center policy to observe outputs from the confined subsystem. For example, suppose the normal computer center policy is to write the date and time on a tape when it is mounted to provide tape usage information. The covert channel in this case is implemented by the confined subsystem requesting a certain tape to be mounted to send a one bit and not requesting the tape to send a zero bit. If the spy has access to the tape vault where the tape is stored and can read the date last mounted, information can be leaked. A larger bandwidth channel occurs if any one of a large number of tapes can be requested. In general, information could be obtained by the spy merely standing in the computer room while the confined subsystem executes. Such channels can only be blocked by instituting physical protection procedures so that unauthorized persons can not enter or see into the computer room. In addition, the interface of the computer room with the user community (e.g. entry to the tape vault) must be strictly controlled.

3.5 Summary

Covert channels are implemented by the confined subsystem altering the state of the system resources. For any one resource there may be many covert channels. An example is a physical disk where the disk head placement, the file placement policy and the controller contention are all possible covert channels. To provide confinement, all these channels must be blocked with either an absolute or a partial confinement mechanism. In designing a system offering partial confinement, a small number of the channels with the greatest impact on system performance should be partially confined with the remainder being absolutely confined. However, it is not enough to just block the channels observable by an executing spy process. Physical security is necessary to prevent the spy from observing anything in the computer room that could be a result of the confined subsystem's execution.

4. THE COVERT CPU SCHEDULING CHANNEL

4.1 Transmitting Information Through the Scheduler

One important covert channel in timesharing systems is the channel implemented by the CPU scheduling policy. Effective CPU scheduling is usually necessary for good system performance. However, absolute confinement obtained by masking requires allocating the CPU to a confined subsystem even though the CPU is not needed and other processes are waiting for the CPU. Wasting CPU time on the confined subsystem degrades the system response time, which is an important part of the productivity of the system as seen by the user. It may often be an important objective in a system design to minimize the impact of the confined subsystem on the response time of the non-confined users and at the same time to provide an acceptable response time to the confined subsystem. This chapter discusses how information is transmitted through the CPU scheduler and outlines the CPU scheduling algorithms to be analyzed in subsequent chapters.

A model of the information channel implemented by the

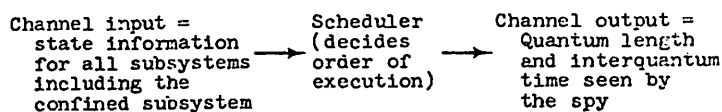


Figure 4.1
The CPU Scheduling Channel

CPU scheduler is shown in Figure 4.1. In this model, the CPU scheduler is viewed as accepting input in the form of process state information from the confined subsystem and the other jobs in the system, and deciding on the order in which all jobs will execute. For example, a pure priority scheduler uses only the priority of the jobs and their status (i.e. whether a job is waiting for the CPU or not) to determine the order of execution. This order of execution is dynamically changed to reflect changes in the state of the active processes consistent with the scheduling policy. For example, in round robin scheduling, a process that uses its CPU quantum is placed at the end of the execution list. Covert information is then transmitted through the scheduler by the confined subsystem affecting the position of the spy in the order of execution. For our analysis, the order of execution is assumed to be hidden from the spy. Thus the number of ways the confined subsystem can send information through the scheduler is limited. The spy can only tell when he is at the top of the list (the spy is executing) or not at the top of the list (the spy is not executing). This limits the channel outputs to observations of the real time between quanta allocated to the spy.

There are three methods for transmitting information by using real time information. Of the three, two can be analyzed using the same channel model while the third does not yield readily to analysis and must be totally blocked by the confinement mechanism. In the first method, the confined subsystem varies the amount of non-zero CPU time used each quantum to send different code letters through the channel. The amount of CPU time used will vary from one quantum to the next as successive bits of information are transmitted. The amount of a quantum used by the confined subsystem will affect the time between successive spy executions. For example, if the scheduling policy uses a quantum length of one second, a simple channel encoding of the information to be sent is for the confined subsystem to use all of

the quantum to send a 1 bit and only a small part of the quantum to send a zero bit. A generalization of this strategy is for the confined subsystem to use $1X$ seconds to send a zero, $2X$ to send a one, $3X$ to send a two, ..., and $(n+1)X$ to send an n where X is no greater than one-nth. of the quantum length and is dictated by the channel characteristics.

At this point it is instructive to determine the maximum bit rate using reasonable system parameter assumptions to emphasize the significance of this channel. The maximum bit rate for the scheduling channel occurs if only the confined subsystem and the spy are executing, and when both subsystems reside in primary memory at the same time. If we assume that process switching overhead is randomly selected from the range 300 to 500 microseconds (the exact number depends on the system overhead functions performed during the switch), then the noise injected by process switching into the channel ranges from a minimum of 600 to a maximum of 1000 microseconds (two process switches) per code letter transmitted. For this noise level, a suitable value of X in the last paragraph would be 401 microseconds ($= 1000-600+1$). To transmit the maximum bit rate, the real time per code letter sent must be minimized. This is accomplished by the confined subsystem encoding information into the two shortest quantum lengths which are one microsecond for the 0 bit (assuming the cycle time of the machine is one microsecond) and 401 microseconds for the 1 bit. The capacity of this channel is one bit per channel use since the channel outputs are always correctly decoded. If the inputs are equally likely and the spy is assumed to take 1 millisecond to execute each quantum, then the average channel bit rate is 1 bit/(201 microseconds for the average service execution time + 800 microseconds for context switching + 1000 microseconds for the spy execution) = 500 bits/real second. At this rate, over eight 10-character/second teletypes could be continuously printing with a 6 bit character set. Thus the rate of information flow through this channel can be significant. The average channel bit rate in normal operation with other users on the system will depend on the system workload.

The second method for sending information through the CPU scheduler is by encoding information in the time between two successive confined subsystem quanta (the interquantum time channel). One realization of this channel is

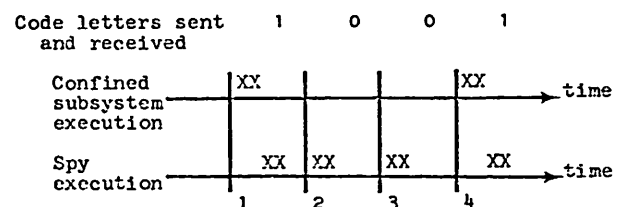


Figure 4.2
The Interquantum Time Channel

shown in Figure 4.2 for the case in which only the confined subsystem and the spy are assumed to be in the CPU queue. To send information, the confined subsystem and the spy agree on the set of times (denoted by t_1 , t_2 , t_3 , and t_4) at which information is to be sent. The transmission strategy is for the confined subsystem to be executing at time t_i if the i^{th} bit is a one and to not be executing at time t_i if the i^{th} bit is a zero. The confined subsystem and spy regulate the times at which they execute by blocking until some event (such as a certain real time) occurs. This feature, which allows a process to interrupt its own execution for a specified

amount of time, is present in many systems in the form of a "suspend" or "sleep" system call. This execution suspension enables the confined subsystem and the spy to guarantee that they will be executing at a specified real time. The spy tests whether the confined subsystem is executing at time t_1 by awaking just after the agreed upon time. If the confined subsystem has previously awakened and is executing, the spy will observe a delay in the beginning of his execution. In Figure 4.2, the spy's execution is delayed at times t_1 and t_4 but not at times t_2 and t_3 . The message sent is then 1001.

The analysis of the interquantum length channel does not require a different channel model from the quantum length channel above. The difference between the two channels is that the quantum length channel sends information by varying its non-zero CPU quantum length while the interquantum time channel sends information by the number of times a zero quantum length is allocated. These two channels can be combined if the allowed quantum length for the confined subsystem can be both zero and non-zero. When a zero length quantum is allocated, the interquantum channel is being used by the confined subsystem, and when a non-zero length is allocated, the quantum length channel is being used. In the following analysis, the equivalent channel model will be used rather than the two separate models.

The third method for transmitting information through the CPU scheduler is by the confined subsystem encoding information in its total execution time. The total confined subsystem execution time may be leaked by the spy noticing a variation in his quantum allocations due to the termination of the confined subsystem. For example, observing the equilibrium steady state of the system (i.e. the distribution of the spy's interquantum times), the spy might be able to determine when the confined subsystem is no longer receiving CPU time. That is, when the system becomes less busy than before, the confined subsystem has stopped execution. The strategy used to block this channel depends on whether the channel is to be part of the channel model (partially confined) or not (absolutely confined). For absolute confinement, the most common blocking method is for the invoker of the confined subsystem (confiner) to specify the time limit. This can be done by either specifying a time limit for the entire terminal session with the timesharing system or specifying a time limit for just the confined subsystem execution. If the entire terminal session is given a time limit, a long running confined subsystem can affect the terminal session length, causing a longer or shorter session to be needed than planned. If a shorter session is needed than was initially specified, the remaining time will be wasted, but with no unexpected information leakage. If a longer session is needed than was initially specified, at least one additional bit of information could be leaked if the confined subsystem is restarted when the time limit expires. The restart could be noticed by the spy as an addition to the workload that affects the spy's quantum allocations. Therefore, the confiner must be wary about the time limit specified and about requesting additional CPU time at the expiration of the time limit.

The alternative to confining the entire terminal session is to confine each confined subsystem execution with its own time limit. The same considerations apply here as for the terminal session time limit. If the confined subsystem finishes early, the remaining time must be allocated even though it can not be used. The confiner may submit the next task to be done to utilize the remaining CPU time, but under the same constraints as are imposed on the confined subsystem. If the confined subsystem requires more time than allocated, the confiner may restart the subsystem but at the risk of leaking at least one bit of additional information.

The advantage of specifying a time limit for the entire

terminal session is that other processes besides the confined subsystem may cause the terminal session length to vary. Thus if the terminal session is continued and the spy notices the continuation, the spy is not certain that the continuation is caused by the confined subsystem trying to leak information. In effect, there is a possibility that more noise is present than might be expected. However, in the worst case the assumption would be that the spy knows about all unconfined processes and that the amount of extra noise is zero. In the confined subsystem time limit case, the spy does know the continuation is a direct result of the confined subsystem execution.

The time limit channel can also be treated as a partially confined channel, which makes the scheduler channel model consist of two parallel channels: the quantum length channel and the time limit channel. The channel capacity would then be the sum of the capacity of each of these channels. For this thesis, it will be assumed that the time limit channel is absolutely confined and that the channel model consists of only the quantum length channel discussed above. If the analysis of the parallel channel case is desired, the techniques of Chapter 6 can be applied to each of the parallel channels separately to obtain the channel capacity.

One consideration in the design of scheduling algorithms is that transmission noise can often be decreased if the scheduler uses process characteristics in its scheduling decision. If the spy knows the property used in scheduling, he can make the spy process assume any value of the property to trick the scheduler into decreasing the channel noise normally generated by the independent processes. This strategy is analogous to Coffman's scheduling countermeasures [Coffman68] for obtaining preferential scheduling for processes. For a priority scheduler in which the process priority is selected from a segment of the real line (e.g. 0.0 to 1.0 inclusive), the spy can decrease the channel noise by setting the priority of the spy process to just less than that of the confined subsystem. Information is transmitted through the quantum length channel by the confined subsystem blocking after using the CPU time required to send a bit, which causes the spy to begin execution. Execution of the confined subsystem and spy alternate to continuously send information. In this priority scheduler, processes with a priority less than the confined subsystem and spy do not contribute noise to the channel. If the priority of the confined subsystem is high enough to block out all other processes, perfect transmission can be obtained. Similar strategies can be used for schedulers using memory length, number of page faults, CPU time estimates or working set size as the scheduling property.

4.2 Restrictions for Confined Scheduling

To offer either absolute or partial confinement, a scheduler must guarantee that the maximum amount of leakage is calculable. For absolute confinement, the channel capacity is measurable (i.e. zero) if and only if the scheduling decision is made independent of any inputs provided by the executing confined subsystem. In this case the channel transition probability matrix (CTPM) is not needed to compute the channel capacity. However, for partial confinement, the channel transition probability matrix must be obtained. With the CTPM, the calculation of the capacity is straightforward. In some cases of interest, the CTPM is so large that constraints must be placed on the scheduling channel to permit the CTPM to be calculated. In the quantum length channel, the confined subsystem might be able to choose any amount of an allocated CPU quantum with a grain-of-time of one machine cycle as the channel input. The real time outputs observed by the spy will also have a grain-of-time of one

machine cycle. In this case, the discrete memoryless channel capacity is not computable in practice when users other than the confined subsystem and the spy are executing, since the elements of the CTPM can not be measured with an acceptable error. In a computer system with a CPU quantum length of 0.1 second and a machine cycle time of one microsecond, 100,000 probability distributions must be measured to obtain the CTPM (one for each possible input). Of course if there is no channel noise (i.e. constant process switching time and only the confined subsystem and the spy are executing), the CTPM can be found but the channel capacity can only be computed if the confined subsystem uses a "small" number of channel inputs (e.g. less than 50). The limitation on the capacity calculation here is the number of iterations needed to find the channel input probabilities that achieve capacity. For these reasons, the straightforward channel capacity calculation method can not be used on a confined subsystem that is not constrained in some manner.

Two restrictions that together allow the channel capacity to be calculated are the limiting of the channel outputs observable by the spy to be a multiple of some fixed constant, and the limiting of the number of possible quantum lengths allocated to the confined subsystem to some small number. The spy restriction is implemented by the operating system starting each user's quantum only at a time that is some multiple of the chosen constant. This policy does cause CPU time to be wasted during process switching, but this is viewed as part of the cost of offering confinement. Another source of wasted CPU time results from the small number of quantum lengths that can be allocated to the confined subsystem. Since the amount of a quantum used by the confined subsystem during execution varies as a function of hardware contention and the task to be performed, it is unlikely that the chosen confined subsystem quantum is always exactly correct. Some CPU time is normally wasted to pad the confined subsystem's CPU requirement to one of the allocatable lengths. In both these restrictions, the wasted CPU time is traded for the ability to calculate the channel capacity. The discrete channel model that results from these restrictions is discussed in detail in Chapter 5.

Both of these restrictions are necessary to make the CTPM computable. If only the times at which processes start execution is discretized, then the confined subsystem can execute for any multiple of a machine cycle. Each of these execution times must be viewed as a different channel input which causes the number of inputs to be too large for a capacity calculation to be made. If only the CPU execution time is quantized, then the variable swapping time, which is a multiple of a machine cycle, causes the spy to begin execution at any multiple of a machine cycle. The channel output, in this case, is defined to be the time between the end of one spy execution and the beginning of the next. This time interval is in multiples of the machine cycle time which leads to a large number of channel outputs. Thus the channel capacity is not computable here due to the large number of outputs. Both restrictions limit the channel inputs and outputs making the CTPM computable.

4.3 Scheduling Algorithms to be Analyzed

Three scheduling algorithms are studied to determine their effect on the capacity of the covert scheduling channel and on the cost of confinement. The algorithms are variations of first-come-first-served (FCFS), round-robin (RR), and two-level feedback (FB). FCFS and RR are examples of non-priority-type algorithms, while FB is a priority-type algorithm in which the priorities are based on allocated CPU time. In all the algorithms, the confined subsystem is handled differently from other processes to permit measurements

of the covert channel capacity. The non-confined subsystems are handled as expected for each algorithm.

4.3.1 First-Come-First-Served Scheduling

The first-come-first-served scheduling policy is implemented by a single queue of active processes waiting for the CPU. New arrivals enter at the end of the queue and the next process to be serviced is taken from the front of the queue (i.e. the process having been in the queue the longest amount of time). When a process is selected for execution, the CPU is allocated to the process until its CPU requirement is satisfied. After completing its CPU requirement, the process normally exits the CPU queue. However, when the CPU requirement of the confined subsystem is satisfied, it is placed at the end of the CPU queue rather than being deleted from the queue. When the confined subsystem is selected for execution, the required amount of CPU time is allocated to satisfy its request (possibly zero CPU time), and this cycle is repeated for the confined subsystem until the entire subsystem execution is terminated. The confined subsystem can only be removed from the CPU queue when its time limit expires. This continuous circulation in the CPU queue is done for the confined subsystem to model the information channel as a quantum length channel only, and not as two parallel channels (the quantum length and interquantum time channels) as discussed earlier in this chapter.

Information is sent through the FCFS scheduler by the confined subsystem providing inputs to the quantum length channel by varying the amount of CPU time required for execution. The spy's channel outputs are obtained by determining the real time between two successive quantum allocations. Due to the implementation of the FCFS scheduler, the confined subsystem is guaranteed to execute exactly once between every two successive spy quanta, provided the spy process causes itself to be placed at the end of the CPU queue immediately after its quantum terminates. Thus synchronization between the confined subsystem and the spy is automatically provided by the scheduler, even if an absolute confinement scheduling algorithm is used to block other covert channels. This makes decoding the channel outputs easier than for the case in which the spy does not know if the confined subsystem has executed between two successive spy executions. A channel model that implements such a case is explored in Chapter 6 and is shown to indeed yield a smaller capacity.

4.3.2 Round-Robin Scheduling

Round-robin (RR) scheduling is implemented like FCFS scheduling with the exception that at most Q CPU seconds is allocated at any one time (Q is the quantum length of the scheduler). If the process does not complete in the allotted CPU time, then it is placed at the end of the queue to await

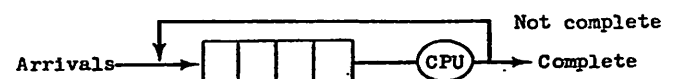


Figure 4.3
Round Robin Scheduling

another quantum allocation (Figure 4.3). As for FCFS, when a non-confined process terminates, it is deleted from the CPU queue until more CPU time is required. Again, the confined subsystem is treated specially since it is always returned to the end of the CPU queue after execution regardless of whether it completed its CPU requirement or not.

The confined subsystem can only be removed from the CPU queue when its time limit expires.

As for FCFS, the confined subsystem is guaranteed to execute exactly once between two spy executions provided the spy process always returns to the end of the CPU queue after its CPU quantum. This fact helps the spy to decode information sent through the quantum length channel.

In RR, the percentage of the CPU used by the confined subsystem is inversely proportional to the number of active processes. This is the type of behavior that is desirable in a scheduler for confinement since the confined subsystem may waste a large percentage of its allocated CPU time under some confined scheduling strategies such as absolute confinement. RR also does not use any properties of the processes in the scheduling decision. Thus the spy process can not obtain preferential scheduling to decrease the channel noise as can occur in other types of schedulers.

4.3.3 Feedback Scheduling

The two-level feedback scheduling algorithm is imple-

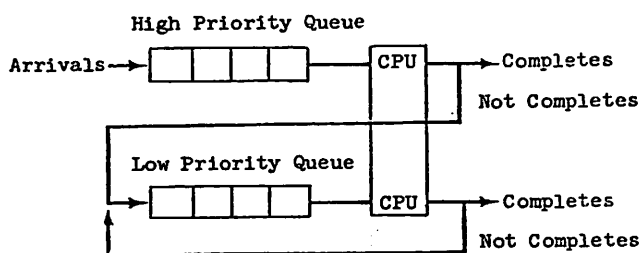


Figure 4.4
Feedback Scheduling

mented by two queues as shown in Figure 4.4. New arrivals enter the high priority queue and are processed in the order of arrival. If the new arrival completes in the high priority queue quantum, Q_h , the process is deleted from the CPU queues until more CPU time is required. Otherwise the process is placed at the end of the low priority queue. When the process reaches the head of the low priority queue and there are no processes waiting for execution in the high priority queue, then a quantum of length Q_l is allocated to the process. If the process does not complete in Q_l CPU seconds, it is placed at the end of the low priority CPU queue to await another quantum allocation. When the process does complete, it is deleted from the CPU queues until more CPU time is needed. The two-level feedback algorithm analyzed here is non-preemptive. Once a low priority queue quantum begins, a newly arrived process waits until the end of the quantum to begin its execution.

The confined subsystem is treated differently than the other processes in the system to make the channel capacity analyzable. If this were not the case, the information channel would have to be treated as two parallel channels rather than one. This is true because in addition to the quantum length channel, a queue-residency channel would be present, since the confined subsystem could transmit information by the queue in which it is residing. Because the optimal sending strategy for this parallel channel representation is a research topic in itself, a less general method for handling the confined subsystem is used to study the effect of priority on the channel capacity rather than studying the more general question of the effect of the general feedback algorithm on channel capacity.

Two ways of scheduling the confined subsystem are to restrict the subsystem to execute in the high priority queue only, or in the low priority queue only. With either of these methods, the quantum length channel can be analyzed as in the FCFS and RR cases in which the confined subsystem is placed at the end of the appropriate CPU queue (high or low priority) after execution, regardless of whether its CPU requirement was satisfied or not. As before, the confined subsystem is allocated a zero length quantum if it becomes eligible for execution but needs no execution time. Of these two methods, constraining the confined subsystem to execute in the low priority queue is the more interesting case since newly arrived processes in the high priority queue will execute before the confined subsystem in the low priority queue. This may provide a highly variable amount of channel noise. If the confined subsystem was constrained to the high priority queue, less channel noise would be present since processes in the low priority queue would not contribute any channel noise. Also, the low priority queue processes would be prevented from executing since there would always be a process (the confined subsystem) in the high priority queue. For this thesis, the confined subsystem will be constrained to always execute in the low priority queue.

Information is transmitted through the quantum length channel by the confined subsystem varying the amount of CPU time it uses every time it becomes eligible for execution. Since the confined subsystem is constrained to execute in the low priority queue, the spy's strategy is to also execute only from the low priority queue. The spy process can fall into the low priority queue by being a new arrival in the CPU queue, which places it in the high priority queue, then using more CPU time than the high priority queue quantum length, Q_h . For the spy to continually circulate in the low priority queue (follow the confined subsystem), the spy process may take on a set of properties to insure it stays in the low priority queue. For example, in the CTSS FB scheduler [Corbato62], the spy process could change its memory length and effectively choose which priority queue it would enter next. This is due to memory size being a determinant of queue placement. Many systems use some type of priority determining rule to increase the throughput. Thus it is reasonable to assume that the spy can utilize such a rule to effectively choose the priority queue in which it will execute. With this ability, the channel output seen by the spy is the real time between two consecutive quantum allocations. Since the confined subsystem and spy both continually circulate in the low priority queue, the confined subsystem is guaranteed to execute exactly once between two consecutive spy executions. This synchronization between the confined subsystem and the spy aids the spy in decoding the channel outputs.

As noted for the RR scheduler, the FB scheduler also has the property that the percentage of the CPU used by the confined subsystem decreases as the number of active processes increases. Since the confined subsystem may waste CPU time, this effect is desirable in a confined scheduler.

For feedback, the type of absolute confinement strategy used on the other covert channels might make a difference in the scheduling channel capacity by permitting certain sending strategies to be fruitful. For example if the spy is able to detect when the confined subsystem actually executes, then in addition to a spy process circulating in the low priority queue, another spy process can be injected periodically into the high priority queue to try and eliminate some of the channel noise caused by the other legitimate users of the system. The channel output, then, is the real time between the two spy executions that most closely brackets the confined subsystem execution. This procedure decreases the amount of noise generated by the unconfined processes if the high

priority queue spy process takes a very small swap time and uses very little CPU time each execution so as to not affect the equilibrium state of the system. In the analysis to be performed, it is assumed that the other covert channels are absolutely confined by either the partitioning method or by a masking policy that does not permit the spy to determine when the confined subsystem receives a CPU quantum. Either of these policies will negate this spy strategy.

4.4 Number of Spies

The channel capacity of the covert scheduling channel is dependent on the number of concurrently executing spy processes as well as the number of independent (non-spy and non-confined) users. There are two different viewpoints that must be considered when observing the current workload of the system. If the system administrator observes a workload of 27 users (including one executing confined subsystem), the parameters for scheduling the confined subsystem may be set on the assumption that there are 25 independent users, one spy and one confined user. If, however, the number of independent users is overestimated, the capacity of the channel may be much larger than anticipated for the partial confinement case. In the case of *perfect collusion* in which all users other than the confined user are cooperating (i.e. 26 spies and one confined user), there is no channel noise introduced by the other users. Thus only the noise injected by the operating system prevents perfect transmission (zero decoding error for each input sent). In order for an operating system to offer partial confinement, assumptions must be made about the characteristics of the user population.

The other viewpoint is from the perspective of the spy who sees a certain workload on the system and wants to increase the channel capacity by adding more spy processes. One possible strategy for increasing the channel capacity is to trap some of the independent processes (but not the confined subsystem) between two spy processes and to eliminate their effect on the observed channel output. For the round robin and FCFS schedulers, this strategy can be implemented as in

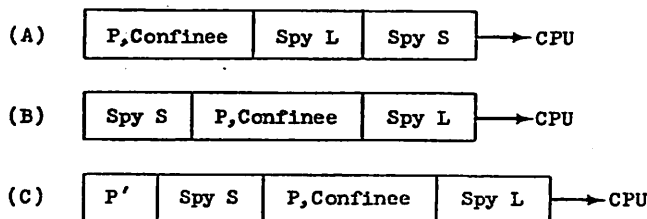


Figure 4.5
Two Spy Strategy for Round Robin/FCFS

Figure 4.5. The spy initiates two processes, one right after the other, with the first process taking a negligible amount of CPU time (Spy S) and the second taking some non-negligible amount of time (Spy L). Figure 4.5A shows the situation when these two spy processes reach the head of the queue. The queue entry labeled P, Confinee represents some combination of independent processes and the confined subsystem. At any time there are usually some processes waiting for input from their console and are not in the CPU queue. These processes are not shown in the figure. After Spy S executes, it is placed at the end of the queue and the situation is now as shown in Figure 4.5B. Since Spy L takes a non-negligible amount of time to execute, there is a non-zero probability that some of the processes waiting for input will receive input, will end their thinking period, and will enter the CPU

queue during Spy L's execution. This leads to the situation in Figure 4.5C where P' represents the processes that joined the CPU queue during Spy L's execution. In this case, the channel output observed by the spy is the time between the termination of the quantum assigned to Spy L and the beginning of the quantum assigned to Spy S. The processes labeled P' do not generate any noise since they are not part of the channel output. Since the amount of noise may decrease, the channel capacity may increase. This trapping strategy will, however, increase the response times seen by the users of the system and will increase the average number of processes in the CPU queue (the average channel noise). This method will be investigated further in Chapter 6 where measurements are given for the round-robin scheduling algorithm.

For the channels to be investigated in this thesis it will be assumed that one spy process is used unless specifically stated to the contrary. This analysis will give the expected channel capacity as a function of the number of independent users. The question of how much collusion a confinement mechanism should be able to tolerate must be an installation-dependent decision. In situations where the users have a thorough background check before being allowed to use the facility (such as in the military clearance procedure), a smaller amount of collusion might be assumed. In systems that process highly sensitive data that must be absolutely protected, perfect collusion must be assumed. This means that the noise attributed to independent users must be ignored in the analysis.

4.5 Summary

This chapter has explored the different ways information can be sent through the CPU scheduler. The major communication channels in the CPU scheduler are implemented by the different quantum lengths that are allocated to the confined subsystem and the total execution time of the confined subsystem. The total execution time channel is assumed to be blocked by the confiner specifying the time. Only the quantum length channel will be studied further. The requirement of being able to calculate the channel capacity for partial confinement was shown to impose restrictions on the confined subsystem and on the system as a whole to limit the number of channel inputs and outputs to a manageable number. There are many interesting channels that can be formulated but not solved due to the restrictions which must be imposed to make them tractable. This is particularly true for scheduling algorithms implemented as information channels with memory. Restrictions on the number of quantum lengths that can be allocated to the confined subsystem and the resolution of the channel outputs seen by the spy make the scheduling covert channels analyzed here tractable.

Another facet of the confinement problem is the determination of the number of spies which might be using the system at one time. Since the channel capacity is dependent on the number of independent users generating channel noise, an evaluation of the trustworthiness of the user community is necessary before using partial confinement.

5. THE MODEL

5.1 Introduction

To study covert channels, the channel capacity must be calculable so that the magnitude of the information leakage problem can be determined. The concept of channel capacity for the scheduling covert channel has been previously discussed at some length for the scheduling algorithms to be analyzed (first-come-first-served or FCFS, round robin or RR, and two-level feedback or FB). This chapter will discuss the method for computing the channel capacity.

5.2 Overview of the Analysis Method

The channel capacity measurements presented in Chapter 6 are based on a simulation model of the Compatible Time Sharing System (CTSS) developed at MIT during the early 1960's [Corbato62]. There are two reasons this system was chosen for study. First, CTSS was a simple operational system that supported a number of simultaneous timesharing users. By basing the capacity measurements on this system, the results should reflect realistic estimates of the channel capacity for real systems. Of course, each different timesharing system will have different channel capacities since different hardware and different system policies are used. However, the procedure for measuring these capacities is the same one presented here. Second, detailed measurements of the system operation are available and an accurate simulation model of the system had been previously developed [Scherr66]. The simulation model is a convenient source of system measurements since the instrumentation can be easily added to obtain the performance characteristics of interest.

The first step in the analysis process is to validate our simulator of a CTSS-like system with the simulator measurements produced by Scherr's CTSS simulator. Our simulation results are only CTSS-like since discretization of the system, as explained in Chapter 4, must be done to make the capacity computable. For validation purposes, the discretization feature has not been used so that a direct comparison with Scherr's results could be made.

The second step is to use the output from our simulator to parameterize a semi-Markov model of the timesharing system which will be needed to calculate the channel capacity. The semi-Markov model represents the effect of allocating CPU time to the confined subsystem on the CPU scheduler. In particular, the number of processes that are requesting CPU time (and hence the noise level) distinguishes the states of the model from one another. For any given state in the model, the next state probability and the probability of the amount of time until the next state transition are obtained from the model parameters.

The third step is to obtain a CTPM for the CPU scheduling channel from the semi-Markov model. Once the CTPM is determined, the capacity can be found using the methods explained in Chapter 2.

The following sections present each of the above steps in the capacity calculation procedure in more detail along with the validation information for each step. A discussion of the outputs obtained from the model is postponed until the next chapter.

5.3 The CTSS Timesharing System Simulator

The hardware used for CTSS is similar in speed and capacity to the minicomputer-based timesharing systems presently available. Specifically, the CTSS host machine was

an IBM 7094 with six tape units, one IBM 1301 disk drive, and one IBM 7320 swapping drum. The IBM 7094 was modified to have two 32K memory banks (one for user programs and one for system programs), and automatic relocation and bounds protection registers. However, in the implementation of CTSS measured by Scherr, the automatic relocation register was not used, causing all user programs to start at absolute location zero in the user memory. Thus at most one complete user program could be in memory at one time.

There are two characteristics of the CTSS system that simplify the simulation. The first is that there is no overlapping of CPU time and I/O time. Thus all I/O time is included in the user's quantum allocation. Since there is at most one complete user process in memory at one time, the system swap time is not overlapped with useful user CPU time. This decreases the complexity of the system. The second characteristic is that Scherr found the non-swapping system overhead associated with a user to be approximately linear with respect to the CPU time requirement of the user. Thus the user's real CPU requirement is suitably lengthened to account for both the user CPU time and for system overhead.

One simplifying assumption made by Scherr in simulating CTSS is that the CPU requirement for each interaction as well as each user think time is independent of all previous CPU requirements and think times. Introducing this assumption did not appreciably affect the simulator validity since the CTSS simulator closely agreed with the measurements of the real CTSS system. This assumption is also made in the simulator implemented here for the capacity measurements.

The CTSS simulator is described by the model in Figure 5.1. This is a closed interaction model of a timesharing sys-

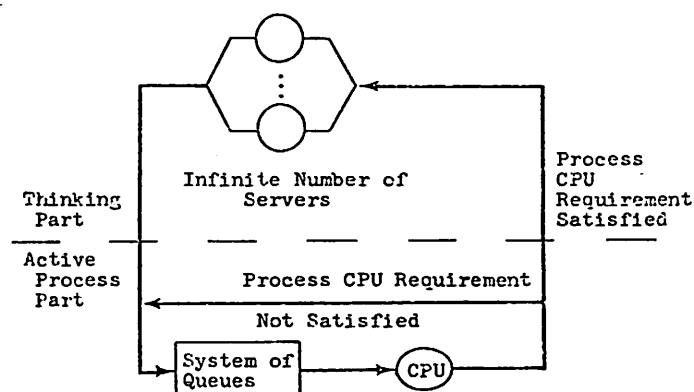


Figure 5.1
CTSS Queueing Model

tem that accounts for all users on the system. No users either enter or exit from the model. A typical interaction in a timesharing system consists of two parts. The first part is the period of time during which the user is thinking of his next input line to be entered. This includes both the actual thinking time and the time required to type in the input. At the moment the input is entered, a process is activated by the operating system to perform the task desired by the user. The process is scheduled for execution through a system of queues and is eventually allocated a quantum of CPU time. If the quantum allocated does not satisfy the CPU require-

ment of the process, the process reenters the system of queues to be scheduled for another quantum allocation. When enough CPU time has been allocated to the process to satisfy the CPU requirement, the interaction is complete and the operating system waits for more input from the user before any more CPU time is allocated to that user. Figure 5.1 consists of two parts that model the thinking user (thinking part) and the allocation of CPU time to the user's process (active process part). The thinking users on the system are modeled by a service station with an infinite number of servers. Thus there is never a queue at this station. The service distribution is the one measured by Scherr which is nearly (but not quite) exponential. The mean of this distribution is 35.2 seconds with a standard deviation of 23 seconds. The think time density has an impulse of area 0.12 at time=0 representing the processes in the workload that generate the next command themselves without user intervention. The active process part consists of a system of queues (which represent the scheduling algorithm) and a single CPU server. The service time for the CPU service station includes the time to swap in the current process, to execute the process and to swap out the process. The CPU service distribution used is the same one used by Scherr. The CPU time distribution is characterized by many CPU requests requiring very small amounts of CPU time, as shown by the distribution median being less than 0.05 seconds. A typical example of this behavior is file editing. In addition to these small interactive-type requests, there are also some fairly large requests, such as for compilations and long program executions. These requests make the mean of the distribution (0.88 seconds) much larger than the median.

In the CTSS model there are three classes of processes: unconfined processes for users not executing confined, the spy process and the confined subsystem. The unconfined processes are free to enter and leave both parts of the model but the spy process and confined subsystem are constrained to never leave the system of queues representing the scheduler, as noted in Chapter 4. The CPU service distribution of each class is distinct due to the different characteristics of each class. The unconfined processes exhibit the characteristics observed by Scherr, the spy process uses very little resources during execution, and the confined subsystem exhibits characteristics which vary depending the scheduling strategy used. Since the interquantum time probability distribution for the spy is needed in the calculation of channel capacity, the queueing theory model of this system is not pursued in favor of the simulation approach.

The objective of the simulator development for the channel capacity study is to duplicate the CTSS simulator results, then to make suitable modifications (such as discretization) to allow the channel capacity to be measured. Table 5.1 is a comparison of the results obtained from our simulator with the results published in [Scherr66] for the round-robin scheduler with a quantum length of 2 seconds and 25 users (no discretization). These measurements show that there is not a large discrepancy between Scherr's simulator and our simulator. In fact, all of the measurements listed in Table 5.1 for the capacity measurement simulator are within 5% of the values obtained from Scherr's CTSS simulator. Based on this comparison, our simulator is believed to reflect the CTSS system behavior closely enough for the purposes of this thesis. Thus the measurements reported here are based on a real timesharing system and should illustrate the magnitude of channel capacities available in such a system.

	Scherr CTSS Simulator	Capacity Measurement CTSS Simulator
Average response time	7.20	7.32
% Disk time utilization (swapping)	24.1	23.6
% User CPU time	52	54

Table 5.1
Comparison of Scherr's CTSS Simulator
with Our CTSS Simulator

5.4 The Semi-Markov Model for Capacity Calculation

The next step in the calculation requires that the output from the simulator be used to formulate a semi-Markov model of the scheduling algorithm to be analyzed. The use of semi-Markov models to analyze a stochastic process in a computer system is not new. A similar approach is taken in [Chu76] for analyzing page replacement algorithms. In that analysis, the semi-Markov model is used to represent the page faulting characteristics of an executing process. In the present analysis, the semi-Markov model represents the interquantum times seen by the spy. Since the confined subsystem can usually be allocated one of several possible quantum length (e.g. any multiple of the discretization interval up to the maximum quantum length), there is usually more than one semi-Markov model representing a scheduling algorithm. Each semi-Markov model represents the effect of allocating one particular quantum length to the confined subsystem. In the following model derivation, only one semi-Markov model is considered. Generation of the other models for the different quantum lengths is performed in the same manner.

Before proceeding with the derivation of the model, a brief introduction to the characteristics of a semi-Markov model will be given. A semi-Markov process is represented by a set of states, S_i ($0 \leq i \leq n$). The Markov property of this representation states that the behavior of the model at any instant of time is governed by the current state and the time of the last transition into the current state. The behavior is not dependent on the history of the states entered before the current state. Given the current model state, S_i ($0 \leq i \leq n$), the probability of the next state being S_j ($0 \leq j \leq n$) is given by a transition probability function. There is one such function for each state in the model. Once the next state, S_j , has been determined by the transition probability function, the amount of time between the transition into the current state, S_i , and the transition to state S_j is given by a probability density function. There is one such time probability density function for each ordered pair of states (S_i, S_j) , $0 \leq i, j \leq n$. For the present analysis, the semi-Markov model is used to determine the probability of a certain state sequence occurring and the probability density of the interquantum times seen by the spy.

The major hurdle in the analysis is defining a state space describing the CTSS system for the semi-Markov model that is small enough to make the computation of the above two probability densities feasible. A state space that exactly models the CTSS system must include a representation of the

number of quanta each process in the CPU queue has been previously allocated for the current interaction, and the amount of time each process that is waiting for user input (the user is thinking) has been in that state. This is a very large state space since a user's think time and previous CPU time allocation could be measured in the hundreds of seconds. To decrease the state space size, certain assumptions are introduced. These assumptions serve to eliminate the necessity of exactly representing the process state and are the topic of the next section.

5.4.1 Model Assumptions

To simplify the representation of the real CTSS system, five assumptions are made about the user and process characteristics. The last three assumptions are actually incorporated into the simulator and all five are used in generating the semi-Markov model parameters. The decision as to the validity of these assumptions will be made by comparing the channel capacity obtained from the semi-Markov model with the capacity obtained from simulation. No individual validation of the assumptions will be done.

The first assumption is that the think time distribution for each user is exponential. As noted in [Estrin67], the observed think time probability distribution for CTSS is very close to exponential so it is expected that this assumption will not greatly affect the validity of the model. The effect of this assumption is to decrease the number of states needed in the semi-Markov model. If the assumed think time probability distribution is not exponential, the amount of time each thinking user has been thinking would have to be encoded into the state space. For example, suppose there are ten users on the system and that these users may either be thinking or requesting CPU time for a process. Also assume that the think time density function has non-zero probability for the interval 0 to 20 seconds and that the discretization used is 0.2 seconds. With these assumptions, the number of states needed to accurately represent just the thinking users is:

$$\left(\frac{20}{0.2}\right)^{10} + \left(\frac{20}{0.2}\right)^9 + \cdots + \left(\frac{20}{0.2}\right) = 1.01 \times 10^{20}$$

The reason for the large number of states is that knowing how long a user has been thinking does affect the probability of this future behavior. With the exponential distribution assumption, the amount of time each user has been thinking gives no clue as to how much longer the user will be thinking, due to the memoryless property of the exponential distribution. Thus the number of states needed for the above example with this assumption is 11.

The second assumption is that the number of quanta required for execution of a process is geometrically distributed. This means that the probability of a process terminating during any given quantum is independent of all previous executions of the process. In the timesharing system model, this distribution is generated by first deciding whether the executing process terminates during the present CPU allocation by flipping a coin biased according to the queue of which the process is a member. Given the outcome of the toss, the CPU time needed is then chosen from either the terminating process distribution for that queue or the non-terminating process distribution for that queue. The terminating and non-terminating distributions are not dependent on the number of quanta previously allocated to the process. These simulations are obtained from simulation measurements and

can be viewed as an average of the behavior of all processes. The assumption of a geometric distribution for the number of quanta required and use of the terminating and non-terminating CPU time distributions from the simulator will produce a process behavior that is different from the one observed in CTSS. If, however, the assumption of a geometrically distributed number of quanta is not made, the amount of CPU time previously allocated must be kept for each executable process, which makes the number of states large for the same reason as for the think time case. The validation measurements to be given later will show that the behavior of the model with this assumption closely parallels the behavior of the real system.

The third assumption is that the behavior of a process is independent of the state of any other process and of the number of users on the system. This assumption is never exactly satisfied by a real system since there is a feedback effect between the system workload and the type of jobs executed by the users. For example, if there are a number of long running, high priority jobs being executed, the user might run non-interactive jobs since the interactivity of the system might be poor. However, in Scherr's analysis, this feedback effect was not included in the simulator, yet the simulation results did accurately represent the CTSS system. Thus our omission of this feedback effect should not significantly affect the results of this analysis.

The fourth assumption is that the entire operating system is discretized with a discretization interval of 0.2 seconds unless otherwise noted. This assumption was previously mentioned in Chapter 4 and serves to make the channel capacity analysis tractable. There are two ways to interpret the capacities computed using this assumption. The first interpretation is that the measurements are an approximation to the operation of a real system. As the discretization interval approaches zero, the capacity measurements more nearly approximate the true channel capacity in an undiscretized system. In this interpretation, the capacity calculation is done by integration rather than by the discrete methods used here. The problem with this approach is in accurately specifying the continuous (or very small discretization interval) probability functions needed for the integration operation without hypothesizing some convenient function such as exponential. Note that the probability function specification is only a problem for determining the channel capacity for partial confinement and not for absolute confinement in which the capacity is zero. The second interpretation is to accept discretization as a constraint on the design of systems offering partial confinement. In the system design, the discretization interval is set to the smallest value for which the channel capacity can be computed. The capacity obtained can then be taken as a good estimate of the actual channel capacity. Operating systems that offer partial confinement must give some kind of guarantee that the amount of information leakage by a process will not exceed a certain amount. The view which most closely supports this is the interpretation of discretization as a constraint. This is also the view that will be taken in this thesis.

The fifth assumption defines the characteristics of the spy process. Since the spy executes continuously, the more CPU and swapping time used by the spy each execution, the larger is the mean average scheduler queue length. Since more unconfined processes executing mean a decrease in channel capacity, the spy's objective is to perturb the system as little as possible to keep the channel noise to a minimum.

For this reason, the spy should use as little CPU time per execution and as little memory as possible. The simulator and the semi-Markov model assume that the spy uses only 1 microsecond of CPU time each execution and that the spy's memory requirement is only one word. Both of these characteristics are chosen to insure that the actual system perturbation due to the spy is greater than that assumed and thus that our results represent an upper bound on the amount of leakage.

5.4.2 Analysis Assumptions

In addition to the assumptions dealing with the CTSS system, three assumptions are also made in the channel capacity analysis. The first assumption is that the confined subsystem and the spy do not know the number of processes requesting CPU time or have any knowledge of the number of processes in any queue. If the spy could tell, for example, how many unconfined processes execute with the confined subsystem between two spy executions, the spy might be able to more intelligently decode the channel outputs and thus raise the capacity of the channel. To satisfy this assumption, the system must not make available to any process the number of processes requesting CPU time. In fact, the entire system performance evaluation effort would have to be carefully evaluated to insure that no leakage of performance information to potential spies could result in more leakage. For example, if the spy is able to find out that the average CPU requirement for the different job classifications (e.g. student, faculty, research) and the spy knows the confined subsystem is a "research" job, then a better guess as to the total running time of the confined subsystem can be made. This information is not generally available to the spy process since the identity of the other concurrently executing processes is not known. This information could leak additional information, although the amount of extra channel capacity is probably small from such gross measures of system performance.

The second assumption for the analysis is that the system workload consists of one confined subsystem, one spy process and n general users ($n = 5, 15, \text{ or } 25$) at any one time. The number of users on the system remains constant (no logons or logoffs) but the proportion of users thinking varies with time. The one confined subsystem restriction could be removed by explicitly accounting for the characteristics of the additional confined subsystems in the model. However, providing more than one spy may cause the channel capacity to increase as mentioned in Chapter 4. The effect of two spies will be analyzed in Chapter 6 to determine the effect of this assumption. A discussion of this point is deferred until then.

The third assumption is that for partial confinement, the confined subsystem will be forced to allow the system to return to some predefined equilibrium state between each series of variable quanta sent. A variable quantum is defined as a quantum with length determined by the confined subsystem. Each set of variable quanta perturbs the system from its equilibrium state. To make the channel capacity measurable, the equilibrium state must be reestablished between each set of variable quanta as explained in Chapter 2. The equilibrium state used in all cases is the state that results from the confined subsystem being allocated a constant length quantum every time it is eligible for execution. This constant quantum length is called the equilibrium quantum length since it defines the equilibrium state. This point is more thoroughly explained in Chapter 6. This formulation of

the covert channel is appealing since the subsystem that pays most of the penalty for confinement, either in long response times or in wasted CPU allocations, is the confined subsystem. This does not mean, however, that the unconfined users will be unaffected by the confined execution.

5.4.3 Model Parameterization

A semi-Markov model of a stochastic process is composed of three parts:

1) A set of i states ($0 \leq i \leq n$) representing the stochastic process

2) A set of state transition probabilities. If the current state is j , the probability that the next state is i is P_{ij} . These transition probabilities obey the usual rules:

$$a) \sum_{k=0}^n P_{kj} = 1, 0 \leq j \leq n.$$

$$b) P_{ij} \geq 0, 0 \leq i, j \leq n.$$

3) A set of transition time probability density functions, $T_{ij}(t)$ for $0 \leq i, j \leq n$. The function $T_{ij}(t)$ gives the probability that the time spent in state j before the transition to the next state i is t , given that the transition from state j to state i occurs.

The model is characterized by the Markov property that the behavior of the process at any time is dependent only on its current state and not on any previous state. Thus any past history that affects the current behavior of the process must be encoded into the current state. The difference between the semi-Markov model and a straight Markov model is that the state residency time probability density function is included in the semi-Markov case and not in the straight Markov model. It is the inclusion of the state residency time probability density function that makes the semi-Markov model suitable for measuring the channel capacity of the scheduling covert channel. For the interested reader, a more formal definition of the semi-Markov model can be found in [Ross70].

To determine the capacity of the covert scheduling channel, there must be a way to model the effect of allocating different quantum lengths to the confined subsystem on the rest of the system. In particular, since the covert channel output observed by the spy is the interquantum time, the effect of allocating different amounts of CPU time to the confined subsystem on the spy's interquantum time must be calculable. This is accomplished by using a set of semi-Markov models (one complete model for each possible CPU quantum length allocated to the confined subsystem) each with the same state space but with different P_{ij} and T_{ij} functions. The remainder of this chapter will explain how one of the semi-Markov models from the set is obtained for a given quantum length allocated to the confined subsystem. The models for the remaining confined subsystem quantum lengths can be generated by using the same technique.

The scheduling algorithms to be analyzed are round-robin (RR), first-come-first-served (FCFS), and two-level feedback (FB) as described in Chapter 4. The semi-Markov model parameters for RR and FCFS are calculated in the same way since the two algorithms are closely related. However, the FB scheduler is very different from either the RR or the FCFS scheduler. In particular, FB is implemented with two queues of different priorities while RR/FCFS is implemented with only one queue. Because of this difference, the method for parameterizing the semi-Markov model for the FB scheduler is more complex than for the RR or FCFS scheduler. For this reason, the RR/FCFS model will be dis-

cussed first and the additional steps needed to parameterize the FB model will be discussed next.

In the RR/FCFS model, the model states represent the number of unconfined processes ahead of the spy in the CPU queue at the instant the spy is placed at the end of the CPU queue. Thus for four unconfined users in the system, state 2 represents the situation in Figure 5.2. For this state, the CPU queue ordering of the unconfined processes and the confined subsystem is immaterial since they will all execute

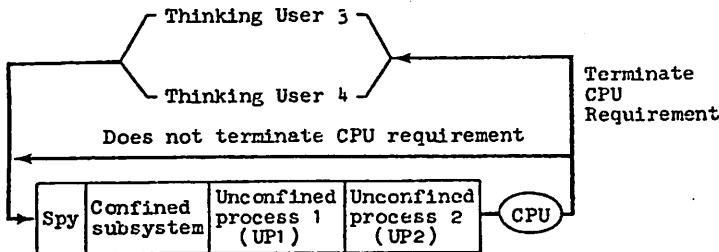


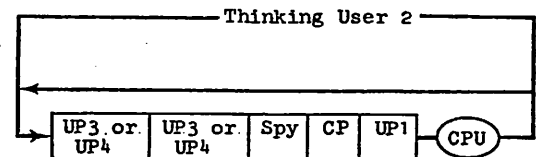
Figure 5.2
Representation of State 2 for the RR/FCFS Model

before the spy process. Note that changing the number of users in the system does not affect the number of users in the CPU queue for this state, but does affect the number of users that are thinking. Because of the exponential think time assumption and the independence of the present CPU requirement and the past CPU time allocations, the state of the scheduler can be given by the total number of users in the CPU queue. For a workload of 25 unconfined users, the complete state space would have 26 states representing 0, 1, ..., 25 unconfined processes in the CPU queue. However, most of the higher numbered states have a very small probability of occurrence and can be neglected for the purpose of the analysis. The contribution of the neglected states to the channel capacity is small since virtually all of the capacity results from the three states representing 0, 1, and 2 unconfined users in the system. For this reason, the RR/FCFS model uses only 12 states (states 0–11) for a workload of 25 unconfined users, 8 states for a workload of 15 unconfined users, and 6 states for a workload of 5 unconfined users.

The next task is to compute the remaining two components of the semi-Markov model: the transition probability density functions, P_{ij} , and the transition time probability density functions, T_{ij} . To do this, it is first necessary to simulate the scheduling algorithm to be analyzed and to obtain the probability that an unconfined process will terminate during its next quantum allocation, the probability density function for the amount of real time a terminating process will take during its next quantum allocation, $TERM(t)$, and the probability density function for the real time a non-terminating process will take during its next quantum allocation, $NTERM(t)$. Both of the probability density functions include any necessary swapping time. With these simulation measurements, the P_{ij} and T_{ij} functions can be computed by starting in state j and computing the probability and the time density function for each event sequence that results in a transition to state i . The P_{ij} function is simply the summa-

tion of the probability of all event sequences leading to state i . The T_{ij} function is obtained by weighting the convolutions of the $TERM(t)$ and $NTERM(t)$ probability densities which represent the event sequence for a transition from state j to state i , by the probability of the event occurring. As an example, take the case of computing the transition probabilities and the transition time probability density function for state 2 with four unconfined users (two unconfined users thinking and two unconfined user processes requesting CPU time). This starting state is shown in Figure 5.2. From this starting point, six events could occur during the execution of Unconfined Process 2 (UP2). The possibilities are:

- 1) UP2 terminates and neither thinking user activates a process that enters the CPU queue.
- 2) UP2 terminates and one thinking user activates a process that enters the CPU queue.
- 3) UP2 terminates and each thinking user activates a process that enters the CPU queue.
- 4) UP2 does not terminate and neither thinking user activates a process that enters the CPU queue.
- 5) UP2 does not terminate and one thinking user activates a process that enters the CPU queue.
- 6) UP2 does not terminate and each thinking user activates a process that enters the CPU queue. From the simulator output and the exponential think time assumption, the probability of each of these events and the real time density function is calculable in a straightforward manner. For case 3



CP = confined process
UP3 or UP4 = either unconfined process for user 3 or for user 4
UP1 = unconfined process for user 1

Figure 5.3
Intermediate State After Unconfined Process 2 Executes

above, the intermediate state entered is shown in Figure 5.3. The transition probability for case 3, PT, is computed by the expression:

$$\sum_{i=0}^{\infty} (C_{22} \times (\text{probability one user quits thinking in real time } \leq t)^2 \times (\text{probability one user continues thinking longer than time } t)^0 \times (\text{probability UP2's execution time is } t, \text{ given UP2 terminates}) \times (\text{probability UP2 terminates}))$$

in which C_{ij} is the number of combinations of j things that can be made without regard to order out of a total of i things. The simulator provides the probability that UP2's execution time is t given UP2 terminates and the probability that UP2 terminates. The probability one user quits thinking in time $\leq t$ and the probability one user continues thinking longer

than time t are computed from the exponential think time density function. The real time density function for the intermediate transition due to the execution of UP2, $T(t)$, is simply:

$$T(t) = C_{22} \times (\text{probability one user quits thinking in real time} \leq t)^2 \\ \times (\text{probability one user continues longer than time } t)^0 \\ \times (\text{probability UP2's execution time is } t \text{ given UP2 terminates}) \\ \times (\text{probability UP2 terminates})$$

When the calculation for the six cases above is completed, there is a partial transition probability defined from state 2 to the allowable intermediate states as shown in Table 5.2. Since there is still one unconfined process ahead of the confined subsystem and the spy in the CPU queue, this procedure must be repeated using each of the intermediate states in Table 5.2 as starting states. The main complication of this

Number of Thinking Users	Total Number of Unconfined Users in Queue Behind the Spy	Cases Contributing to Transition
0	3	6
1	2	3, 5
2	1	2, 4
3	0	1

Table 5.2

Intermediate States After One Unconfined Process Execution

calculation is in computing the time density function from the convolution of the density obtained for the transition from state 2 to the first intermediate state, with the density for the transition from the first intermediate state to the second intermediate state. This operation is, however, straightforward.

After the execution of UP1 has been accounted for, only the confined subsystem remains to execute before the spy. The procedure for calculating the state transition probability and the transition time density for the confined subsystem is the same as for the unconfined processes with one difference. The difference is that the confined subsystem takes a fixed amount of time to execute, which is the CPU quantum length for the confined subsystem execution for which this model is being computed. The probabilities of processes entering the CPU queue at the completion of a user think time, in particular, are dependent on the CPU allocation given to the confined subsystem. Thus the formula for PT above has only one non-zero term in the summation and the formula for $T(t)$ is non-zero only for t equals the confined subsystem quantum length. After the confined subsystem execution has been accounted for, the remaining factor that determines the state transition is the spy execution. As noted before, the spy takes as little CPU time as possible to keep its system perturbation to a minimum. The assumed execution time for the spy is one discretization interval (0.2 seconds unless specifically noted otherwise). The effect of

the spy is computed in the same manner as for the confined subsystem only the execution time is a constant 0.2 seconds. The resulting transition probabilities and transition time density functions after the spy computation are the final functions.

The above procedure stepped through the computation for only state 2 of the semi-Markov model. Each state must be similarly analyzed with the same CPU quantum length for the confined subsystem to obtain a complete semi-Markov model. The CPU allocation to the confined subsystem is then changed to another possibility and another entire model computed. This results in the necessary one semi-Markov model for each possible confined subsystem CPU allocation.

The FB analysis is slightly more complex since it is not known how many unconfined processes will execute between two successive spy executions in the low priority queue, due to processes entering the high priority queue. This two queue structure also complicates the description of the intermediate states since the number of processes in both queues must be kept.

The state space for the FB algorithm excluding the intermediate states is composed of a pair of numbers, (A,B) in which A is the number of processes in the high priority queue and B is the number of processes in the low priority queue. Since the spy always executes in the low priority queue, A is zero when the spy begins execution. To reduce the size of the model state space, the model states represent the state of the CPU queue when the spy begins execution (all pairs for which A=0). Thus the state space can be represented by the single number B.

To obtain the transition probabilities, P_{ij} , and the transition time probability density functions, $T_{ij}(t)$, the same method is used as for RR/FCFS but with two major differences. The first difference is that the number of intermediate states is much larger—i.e. all allowable values of (A,B). This makes computation of the model parameters more difficult. The second difference is that there is a non-zero probability that the spy will not execute again within time T_1 , where T_1 is an arbitrarily large number. This could occur if processes enter the high priority queue and terminate in one quantum allocation in such a way as to lock out all low priority queue processes from execution. Thus there is a finite probability that no transition will occur within a finite time. The approach taken to this problem is to establish a probability threshold (0.995 unless otherwise stated): the transitions from intermediate states to the intermediate states and to the final model states (those with A=0) are computed until the sum of the transition probabilities into the final states exceed the threshold. All transition probabilities are then normalized to 1.0 and the computation terminates. For example, take the case of determining the transition probabilities and the transition time density function for the (0,2) state (0 unconfined processes in the high priority queue and 2 unconfined processes in the low priority queue) in a system with four unconfined processes. After the first unconfined process in the low priority queue executes, the next state could be any legal state (A,B) where $1 \leq B \leq 3$. To calculate the state probabilities for the next transition, all the intermediate states from the first calculation are used to compute the second order transitions as in the RR/FCFS analysis. However in the FB analysis, the same state may occur more than once in a state transition sequence unlike the RR/FCFS analysis. The transition computations are continued until the amount of state probability in all the terminal

states (i.e. those states (A,B) in which $A=0$) is greater than 0.995. The state transition probabilities and the transition time density functions are then normalized so that all probabilities sum to 1.0. The effect of the confined subsystem and the spy are then taken into account as for the RR/FCFS analysis which yields the final model parameters. As for RR-FCFS, the number of states used in the model is not the number of unconfined processes in the system since some of the states have only a small probability of occurrence. The FB model uses only 10 terminal and 70 intermediate states for a workload of 25 unconfined processes, 8 terminal and 40 intermediate states for a workload of 15 unconfined processes, and 6 terminal and 30 intermediate states for a workload of 5 unconfined processes.

Since the number of states for a CPU queue may be less than the number of unconfined users in both the RR/FCFS and FB models, some special action must be taken when the number of unconfined processes in a CPU queue exceeds the maximum allowable by the model. This problem is handled by having the highest numbered state represent all states that are not explicitly represented. For example, in the RR/FCFS model with 25 unconfined processes and only 12 states, the scheduler queue with 0 to 10 unconfined processes is explicitly represented and the remaining state is used to represent 11 to 25 unconfined processes in the scheduling queue. For the FB algorithm with a maximum state representation of (A,B) for $A, B > 0$, the state (A,Y) receives all transitions for non-represented states (X,Y) for $X \geq A$ and $Y \leq B$, the state (X,B) receives all transitions for non-represented states (X,Y) for $X \leq A$ and $Y \geq B$, and the state (A,B) receives all transitions for non-represented states (X,Y) for $X > A$ and $Y > B$. The validity of restricting the state space in this manner will be determined when the channel capacity computed from the semi-Markov model is compared with the channel capacity obtained from simulation.

5.4.4 Channel Transition Probability Matrix Generation

To calculate the channel capacity, a channel transition probability matrix (CTPM), which gives the probability of the spy observing a certain interquantum time given the CPU time allocated to the confined subsystem, must be computed. From the definition of the semi-Markov model, the CTPM can be generated for both the discrete memoryless channel (DMC) case and the discrete channel with memory (DCWM) case.

By definition of the DMC, the channel must return to its equilibrium state before each variable quantum length is allocated. The equilibrium state is determined by the equilibrium quantum length allocated to the confined subsystem. That is, the quantum length that is allocated to the confined subsystem between variable quanta. For example, if the transition probabilities for the semi-Markov model with at most two unconfined users in the CPU queue at one time is given in Figure 5.4, then the model state probabilities at equilibrium, S_i for $i=0, 1, 2$, can be found by solving the following transition equations based on the transition diagram [Kleinrock75]. The equations for Figure 5.4 are:

$$\begin{aligned} S_0 + S_1 + S_2 &= 1.0 \\ 0.70S_0 + 0.25S_1 + 0.30S_2 &= S_0 \\ 0.20S_0 + 0.60S_1 + 0.30S_2 &= S_1 \\ 0.10S_0 + 0.15S_1 + 0.40S_2 &= S_2 \end{aligned}$$

The first equation is the requirement that the state probabili-

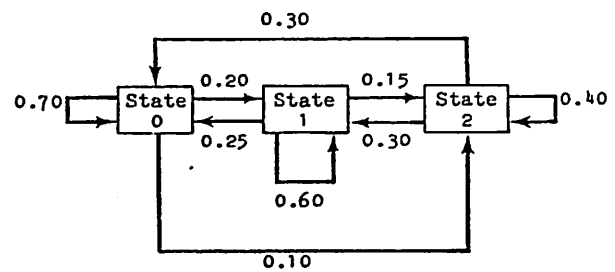


Figure 5.4
Semi-Markov Model of Equilibrium Quantum

ties must sum to 1.0. The remaining three equations (of which only two are independent) result from the requirement that the probability of entering a state during the next transition is equal to the probability of the state at equilibrium. Solving these equations gives the equilibrium state probabilities:

$$S_0 = 0.470, S_1 = 0.361, S_2 = 0.169$$

The CTPM for the DMC case is computed entirely from the semi-Markov model computed from the requirement that the confined subsystem is allocated the equilibrium length quantum. The CTPM row corresponding to zero CPU time allocated to the confined subsystem is computed by weighting each transition time density function, $T_{ij}(t)$, by its probability of occurrence since the equilibrium state probabilities are known, then summing over all the transition time density functions. The CTPM row corresponding to 0.2 seconds allocated to the confined subsystem is found by shifting the zero CPU time CTPM row found above one discretization interval (one discretization interval = 0.2 seconds). That is, if the CTPM row for zero CPU time allocated to the confined subsystem is:

Interquantum time	0.0	0.2	0.4	0.6	0.8
Probability	0.0	0.6	0.2	0.2	0.0

then the CTPM row for 0.2 seconds allocated to the confined subsystem is:

Interquantum time	0.0	0.2	0.4	0.6	0.8
Probability	0.0	0.0	0.6	0.2	0.2

Simple shifting of the density produces the correct result since the time used by the confined subsystem is added to the effects of the unconfined processes. For the transition time probability function of the equilibrium quantum given in Table 5.3 and the state transition probabilities given in Table 5.4, the CTPM row corresponding to zero CPU time allocated to the confined subsystem is:

Interquantum Time	0.2	0.4	0.6	0.8
Probability	0.676	0.161	0.163	0.0

To compute the probability for a confined subsystem quan-

From State	To State	Interquantum Time Probabilities		
		0.2 sec	0.4 sec	0.6 sec
0	0	0.90	0.05	0.05
0	1	0.70	0.20	0.10
0	2	0.40	0.25	0.35
1	0	0.60	0.20	0.20
1	1	0.50	0.30	0.20
1	2	0.40	0.30	0.30
2	0	0.65	0.25	0.10
2	1	0.45	0.20	0.35
2	2	0.35	0.10	0.55

Table 5.3

Example Transition Time Probability Density Function for the Equilibrium Quantum

From State	To State	Probability
0	0	0.80
0	1	0.10
0	2	0.10
1	0	0.60
1	1	0.25
1	2	0.15
2	0	0.50
2	1	0.30
2	2	0.20

Table 5.4

Example Transition Probabilities for a Variable Quantum

tum length of 0.2 seconds, the formula is:

$$\sum_{i=0}^2 (\text{Equilibrium probability of state } i) \times \sum_{j,k=0}^2 ((\text{Probability of 0.2 seconds given the transition from state } j \text{ to state } k) \times (\text{probability of the transition from state } j \text{ to state } k))$$

A similar calculation is made for the 0.4 and 0.6 second probabilities. As noted before, the CTPM row corresponding to 0.2 seconds would be:

Interquantum Time	0.4	0.6	0.8
Probability	0.676	0.161	0.163

and for 0.4 seconds would be:

Interquantum Time	0.6	0.8	1.0
Probability	0.676	0.161	0.163

The calculation of the CTPM for the DCWM is more complex since there is one row for each n-tuple of possible confined subsystem quantum lengths and there is one column for each n-tuple of possible interquantum times. In this analysis, n will usually have a value of three. The calculation of the probability of each n-tuple of interquantum times is done by iterating on all possible n-tuples of state sequences. For every n-tuple of possible state occurrences (S_1, S_2, \dots, S_n), the probability of the state sequence is

given by the equilibrium state probability for S_1 multiplied by the appropriate transition probability to each succeeding state. The n-tuple of interquantum times given the state sequence can be computed from the semi-Markov model in a straightforward manner. Multiplying the probability of the state sequence by the probability of the interquantum time n-tuple given the state sequence and summing over all possible state sequences gives the probability of the interquantum time n-tuple.

To illustrate the method outlined above, the example used for the DMC will be extended to the DCWM case for $n=2$. Assume the confined subsystem chooses to send the equilibrium quantum, characterized by the parameters in Tables 5.3 and 5.4, twice in succession as its variable quanta. The equilibrium state probabilities of (0.470, 0.361, 0.169) for state S_0, S_1, S_2 respectively are remembered from the earlier analysis. The first thing to do is to choose a state sequence which gives the scheduler states when the variable quanta are allocated to the confined subsystem. The state sequence initially chosen is not important since all possible state sequences will be used in the final summation. Assume the sequence (S_0, S_1) is chosen. The probability that the spy observes the interquantum pair (0.2 seconds, 0.4 seconds) is to be calculated. The contribution to the probability of the outputs (0.2 seconds, 0.4 seconds) being observed by the spy from the state sequence (S_0, S_1) is computed by:

$$\begin{aligned} &\text{Prob}((0.2\text{seconds}, 0.4\text{seconds}) \text{ given } (S_0, S_1)) \\ &= \text{Prob}(S_0 \text{ is the equilibrium state}) \\ &\quad \times \text{Prob}(\text{transition from } S_0 \text{ to } S_1) \\ &\quad \times \text{Prob}(0.2 \text{ seconds given the transition from } S_0 \text{ to } S_1) \\ &\quad \times \text{Prob}(0.4 \text{ seconds given } S_1) \\ &= (0.470) \times (0.10) \times (0.70) \times [(0.2 \times 0.6) + (0.3 \times 0.25) \\ &\quad + (0.3 \times 0.15)] \\ &= 0.008 \end{aligned}$$

Remember that this is the probability given the selected state sequence. The probability of (0.2 seconds, 0.4 seconds) must be summed for all state sequences before the correct value for this CTPM element is obtained. This procedure is then repeated for all the elements needed for the CTPM matrix.

The major problem with calculating the CTPM for the DCWM is that the number of possible n-tuple outputs seen by the spy is very large. This makes both the number of elements in the CTPM and the CPU time required to compute the CTPM large. To make the CPU time requirement reasonable, the number of outputs seen by the spy for the CTPM must be limited. For the calculations in Chapter 6, the number of spy observations is limited to the first twenty time intervals for the DCWM. For a discretization of 0.2 seconds, the channel outputs actually modeled are 0, 0.2, 0.4, ..., 3.4, 3.6, and greater than 3.6 seconds. The effect of this limitation on the results of Chapter 6 is discussed next.

5.4.5 Semi-Markov Model Validation

The semi-Markov model developed above can be validated with respect to our CTSS simulator. The validation consists of simulating the scheduling algorithms of interest, keeping track of the scheduler's equilibrium state probability and interquantum time density function, and comparing these figures with those obtained from the model for the DMC. One good measure of the model validity is obtained by comparing a representative set of channel capacities based on the simulator density function and on the model density func-

second, a 0.4 second or a 1.0 second quantum length. The unconstrained channel allows the confined subsystem to choose a 0, 0.2, ..., Q quantum length where Q is infinite for FCFS, is 2.0 seconds for FB and RR2, and is 1.0 for RR1. The capacities for the two methods are very close for the three-input channel (which is the channel to be extensively studied in Chapter 6) and are within 10% for the unconstrained channel. The 10% error is considered reasonable for our purposes since most of the results are concerned with varying the capacity by orders of magnitude rather than by 10%.

For the DCWM, it is impossible to obtain an accurate probability density on even 2-tuples of spy interquantum times from the simulator. Thus the validation of the semi-Markov model done by the DMC analysis above must suffice. The validation procedure above does include the transition probabilities and the transition time density functions in the results of Tables 5.5 through 5.7 since they are needed to compute the equilibrium state probabilities. Thus these parameters are, at least indirectly, validated by the DMC procedure.

The remaining question is whether the truncation of the channel inputs to the first 20 observations will affect the results. This question for the DMC channel (which is the only one for which a capacity comparison can be made) is

by Scherr. The semi-Markov model itself is validated by comparing the channel capacities obtained directly from the simulator with the capacities obtained from the model. The approach of using only 20 channel outputs in the capacity calculation for the DCWM does not appear to affect the channel capacity calculated. Now that the method for obtaining the covert channel capacities has been discussed, the model results can be analyzed in the next chapter.

<u>Number of Users</u>	<u>Scheduler</u>	<u>Capacity All Outputs</u>	<u>Capacity 20 Outputs</u>
25	FB	1.132	1.130
25	FCFS	1.116	1.114
25	RR2	0.973	0.973
25	RR1	0.782	0.782
15	FB	1.365	1.365
15	FCFS	1.361	1.359
15	RR2	1.320	1.319
15	RR1	1.256	1.256
5	FB	1.517	1.517
5	FCFS	1.521	1.520
5	RR2	1.512	1.512
5	RR1	1.497	1.497

Table 5.8
DCWM with Limited Channel Outputs

answered in Table 5.8. Here the capacity calculated using only 20 interquantum observations is compared with the capacity calculated using all the observations for the channel. The channel used in Table 5.8 allows the confined subsystem to choose between a 0, 0.4 or 1.0 second variable quantum. The conclusion is that limiting the outputs to twenty does not significantly affect the DMC capacity. This means that most of the capacity is determined by a few high probability interquantum times that are contained within the first 20 interquantum times. Thus it is reasonable to expect that for small n , the capacity of channels sending n consecutive letters will also be closely approximated by the DCWM model used.

5.5 Summary

This chapter defines the semi-Markov model used in the capacity calculations and how the channel capacities can be calculated from the model. Our CTSS simulator used in generating probability densities for the model parameter calculation algorithm is validated against a similar simulator written

6. ANALYSIS OF CONFINED SCHEDULING POLICIES

6.1 Introduction

Designing an operating system involves making difficult choices between sometimes opposing system objectives. One typical trade-off is the evaluation of the cost of implementing a certain feature in the operating system versus the benefit it will provide. If the benefit is perceived to outweigh the cost, then the feature is implemented. A good example of this type of consideration is in the design of a confinement mechanism. However, in the case of confinement, there are very few studies and even less experience in implementing confinement for covert channels in timesharing systems. The objective of this chapter is to answer some of the questions about the cost of confinement and the characteristics of the CPU scheduler covert channel for our simple CTSS-like timesharing system, by using the semi-Markov model developed in Chapter 5. Such an investigation will yield insight into the confinement of other similar covert channels that are present in a typical timesharing system.

This chapter explores different issues relevant to making design decisions about confinement. In discussing these issues, data for each of the scheduling algorithms discussed in Chapter 4 (i.e. round-robin, first-come-first-served, and two-level feedback) is presented for the cases in which the algorithm used may make a difference in the results. The first issue discussed is the amount of information leakage that can be transmitted through the covert CPU scheduling channel in the event that no confinement mechanism is implemented. Since the amount of leakage will vary with both the scheduling algorithm and the number of users on the system, both of these factors are discussed. Other important factors in these measurements are the choice of the discretization interval for the semi-Markov model, equilibrium quantum length and system workload. The effects resulting from all these variables are analyzed.

If the amount of leakage that occurs for no confinement is more than can be tolerated, the cost of completely closing this channel by absolute confinement must be evaluated. The cost function used is a combination of the response time and the wasted CPU time. If neither absolute confinement nor no confinement is satisfactory, then an intermediate choice is provided by implementing different channel models in the CPU scheduler to give a reasonable trade-off between cost and information leakage. The channel models analyzed are constructed by using the information channel synthesis techniques explained in Chapter 2. Other techniques that are investigated to obtain a favorable trade-off include basing the return to equilibrium for the DMC channel model on real time and implementing the channel with memory model. The last issue to be discussed examines the variation of channel capacity with the number of spy processes to determine if several spies can do better than one spy in receiving leaked information. Finally, implications for the other covert channels present in operating systems are drawn from the results for the scheduler channel.

Before the results are presented for the scheduling covert channel, a brief summary of the analysis assumptions for the semi-Markov model and the cost function are presented. This is done in the next section.

6.2 Background for the Channel Analysis

The purpose of this section is to briefly review the major points in Chapters 3 through 5 that affect the analysis presented in the remaining sections of this chapter and to define in detail the DMC and DCWM channel models. This section provides the necessary background for the remainder of the chapter.

6.2.1 Operation of the Scheduling Covert Channel

The scheduling covert channel leaks information from one process to another by allowing each process to choose the amount of CPU time it will actually use each quantum that it is allocated. The choice of quantum length affects the amount of time the other processes spend in the CPU queue waiting for service. The process can regulate its quantum usage by executing for a measured amount of time before waiting on input from the user or by suspending its own execution. For example, in a single CPU system, if process A chooses to use all of its quantum, the processes waiting for the CPU will be delayed longer than if only a small part of the quantum is used. This dependence of the time between quantum allocations (interquantum time) of one process (the spy) on the CPU time used by another process (the confined subsystem) is the method by which information is transmitted through the scheduling covert channel.

To execute a process confined, the user makes a special request to the operating system. The effect of this request is to make the operating system aware of which process may try to transmit information. However, there is no method by which the operating system can identify the spy process since it may be disguised. Since the spy can not be prevented from observing the output of the scheduling covert channel because its identity is unknown, the amount of information transmitted by the confined subsystem must be restricted and measured. The method for accomplishing this is to allocate quanta to the confined subsystem based on some information channel model with a measurable channel capacity. Several different types of channel models will be analyzed in later sections. The measurements taken are based on a simple timesharing system developed at MIT called the Compatible Time-Sharing System (CTSS). A brief description of this system is given next.

6.2.2 Important Characteristics of the CTSS System

The CTSS system developed in the early 1960's was one of the first timesharing systems. The host machine was an IBM 7094 with a maximum user memory of 32K words. The operating system component that most affected the performance of the system was memory management. The memory management algorithm swapped in every program starting at the same memory location (address zero of the user memory). Thus at most one complete program was in memory at any one time. For this reason, the process switching procedure always required a swap out of the current user and a swap in of the next user before the next user could be executed. Another consequence of the memory management algorithms was that no overlapping of one user's CPU time and another user's I/O time could occur. This means that the I/O characteristics of a user's program do not need to be explicitly modeled. Instead, the CPU quantum allocated includes any user I/O that is done. Also, the system overhead (not including swapping) was found by Scherr [Scherr66] to be proportional to the amount of CPU time actually used during the quantum. Like the user I/O time, the system overhead is included in the user CPU time and is not separately modeled.

The above system features are incorporated in a closed system simulation model that is shown in Figure 5.1 from Chapter 5 and consists of two service stations. One station representing thinking users has an infinite number of servers. The service distribution is nearly exponential with a mean of 35.2 seconds as found by Scherr. The other station represents the CPU and has a single server. Processes completing a CPU quantum either return to the system of queues for the CPU if more CPU time is required or go to the service station representing thinking users. (A user can have at

most one process competing for CPU time at any given time.) The confined subsystem and the spy are, however, constrained to always return to the CPU queue and to never enter the thinking user station as explained in Chapter 4. A more detailed discussion of the simulator model is presented in Chapter 5.

6.2.3 Summary of Conventions and Major Assumptions Used in the Analysis

There are several conventions to remember when reading the following sections. The first is that when a quantum length is specified for unconfined processes, it refers to the maximum amount of useful CPU time that can be allocated to an unconfined process each time the process is eligible for execution. Thus any swap time needed to make the process ready for execution is not included in the quantum length. On the other hand, when a quantum length is specified for the confined subsystem, it includes both the useful CPU time used and any time needed to ready the process for execution (including swap time). This difference in the definition of quantum length for the confined subsystem is necessary to obtain an accurate estimate of the covert channel capacity. The channel capacity computed can only be accurate if the entire perturbation made by the confined subsystem is included in the channel model.

Another convention used in the analysis is that whenever a system load in terms of number of users is specified, the number does not include either the confined subsystem or the spy. Thus if it is stated that there are 5 users on the system, there are really 5 unconfined users, the confined subsystem, and the spy.

The last convention used is the naming of the scheduling algorithms analyzed. RR1 stands for round-robin with a 1 second quantum length, RR2 stands for round-robin with a 2 second quantum length, FCFS stands for first-come-first-served, and FB stands for two-level feedback as explained in Chapter 4. Remember that for the FB algorithm, the confined subsystem is constrained to circulate in the lower priority queue while the unconfined processes move between the two queues depending on their execution characteristics.

There are two major constraints on the system operation to allow the channel capacity to be measured. The first is that the system is assumed to be discretized. This simply means that processes begin execution at some multiple of the discretization constant. For this analysis, the discretization constant is 0.2 seconds unless noted otherwise. This constraint permits the covert channel capacity to be computed using the discrete memoryless channel (DMC) and discrete channel with memory (DCWM) models by ensuring that the spy observes only discrete channel outputs. A more detailed discussion of this constraint was presented in Chapter 5.

The second constraint prevents the confined subsystem from sending arbitrarily long sequences of continuously transmitted code letters. With the present state-of-the-art of information theory, this type of transmission strategy through a channel with memory (which characterizes the usual scheduling channel) can not be analyzed with respect to the channel capacity in the general case. Thus the channel is constrained to return to a confiner chosen equilibrium state at certain intervals to allow analysis. This is an inherent constraint in providing partial confinement having a measurable capacity as explained in Chapter 5.

In the analytic modeling of the CTSS system, several assumptions are made to permit the creation of a semi-Markov model and the calculation of channel capacity. The assumptions used are: 1) Each user has an exponential think time. 2) The number of CPU quanta required for each user interaction is geometrically distributed. The distribution of the

total CPU time required for each interaction is a function of the geometric quantum requirement distribution, and the terminating and non-terminating probability functions obtained from the simulator. 3) The behavior of a process is not a function of the load on the system. 4) The spy process takes a minimal amount of CPU time each execution and has a minimal program length. Also the spy process can not be identified by the operating system. 5) The confined subsystem is identified to the operating system by the confiner before execution begins which allows the operating system to schedule the confined subsystem differently than unconfined processes. 6) Neither the confined subsystem nor the spy know the total number of processes requesting CPU time or the number of processes in any CPU queue (if there is more than one queue). The above assumptions are covered in more detail in Chapters 4 and 5.

6.2.4 The Semi-Markov Model

The purpose of the semi-Markov model defined in Chapter 5 is to enable the channel transition probability matrix (CTPM) of the scheduling covert channel to be calculated, to determine how many equilibrium quanta must be allocated to return the system to near equilibrium after a perturbation caused by the confined subsystem, and to investigate the DCWM channel model. The model states represent the state of the scheduler just after the spy is allocated a quantum and thus has state transition times representing the interquantum times seen by the spy. This is a natural formulation of the semi-Markov model since information is transmitted to the spy by the spy process observing its interquantum times. The model state transition probabilities and transition times can be calculated from parameters supplied by the CTSS simulator as explained in Chapter 5. In particular, the probability of a process terminating during the current quantum, the time density of a terminating quantum and the time density of a non-terminating quantum are used. To the effect of the unconfined processes is added the effect of the spy process and the confined subsystem to compute the transition probabilities and transition time densities for each state. Computing the CTPM from the semi-Markov model uses only the structure of the model in the calculation since brute force techniques are needed to obtain the required probability densities from the model. All the probability information needed is included in the model. The remaining two sections explain the two types of channel models that are analyzed with the semi-Markov model.

6.2.5 The DMC Model

To permit the scheduler channel in a timesharing system to be represented as a DMC for the calculation of channel capacity for partial confinement, the same CTPM must be used for each code letter sent. For the scheduling covert channel, a code letter corresponds to a quantum allocated to the confined subsystem with a length chosen by the confined subsystem (i.e. a variable quantum). In this situation, the CTPM represents the "equilibrium state" of the system before the variable quantum is allocated. In terms of the model developed in Chapter 5, the equilibrium state is characterized by the state probabilities being close to the equilibrium state probabilities. Of course, immediately after the confined subsystem is allocated a variable length quantum, the equilibrium state may no longer exist due to the disturbance caused by the confined subsystem. To reestablish the equilibrium state, the confined subsystem is allocated a fixed length quantum (called an equilibrium quantum length) for a sufficiently long time before another variable quantum is allocated. Since the system must be in equilibrium before every variable quantum is allocated, the channel can not "remember"

previously transmitted input letters by having the CTPM be dependent on the previous channel inputs. The allocation of equilibrium quanta is done here to enable the channel capacity to be calculated, as pointed out in Chapter 2.

In practice, a timesharing system does not return exactly to an equilibrium state in a finite amount of time after a perturbation. Thus the DMC model requirement, which is that the equilibrium state be reestablished before each variable quantum is allocated, is never precisely satisfied. For this reason, an approximation to the equilibrium state is used to permit analysis of the channel capacity. When the system state is "close enough" to the equilibrium state, it is assumed that the equilibrium state has in fact been reestablished. A natural metric for measuring the nearness to equilibrium is the DMC capacity function. For the channels to be investigated, the system is assumed to have returned to its equilibrium state when the DMC channel capacity (which is a function of the state probabilities) returns to within 10% of its equilibrium value, given that the largest system perturbation occurs. Thus the return to equilibrium is given by a measure of the closeness of the state probability distribution after the perturbation, to the equilibrium state probability distribution. The time required to return to equilibrium is measured in *rounds* where one round is defined to be the time between two successive executions of one process (e.g. the spy). From the semi-Markov model, which is based on the rounds seen by the spy, the equilibrium requirement is computed by (1) solving for the equilibrium state probabilities and computing the equilibrium DMC channel capacity based on these probabilities, (2) simulating the effect of allocating the quantum producing the largest perturbation to the confined subsystem by using the state transition probabilities of the model, and (3) simulating the allocation of a constant quantum length (equilibrium quantum length) to the confined subsystem by using the state transition probabilities of the model until the DMC channel capacity is within 10% of the equilibrium value. This definition of equilibrium is, of course, not the only possible definition. Any function of the system state could be used instead of the capacity measure. However, the capacity measure seems most appropriate since the channel capacity is the quantity of interest.

The choice of the equilibrium quantum length to be used affects both the confiner and the unconfined processes. If the equilibrium quantum length is zero, the equilibrium state of the system represents the confined subsystem not being present in the CPU queue. This choice yields the least effect on the unconfined processes as measured by their response times. However, for this equilibrium quantum choice, the number of equilibrium quanta that must be allocated between two variable quanta is usually large, which increases the response time of the confiner. For example, based on the model of the CTSS-like system introduced in Chapter 5, the number of equilibrium quanta allocated for a 25 user load can be as large as eight for the round-robin scheduler with a quantum length of one second. This means that if the confiner's think time ends just after a variable quantum is allocated, the confiner is not allocated more CPU time until eight rounds have occurred, which could take several minutes. Also, if the CPU requirement of the confined subsystem is long and requires several quanta to complete, the total real time needed for the interaction may be large due to the possibly large amount of real time between variable quanta.

Using equilibrium quanta to reestablish the equilibrium condition can affect the comparison of two scheduling algorithms with respect to capacity and cost. If scheduler A requires four equilibrium quanta to reestablish equilibrium and scheduler B only requires two equilibrium quanta for the

same workload, then scheduler B could have a lower cost than A due to the larger number of variable quanta allocated. That is, for 20 quanta allocated by both A and B, A would allocate five variable quanta but B would allocate ten. The capacity measured in bits/useful CPU second is directly proportional to the number of variable quanta allocated. Thus in the above example, if A and B had the same capacity in bits/channel use, then B would usually have a higher capacity in bits/useful CPU second.

To make confined scheduling more attractive to the confiner, the average confiner response time can be decreased by allocating an equilibrium quantum length that is greater than zero. Now if the confiner's think time ends just after a variable quantum is allocated, the CPU requirement may be satisfied by the equilibrium quantum allocations without ever using a variable quantum allocation. This method usually increases the CPU time wasted by the confined subsystem and also increases the response time of the unconfined users. But if good response time is important to the confiner, the extra cost may be justified. A side benefit to the confiner of allocating a non-zero equilibrium quantum is a decrease in the channel capacity measured in bits/channel use. The decrease is caused by an increase in the average CPU queue length resulting from the heavier workload represented by the confined subsystem. For example, in round-robin scheduling with 25 users, a maximum quantum length of 2 seconds, and possible variable quanta values of 0.0, 0.2, 0.4, ..., 2.0 seconds with an equilibrium quantum of 0.4 seconds, the channel capacity in bits/channel use is 40% of the channel capacity with an equilibrium quantum of 0.0 seconds.

6.2.6 The DCWM Model

The other type of channel to be analyzed is the discrete channel with memory (DCWM). For this channel, the CTPM is not the same for each input sent since the equilibrium state is not reestablished between consecutively sent code letters as noted in Chapter 2. However, in Chapter 2 it was pointed out that in order for the capacity of the channel with memory to be computable with a limited amount of computer resources, the equilibrium state must be reestablished after some number of consecutive variable quantum allocations. In a DCWM, the current state of the channel (i.e. the number of active processes in the CPU queue) will affect the channel outputs seen by the spy for all code letters sent until the equilibrium state is reestablished. For example, the probability of the spy observing a round time of 20 seconds followed by a round time of 0.2 seconds for two consecutively transmitted variable quanta of 0.0 seconds (no equilibrium quantum in between) is different than the probability of the spy observing the same sequence of outputs in the DMC channel model in which there are equilibrium quanta in between. The reason is that in the DCWM model, the long round time indicates that there are many processes in the CPU queue waiting for service. For a short interquantum time to be observed after the long interquantum time, all the unconfined processes must terminate during their quantum and no processes can enter the CPU queue from the think mode. The 0.2 second round time represents the execution time of the spy only. A typical DCWM CTPM would have a few large probability entries representing certain combinations of consecutively sent inputs and would have many very small probability entries.

The DCWM model is different from the DMC model in that the average capacity per code letter transmitted may either increase or decrease with the number of consecutively allocated variable quanta. For example, assume the scheduling covert channel has an equilibrium quantum length of 0.0

seconds and quantum lengths of 0.0 and 1.0 seconds allocatable to the confined subsystem. The equilibrium state is found for the case of zero time being allocated to the confined subsystem. When a variable quantum of length 0 seconds is allocated to the confined subsystem, the state of the system continues to be the same as the equilibrium state and the CTPM for the next consecutive code letter sent is the same as the equilibrium CTPM. If, however, a 1.0 second quantum is allocated to the confined subsystem, the system is perturbed, resulting in larger average CPU queue lengths for the second letter sent than for the equilibrium state. Thus the states in the semi-Markov model representing a large number of users in the CPU queue become more probable at the expense of the states representing a small number of users. A representation of this situation is shown

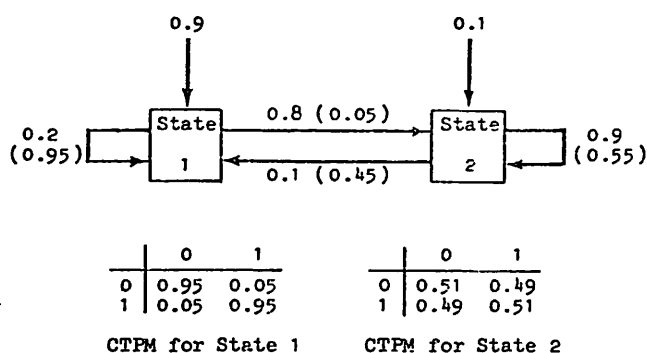


Figure 6.1

Channel with Memory Having Decreasing Capacity

in Figure 6.1. State 1 represents a small number of processes in the CPU queue (high capacity state) and state 2 represents a large number of processes in the CPU queue (low capacity state). The state transition probabilities in parentheses are for the confined subsystem being allocated a 0 second quantum and the transition probabilities not in parentheses are for the confined subsystem being allocated a 1.0 second quantum. The equilibrium state probabilities (0.9 for state 1 and 0.1 for state 2) are obtained by using the parenthesized transition probabilities since the equilibrium quantum length is zero. The CTPM for the DMC model of this channel is

	0	1
0	0.906	0.094
1	0.094	0.906

Table 6.1

CTPM for the DMC Model of Figure 6.1

	00	01	10	11
00	0.8295	0.0765	0.0765	0.0175
01	0.0765	0.8295	0.0175	0.0765
10	0.0541	0.0399	0.5395	0.3665
11	0.0399	0.0541	0.3665	0.5395

Table 6.2

CTPM for the 2-Bit DCWM Model of Figure 6.1

shown in Table 6.1 and is computed by weighting each state CTPM by its equilibrium probability of occurrence. The 2-

letter DCWM model of this channel has the CTPM shown in Table 6.2 and is computed by tracing each pair of inputs through all possible state combinations using the state transition probabilities. For this channel, the DMC capacity is 0.550 bits/channel use and the DCWM capacity is 0.439 bits/channel use. Note that the actual capacity of the DCWM channel is 0.878 bits/two quantum allocations, but since each channel input consists of two quantum allocations to the confined subsystem, the capacity is 0.439 bits/channel use. The decrease in capacity for the DCWM case is a result of the first input letter having a high probability of being transmitted in the large capacity state (state 1) and of the second input letter having a lower probability of being transmitted in the large capacity state. Thus the average capacity per input letter for the 2-letter DCWM is less than the DMC capacity.

An example of a channel in which the capacity in bits/channel use for two consecutively transmitted input letters is greater than for one consecutively transmitted input

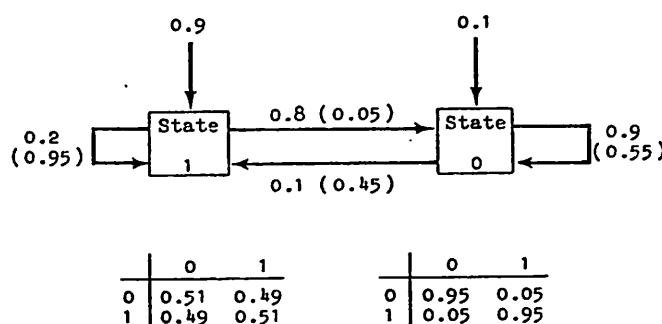


Figure 6.2

Channel with Memory Having Increasing Capacity

is shown in Figure 6.2. This is the same channel as in Figure 6.1 except that the CTPM for state 1 is exchanged with the CTPM for state 2. The CTPM for the DMC case is shown in

	0	1
0	0.554	0.446
1	0.446	0.554

Table 6.3

CTPM for the DMC Model of Figure 6.2

	00	01	10	11
00	0.3156	0.2384	0.2384	0.2076
01	0.2384	0.3156	0.2076	0.2384
10	0.3847	0.0613	0.4817	0.0723
11	0.0613	0.3847	0.0723	0.4817

Table 6.4

CTPM for the 2-Bit DCWM Model of Figure 6.2

Table 6.3 and for the 2-bit DCWM case in Table 6.4. The DMC capacity for this channel is 0.0843 bits/channel use and the 2-bit DCWM capacity is 0.216 bits/channel use. The increase in capacity for the DCWM case is a result of the first input having a high probability of being transmitted in the

low capacity state (state 1) and of the second having a lower probability of being sent in the low capacity state. Because the capacity of a DCWM might increase or decrease over the DMC case, the continuous sending policy for scheduling algorithms must be carefully analyzed to avoid underestimating or grossly overestimating the channel capacity. The question of how the scheduling channel capacity varies with the number of consecutively allocated variable quanta is discussed in a later section.

The possibility of the DCWM capacity varying with the number of consecutively transmitted code letters means that even if the capacity of the general DCWM for an infinite number of consecutively transmitted code letters is calculable, it is not necessarily the correct capacity when only a finite number of quanta are allocated. For example, if the channel being analyzed has a strictly decreasing channel capacity, then the infinite consecutive code letter case has a capacity of zero bits/channel use while the finite consecutive code letter case has a positive capacity. In the scheduling covert channel, since all processes terminate in a finite amount of time, the number of consecutively sent code letters is finite and the capacity calculation must account for this. In practice, for the covert scheduling channel capacity to be computable for a finite number of consecutively sent code letters, the number of consecutively allocated variable quanta can not exceed three due to the amount of resources needed to perform the calculation. This is an inherent limitation in the analysis method. For example, assume the number of channel outputs observable by the spy is 20. This is a very conservative figure and is generally much lower than the usual number of observations. If four consecutive variable quanta are allocated, the spy can observe any four-tuple of channel outputs. Each row of the CTPM would then consist of 20^4 or 160,000 elements. If the confined subsystem can choose from any one of three possible variable quantum lengths, the number of rows in the CTPM is 3^4 or 81. The total number of computer words necessary to hold the CTPM is then 12.96 million which is difficult to process efficiently on present computer systems. In addition, the amount of computer time needed to compute the CTPM elements and to compute the capacity would be prohibitive. Thus the amount of effort and resources needed to compute the channel capacity is large even for a modest number of consecutive variable quanta. In this thesis, the DCWM capacity will be computed for the allocation of at most three consecutive variable quanta with a choice of three different quantum lengths for each variable quantum. To decrease the number of channel outputs seen by the spy, only the first twenty outputs (corresponding to 0, 0.2, 0.4, ..., 3.6, and greater than or equal to 3.8 seconds) are used in the calculation. The difference in the DMC channel capacity between using 20 outputs and all outputs is less than 0.1%. By limiting the number of assumed spy observations and the number of possible variable quantum allocations, the CTPM size and capacity calculation time is feasible.

For the above reasons, the DCWM model to be investigated will allocate at most three consecutive variable quanta before forcing the system to return to equilibrium by allocating a sufficient number of equilibrium length quanta. The number of equilibrium quanta allocated is computed by determining how many are required to return the capacity for the first code letter sent in the block of consecutively sent code letters (DMC model) to within 10% of its original DMC capacity given the worst case perturbation. The method for tracing the effects of the perturbation through the semi-Markov model is the same as outlined in Chapter 5. The same considerations for the choice of the equilibrium quantum length (zero versus non-zero) apply to the DCWM as to the DMC.

6.3 Cost and Information Leakage Functions

Two measures of the performance of a timesharing system are the average response time per user interaction and the amount of resources needed to complete a given task. The average response time is indicative of the amount of work that can be performed in a given amount of real time, and the resource usage determines the computation cost of using the system. These measures are not independent and represent a trade-off decision for the confiner. In order to capture the true confiner cost associated with a confined scheduling algorithm, both response time and resource usage must be used in the cost equation. Otherwise, the scheduler that optimizes the confiner's response time allocates an infinite quantum length to the confined subsystem and the scheduler that optimizes the computer resources used (i.e. wastes no CPU time) allocates a near-zero quantum length to the confined subsystem. Neither of these extremes is desirable since in the first case the unconfined users are provided almost no CPU time and in the second the confiner has a very long response time. A combination of response time and CPU cost was previously used in [Gold69] to compare timesharing and batch systems. A slight modification to the cost function used in [Gold69] to account for confinement is:

$$\text{Confiner Cost} = \$/\text{useful CPU second} = \{(\text{Cost of a CPU second}) \times (\text{Average number of wasted plus useful CPU seconds allocated for each useful CPU second})\} + \{(\text{Cost of a user's real second}) \times (\text{Average response time per interaction}) \times (\text{Average number of interactions per useful CPU second})\}$$

The first term in {...} represents the cost of allocating enough CPU time on the average to obtain one CPU second of useful work. This term does not include process swap time but does include useful CPU time and wasted CPU time. A point estimate of the term (average number of wasted plus useful CPU seconds allocated for each useful CPU second) can be obtained from a simulation of the scheduling algorithm by dividing the sum of the total number of wasted CPU seconds and the total number of useful CPU seconds allocated to the confined subsystem by the total number of useful CPU seconds. The (cost of a CPU second) term is determined by amortizing the cost of a system with the approximate power of the CTSS IBM 7094 used in the simulations over a five year period. A comparable machine on today's market is the PDP-11/70 or its equivalent which can be purchased with a "reasonable" configuration of memory, disks and a printer for approximately \$250,000. The cost for running the system for five years including personnel and material costs will be approximately another \$250,000, based on experience with the PDP-11/70's at the University of California, Berkeley. Amortizing the total amount of \$500,000 over five years means that there must be a recovery of \$100,000/year. By assuming the system runs a full eight-hour shift, 5 days a week, 50 weeks a year (which accounts for holidays and emergency maintenance time), there are 2000 hours/year. The cost per hour is then \$100,000/2000 or \$50/CPU hour. This \$50/hour figure is the one used in the cost calculations.

The second term in {...} is the cost to the confiner resulting from poor response time. This term equates the user's real-time cost with his salary as an estimate of the response time cost. The average response time per interaction is measured from the simulations. The number of interactions per useful CPU second is given by Scherr in [Scherr66] as (1.0/0.88). The cost of a user's real second is determined from an assumed salary of \$24,000/year or \$0.0032/real second. This value assumes that programmers and not managers, who are generally more highly paid, will be the

ones who actually interface to the timesharing system. If people in a different salary range interface with the system, the user cost changes and the trade-off between the CPU time expended and tolerable response time changes. An example of the user cost being very high is if a very high priority is placed on completing a confined job within a specified time limit. In this case it could be more cost effective to waste large amounts of CPU time to insure completion. The values for both CPU cost and user cost in this thesis are chosen to represent values in an everyday situation. The analysis can be carried out for other values of these parameters representing special situations in a straightforward manner. An analogous cost function could be obtained for the unconfined users of the system which would also include response time and CPU time but not swapping time. Such a function will be used for this purpose in several places in the following sections.

To evaluate the cost function, several of the parameters will be obtained from simulations of the CTSS-like system. To determine the simulation error for these parameters, the methods explained in [Crane74] and [Lavenberg75] are used to obtain a confidence interval. The methods are based on the fact that the simulation contains points in time at which the system being simulated returns to the same state (called regeneration points). The simulation between any two consecutive regeneration points is then independent of the simulation between any other two consecutive regeneration points. This produces a series of independent, identically distributed (iid) intervals. Classical statistics is then used on the iid intervals to obtain a variance estimate and a confidence interval for each of the parameters. For the measurements reported here, the confiner cost for the 5 user case has an error of ± 3 to 8 percent; for the 15 user case has an error of ± 5 to 9 percent, and for the 25 user case has an error of ± 9 to 15 percent.

For partial confinement, the channel capacity for the covert scheduling channel must be obtained. The channel capacity is expressed in units of bits per useful CPU second allocated to the confined subsystem, to be compatible with the computer cost function expressed in units of \$ per useful CPU second (or \$/useful CPU second). The expression for the capacity is:

$$\text{Capacity} = \text{Bits/useful CPU second} = (\text{Capacity per variable quantum allocated in bits/channel use}) \times (\text{Average number of variable quanta per useful CPU second})$$

Remember that the capacity of a channel assumes that the optimal encoding and decoding strategy is used by the confined subsystem and the spy. Any information rate from zero up to the channel capacity is theoretically possible. The number of variable quanta allocated per useful CPU second is obtained from the simulation of the scheduling policy. The confidence intervals for the capacity in units of bits/useful CPU second range from 8 to 12 percent for the lightly loaded case (5 users), to 12 to 18 percent for the fully loaded case (25 users). Again, the higher error is for the low capacity cases and the lower error is for the high capacity cases. The error for the capacity measured in bits/channel use is much lower (on the order of 3%) than the bits/useful CPU second measure. The channel capacity is achieved by the confined subsystem varying his quantum length to send the maximum amount of information per quantum without regard for the response time seen by the confiner. A confined subsystem that achieves the channel capacity does not usually have the characteristics of a typical process. A typical process is characterized by a relatively short CPU requirement (0.88 CPU seconds on the average) followed by a relatively long

user think time (35.2 seconds on the average). This behavior will usually not yield the channel capacity since too many short quanta (e.g. when the confiner is thinking) are allocated. Thus the confined subsystem that is trying to leak the maximum amount of information will generally be allocated more CPU time to perform a task than a normal process doing the same task.

For the simulations here, the confined subsystem is assumed to be a normal process. Thus the cost and capacity measures reported do not explore the trade-off between actual encoding/decoding strategies and CPU cost. Here, the cost function represents the penalty paid by the confiner for executing a non-sending confined subsystem under confinement. However, the capacity function represents the channel capacity for the confined subsystem and the spy performing the optimal encoding and decoding strategy. This gives the maximum amount of information that could be leaked if the confined subsystem and the spy were trying to communicate.

6.4 Information Leakage in a System Without Confinement

To study the extent of the information leakage problem on systems not providing confinement, the unconstrained channel model is used. The unconstrained channel model is a DMC that permits the "confined subsystem" (or sender in systems not offering confinement) to choose which quantum length is allocated each time it is eligible for execution. Since the semi-Markov model is a discrete model, there are a finite number of quantum lengths to choose from. For the FB and RR2 schedulers, the maximum quantum size is 2.0 seconds and for the RR1 scheduler is 1.0 second. Thus for the discretization interval of 0.2 seconds, the sender can choose quantum lengths of 0.0, 0.2, ..., 2.0 seconds (11 inputs) for the FB and RR2 schedulers, and quantum lengths of 0.0, 0.2, ..., 1.0 seconds (6 inputs) for the RR1 scheduler. To make use of this channel, the confined subsystem must be able to regulate the quantum allocated (for a non-zero quantum length) and skip quanta (for a zero quantum length). To regulate the length of non-zero quanta, the sender can time its execution by using the real time clock which is accessible in most systems. When the sender has used the amount of CPU time it wanted to for this quantum allocation, it gives up the CPU to the next user. In the original CTSS system, this could be done by the sender process changing its length. In most systems, there is some facility for doing a similar thing, such as issuing a "sleep" system call which suspends the process until some event occurs. To regulate the number of zero length quanta allocated, an event type mechanism could be used that is present on some systems. The method is for the spy process to cause a different event to occur each time it executes. The confined subsystem then waits on the event that it knows will occur a specified number of spy executions in the future. Thus the sender process is "allocated" zero length quanta until it is activated by the event for which it is waiting. It is permissible for the sender and the spy to use the event mechanism in this way since there is no restriction on information given to the sender under confinement. Only information from the sender is to be confined. If the operating system being used does not have an event mechanism or some similar feature to allow the sender process to skip executions, the zero quantum length can be approximated by the confined subsystem using a very small quantum. For this analysis in this section, the sender is assumed to be able to choose a zero length quantum.

The unconstrained DMC has the largest capacity (in bits/channel use) of any DMC for a given maximum quantum length and a given discretization size. The capacity obtained is also achievable since the sender and the spy can implement the DMC transmission method. As shown in Table

		Scheduler		
		FB	RR2	RR1
Number of Users	0	3.46	3.46	2.58
	5	3.24	3.23	2.38
	15	2.84	2.72	1.93
	25	2.32	1.94	1.17

Equilibrium Quantum Length = 0.0 Seconds

Table 6.5
Unconstrained Channel Capacity
in Bits/Channel Use

6.5 for FB, RR2 and RR1, with an equilibrium quantum length of zero seconds, the amount of leakage can vary by a factor of three depending on the system workload and the scheduler. The capacity of this scheduling covert channel model ranges from 1.17 bits/channel use for the system with 25 users and the RR1 scheduler to 3.24 bits/channel use for the system with 5 users and the FB scheduler. The theoretical maximum capacity for the FB and RR2 algorithms with zero unconfined users (zero channel decoding error) is 3.46 bits/channel use ($= \log \text{ base } 2 \text{ of } 11 \text{ inputs}$) and for the RR1 algorithm is 2.58 bits/channel use ($= \log \text{ base } 2 \text{ of } 6 \text{ inputs}$). The capacity measurements for the 5, 15, and 25 user cases are obtained from the semi-Markov system model.

The order of the scheduling algorithms with respect to the channel capacity in Table 6.5 is not surprising. The ordering of the algorithms with respect to decreasing channel capacity is the same as the ordering with respect to increasing mean CPU queue length. The longer mean CPU queue length is indicative of more channel noise. The reason for the large difference in capacity between the FB/RR2 algorithms and the RR1 algorithm is that the FB/RR2 algorithms have 11 possible sender inputs and the RR1 algorithm only has 6 possible inputs.

The capacities in Table 6.5 can be used to determine the maximum number of variable quanta that can be allocated to the sender to keep the number of bits leaked through the channel below the total permissible leakage specified by the confiner. For example, if the confiner can allow no more than 100 bits to be leaked and there are 15 users on a system using the RR1 algorithm (channel capacity = 1.93 bits/channel use) then no more than 51 variable quanta can be allocated to the confined subsystem during its execution.

Now suppose that the confined subsystem is not trying to send information to a confederate but is a normal job. In this case, the system uses the unconstrained channel model to measure the amount of potential information leakage based on the capacities in Table 6.5. The unconstrained channel model is chosen so that the confined subsystem can run efficiently by having a large choice of quantum lengths from which to match the CPU time needs of the confined subsystem. Since the confiner can not tell whether the confined subsystem is trying to communicate with a spy or not, the confiner must assume that every variable quantum allocated to the confined subsystem leaks, on the average, the number of bits shown in Table 6.5. From simulations of each scheduling algorithm and workload combination, the average number of variable quanta allocated for each second of CPU time that is spent doing useful work for the confined subsystem (useful CPU second) can be determined. By multiplying this number by the capacities in Table 6.5, a capacity in bits/useful CPU second ($= [\text{bits/channel use}] \times [\text{number of channel uses/useful CPU second}]$) can be found as shown in Table 6.6 for an equilibrium quantum length of zero

		Scheduler		
		FB	RR2	RR1
Number of Users	5	260.7	153.8	196.6
	15	75.7	44.8	35.3
	25	13.0	9.4	6.3

Equilibrium Quantum Length = 0.0 Seconds

Table 6.6
Unconstrained Channel Capacity
in Bits/Useful CPU Second

seconds. In this thesis, the term "useful CPU second" means the number of seconds of CPU time spent actually executing the confined subsystem program. Any wasted CPU time due to discretization, to intentionally allocating the wrong size quantum length, or to swapping is not included. Of course, a confined subsystem that is actively trying to send information may have its capacity in bits/useful CPU second higher or lower than the value obtained through the simulation. However, Table 6.6 demonstrates that in using confinement for a subsystem that does not try to communicate with a spy process, a significant amount of leakage must be assumed, ranging from 260.7 bits/useful CPU second for the 5 user FB scheduler to 6.34 bits/useful CPU second for the 25 user RR1 scheduler. This translates into 29,328 32-bit words per CPU hour for the 5 user case and 713 32-bit words per CPU hour for the 25 user case.

Table 6.6 does not have the same ordering of capacities for the 5 user case as does Table 6.5. This is due to the number of equilibrium quanta allocated for each scheduling algorithm for this workload. It takes one equilibrium quantum (of length 0 seconds) to return the system with a FB or RR1 scheduler to equilibrium but it takes two equilibrium quanta for the system with the RR2 scheduler. This means that when a variable quantum length is allocated by the RR2 scheduler to the sender, there is a higher probability that some amount of useful CPU time will be used. This is a direct consequence of waiting longer between variable quantum allocations for the RR2 scheduler. The result is that the number of channel uses/useful CPU second is much less than for FB or RR1. This causes the capacity in bits/useful CPU second for the 5 user RR2 scheduler to be less than for the 5 user FB or RR1 schedulers. For the RR1 and RR2 schedulers with 15 and 25 users, the number of equilibrium quanta allocated between two variable quanta does not influence the ordering of the schedulers as much as the channel capacities (in bits/channel use). Thus the ordering for the 15 and 25 user workloads is the same as in Table 6.5.

The first-come-first-served scheduler (FCFS) is not shown in either Table 6.5 or 6.6 because its capacity (in bits/channel use) is theoretically infinite for any workload. This is due to the ability of the confined subsystem to choose any positive quantum length for allocation which means there are an infinite number of possible channel inputs. For example, if the quantum lengths chosen by the confined subsystem are in multiples of some large CPU time (e.g. 10 minutes) so that the probability of a decoding error is nearly zero, then the capacity in bits/channel use is the log (base 2) of the number of inputs. Since there are an infinite number of possible inputs, the capacity is infinite. Since the FCFS algorithm is not interesting for the unconstrained channel model, the RR2 algorithm is used to demonstrate the effect of a longer quantum on the round-robin scheduler. In subsequent sections, the RR2 algorithm will not be used but will be replaced by the FCFS algorithm.

The channel capacities given in this section for the unconstrained channel model are based on the discretization interval of 0.2 seconds. Higher channel capacities can be obtained by using a smaller discretization interval as will be shown in the section discussing discretization. One of the problems with designing a system offering confinement is that the system designer does not know the encoding strategy used by the confined subsystem and the spy. To insure that the amount of leakage is not underestimated, the number of bits specified by the channel capacity must be assumed to be leaked each time the confined subsystem chooses its quantum length. This leads to a very large amount of assumed leakage in bits/useful CPU second. The following sections explore techniques for decreasing the amount of assumed leakage but at an added cost in wasted CPU time.

6.5 The Effect of Workload on Channel Capacity

The results in Table 6.5 show that the capacity of the scheduling covert channel decreases as the number of users increases, if all other factors are held constant. This effect is due entirely to the increase in channel noise caused by a larger average number of unconfined processes in the CPU queue. As the number of users increases, the high capacity states in the semi-Markov model that represent fewer users become less probable. For the DMC model, the equivalent CTPM for the semi-Markov model produced by this probabilistic shift yields a lower channel capacity.

The capacity measure that is of concern to the system designer, which is bits/useful CPU second in Table 6.6, also decreases as the number of users increases. In this case, the difference between the extremes is more than an order of magnitude. The difference is caused by two factors. The first factor is the decrease in the capacity measured in bits/channel use mentioned above. The second factor is that for a large number of users, a perturbation to the system requires more equilibrium quanta to bring the system back to its equilibrium state. Thus the number of equilibrium quanta needed to reestablish the equilibrium state for the 25 user case is greater than for the 5 user case, which causes the number of variable quanta allocated to the confined subsystem per useful CPU second to be less for the 25 user case. The decrease in capacity in bits/channel use and the decrease in the number of variable quanta/useful CPU second for the 25 user workload over the 5 user workload produce the inverse relation between the number of users and the channel capacity in bits/useful CPU second.

The implication of this inverse relationship for system design is that fewer restrictions are needed on the confined subsystem as the load on the system increases to produce a constant capacity. However, using just the workload to regulate the channel capacity is too costly for both the confiner and the unconfined processes. To obtain a small channel capacity, the system must be heavily saturated, which causes the system response time to be large. Methods for decreasing the capacity other than by regulating the workload must be explored.

6.6 The Effect of the Equilibrium Quantum Length

The equilibrium quantum length chosen for the confined subsystem affects both the channel capacity and the cost of confinement. As the equilibrium quantum length increases from zero, the steady state noise level of the system increases and the capacity in bits/channel use decreases. The number of variable quanta/useful CPU second increases until the equilibrium quantum is about midway between its two extremes, and then decreases. This is due to the number of equilibrium quantum allocations needed to reestablish the equilibrium state decreasing until the midpoint is reached,

then increasing. The cost is affected by the equilibrium quantum length chosen since a longer quantum length gives a shorter confiner response time but usually results in more wasted CPU time. The converse is also true.

A typical graph of capacity versus cost for non-zero

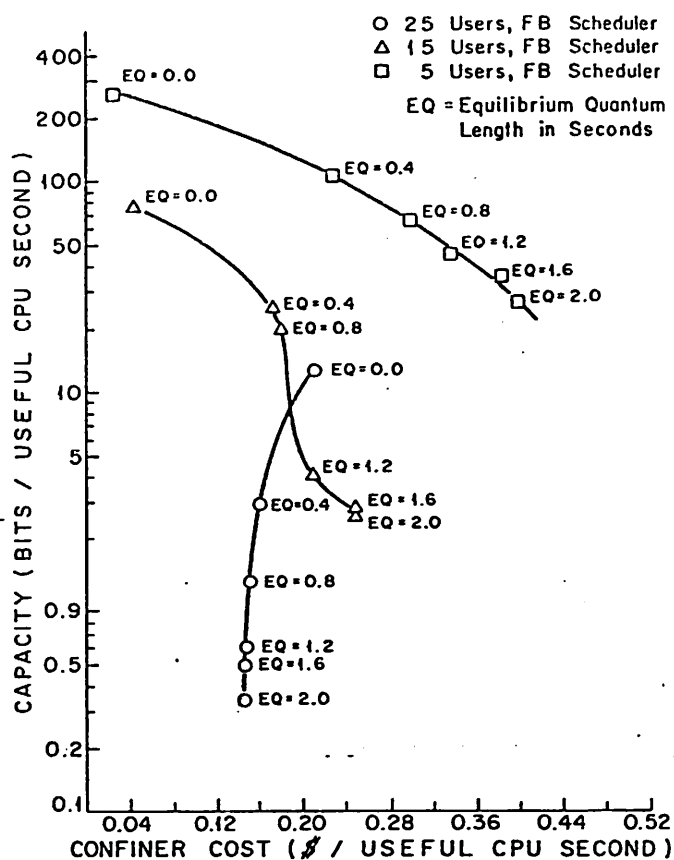


Figure 6.3
FB Unconstrained Channel Capacity vs. Cost

equilibrium quanta is shown in Figure 6.3 for the FB scheduler with different workloads. The graphs of the other scheduling algorithms exhibit features similar to the FB graph and are not shown. The most striking feature of Figure 6.3 is the different shaped curves for the different numbers of users. Both the 5 and 15 user curves show a distinct trade-off of capacity versus cost due to the negative slope. The 25 user curve with its slightly positive slope indicates just the opposite. In the 25 user case it is more cost effective for the confiner to use a larger equilibrium quantum length to decrease both the cost and channel capacity. The cost decrease for increasing equilibrium quantum lengths is due to a marked decrease in the confiner's response time. For example, for the zero equilibrium quantum length, the response time is 54.2 seconds and for the 1.2 second equilibrium quantum length, the confiner response time is 17.3 seconds. The response time cost component and the CPU time cost component are separated in Figure 6.4A for the 25 user case. The conclusion is that the confined subsystem should be allocated a large equilibrium quantum length to avoid having an extremely large response time cost. If the user cost is increased by a factor of 10, which places a very high premium on completing the job quickly, the conclusion would still be that a large equilibrium quantum should be allocated since both the CPU and response time cost curves are fairly unresponsive to changes in equilibrium quantum length for lengths in the 0.8 to 2.0 second range.

The 15 user case exhibits a definite trade-off between

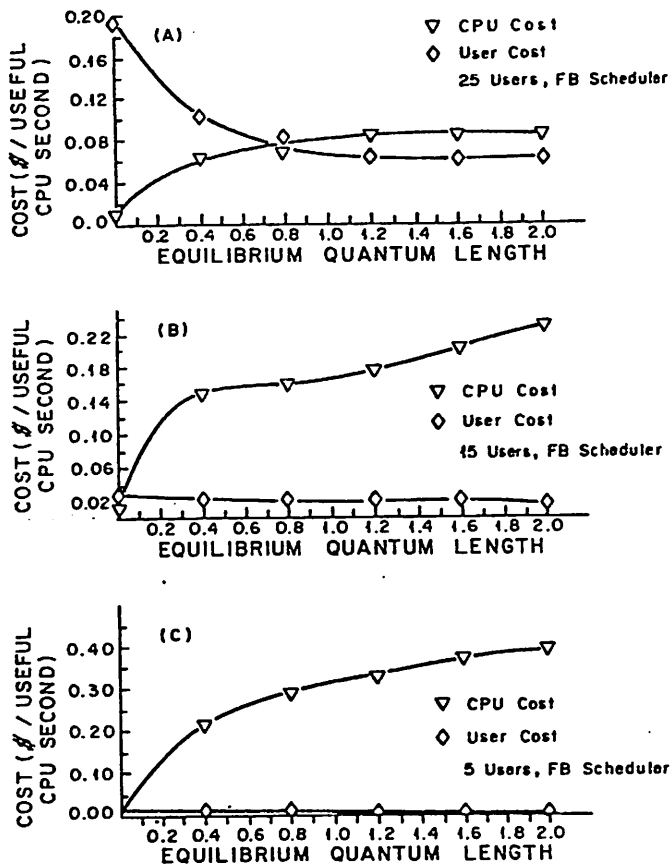


Figure 6.4
Unconstrained Channel Cost Components

channel capacity and confiner cost in Figure 6.3. As the equilibrium quantum length increases from 0 to 1.2 seconds, the capacity decreases much faster than the cost increases as shown in Figure 6.3. Equilibrium quantum lengths in excess of 1.2 seconds show a tendency to have a less favorable capacity-cost trade-off. This behavior is a result of the factors discussed at the beginning of this section. Increasing the equilibrium quantum length may be a cost effective method for decreasing the channel capacity for a limited capacity range in the 15 user case since the capacity per useful CPU second decreases by a factor of 18.5 when the equilibrium quantum length is increased from 0 to 1.2 seconds, while the cost only increases by a factor of 4.8. The dominant factor in the cost for the 15 user case is the wasted CPU time for non-zero values of the equilibrium length as illustrated in Figure 6.4B. For a zero equilibrium quantum length, the cost is much less than with a non-zero equilibrium quantum length since there is no wasted CPU time.

For the 5 user case, as the equilibrium quantum length increases from zero seconds, most of the extra CPU time is wasted, with only a small part being used to reduce the confiner's response time as shown in Figure 6.4C. Thus the cost function steadily increases as the equilibrium quantum length increases. In a similar manner, the capacity in bits/useful CPU second shows a steady decline in Figure 6.3 due to a decrease in capacity and a decrease in the number of variable quanta/useful CPU second as the equilibrium quantum length increases. As for the 15 user case, the cost for a non-zero equilibrium quantum in the 5 user case is extremely high.

The large cost incurred for non-zero equilibrium quantum lengths for the 5 and 15 user workloads is due to the linear function used for the CPU time cost. In both of these cases,

since excess CPU capacity is available, the confined subsystem could conceivably be charged a lower rate for the CPU time used. This probably would not occur if computer time is being purchased from an outside service bureau but could occur for an in-house system. Many systems offer lower rates for non-prime time computing and the extension of this rate philosophy to accommodate confinement may become a common practice. Such a CPU charging philosophy would alter the relationship of the workload curves in Figure 6.3. If the CPU time charge is decreased to zero, the entire cost would be the user cost, which makes the 5 user case the least expensive and the 25 user case the most expensive. This is the opposite ordering from Figure 6.3. Other cost functions could be found which would also change the results presented here. However, the linear CPU cost function is one of the most common charging algorithms currently in use and will be utilized in the remainder of this thesis.

The problem with using the equilibrium quantum length alone to decrease the channel capacity is that the capacity of the channel may never become low enough to suit the application, or the cost becomes extremely high. Later sections examine other capacity limiting strategies that can decrease the capacity to zero or near-zero by changing one channel parameter.

In subsequent sections the 5 user FB scheduler channel and all 15 user scheduler channels will be investigated only for an equilibrium quantum length of zero seconds. This choice minimizes the confiner cost. The objective in later sections is to produce a less costly partial confinement method than that produced by varying the equilibrium quantum length. The cost of wasted CPU time for any other value of equilibrium quantum length would be too great to meet this objective.

6.7 The Minimum Cost of Partial Confinement

The unconstrained channel model analyzed in the last three sections provides the starting point for the channel models to be investigated in later sections. This model yields the lowest cost to the confiner of the models to be studied since it allows the confined subsystem a wide choice of CPU quantum lengths when a variable quantum length is to be allocated. Since this is a low cost model, the cost to the confiner for the ability to measure the amount of potential information leakage is an interesting statistic. This cost includes the cost of discretization and the cost of implementing a DMC channel through the introduction of equilibrium quanta. The cost to the unconfined process only includes the cost of discretization. The confiner and unconfined costs for the FB scheduler are shown in Table 6.7 for the different values of the equilibrium quantum length. The costs for the other schedulers are similar and are not shown.

The minimum cost factor for partial confinement (= confiner cost/unconfined cost) ranges from 2.71 for 25 users to 1.08 for 5 users. The lower cost factors corresponding to large equilibrium quantum lengths for 25 users are not considered since most of the factor decrease is due to a rise in the unconfined cost with no significant change in the confiner cost. A light load is necessary to obtain the least cost for executing a confined subsystem as shown in Table 6.7. However, this results in a very large potential confinement leakage as shown in the previous section. The trade-off of these two factors must be weighed before the confined subsystem is executed.

Table 6.7 shows that it can be very expensive to execute a partially confined subsystem as compared to unconfined execution. Of course, execution of an absolutely confined subsystem costs even more as explained in Chapter 3. The cost of partial confinement and the cost of absolute confinement

Number of Users	Quantum Length (Seconds)	Confined Cost (=A) (\$/Useful CPU Second)	Unconfined Cost (=B) (\$/Useful CPU Second)	Cost Ratio (A/B)
25	0.0	0.213	0.0493	4.32
	0.4	0.165	0.0496	3.33
	0.8	0.151	0.0531	2.84
	1.2	0.147	0.0543	2.71
	1.6	0.147	0.0561	2.62
	2.0	0.145	0.0574	2.53
15	0.0	0.044	0.0303	1.45
	0.4	0.174	0.0322	5.40
	0.8	0.183	0.0328	5.58
	1.2	0.210	0.0333	6.31
	1.6	0.250	0.0358	6.98
	2.0	0.251	0.0369	6.80
5	0.0	0.026	0.0240	1.08
	0.4	0.230	0.0246	9.35
	0.8	0.299	0.0255	12.15
	1.2	0.338	0.0263	12.85
	1.6	0.384	0.0273	14.07
	2.0	0.397	0.0283	14.03

FB Scheduler

Table 6.7
Cost for the Unconstrained Channel

are compared in a later section to determine if partial confinement can be an alternative to absolute confinement.

6.8 The Effect of Discretization

The concept of discretization is used in the hardware implementation of all digital computer systems. The hardware sequences for the computer operations are based on a clock pulse that occurs at fixed time intervals (called the machine cycle time). Thus all events in a computer system can only occur at one of these discrete times. For example, the spy process in the covert scheduling channel can only receive channel outputs that are multiples of the machine cycle time since it must measure its interquantum time in multiples of the machine cycle time. The discretization inherent in a computer system is used in Chapter 5 to develop a semi-Markov system model. Due to the difficulty in obtaining the CTPM for a discretization interval the length of a machine cycle, the larger value of 0.2 seconds is chosen for the analysis here, as explained in Chapters 4 and 5.

In a real system providing partial confinement, the discretization interval used in the system analysis will also have to be substantially greater than a machine cycle to obtain a reasonable estimate of the channel capacity with a discrete channel model. The discretization interval chosen must also be enforced, which means that all users start their execution quanta only at times that are multiples of the discretization interval. This enforcement produces a trade-off decision for the system designer as to the appropriate value of the discretization interval. As the interval increases, the amount of CPU time that is wasted between the completion of the quantum of one user (which includes swap time) and the beginning of the quantum of the next user is usually approximately one-half of the discretization interval. This wastage affects both confined and unconfined subsystems and discretization intervals. The effect of this wasted CPU time can be viewed as an increase in the workload generated by a certain number of users. For example, a 15 user load with a discretization interval of 0.2 seconds may be equivalent, with respect to response time, to an 18 user load with a discretization interval of a machine cycle.

Balanced against this CPU wastage is a decrease in the channel capacity for a given workload as the discretization in-

terval is increased. The reason for the capacity decrease is explained in the Data Processing Theorem (discussed in Chapter 2) which states that post-processing of channel outputs can only decrease channel capacity. In this case, the post-processing involves consolidation of a large number of distinct channel outputs for the smaller discretization system into a smaller number of distinct outputs for the larger discretization system. This consolidation can only cause the capacity to remain the same or decrease, but never to increase. An example of the magnitude of the capacity changes that can be expected by varying the discretization interval is

Number of Users	Scheduler	0.2 Second Discretization Capacity (Bits/Channel Use)	0.05 Second Discretization Capacity (Bits/Channel Use)
25	FB	2.32	4.81
25	RR2	1.94	4.68
25	RR1	1.17	3.70
5	FB	3.24	5.27
5	RR2	3.23	5.27
5	RR1	2.38	4.31

Table 6.8
Unconstrained Channel Capacity for Different Discretization Intervals

shown in Table 6.8 for intervals of 0.2 and 0.05 seconds. These measurements are for the channel capacity extremes (5 and 25 users) for an equilibrium quantum length of zero to give the range of possible channel capacities of interest. For the largest amount of leakage (5 users, FB), the capacity in bits/channel use increases by a factor of 1.63 over the 0.2 second interval length case. This increases the potential leakage in bits/useful CPU second to greater than 47,700 32-bit words for a one hour production run. On the other end of the scale, the capacity for the small amount of leakage (25 users, RR1) increases by a factor of 3.16 over the 0.2 second interval case. This boosts the potential leakage in bits/useful CPU second from 713 words to greater than 2,250 32-bit words for the one hour production run. As the discretization interval approaches a machine cycle, the channel capacity with zero users and a constant system overhead for process switching becomes the log (base 2) of the quotient of the quantum length divided by the machine cycle time. For a two second quantum length and a machine cycle time of one microsecond, the largest capacity is 20.93 bits/channel use, which is a significant amount of information. If the number of unconfined users is not zero or the system overhead for process switching is not a constant, the capacity will be less.

This discretization analysis confirms the conclusion of an earlier section that the potential information leakage of current systems can be quite large since these systems have a discretization interval equal to the machine cycle time. The beneficial aspects of introducing a larger discretization interval into a system design (lower channel capacity, easier model analysis) must be weighed against the performance lost. In particular, the system will respond more slowly with a given workload and a large discretization interval than with a small discretization interval. It is our belief that for partial confinement to be a viable option, the discretization interval should be substantially larger than the machine cycle time.

6.9 The Cost of Absolute Confinement

Absolute confinement is the most expensive form of confinement for the scheduling covert channel since the scheduling algorithm must not use any information from the

state of the confined subsystem in making scheduling decisions, as pointed out in Chapter 3. This results in CPU time being allocated when it can not be used by the confined subsystem and in CPU time not being allocated when it is needed. The best absolute confinement scheduling algorithms, with respect to confiner cost, take into account the general execution characteristics of the confined subsystem to be executed. For example, if the confined subsystem is known to be I/O intensive, less CPU time could be allocated than if it is CPU intensive. But when the entire terminal session, which consists of executing many programs having different characteristics is to be confined, tailoring the scheduling algorithm can be difficult. An example is the program debugging procedure in which file editing, compilation and program execution are done in a cyclic manner. Each of these programs has distinct characteristics so that a scheduling algorithm that is optimal for one program is generally not optimal for another.

The following sections discuss the absolute confinement algorithm used in this thesis and the difference in cost between absolute confinement, partial confinement (using the unconstrained channel model), and no confinement. A cost function that accounts for the users executing unconfined programs as well as the confiner is also discussed.

6.9.1 The Absolute Confinement Scheduling Algorithm

Rather than taking into account all the different types of programs a user might execute during a terminal session, an absolute confinement algorithm that allocates a fixed non-zero quantum length to the confined subsystem every n th time the confined subsystem is eligible for execution (i.e. comes to the head of the CPU queue) is used. At all other times the confined subsystem is eligible for execution, a quantum length of zero seconds is allocated.

To obtain the quantum length and the value of n for the absolute confinement scheduler, a two step procedure is used. The first step is to determine the best choice of the non-zero quantum length by simulating an absolute confinement scheduler for $n=1$. For the 25 user FB or FCFS scheduler, this quantum length is 1.4 seconds, for the 25 user RR2 scheduler the length is 1.2 seconds, for the 25 user RR1 scheduler the length is 1.0 seconds, for the 15 user FB, RR1, and FCFS algorithms the length is 0.4 seconds, and for the 5 user FB algorithm the length is also 0.4 seconds. In the 15 user and 5 user cases, the small quantum length is necessary to avoid wasting a large amount of CPU time. A quantum length of 0.2 seconds causes a large increase in the user cost due to the inability of large confined programs to be completely swapped in and executed in one quantum allocation. Thus the 0.4 second quantum is the better choice. For the 25 user case, the large quantum length indicates that the user cost factor is much more important than in the 15 and 5 user cases. This requires a long quantum allocation to reduce the confiner's response time.

The second step is to determine the value of n which gives the lowest confiner cost given the quantum length from step 1. For 25 users with all schedulers, skipping no quantum allocations yields the lowest confiner cost. This is not surprising since this step is designed to reduce the CPU time cost at the expense of user cost. For 15 users with the FCFS, RR1 and FB schedulers, skipping three quantum allocations yields the lowest confiner cost. For 5 users with the FB scheduler, skipping nine quantum allocations yields the lowest confiner cost.

Even though the procedure for determining the absolute confinement algorithm scheduling parameters is a two step procedure, only one variable is really being varied for each workload. For the 25 user workload, n must be one to keep

the user cost low. For the 15 and 5 user cases, the quantum length allocated must be the smallest feasible length or else excess wasted CPU time results. Thus instead of exploring a two dimensional space of absolute confinement scheduling algorithms, only a one dimensional space is analyzed in each case.

Subsequent references to minimum absolute confinement cost refer to the cost obtained by the above procedure. This is not necessarily the lowest absolute confinement cost since all possible absolute confinement algorithms were not analyzed. However it will provide a benchmark for comparing the cost of absolute confinement with other confinement policies.

6.9.2 The Cost Penalty for Absolute Confinement

The absolute confinement cost given by the above algorithm is useful in answering two questions about confinement. The first question is whether partial confinement can be less costly than absolute confinement. If partial confinement is less costly, it may be preferable to use partial confinement in certain applications. To answer this question, the confiner cost of absolute confinement is compared with the confiner cost for the unconstrained channel analyzed previously. Having a lower confiner cost for the unconstrained channel does not guarantee that partial confinement would be used since the information content cost of leaking some data must be added to the cost of partial confinement. Thus a lower cost for partial confinement is a necessary but not sufficient condition for partial confinement to be used. In the analysis presented here, only the confiner cost is considered and not the information content cost for partial confinement. The second question is how much more expensive is it to execute a subsystem with an absolute confinement scheduling algorithm than to execute the subsystem with an algorithm that does not offer confinement. The comparison will assume that both algorithms are used on discretized systems. The effect of this assumption is examined critically in a later section.

The behavior of an absolutely confined subsystem with an unconfined workload of 25 users differs greatly from

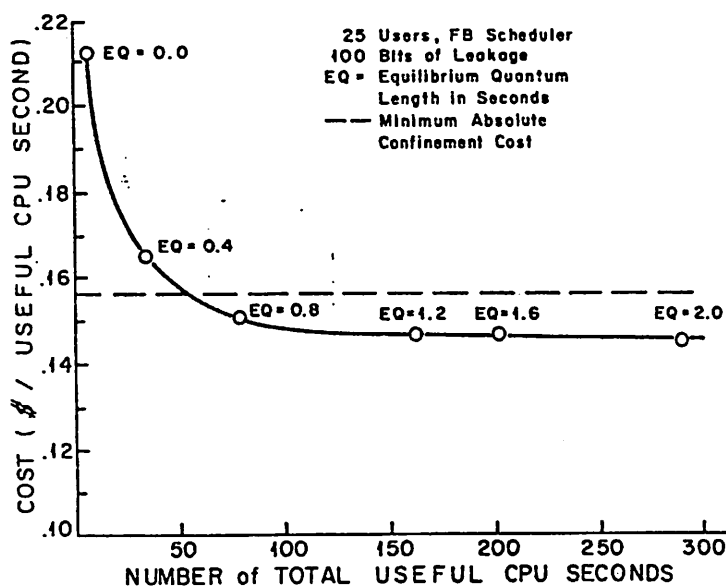


Figure 6.5
Unconstrained Channel Cost vs. Absolute Confinement Cost
25 Users, FB Scheduler

unconfined workloads of 15 and 5 users. Figure 6.5 is the graph of the total cost per useful CPU second for a 25 user workload versus the amount of CPU time obtainable before

100 bits may be leaked. In other words, the confiner estimates his CPU requirement for the terminal session (in useful CPU seconds), finds that number on the horizontal axis, then determines the cost of CPU time from the appropriate curve. The horizontal dotted line is the lowest cost absolute confinement scheduler, which in this case represents allocating the confined subsystem a quantum length of 1.4 seconds every time it is eligible to execute. The solid line is the cost to the confiner of using the unconstrained channel to obtain CPU time, given that at most 100 bits can be leaked, with the equilibrium quantum length as the parameter. The leakage value of 100 bits is an arbitrary choice. Any other value would change the scale of the horizontal axis, but not the shape of the curve. For example, a leakage value of 50 bits would multiply all the horizontal axis marks by one-half to obtain the graph for this case. The relationship between the number of total useful CPU seconds and the channel capacity is an inverse relationship: as the capacity decreases, the number of total useful CPU seconds increases.

The interesting feature of Figure 6.5 is that for a large range of CPU times (and channel capacities), the cost difference between absolute confinement (the most expensive form of scheduling since the needs of the confined subsystem are not considered) and unconstrained scheduling is only about 8%. This result is not entirely unexpected after finding that a very large equilibrium quantum length minimized the cost for the unconstrained channel. For the 25 user case, the user cost is greater than the CPU cost, which allows CPU time to be traded for a faster user response time to produce a lower total cost. Since the majority of the quanta allocated to the confined subsystem are equilibrium quanta, it seems likely that the absolute confinement scheme of allocating a large quantum every time to the confined subsystem would not differ greatly in cost from the unconstrained channel scheme. The conclusion reached from Figure 6.5 is that for the FB scheduler, the overhead of making the channel capacity measurable (i.e. the inclusion of equilibrium quanta) introduces enough overhead that the cost to the confiner of partial confinement and absolute confinement are nearly the same. Thus for the heavily loaded system, partial confinement does not appear to give a satisfactory trade-off of capacity versus

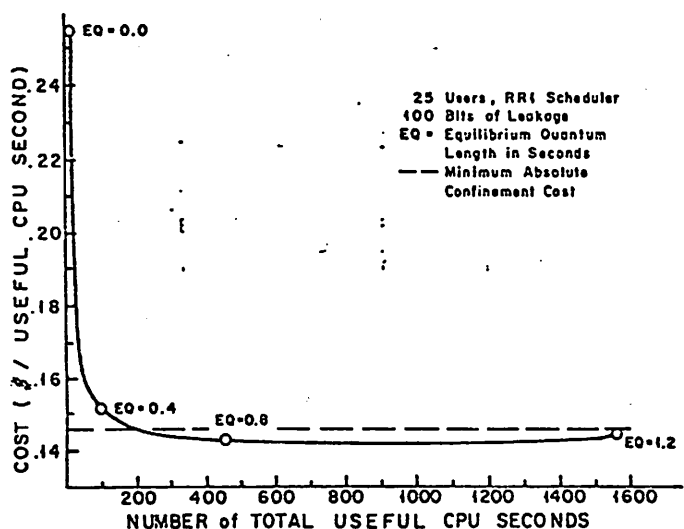


Figure 6.6
Unconstrained Channel Cost vs. Absolute Confinement Cost
25 Users, RR1 Scheduler

confiner cost. As shown in Figures 6.6 and 6.7, this phenomenon is not restricted to the FB scheduler, but is also exhibited by RR1 and RR2.

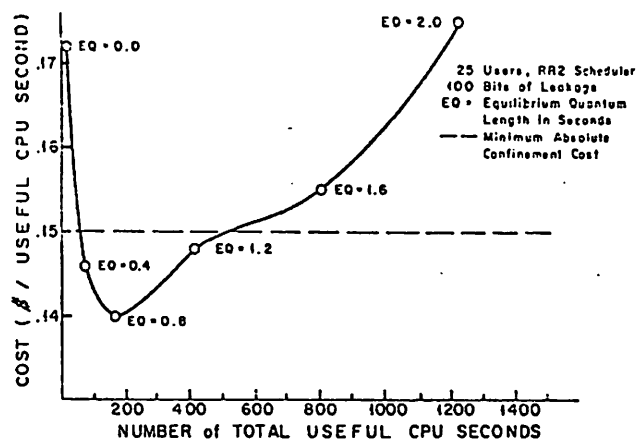


Figure 6.7
Unconstrained Channel Cost vs. Absolute Confinement Cost
25 Users, RR2 Scheduler

The 5 user and 15 user workload cases exhibit an entirely different cost behavior than the 25 user workload case. The confiner cost versus number of useful CPU seconds

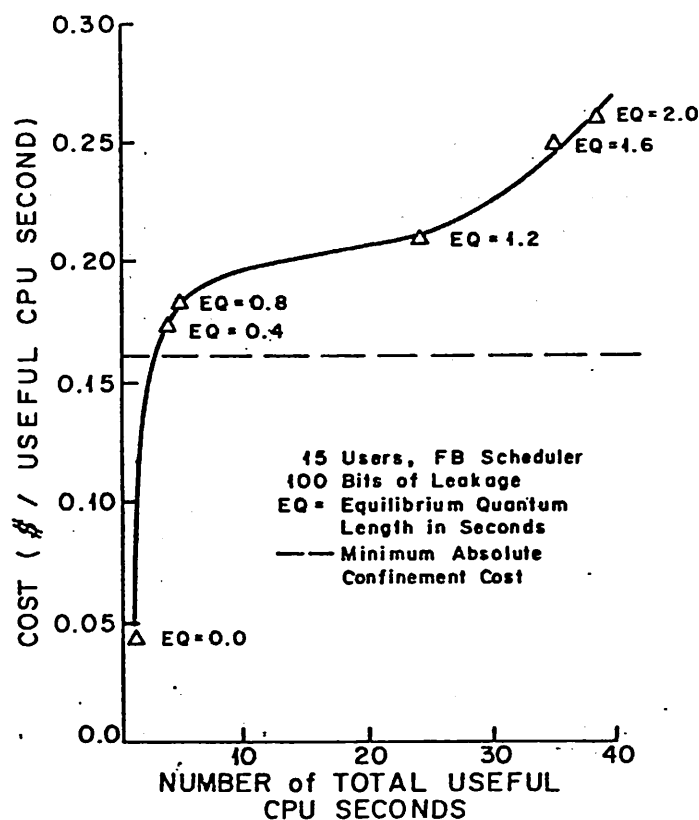


Figure 6.8
Unconstrained Channel Cost vs. Absolute Confinement Cost
15 Users, FB Scheduler

graph for the 15 user FB scheduler is shown in Figure 6.8. The graphs for the other schedulers with a 15 user workload and all schedulers with a 5 user workload have the same characteristics as Figure 6.8 and are not shown. The dotted horizontal line in Figure 6.8 is the minimum absolute confinement cost and the solid line is the unconstrained channel cost for different values of the equilibrium quantum length. Unlike the 25 user workload case, the confiner cost of the unconstrained channel for an equilibrium quantum

length of zero seconds is 27% of the cost of absolute confinement. As the equilibrium quantum length increases, the cost eventually exceeds the absolute confinement cost. The fact that the cost of partial confinement can be significantly less than absolute confinement for the 15 user and 5 user workloads shows that a trade-off of cost versus capacity can be made in these cases. Since the unconstrained channel cost curve rises abruptly as the equilibrium quantum length increases, using the equilibrium quantum length as a "knob" for adjusting the capacity/cost trade-off does not produce a large range of CPU times for which partial confinement is cheaper than absolute confinement. In subsequent sections, other knobs are explored that provide better control over the cost and capacity.

There is still the question of how much absolute confinement costs compared to no confinement. The answer

Number of Users, Scheduler	Unconfined Cost (=A) (\$/Useful CPU Second)	Confiner Cost (=B) (\$/Useful CPU Second)	Cost Factor (B/A)
25,FB	0.0568	0.156	2.75
25,RR2	0.0626	0.163	2.60
25,RR1	0.0705	0.146	2.07
15,FB	0.0302	0.161	5.33
15,FCFS	0.0330	0.159	4.82
15,RR1	0.0335	0.169	5.04
5,FB	0.0235	0.138	5.87

Table 6.9

Cost of Absolute Confinement Versus No Confinement

is contained in Table 6.9 which gives the cost factor for absolute confinement (= cost of absolute confinement for the confiner divided by the cost of no confinement for each unconfined process). The cost factor for absolute confinement for 25 users and the FB scheduler (= 2.75) is nearly the same as for the unconstrained channel model in Table 6.7 (= 2.71). The major difference is in the 15 and 5 user workloads. The cost of absolute confinement is 4 to 6 times larger than the unconfined cost for these workloads. The corresponding cost of partial confinement is 1 to 1.5 times the unconfined cost. As mentioned before, absolute confinement is very expensive compared to partial confinement for the cost function defined in section 6.3 for the 5 user and 15 user workloads.

6.9.3 Using System Cost for the Comparison

Although the cost function defined in section 6.3 will be used throughout this thesis, an alternative would be to include the cost for all users in the cost function. Such a cost function could be called the system cost function. This type of cost function would prevent the scheduler from being optimized for the confiner at the expense of the users executing unconfined jobs. This type of cost function might be used by the system administrator to keep the average level of satisfaction with the system response high.

One natural system cost function would be a user-weighted cost function in which the cost to all users of the system is averaged. For example, if 25 users are executing unconfined programs and one user is executing a confined program, the cost of execution (including both CPU time and user time) for one unconfined user would be multiplied by 25 and the product would be added to the cost for the one confined user. The function used to determine the cost for both the confined and unconfined users is the one defined in section 6.3. The interesting question involving the system cost function is whether the cost relationship between abso-

lute confinement and partial confinement as implemented by the unconstrained channel changes if the system cost function is used rather than the confiner cost alone. The analysis results could change if the scheduling parameters chosen to minimize system cost are different than the parameters chosen to minimize confiner cost.

The most important parameter chosen is the value of the equilibrium quantum length since this parameter is the primary factor in determining cost. The data for the equilibrium quantum selection for the 5 and 15 user workloads is given in Table 6.7. For these two cases, the confiner cost and the unconfined user cost are minimized at the same equilibrium quantum length (= 0.0 seconds). Thus the equilibrium quantum length used to minimize system cost is the same as the equilibrium quantum length that minimizes confiner cost. To compare the difference in system cost between absolute confinement and the unconstrained channel (partial confinement), the data in Table 6.9 and Table 6.7 are used. The system cost for the 15 user FB scheduler, which is typical of the other schedulers, for absolute confinement is 0.614 \$/useful CPU second (=0.161 for confiner cost + (0.0302×15) for user cost) and for the unconstrained channel is 0.499 (=0.044 for confiner cost + (0.0303×15) for user cost). The difference between the user costs for these two cases is small but the difference between the confiner costs is large. The small difference in cost for the unconfined jobs is typical of all the measurements (about 3%), so the scheduling parameters for confinement do not significantly affect the unconfined user cost. Since the type of confinement does not materially affect the cost to the unconfined users, the confined scheduling parameters should be chosen to minimize the confiner cost as has been done in the subsequent analysis. This conclusion also holds for the 5 user case.

For the 25 user case, the equilibrium quantum length chosen will depend on the cost function used. As shown in

Equilibrium Quantum Length (Seconds)	Absolute Confinement System Cost (\$/Useful CPU Second)	Unconstrained Channel System Cost (\$/Useful CPU Second)
0.0	X	1.446
0.4	1.478	1.405
0.8	1.488	1.479
1.2	1.571	1.505
1.6	1.646	1.550
2.0	1.657	1.580

Table 6.10

System Cost vs. Confiner Cost
for the 25 User, FB Scheduler

Table 6.10, the equilibrium quantum length that minimizes system cost is 0.4 seconds. This is different than the equilibrium quantum length of 1.4 seconds that minimizes system cost. This difference is due to the large effect of absolute confinement on the cost to the unconfined users. For the equilibrium quantum length of 0.4 seconds, the difference in system cost between the unconstrained channel and absolute confinement is on the order of 5%. This result is consistent with the difference of 8% found for the equilibrium quantum length that minimizes confiner cost. Since the difference between the absolute confinement cost and partial confinement cost is small with both cost functions, absolute confinement would probably be used instead of partial confinement as noted before.

In summary, the use of system cost in choosing the

scheduling parameters should not alter the conclusions of this thesis. For the 5 and 15 user cases, the equilibrium quantum length parameter chosen is the same for either cost function and for the 25 user case, the equilibrium quantum chosen according to either cost function results in partial confinement (with the unconstrained channel) not being significantly cheaper than absolute confinement. A cost function that charges the confiner a certain rate according to the workload (i.e. charge less per CPU second when there is an excess of CPU time available as in the 5 user case) might change the conclusions. The feasibility and usage of such a cost function is an open area of research.

6.9.4 The Effect of Discretization on the Absolute Confinement Cost

The cost of absolute confinement could be measured without any system discretization since discretization is not needed to compute the channel capacity. Without discretization, the absolute confinement cost might be lower than that measured. However, by using the appropriate design strategy, the cost of partial confinement could also be lower than that given here. If a system is to be designed with a partial confinement feature, the correct procedure would be to first model the system with a discretization that is large enough to permit the set of computations of the channel capacity at a reasonable cost. This is the method for choosing the 0.2 second discretization interval used in this thesis. Once the correct neighborhood is found for the scheduling parameters, a once-only computation could then be made of the channel capacity for a small enough discretization interval that the system overhead due to discretization would be small.

To obtain a better idea of the magnitude of the absolute confinement cost difference with no discretization, the 25 user and 5 user FB scheduler was simulated with a discretization interval of 0.05 seconds. At this level of discretization, the percentage of wasted CPU time for unconfined users is 2 to 3% for a 25 user workload. The cost increase of the 0.2 second discretization over the 0.05 second discretization for unconfined users is 1.16 for 25 users and 1.05 for 5 users. The confiner cost increase is a factor of 1.05 for 25 users and is 1.01 for 5 users. These figures suggest that the unconfined user cost will drop more rapidly than the confiner cost. Thus as the discretization goes to zero, the cost factor increases for executing absolutely confined as opposed to unconfined. The reason that the unconfined cost decreases more rapidly than the absolute confinement cost is that by reducing the discretization interval, the response time cost and the wasted CPU time decrease for the unconfined case. For absolute confinement, the response time cost will decrease but the wasted CPU time component will not decrease. This is due to the absolute confinement scheduling algorithm being unable to allocate a quantum length based on the needs of the confined subsystem. Thus the increasing cost factor for absolute confinement as the discretization interval decreases should be expected. These measurements suggest that the cost factor of executing absolutely confined versus unconfined will increase slightly as the discretization decreases. Also the variance in cost of absolute confinement as the discretization interval decreases is small.

6.9.5 Summary

This section has presented a major result of this thesis: partial confinement yields only a 8% reduction in the confiner cost for heavy workloads but yields reductions of 80% to 85% for the lighter workloads over absolute confinement. For the heavy workload case, the cost of making the capacity measurable causes the very small cost advantage of partial confinement. For most applications, the advantage is small

enough that absolute confinement would probably be preferred over partial confinement. For the medium and light workload cases, however, the cost advantage of partial confinement is significant and could provide an alternative to absolute confinement. The last part of this section demonstrates that absolute confinement is very costly compared to unconfined execution.

6.10 Partial Confinement Using DMC Channel Models

The previous section demonstrated that partial confinement can be less costly than absolute confinement for medium and light workloads. The penalty for using partial confinement is that the amount of leakage is not zero. The non-zero leakage and lower cost result from the operating system choosing which quantum length to allocate to the confined subsystem based on the state of the confined subsystem. Those scheduling strategies in which the quantum choice is heavily dependent on the confined subsystem state usually leak more information and cost less than strategies that are not as heavily dependent. A desirable feature of a confinement mechanism is to provide a "knob" that regulates the amount of dependence by adjusting a small number of parameters. This feature is included in the three DMC channel models investigated in this section.

The objective of this section is to show that partial confinement can provide a choice of channel capacities that vary more than an order of magnitude for less cost than absolute confinement. This allows the confiner to have a wide selection of cost/capacity choices for an application in which a non-zero amount of leakage can be tolerated. The channel models that are implemented by the scheduler to provide partial confinement are based on the cascaded, the compound and the cascaded compound channel models presented in Chapter 2. These models are not the only ones that can be used. Any channel model that can be implemented in a computer scheduler is a possible candidate. But these models do provide a wide range of cost/capacity choices.

The next section explains the constrained channel model that will be used throughout the rest of this chapter. The constrained model is shown to have the same characteristics as the unconstrained model analyzed earlier but with fewer inputs to make the analysis easier. Constraining the channel is necessary for a comparison of the DMC and the DCWM channel models since the DCWM model can have at most three inputs for computational reasons.

6.10.1 The Constrained Channel Assumption

The constrained channel is a channel in which only a subset of the possible quantum lengths can actually be allocated to the confined subsystem. For example, in RR2 with a discretization interval of 0.2 seconds, the confined subsystem can choose a quantum length of 0.0, 0.2, ..., 2.0 seconds as in the unconstrained channel. In the constrained channel model, only three of these possible quantum lengths can be allocated. Decreasing the number of inputs for the confined subsystem can only decrease the channel capacity as explained by the Data Processing Theorem. For example, in the channel that transmits data without any decoding error, decreasing the number of channel inputs from 11 to 3 decreases the channel capacity from $\log_2 11$ (≈ 3.46) bits per channel use to $\log_2 3$ (≈ 1.58) bits per channel use. The reason for the decrease is that the data must be encoded into a smaller number of symbols, which decreases the data rate of the channel.

The choice of the quantum lengths that can be allocated to the confined subsystem for the constrained channel model must be a function of the job characteristics. Since our view of confinement is that the entire terminal session must be

confined, the characteristics to use are the ones for the entire workload. In the CTSS workload, most of the jobs require a very small amount of CPU time (less than 0.05 seconds) for each interaction. A typical job with this characteristic is an interactive editor that waits for an editing command, performs the editing function specified, then waits for the next editing command. If swap time is included with the small CPU time requirement, most of these interactions complete in less than 0.4 seconds. The remainder of the workload represents interactions that require more than a trivial amount of CPU time, such as compilations. To handle these requests efficiently, a quantum length longer than 0.4 seconds is required. After experimentation with several values of a longer quantum length, the value of 1.0 seconds was chosen to handle interactions in this class, although other quantum lengths in the neighborhood of 1.0 seconds (e.g. 0.8 and 1.2 seconds) yielded similar results. The most frequent type of quantum request occurs when the CPU is to be allocated to the confined subsystem, but the CPU time can not be used (i.e. the confiner is thinking). This requires that a zero length quantum also be a channel input. This case occurs frequently in the light and medium workload cases in which there is excess computer capacity. Based on the above considerations, a three input constrained channel allows the confined subsystem to have possible channel inputs of 0.0, 0.4 and 1.0 second quantum lengths.

The comparison of channel capacity and confiner cost for the unconstrained and constrained channels with a zero

Number of Users, Scheduler	Unconstrained Channel Capacity (Bits/Useful CPU Second)	Constrained Channel Capacity (Bits/Useful CPU Second)	Unconstrained Channel Cost (\$/Useful CPU Second)	Constrained Channel Cost (\$/Useful CPU Second)
25,FB	0.62	1.23	0.147	0.162
15,FB	75.69	51.12	0.044	0.047
15,FCFS	Infinity	49.19	X	0.052
15,RR1	35.31	30.53	0.056	0.051
5,FB	260.7	121.6	0.026	0.029

The equilibrium quantum length is zero seconds except for the 25 user FB scheduler. In this case the unconstrained channel equilibrium quantum length is 1.2 seconds and the constrained channel equilibrium quantum length is 1.0 seconds.

Table 6.11
Unconstrained Channel vs. Constrained Channel

length equilibrium quantum is given in Table 6.11. The largest decrease in channel capacity, of course, occurs for the FCFS scheduler in which the capacity is finite for the constrained channel and infinite for the unconstrained channel. Excluding the FCFS algorithm, the average percentage capacity decrease for all the algorithms is about 32%. The confiner cost for the constrained channel is within 10% of the unconstrained channel for all schedulers except FCFS. The cost for the FCFS scheduler is undefined for the unconstrained channel since the number of equilibrium quantum allocations required to return the system to equilibrium can not be determined. This is a result of there being no largest perturbation that can be caused by the confined subsystem. The unconstrained channel does not necessarily have a lower cost than the constrained channel since the number of equilibrium quanta allocated between two variable length quanta may be greater for the unconstrained channel model, which increases the cost of the unconstrained channel. This effect occurs for the RR1 scheduler. The difference in cost is on the order of 10%. Also the capacity of the 25 user FB unconstrained channel capacity is less than the constrained channel due to the smaller number of equilibrium quanta allocated per useful CPU second for the constrained channel.

Since the cost of the constrained channel is not much

different from the unconstrained channel, the channel inputs that are omitted in the constrained channel do not appear to have a large effect on the cost of confinement. Since the cost for the constrained channel is nearly the same as the unconstrained channel but with a smaller channel capacity, the constrained channel will be used as the base channel for the subsequent channel models in this chapter.

In subsequent sections, the only heavy workload and light workload scheduler to be analyzed is the FB scheduler. For a heavy workload, all the schedulers exhibit similar characteristics and yield capacity and cost values in the same neighborhood. Since the FB type of algorithm is frequently used in practice, it is the one chosen to represent the heavy workload case. For the light workload, the excess CPU time available makes the characteristics of all schedulers similar. Again FB is chosen as a typical example of this workload.

6.10.2 Constrained Cascaded Channel Model

One method for decreasing the channel capacity is to not always allocate the confined subsystem the quantum length it needs every time. The effect of this strategy is to make the decoding task of the spy much more difficult. This problem can be overcome by the confined subsystem and spy doing more sophisticated encoding and decoding, but at a reduced rate of data transmission. One implementation of this chan-

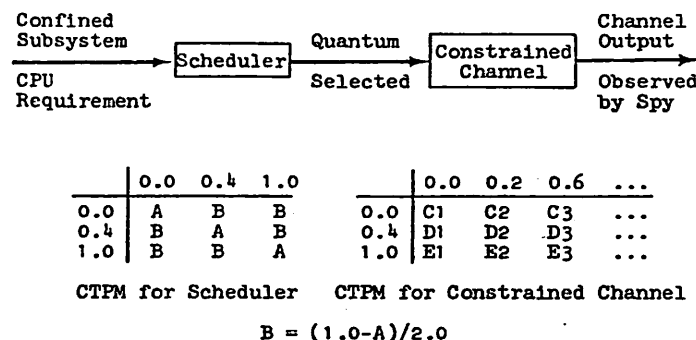


Figure 6.9
Constrained Cascaded Channel Model

nel is shown in Figure 6.9. Here, the confined subsystem makes its CPU requirements known to the scheduler which allocates a quantum length according to the scheduler's CTPM. That is, with probability A, the desired quantum length is allocated, and with probability $(1.0 - A) / 2.0$ each of the unwanted quantum lengths may be allocated. The allocated quantum is then used as the input to the constrained channel analyzed in the last section which represents the other users of the system and which has as outputs the inter-quantum times seen by the spy.

The entire covert channel (scheduler channel + constrained channel) can be analyzed with the cascaded channel model discussed in Chapter 2. The capacity of the entire channel can be regulated to range from zero (for $A = 1/3$) to the constrained channel capacity (for $A = 1.0$). The scheduler channel capacity is zero exactly when $A = 1/3$ because all the rows of the scheduler CTPM are equal. By the Data Processing Theorem, the capacity of the entire cascaded channel is then shown to be zero. Of course, when $A = 1.0$, the scheduler always gives the confined subsystem the quantum it wants so the cascaded channel model has the same capacity as the constrained channel alone. The ability to continuously adjust the channel capacity by varying one parameter is desirable in a partially confined environment. If this feature is not

provided, an excessively restrictive (lower capacity but higher cost) channel model may have to be used to insure the information leakage specifications of the confiner are met.

The result of using the cascaded channel model for the heavily loaded system with the FB scheduler is shown in Fig-

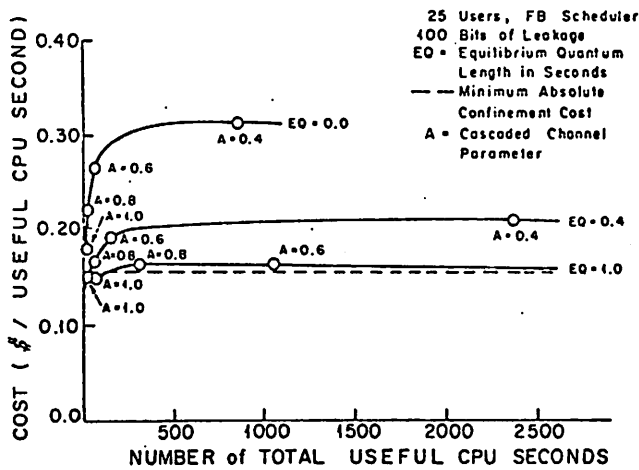


Figure 6.10
Constrained Cascaded Channel Cost vs. Absolute
Confinement Cost 25 Users, FB Scheduler

ure 6.10. As the value of A decreases from 1.0, the confiner cost must increase since the quanta allocated depends less and less on the needs of the confined subsystem. Since the cost of the constrained channel alone (A=1.0) is within 5% of the absolute confinement cost, the cascaded model does not provide much incentive to use partial confinement. As shown in Figure 6.10, for an equilibrium quantum value of 1.0, which yields the least cost for the constrained channel, the cost quickly exceeds the absolute confinement cost. The increase in cost is due to both a longer average response time

Equilibrium Quantum (Seconds)	Channel Parameter (A)	CPU Cost (\$/Useful CPU Second)	User Cost (\$/Useful CPU Second)
0.0	1.0	0.0175	0.163
0.0	0.8	0.0307	0.193
0.0	0.6	0.0407	0.224
0.0	0.4	0.0593	0.256
0.4	1.0	0.0636	0.0888
0.4	0.8	0.0580	0.110
0.4	0.6	0.0738	0.115
0.4	0.4	0.0796	0.132
1.0	1.0	0.0711	0.0776
1.0	0.8	0.0839	0.0809
1.0	0.6	0.0798	0.0876
1.0	0.4	0.0757	0.0947

Table 6.12

Cost Components for 25 Users, FB Scheduler for the Constrained Cascaded Channel Model

and more wasted CPU time as shown in Table 6.12.

The medium workload case is shown in Figure 6.11 for 15 users and the FB scheduler. The graphs for the RR1 and FCFS algorithms for this workload exhibit similar behavior and are not shown. The cascaded channel gives a lower cost for partial confinement for applications that can tolerate a leakage of more than 3.92 bits/useful CPU second with a zero second equilibrium quantum. If 100 bits can be leaked in the total execution of the confined subsystem, then 25.5 seconds of useful CPU time can be obtained. If a non-zero equilibrium quantum length is used, the confiner cost increases significantly as shown in Figure 6.11 for an equilibri-

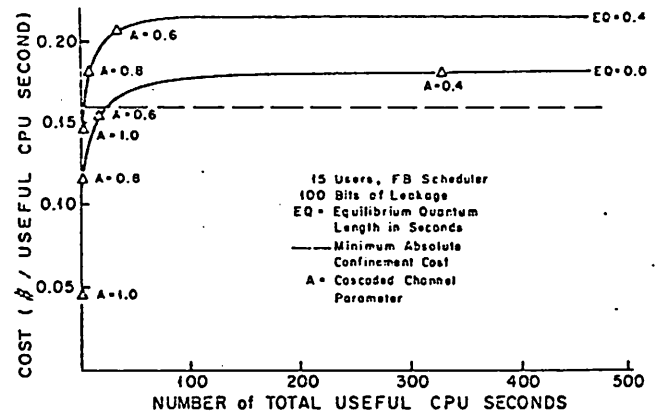


Figure 6.11
Constrained Cascaded Channel Cost vs. Absolute
Confinement Cost 15 Users, FB Scheduler

um quantum length of 0.4 seconds. The lower channel capacity induced by the non-zero equilibrium quantum is more than offset by the extra amount of CPU time wasted. This makes the zero length equilibrium quantum the reasonable choice for partial confinement.

In the light workload case, partial confinement also costs less than absolute confinement for the cascaded channel model. This cost advantage only occurs for large amounts of

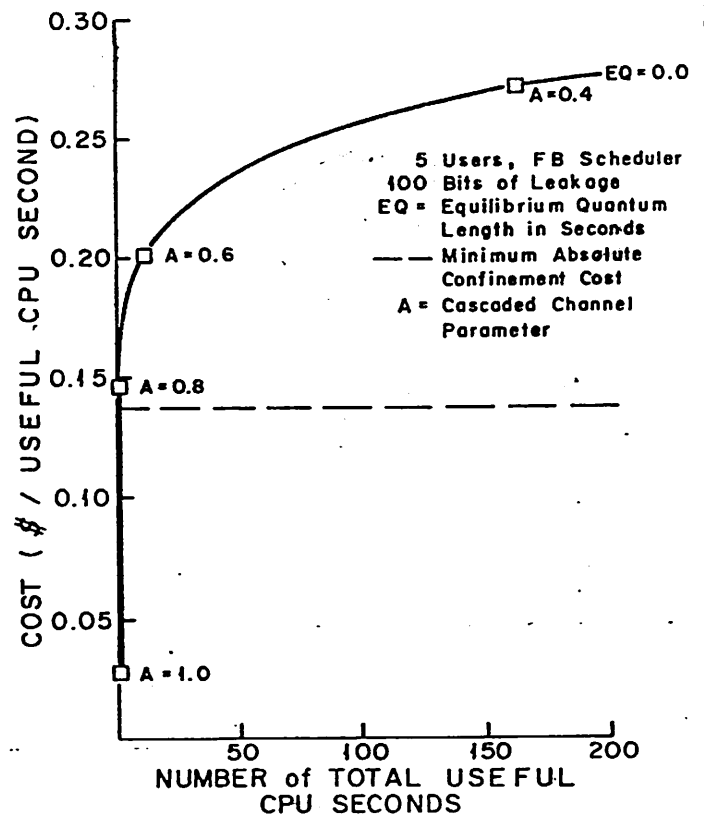


Figure 6.12
Constrained Cascaded Channel Cost vs. Absolute
Confinement Cost 5 Users, FB Scheduler

leakage as shown in Figure 6.12. The confiner must be able to tolerate leakage at a rate greater than 40 bits/useful CPU second to have this channel model cost less than absolute confinement. This translates into only 2.5 useful CPU seconds if a total of 100 bits may be leaked. This exception-

ally high rate of leakage would most likely make absolute confinement the most frequently used alternative. Another alternative is for the system to simulate the effect of additional users to decrease the channel capacity but at a large increase in the system overhead. This strategy is discussed in a later section after all the channel models have been introduced.

The constrained cascaded model by itself does not decrease the channel capacity enough to make partial confinement very cost effective over absolute confinement. For the heavily loaded system, there is almost no cost advantage for partial confinement and in the lightly loaded system, the leakage rate is much too large. For the medium workload, the minimum leakage to make partial confinement less costly is still rather large at 3.92 bits/useful CPU second.

6.10.3 Constrained Compound Channel Model

Part of the problem with the cascaded channel model is that the zero second quantum length does not have a large enough probability to decrease the cost for the medium and light workload cases. The typical interaction is characterized by a long user think time followed by a short CPU time requirement. Thus in the medium and lightly loaded system, the most frequently allocated quantum length is zero seconds. To decrease the channel capacity and to weight the zero second quantum length more heavily, the compound channel model explained in Chapter 2 is used. As for the cascaded channel model, the compound channel model is also constrained since the confined subsystem can only be allocated a quantum length of 0.0, 0.4, or 1.0 seconds. The strategy in using the compound channel model is to decrease the capacity by allowing the confined subsystem to choose the quantum length to be allocated only at random times. When the confined subsystem is prevented from choosing the quantum length, a quantum length of zero seconds is allocated. The constrained compound channel model is shown in Figure

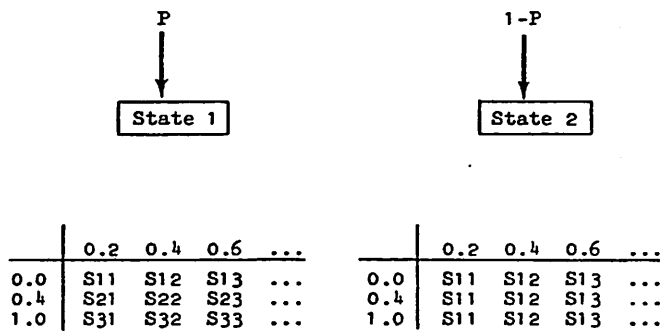


Figure 6.13
Constrained Compound Channel Model

6.13. State 1 represents a variable quantum being allocated to the confined subsystem (with probability P) and state 2 represents a 0.0 second quantum being allocated with probability $1.0 - P$. In state 2, the 0.0 second quantum is allocated without regard to the actual CPU needs of the confined subsystem. The capacity of the CTPM for state 2 is, of course, zero. Since this is a DMC model, after the appropriate quantum has been allocated based on the selected state, a sufficient number of equilibrium quanta are allocated to restore the system to its equilibrium state given the worst perturbation by the confined subsystem before the next model state is selected.

The behavior of the 25 user workload case for this chan-

nel model is similar to the constrained cascaded model. In both cases, the cost difference between absolute confinement and the minimum cost for this model ($P=1.0$ and the equilibrium quantum length = 1.0 seconds) is about 5%. Since the cost can only increase as the compound channel parameter, P, decreases from 1.0, the usefulness of this model for partial confinement is very limited. The cost versus capacity graph

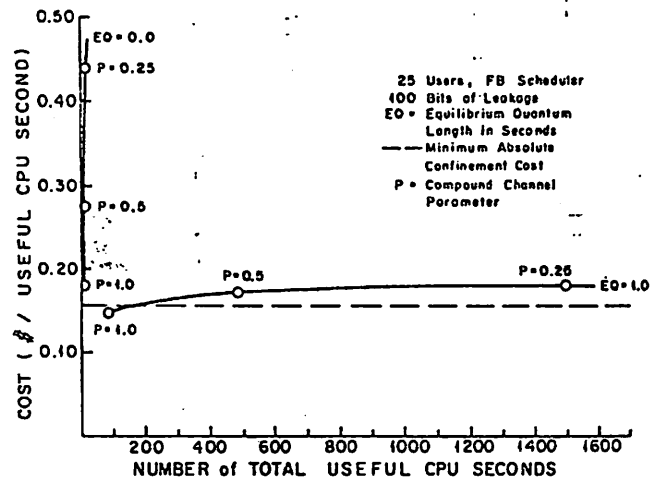


Figure 6.14
Constrained Compound Channel Cost vs. Absolute Confinement Cost 25 Users, FB Scheduler

is shown in Figure 6.14.

The point at which partial confinement becomes less costly than absolute confinement for the medium workload case is nearly the same as for the constrained cascaded chan-

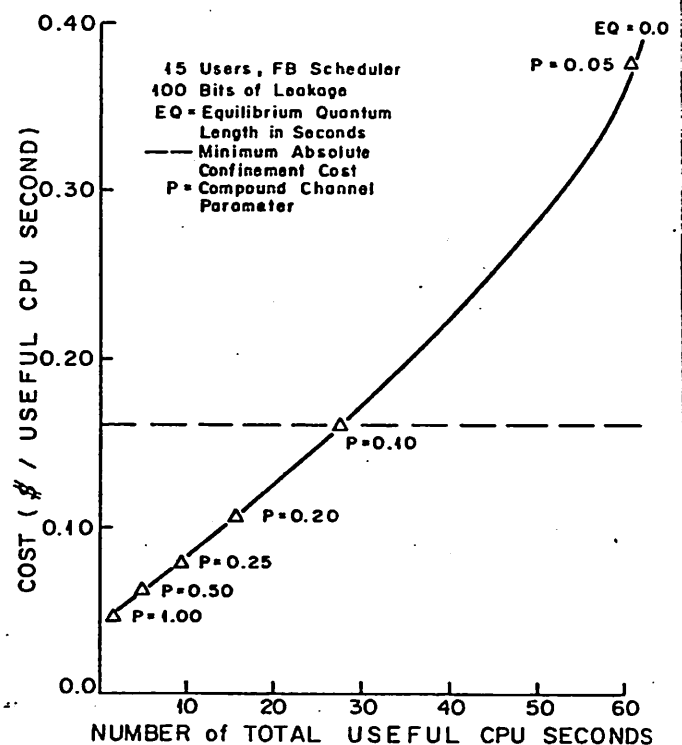


Figure 6.15
Constrained Compound Channel Cost vs. Absolute Confinement Cost 15 Users, FB Scheduler

nel. This is shown in Figure 6.15. The rate for the compound channel is about 3.59 bits/useful CPU second and for the cascaded channel is about 3.92 bits/useful CPU second.

If 100 bits can be leaked by the confined subsystem, then the compound channel model permits 27.9 useful CPU seconds of computation.

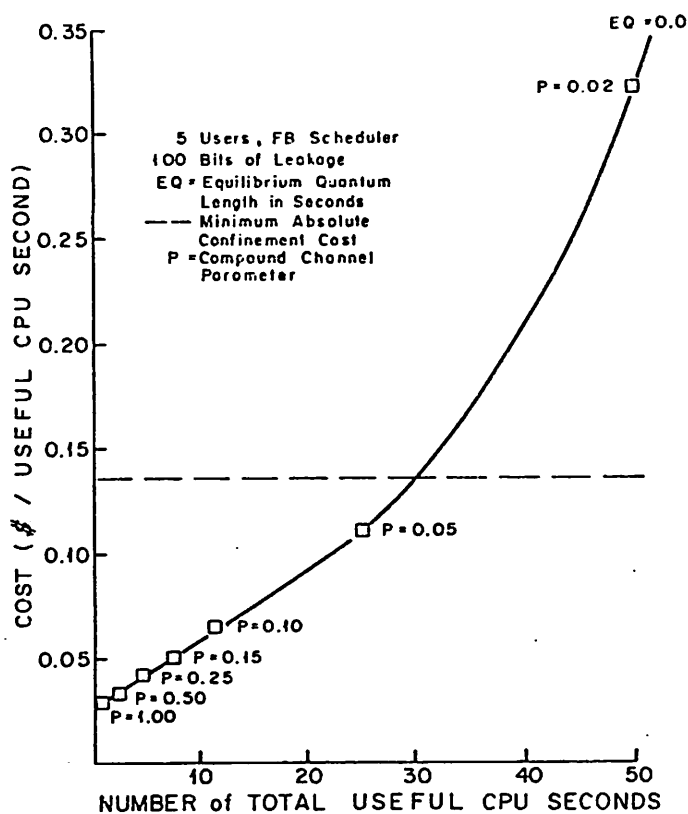


Figure 6.16
Constrained Compound Channel Cost vs. Absolute Confinement Cost 5 Users, FB Scheduler

The light workload case in Figure 6.16 for the constrained compound channel model shows a marked difference from the cascaded model. The minimum leakage at which partial confinement is less costly for the compound channel is 3.3 bits/useful CPU second as opposed to 40 bits/useful CPU second for the cascaded model. The difference in performance is due to the large probability of allocating a zero second quantum length to the confined subsystem. A comparison of the minimum leakage for cost-effective partial confinement for both the 15 and 5 user workloads for the compound channel model shows that the workload difference only affects the CPU time cost and not the amount of CPU time that can be obtained with partial confinement. The 5 user workload even costs less than the 15 user case for comparable amounts of useful CPU time.

The compound channel model is much better suited to light workloads than the cascaded channel since it is able to weight the zero second quantum length heavily. For the medium and heavy workloads, the compound channel performs about the same as the cascaded channel. The next step is to obtain a channel that combines the different characteristics of each of the two channel models in order to give a larger range of capacities over which partial confinement is cheaper than absolute confinement.

6.10.4 Constrained Cascaded Compound Channel Model

The constrained cascaded compound channel model is a synthesis of the constrained cascaded channel and the constrained compound channel analyzed in the last two sections. Figure 6.17 shows graphically why a combined channel is better for the 15 user FB scheduler. For relatively large channel capacities, the compound-channel is better than the

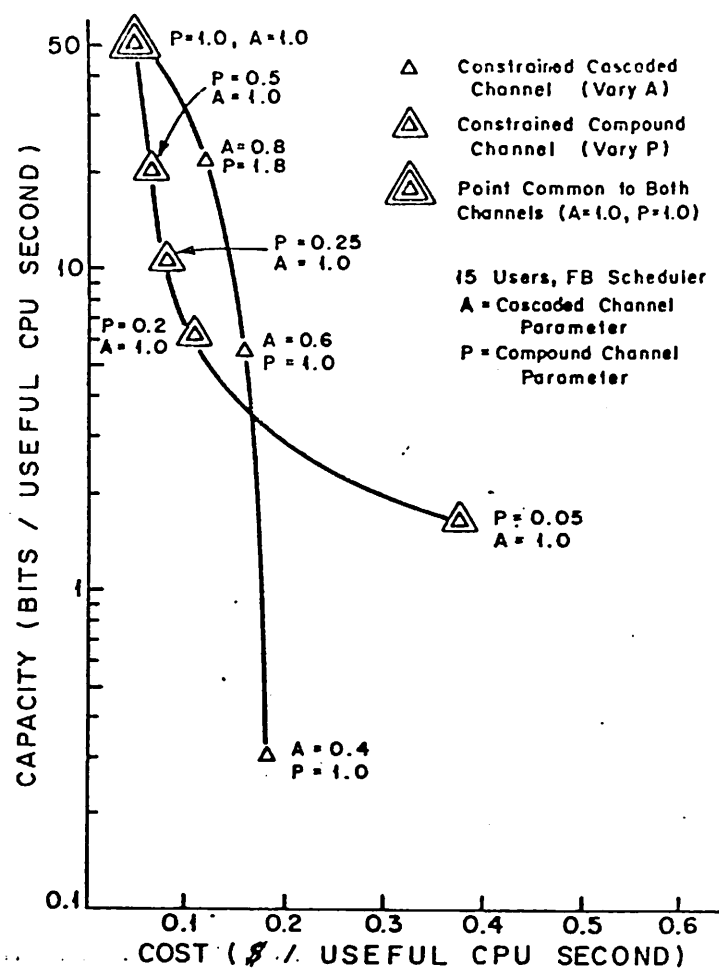


Figure 6.17
Constrained Cascaded Compound Channel Models, 15 Users, FB Scheduler

cascaded channel since decreasing the compound channel parameter, P , from 1.0 causes the capacity to decrease sharply. But as P approaches 0, the slope of the curve steadily decreases. At some point the cascaded channel becomes more cost effective than the compound channel and continues to be more cost effective as the capacity decreases to zero.

The synthesized channel model is formed by making the equivalent CTPM for the cascaded channel of Figure 6.9 (which was derived in Chapter 2) the CTPM for state 1 in Figure 6.13. This means that there are two parameters that can be used to change the characteristics of the channel: the parameter for the cascaded channel (A) and a parameter for the compound channel (P). An example of the interaction of these two parameters is shown in Figure 6.18 for the heavy workload FB scheduler. In this graph of capacity versus cost, concentric curves that are closer to the origin represent parameter choices that are superior to those with curves farther from the origin. The reason is that for any channel capacity, the cost to the confiner is less. For the heavy workload FB case, the curves representing choices of the parameter P that are less than 1.0 are farther from the origin than the $P=1.0$ curve. Thus any partial confinement channel model for the heavy workload case should fix P to a value of 1.0 and just vary A to obtain the desired capacity. As stated before, since the cost of absolute confinement is within 5-8% of all points on the $P=1.0$ curve, absolute confinement is probably a more attractive alternative than partial confinement.

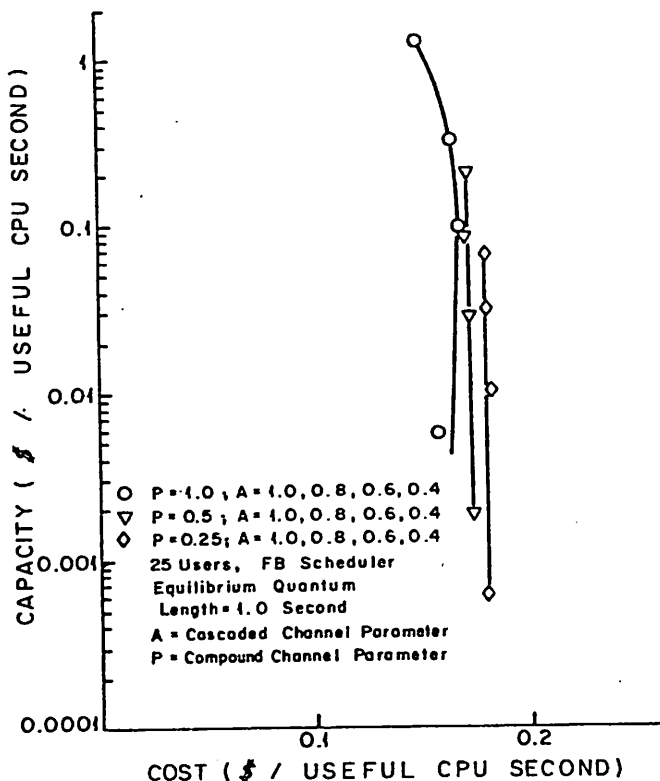


Figure 6.18
Constrained Cascaded Compound Channel
25 Users, FB Scheduler

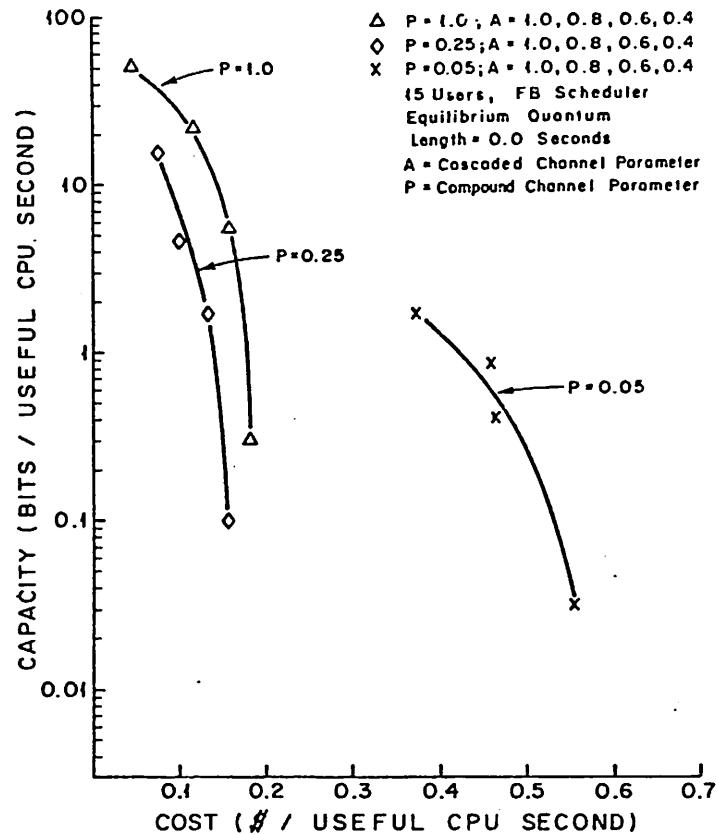


Figure 6.19
Constrained Cascaded Compound Channel
15 Users, FB Scheduler

The channel model results for the medium workload case in Figure 6.19 are much different than the heavy workload case for the FB scheduler. This figure is typical of the FCFS and RR1 algorithms for a medium workload and of the FB algorithm under light load. Figure 6.19 shows that as the value of P decreases from 1.0 to 0.25, the curves formed by varying A for constant values of P are more cost effective for values of P closer to 0.25. Curves having values of P greater than 0.25 shift away from the origin and thus become less cost effective. For values of P less than 0.25, the increased user response time cost more than outweighs the decrease in capacity. An example is the value $P=0.05$ in Figure 6.19. Thus there exists a value of P at which the curve formed by varying the value of A is the best for this channel model. The method for finding the correct value of P in Figure 6.19 is by experimentation. The curves for the medium workload FCFS and RR1 schedulers, and for the light workload FB scheduler are similar to Figure 6.19 and are omitted. The value of P that produces the minimum capacity/cost curve for 15 users with the FCFS algorithm is 0.5, for 15 users with the RR1 algorithm is 0.7, and for 5 users with the FB algorithm is 0.1.

As a comparison of the various scheduling algorithms, Figure 6.20 gives the capacity versus cost points for the 5 user FB scheduler, the 15 user FB, FCFS, and RR1 schedulers, and the 25 user FB scheduler. The points are for the value of P giving the minimum cost curve and for $A=1.0, 0.8, 0.6$ and 0.4 . The curves in Figure 6.20 are ordered according to workload. The most cost effective curve represents the light workload, the next cost effective curve is for the medium workload, and the least cost effective curve is for the heavy workload. This is surprising since it might seem better to run a confined subsystem when there are many other users on the system to generate noise.

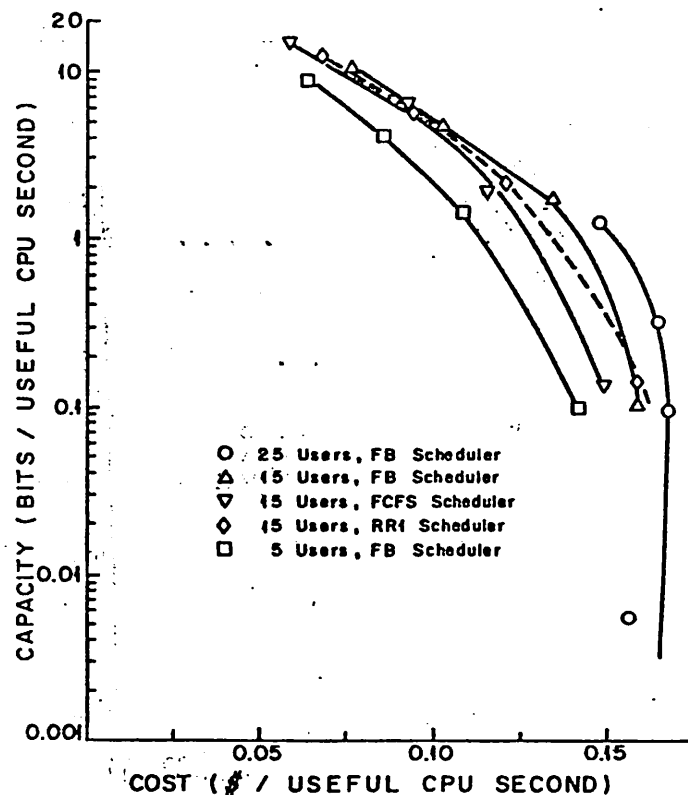


Figure 6.20
All Work Loads and Schedulers
Constrained Cascaded Compound Channel

The constrained cascaded compound channel allows the confiner to choose a range covering nearly two orders of

magnitude of channel capacity in bits/useful CPU second. The low end of the capacity range includes capacities on the order of 0.1 bits/useful CPU second, which is much lower than for the previous two channel models, and at a cost that is less than absolute confinement. The results of this section can be used to make a comprehensive confinement strategy to cover all workloads. This is done in the next section.

6.10.5 The Scheduling Strategy for Partial Confinement

Since the objective of the confiner is to execute the confined subsystem at the lowest cost, the curves in Figures 6.18 to 6.19 suggest how this can be done for a given workload and scheduling algorithm for the constrained cascaded compound channel. The minimum cost curve is found by holding A at 1.0 and decreasing P from 1.0 until the point on this curve closest to the origin is found. Assume that the value of P at this point is P' . The minimum cost curve then follows the capacity versus cost curve for $P=P'$ and decreasing values of A . When (or if) this curve meets the absolute confinement curve, the capacity versus cost curve becomes vertical, denoting that any desired capacity requirement can be satisfied at the absolute confinement cost. The minimum cost curve can be thought of as the envelope of all the capacity versus cost curves investigated in previous sections. For example, take the cost versus capacity curves for the heavy workload case with the FB scheduler that are shown in Figure 6.18. In this case, decreasing the value of P from 1.0 only shifts the capacity versus cost curve away from the origin (i.e. see Figure 6.18) so P' is 1.0. The value of A is decreased until the cost equals the absolute confinement cost, at which point the minimum cost curve becomes vertical. There is not a wide range of capacity requirements that can be filled by the FB scheduler under a heavy workload as pointed out previously. For the medium and light workload cases, P' does not equal 1.0 as can be seen in Figure 6.19. The result of plotting these minimum cost curves for the workloads and schedulers under investigation is shown in Figure 6.21.

There are four interesting features of Figure 6.21. The first item is that the range of capacities available that cost less than absolute confinement is over two orders of magnitude and that capacities as low as 0.1 bits/useful CPU second are cheaper than absolute confinement. This makes partial confinement attractive even when fairly low data rates are required. The second item is that partial confinement can cost anywhere from 20% of the absolute confinement cost to 100% depending on the permissible bit rate. This means that significant cost savings can be realized by using partial confinement. The third item is that there is no algorithm in the medium workload case that does best with respect to capacity and cost. All three scheduling algorithms display the same type of behavior and with values of capacity and cost in the same neighborhood. The fourth item is that for partial confinement, the algorithms appear to be ordered by workload with respect to their cost effectiveness. The light workload case is most cost effective followed by the medium workload then by the heavy workload. This should be considered in choosing when a confined subsystem is to be executed. One possible implementation of partial confinement for the light to medium workload cases would be for the system to add pseudo-users until the channel noise generated by the pseudo-users and the real users is enough to lower the channel capacity to an acceptable level. The basis for this strategy is that with a light workload, for example, there are more than sufficient resources to handle all the user's computing needs. Rather than waste those excess resources (or give the users an improved response time), the resources might be used to generate channel noise. From Figure 6.21, this strategy for the FB scheduler does not give the best

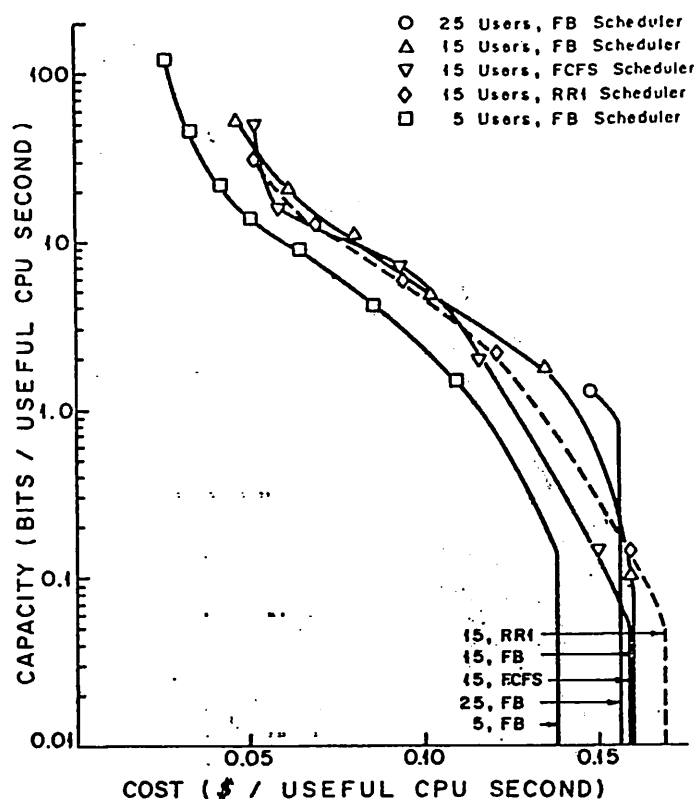


Figure 6.21
Minimum Cost for all Schedulers and Work Loads
Constrained Cascaded Compound Channel

cost/capacity trade-off for the light workload since a better cost can be obtained without adding pseudo-users for any desired capacity. For the medium workload case, a small advantage might be gained for very small capacity requirements by adding pseudo-users due to the slightly smaller absolute confinement cost of the 25 user workload. However, the increased response time seen by all the unconfined users would cause many users to become unhappy with such a policy. This might make such a small cost gain for the confiner unjustifiable with respect to the other inconvenienced users.

Before a system designer implements partial or absolute confinement, the cost of CPU time to the confiner must be compared with the cost of CPU time to an unconfined process. Such a comparison is made in Figure 6.22 for the light, medium and heavy workload cases for the FB algorithm. In this figure, the confiner cost normalized by the unconfined process cost is given. The workload and scheduler that have the lowest normalized cost for low capacity requirements is the heavy workload FB scheduler. In this case, the user can expect to pay 3 times more than an unconfined job. As the number of unconfined processes decreases, the penalty factor for executing confined subsystems rises to between 5.6 and 6.3 times. Only for very large capacities (greater than 20 bits/useful CPU second) with the light and medium workload can a cost penalty of less than double the unconfined process CPU time cost be achieved. This shows that any type of confinement is a very expensive feature to implement in a computer system.

This section has developed a method for implementing partial confinement that is based on information theoretic channel models. Partial confinement implemented by this method permits a wide selection of possible channel capacities and cost savings. Partial confinement is a viable option for a system designer needing a flexible confinement mechanism.

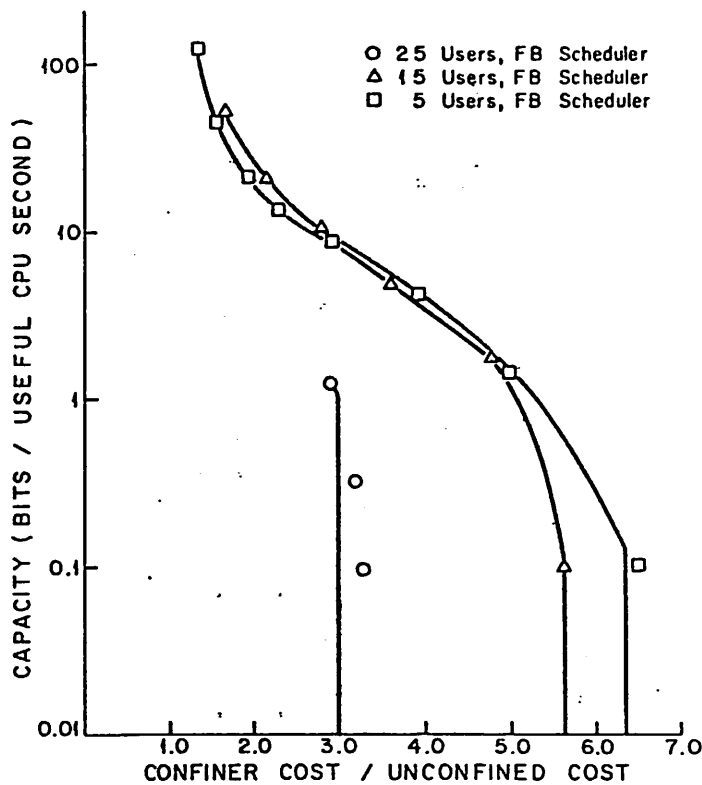


Figure 6.22
Partial Confinement Penalty Graph
Minimum Cost Channel

6.11 Partial Confinement Using the DCWM Channel Model

An alternative to the DMC channel model of the scheduling covert channel is to allow the confined subsystem to choose the quantum length to be allocated n consecutive times. Since the system does not settle back to an equilibrium state between the n quanta, this strategy implements a DCWM. To make the capacity of this scheduling strategy calculable, the system does return to its equilibrium state between each series of n quanta. The equilibrium state test for the DCWM is similar to the test for the DMC. The equilibrium state is assumed to be restored when the capacity of one variable quantum allocation is within 10% of the capacity at true equilibrium. The number of equilibrium quanta that must be allocated is determined from the semi-Markov system model, as for the DMC model. As explained in the first part of this chapter, n must be less than or equal to three for computability with a limited amount of computer resources.

The DCWM model is used to determine if a consecutive quantum allocation strategy is better than the DMC allocation strategy for the scheduling covert channel in terms of capacity and cost. A decision of which model is best suited for confinement modelling must take into consideration the confiner cost, the cost for the users not using the confinement mechanism, the capacity of the different channel models and the ease with which the model analysis can be done. The channel used for the analysis is the constrained cascaded compound channel in which the confined subsystem can choose a 0.0, 0.4 or 1.0 second quantum when a variable quantum is allocated. The parameter values analyzed represent the capacity extremes so that the behavior of the model for both high capacity/low cost and low capacity/high cost situations can be examined.

The careful reader will have observed that the channel operation of the DCWM is slightly different than the

corresponding DMC. The DCWM allocates three consecutive variable quanta rather than one for the DMC case. In the compound channel section of the cascaded compound channel, the P-biased coin for the DCWM determines whether the input letter corresponding to three zero quantum lengths is allocated, or whether three quantum lengths needed by the confined subsystem will be allocated. In the cascaded channel section, the decision of allocating the quanta desired by the confined subsystem is also made once for every three quanta allocated. This is different from the DMC in which the decision is made for each variable quantum allocated to the confined subsystem.

The reason the DCWM model is interesting is that the channel capacity may decrease due to the channel memory. If the channel capacity shows a marked decrease when the DCWM model is used, a confined scheduler should allocate several consecutive variable quanta to the confined subsystem before forcing the system to return to its equilibrium state. Also if the capacity is monotonically decreasing, there would be the possibility that the DMC capacity could be used as an upper bound for each variable quantum allocated in the DCWM. Thus equilibrium quanta would no longer be necessary in the channel model. However, Table 6.13 shows that although there is some capacity decrease for consecutively allocated variable quanta, there is not an order of magnitude change. For the medium and light workload cases, the capacity decrease is small as more consecutive variable quanta are allocated. Table 6.13 suggests that allocating more than one variable quantum at a time does not provide a significant additional channel capacity decrease over the DMC model. In many of the cases listed, the capacity of the DCWM even exceeds the DMC capacity in bits/useful CPU second. Thus the DCWM model is not significantly better than the DMC model on the basis of its capacity.

Another item of comparison is the costs for each model. The results in Table 6.13 are mixed. For the higher capacity channel models for each combination of workload and scheduler, the DCWM model is slightly better for the confiner cost. For the lower capacity channel models, the DMC model tends to be better. The maximum penalty for using the DMC model over the DCWM model is 11.9% of the DCWM confiner cost while the maximum penalty for using the DCWM model is 18.5% of the DMC confiner cost. The unconfined user cost (the cost for the users not executing confined) is very nearly the same for both the DCWM and the DMC models. The cost results do not indicate a preference of one model over the other.

Due to the nearly equal performance of the DCWM and the DMC models, the system designer should make the decision of which model to implement based on the difficulty in analyzing the models. Of the two models, the DCWM model is by far the harder to analyze in terms of computing resources expended. The cost is due to the large size of the CTPM. Since the DMC model requires much less computing time and space and since both models perform about the same with respect to cost and capacity, the DMC model should be used in practical partial confinement applications.

6.12 Effect of Two Spies on Channel Capacity

The DMC analysis assumed that only one spy process was executing with the confined subsystem to monitor the scheduling covert channel. In real systems, this assumption may not be accurate since one spy could be executing several spy processes at once. The purpose of this section is to analyze the capacity of one two-spy transmission strategy to see if more than one spy can increase the channel capacity.

In this experiment, two spies are placed in the CPU scheduling queue at the same time with the spy process spy₁

No. Users	Scheduler	P	A	DMC Confiner Cost (\$/Useful CPU Second)	DCWM Confiner Cost (\$/Useful CPU Second)	DMC Unconfined Cost (\$/Useful CPU Second)	DCWM Unconfined Cost (\$/Useful CPU Second)	DMC Capacity (Bits/Channel Use)	DCWM Capacity (Bits/Channel Use)	DMC Capacity (Bits/Useful CPU Second)	DCWM Capacity (Bits/Useful CPU Second)
25	FB	1.0	1.0	0.180	0.174	0.0483	0.0483	1.132	0.874	10.24	10.27
			0.4	0.315	0.285	0.0496	0.0476	0.0103	0.00798	0.116	0.104
		0.5	1.0	0.275	0.276	0.0479	0.0494	0.371	0.360	4.517	6.329
			0.4	0.448	0.445	0.0492	0.0505	0.00371	0.00331	0.0683	0.0503
		0.25	1.0	0.441	0.495	0.0487	0.0485	0.154	0.162	2.945	3.565
			0.4	0.689	0.693	0.0481	0.0478	0.00157	0.00140	0.0392	0.0306
15	FB	1.0	1.0	0.0471	0.0450	0.0300	0.0304	1.365	1.185	51.12	63.93
			0.4	0.181	0.193	0.0310	0.0316	0.0123	0.0109	0.301	0.289
		0.5	1.0	0.0616	0.0661	0.0305	0.0306	0.464	0.530	20.35	25.90
			0.4	0.157	0.179	0.0300	0.0310	0.00457	0.00474	0.154	0.215
		0.25	1.0	0.0797	0.0901	0.0305	0.0301	0.199	0.246	10.59	15.67
			0.4	0.159	0.173	0.0305	0.0304	0.00201	0.00216	0.100	0.119
15	FCFS	1.0	1.0	0.0518	0.0463	0.0340	0.0332	1.361	1.206	49.19	56.99
			0.4	0.172	0.182	0.0332	0.0345	0.0123	0.0109	0.281	0.275
		0.5	1.0	0.0593	0.0582	0.0332	0.0326	0.456	0.549	15.27	27.45
			0.4	0.149	0.151	0.0329	0.0326	0.00452	0.00491	0.139	0.171
		0.25	1.0	0.0950	0.0848	0.0328	0.0324	0.193	0.258	6.267	14.23
			0.4	0.172	0.164	0.0325	0.0330	0.00196	0.00228	0.0766	0.123
15	RR1	1.0	1.0	0.0514	0.0516	0.0333	0.0344	1.256	1.136	30.53	34.91
			0.4	0.161	0.164	0.0345	0.0353	0.0114	0.103	0.218	0.222
		0.5	1.0	0.0844	0.0778	0.0338	0.0334	0.416	0.516	7.677	17.86
			0.4	0.169	0.166	0.0339	0.0340	0.00415	0.00464	0.0941	0.145
		0.25	1.0	0.122	0.124	0.0334	0.0334	0.173	0.243	4.800	10.08
			0.4	0.230	0.204	0.0314	0.0338	0.00177	0.00214	0.0545	0.0797
5	FB	1.0	1.0	0.0292	0.0283	0.0240	0.0235	1.517	1.456	121.6	166.7
			0.4	0.271	0.311	0.0249	0.0252	0.0136	0.0131	0.610	0.682
		0.5	1.0	0.0334	0.0329	0.0237	0.0238	0.542	0.687	45.47	81.86
			0.4	0.199	0.221	0.0244	0.0250	0.00517	0.00611	0.324	0.430
		0.25	1.0	0.0421	0.0428	0.0241	0.0245	0.0245	0.332	21.65	43.22
			0.4	0.146	0.173	0.0240	0.0245	0.00238	0.00294	0.185	0.294

Table 6.13

Comparison of the DMC and DCWM Models

in front of the spy process spy_2 in the queue. Both spy processes are assumed to take one microsecond of CPU time and to have a memory requirement of one word in order to obtain an upper bound on the channel capacity. Due to the discretization assumption, these assumptions mean that each spy process executes within one discretization interval (= 0.2 seconds). Since the confined subsystem is assumed to already be in the CPU queue, the confined subsystem must execute after spy_2 and before spy_1 . Knowing this, the spy can ignore the channel noise contribution of any unconfined process that executes after spy_1 and before spy_2 . Thus the channel outputs consist of the real time between the execution of spy_2 and the execution of spy_1 . The constrained channel model (with 0.0, 0.4 and 1.0 second quanta that can be allocated to the confined subsystem) was simulated for this two-spy case and the channel capacity was computed by the method outlined in Chapter 5 and Chapter 2.

There are two opposing forces that affect the channel capacity. The capacity may increase due to some of the channel noise being filtered out by the spy execution strategy. The capacity may decrease due to the extra overhead induced by the additional spy execution. The measurements are shown in Table 6.14. For all the cases shown, the difference in capacity between using one spy and using two spies is less than 3%. In an attempt to increase the channel capacity by trapping more unconfined process noise, the execution of spy_2 was increased so that the execution of spy_2 would take longer than one discretization interval. This only resulted in lower channel capacities and a lower system performance with respect to user response time due to the increased workload attributed to spy_2 .

The use of a second continuously executing spy process does not significantly increase the scheduling covert channel

Number of Users, Scheduler	2 Spy Capacity (Bits/Channel Use)	1 Spy Capacity (Bits/Channel Use)
25, FB	1.11	1.13
15, FB	1.36	1.37
15, FCFS	1.36	1.36
15, RR1	1.28	1.26
5, FB	1.51	1.52

Table 6.14

Effects of Two Spies on the Channel Capacity for a Zero Length Equilibrium Quantum

capacity. For systems like the CTSS system, the spy appears to be better off by using only one spy executing at a time.

6.13 The Constrained Time-Average Equilibrium Condition Channel

All the DMC models previously analyzed based the return to the equilibrium state on the number of times the confined subsystem was eligible to execute and was allocated an equilibrium quantum. After a certain number of equilibrium quanta were allocated, the confined subsystem could choose the next quantum length. One drawback to this approach is that if the CPU queue is very congested, it may take a long real time for the confined subsystem to complete the required number of equilibrium quanta before another variable quantum length can be allocated. If the equilibrium quantum length chosen is zero, this means that the time between possible non-zero quantum allocations to the confined subsystem is long, which increases the confiner's response time. On the other hand, if the CPU queue is al-

most empty, many variable quanta could be allocated in a small real time which increases the number of bits potentially transmitted per useful CPU second. Such rapid allocation of variable quanta is usually not needed by the confined subsystem since an average user is characterized by a long average think time (35.2 seconds) compared to a short average CPU requirement (0.88 seconds).

Both of the above problems can be attacked by specifying when the system returns to its equilibrium state in terms of real time instead of the number of equilibrium quanta allocations needed. In this case, two variable quanta could be allocated in succession if the CPU queue is congested, and the number of variable quanta allocated (and hence the number of bits leaked per useful CPU second) could be decreased if the CPU queue is empty.

The algorithm for determining how much time is needed for equilibrium to be reestablished is analogous to determining the number of equilibrium quanta needed. To compute the channel capacity at time t (for t a multiple of the 0.2 second discretization interval), a table is computed with row n (n greater than or equal to 0) corresponding to time ($n \times 0.2$) seconds and columns corresponding to the states of the model. The entry in column m , row n represents the probability that the system entered state m at time ($n \times 0.2$) and is still in state m at the current time. To start the calculation, the current time variable, t' , is set to 0 and the 0th row of the table is initialized to the model state probabilities that occur after the largest system perturbation due to the confined subsystem using the semi-Markov model of Chapter 5. The parameters of the semi-Markov model in Chapter 5 were chosen so that the channel outputs seen by the spy can be computed. Thus the probabilities in row 0 correspond to the system state probabilities that occur when the spy next executes. If the spy executes just before the confined subsystem, the time corresponding to row 0 of the table occurs just before the execution of the confined subsystem following the variable quantum allocation. Since the spy can not be identified, the clock does not start running until the confined subsystem is allocated its next quantum after the variable quantum. Thus at least one equilibrium quantum is allocated to the confined subsystem between two variable quanta.

The iteration step starts by increasing t' by 0.2 seconds. For each entry in the rows corresponding to a time less than t' , the probability of making a state transition at t' , P' , is obtained from the semi-Markov model of Chapter 5 and is entered in the appropriate column of t' . Of course the entry corresponding to the time of entry into the state from which the transition is made is decremented by P' . When all rows corresponding to times less than t' have been processed, the DMC channel capacity is computed by using the sum of each column as the corresponding state probability, then using the method explained in Chapter 5 to generate a CTPM from the state probabilities. The algorithm in Chapter 2 is used to compute the channel capacity from the CTPM. This iteration step is repeated until the DMC capacity of the channel returns to within 10% of its equilibrium value. The last value of t' is the amount of real time that must pass until the time-average equilibrium is assumed to be restored.

The comparison of the constrained channel using the number of equilibrium quanta (rounds) in its return to equilibrium calculation and the constrained channel using time in its return to equilibrium calculation for a zero length equilibrium quantum is shown in Table 6.15. The most significant feature of the table is the large decrease in the channel capacity for the time-average channel. The smallest percentage decrease is 82% (15 users, RR1) and the largest is 91% (5 users, FB). The decrease is mainly due to the time-average method of determining the state probabilities. In the

Number of Users, Scheduler	Round-Ave Capacity (Bits/Useful CPU Second)	Time-Average Capacity (Bits/Useful CPU Second)	Round-Ave Cost (\$/Useful CPU Second)	Time-Average Cost (\$/Useful CPU Second)
25, FB	10.24	1.522	0.180	0.158
15, FB	51.12	5.403	0.0471	0.0836
15, FCFS	49.19	7.378	0.0518	0.0610
15, RR1	30.53	5.537	0.0514	0.0569
5, FB	260.7	23.434	0.0292	0.0402

Table 6.15
Round-Average vs. Time-Average Equilibrium Channel
for Zero Length Equilibrium Quantum

semi-Markov model, the lower numbered states represent fewer unconfined processes in the CPU queue, which means less channel noise and a higher channel capacity. When all the rounds are weighted equally, these high capacity states occur with a large probability. However, when time-averaging is used, these states are less probable due to the short amount of time (but large number of rounds) spent in these states. For example, the state representing a large number of unconfined users in the system will have a higher probability under time-averaging than under round-averaging due to the large average amount of time spent in the state before the next transition. The remaining factor in the capacity decrease is the decrease in the number of variable quanta allocated per useful CPU second due to fewer variable quanta being allocated during a succession of short round times as noted before.

The capacity decrease is achieved at the expense of a larger confiner cost for the medium and light workloads. As

Number of Users, Scheduler	Round-Ave CPU Cost (\$/Useful CPU Second)	Round-Ave User Cost (\$/Useful CPU Second)	Time-Average CPU Cost (\$/Useful CPU Second)	Time-Average User Cost (\$/Useful CPU Second)
25, FB	0.0175	0.163	0.0181	0.125
15, FB	0.0172	0.0299	0.0177	0.0540
15, FCFS	0.0180	0.0338	0.0177	0.0374
15, RR1	0.0182	0.0332	0.0184	0.0356
5, FB	0.0179	0.0113	0.0177	0.0206

Table 6.16
Cost Components for the Constrained Channel and the Time-Average Channel for a Zero Length Equilibrium Quantum

shown in Table 6.16, the major difference in the confiner cost for the round-average and the time-average channels is the user cost (i.e. response time). The round average channel permits smaller response times for the medium and lightly loaded cases but the time average channel has smaller response times for the heavy workload case. This advantage is due to the confined subsystem being allocated more variable quanta during times of great congestion as pointed out before.

The time-average constrained channel model significantly decreases the channel capacity over the round-average case but at an increase in cost for the medium and light workloads. Comparing the capacity/cost figures of Table 6.15 with the minimum cost curves in Figure 6.21 shows that the time-average cost for the capacity in Table 6.15 is less than the minimum cost curve for the 15 user FCFS and the 15 user RR1 algorithms. The time-average cost is 66% of the round average cost for FCFS and 62% of the round-average cost for RR1. The time-average channel points for the FB algorithm for the three workloads falls on the minimum cost curve in Figure 6.21. This suggests that the time-average channel model could be profitably used in the FCFS and RR1 algorithms but perhaps not in the FB algorithm.

6.14 Conclusions

The CPU scheduling covert channel is a channel that is the most heavily used in many systems. The characteristics of this channel for transmitting information are much like the other resource scheduling channels (e.g. the disk mass storage I/O scheduler). The major difference between the different resource schedulers as far as confinement is concerned is the workload. In most systems there is a single CPU that must be shared among all processes, which leads to a large amount of contention under heavy load. The disk I/O scheduler, on the other hand, usually services requests for a number of independent disk units in a large system, which results in a set of parallel channels - one channel for each disk unit. The contention for the I/O devices with a good space allocation algorithm could be much less than for the CPU since several independent requests can be processed in parallel. This reduced contention means that the disk scheduling covert channel could have a much higher capacity than the CPU channel studied for a given system user load. Thus there are several potentially high capacity covert channels in a typical operating system since the CPU channel is shown to have a significant information capacity, particularly under light load. Thus the CPU scheduler channel is not the only channel that must be confined in the operating system.

For resource scheduling channels, partial confinement is shown to be more cost effective for medium and light workloads than absolute confinement. In the heavy workload case, the cost of partial confinement is nearly the same as absolute confinement, which is a surprising result. This means that the absolute confinement strategy is probably preferable for the (hopefully few) resources that are system bottlenecks. The remaining scheduling channels (which usually comprise the majority of channels) then correspond to the medium and light workload cases investigated for the CPU scheduling channel. Analogous methods for reducing the channel capacity for these channels, such as defining an appropriate channel model or using a time-averaging scheme, can be devised. The same large range of trade-offs of capacity versus cost should be present for all resource scheduling channels as for the CPU channel. This does not mean that executing a process under confinement will not be more expensive than not using confinement. The measurements taken for the scheduling channel show that the confiner can expect to pay from one to six times the unconfined rate for a CPU second depending on the workload, the type of confinement, and the scheduler.

In some applications, the weighting factor for the user response time might vary from the "normal" values assumed. If a process is given a very high priority for quick completion, user cost might be ten times its usual value. In this case, the CPU cost per second is less than the user cost per second. To minimize the confiner cost, a larger amount of CPU time would be devoted to the confiner than usual. For absolute confinement, the process would most likely receive a large fixed percentage of the CPU. If the process to be executed requires extensive interaction, a channel model like the constrained cascaded compound channel could be used with perhaps a different set of quantum length choices that causes more CPU time to be allocated to the confined subsystem. Such a policy could involve allocating a non-zero equilibrium quantum to the confined subsystem. If instead of increasing, the user cost decreases by a factor of ten, there would be an incentive for the scheduler to allocate less of the CPU to the confined subsystem. For an interactive process, this could be implemented in the constrained cascaded compound channel by decreasing the value of P or by decreasing the equilibrium quantum length. For absolute confinement, this could be implemented by decreasing the quantum length

allocated or the frequency of allocation.

With the analysis technique outlined in this and preceding chapters, it is possible to select a scheduling policy and the policy parameters needed to keep the information leakage at a prescribed level. For example, at every user logon or logoff, the calculation of the information channel parameters could be made. For a channel implemented by a DMC, this calculation could be done rapidly. (Since there is no advantage to using a DCWM model, all channels would be DMC.) Such an adaptive scheduler is needed to be able to quickly adapt to changing workloads.

From the analysis presented in this chapter, partial confinement is an option that should be included in confinement mechanisms implemented for timesharing systems. The demonstrated feasibility of implementing this concept is perhaps the major contribution of this thesis.

7. CONCLUSIONS

7.1 Summary

In many cases it is no longer sufficient to provide a security mechanism that only prevents one user from directly accessing another user's data in an unauthorized manner. The concept of security should now include a provision for blocking the subtle unauthorized information leakages that are implemented by storage, legitimate, and covert channels. This thesis has analyzed in detail the resource sharing covert channel that is implemented by the CPU scheduler. There are many other covert channels in a computer system that result from resource sharing - e.g. the file disk channel and the swapping disk channel. All channels of this type can be analyzed by the methods used in this thesis.

In the CTSS system studied, the scheduling covert channel had the most contention of any resource scheduling channel since there is no overlap of CPU and I/O. On other systems in which the contention for a resource scheduling channel is not as great, higher or lower channel capacities could occur. The higher capacities occur if the confined subsystem and the spy can execute concurrently so that the confined subsystem resource request affects the spy's request. If, however, the contention for the channel is light enough that all requests complete and the resource is returned to a standard state before the next request arrives, the channel capacity would be zero. Thus the results derived here for the CPU scheduling channel can not be used to approximate the other resource scheduling channels. However, the technique developed for analyzing the CPU channel can be used for the other channels. The technique used is enumerated in the first six chapters.

Chapter 1 lists the recent developments in operating system design that have improved the state-of-the-art in security mechanisms. The problem of preventing direct access to one user's objects by an unauthorized user and the prevention of information leakage through overt and legitimate channels has been studied. However, the problem of covert leakage through resource scheduling channels has not been studied and is an active area of research. The objective of this thesis is to analyze one such channel, the CPU scheduling channel, in order to better understand the problem of covert leakage and to find methods to control the leakage.

A general discussion of covert channels is given in Chapter 3 to define the two alternatives for containing the leakage through covert channels. Absolute confinement allows zero leakage but with the restriction that no resource allocation decisions can be based on the state of the executing confined subsystem. All the implementations of absolute confinement surveyed (masking, partitioning, denial of usage, and restoration of a standard state) could cause a large increase in cost to either the confiner or the users not executing confined, or both. In an attempt to decrease this cost, partial confinement is offered as an alternative. Under partial confinement the resource allocation could depend on the state of the confined subsystem but only if a non-zero amount of leakage could be tolerated. The major focus of the remainder of the thesis is on investigating this cost/capacity trade-off.

Chapter 4 discusses the details of the implementation of the CPU scheduling covert channel. The method for sending information through the waiting time between quantum allocations to the spy is explained. In order to make the channel capacity measurable for partial confinement, two restrictions on the operation of the channel are made to permit modelling the channel by a discrete memoryless channel (DMC) and a discrete channel with memory (DCWM). The first restriction is that the system must return to a chosen equilibrium state

before each quantum that is dependent on the state of the confined subsystem is allocated. The metric chosen for determining when equilibrium is established is the DMC capacity function (as explained in Chapter 6). The second restriction is that the operation of the channel is discretized with a discretization interval that is much larger than a machine cycle. This reduces the size of the channel transition probability matrix (CTPM) to reasonable dimensions. The discretization interval size chosen is 0.2 seconds unless otherwise noted. This chapter also describes the scheduling algorithms used in the remainder of the thesis: round-robin with a 1 second quantum length (RR1), round-robin with a 2 second quantum length (RR2), first-come-first-served (FCFS), and two-level feedback (FB).

Chapter 5 explains the CTSS system used as a basis for the simulator measurements taken and the derivation of a semi-Markov model for the system. The semi-Markov model is needed to generate the information channel model CTPM from the simulator output. A major contribution of this thesis is the development of the technique using this model in the analysis of covert channels. This technique is applicable to the other covert channels which transmit information through the spy observing a time function. No use is made of the stochastic properties of the model in the generation of the CTPM; only the structure of the model is needed.

Chapter 6 builds on the discussion of Chapters 3 and 4, and the semi-Markov model of Chapter 5 to present measurements on the CPU scheduling covert channel for a CTSS-like system. The cost and capacity results presented assume that the confined subsystem is not trying to send information. In this case, the cost measurement is the penalty the confiner pays for executing confined and the capacity measurement is the amount of information leakage that must be assumed is being leaked. No tests were made to find the cost or capacity of a confined subsystem that does try to actively transmit information.

- The information leakage that must be assumed through the CPU scheduling covert channel in a system not offering confinement can be very large. The number of bits per useful CPU second allocated to the confined subsystem ranged from 260 for the 5 user, FB scheduler to 1.17 for the 25 user, RR1 scheduler. As the discretization interval decreases, the channel capacity is shown to be even greater.
- There exists an inverse relationship between workload and the channel capacity, as would be expected. The heavier workloads generate more channel noise which gives a lower channel capacity than the light workloads with less channel noise. The cost of confinement, measured as the cost of a useful CPU second for the confined subsystem, varies directly with the workload. The cost for the 25 user workload is 8 to 9 times the cost for the 5 user workload for the unconstrained channel model. Thus simply increasing the workload is not a good method of regulating information leakage.
- The cost for making the channel capacity measurable for partial confinement, which is the result of introducing equilibrium quanta, causes the unconstrained channel to cost 2.71 times as much as an unconfined subsystem for each useful CPU second for the 25 user, FB scheduler. For the 5 user, FB scheduler, the cost is only 1.08 times as much. The penalty the confiner pays for confinement definitely increases as the number of users increases.
- A major result of this thesis is that for workloads that border on system saturation, the cost of absolute confinement is only about 10% greater than partial confinement. For this workload, most users would be likely to choose absolute confinement for such a small

cost penalty. This result is counter to some predictions that partial confinement is always more advantageous than absolute confinement.

- The usefulness of partial confinement is shown for the 5 and 15 user workloads in which the constrained cascaded compound model afforded the confiner a large range of channel capacities that cost less than absolute confinement. This result means that partial confinement can be significantly cheaper than absolute confinement for certain workloads.
- The DCWM model is shown to yield almost the same performance characteristics as the DMC channel model for the types of channels studied. Since the DMC is much easier to analyze and makes computation of the covert channel parameters feasible in real time to adapt to changing load, the DMC channel model should be used for partial confinement implementations.
- The effect on channel capacity of introducing two spies into the scheduling covert channel is to generally decrease the capacity for the trapping mechanism that is tried. This results from the second spy generating more channel noise by increasing the workload of the system than it eliminates. This does not say that all two spy strategies are inferior to one spy strategies since the system being used may contain a feature that can be exploited by two spies to increase the channel capacity. Also an exhaustive simulation of all strategies was not done. However, this experiment is an important point in the space of two spy strategies since it says that the effect on capacity may not be large.
- The last result is that an alternate method for determining equilibrium (i.e. time-averaging) may yield a better cost/capacity curve (i.e. less cost for equal capacities) than the round average method used for all the preceding measurements. This method did yield points below the minimum cost curve for the constrained cascaded compound channel model for the RR1 and FCFS algorithms but yielded points on the minimum cost curve for the FB algorithm.

7.2 Future Research

There are four areas that are logical extensions of the work in this thesis. The first is to use the methods outlined in this thesis to analyze the other common covert channels in timesharing computer systems. The CPU scheduling covert channel is by no means the only potentially large capacity channel in a computer system. In any practical system design effort, a comprehensive analysis of all the resource sharing channels is necessary to determine the ones that should be partially confined and the ones that should be absolutely confined. This decision can have major consequences on the performance of the system.

The second area is to investigate the synthesis of channel models that are tailored to certain confined subsystem characteristics. For example, if a timesharing program is to be executed confined, can a channel model be found that gives the lowest channel capacity for any cost. The problem of optimizing channel models for different types of tasks is still an open question.

The third area of research is to implement a timesharing system with partial confinement to determine if the system can in practice adjust the channel model parameters as users log on and log off to maintain a constant capacity for the scheduling CPU channel. Data on the real data rate achievable through the covert channel is needed to determine if the capacity function used in this analysis is much larger than that achievable in practice.

The fourth area involves experimenting with other rea-

sonable cost functions to determine if other partial confinement strategies are feasible in an actual computer installation. The linear CPU cost function used in this thesis may not always be the one used in practice. For example, non-prime time computing can be done at a discount in many installations. The effect of artificially increasing the workload with "pseudo-users" to decrease the capacity when excess CPU capacity is available might be feasible in certain situations.

BIBLIOGRAPHY

- [Andrews74] Andrews, G. R. "COPS - A Mechanism for Computer Protection," *Proceedings of the International Workshop on Protection in Operating Systems*, IRIA, Paris, France, (August 1974), pp. 5-25.
- [Ash65] Ash, Robert. *Information Theory*, Interscience Publishers, John Wiley and Sons, (1965).
- [Belady74] Belady, L. A. and C. Weissman. "Experiments with Secure Resource Sharing for Virtual Machines," *Proceedings of the International Workshop on Protection in Operating Systems*, IRIA, Paris, France, (August 1974), pp. 27-33.
- [Bell73] Bell, D. E. and L. J. LaPadula. "Secure Computer Systems: Mathematical Foundations," Report ESD-TR-73-278, Volume 1, The MITRE Corporation, (November 1973).
- [Blahut72] Blahut, Richard E. "Computation of Channel Capacity and Rate Distortion Functions," *IEEE Transactions on Information Theory*, IT-18:4 (July 1972), pp. 460-473.
- [Buzen73] Buzen, J. P. and U. Gagliardi. "The Evolution of Virtual Machine Architecture," *Proceedings of the National Computer Conference*, 42 (1973), pp. 291-299.
- [Chu76] Chu, Wesley W. and Holger Opderbeck. "Analysis of the PFF Replacement Algorithm via a Semi-Markov Model," *Communications of the ACM*, 19:5 (May 1976), pp. 298-304.
- [Coffman68] Coffman, E. G. and L. Kleinrock. "Computer Scheduling Methods and Their Countermeasures," *Proceedings of the Spring Joint Computer Conference*, 32 (1968), pp. 11-21.
- [Cohen75] Cohen, E. and D. Jefferson. "Protection in the HYDRA Operating System," *Proceeding of the Fifth Symposium on Operating Systems Principles*, (1975), pp. 141-160.
- [Corbato62] Corbato, F. J. and M. Merwin-Daggett and R. C. Daley. "An Experimental Time-Sharing System," *Proceedings of the Spring Joint Computer Conference*, 21 (1962), pp. 335-344.
- [Corbato72] Corbato, F. J. and J. H. Saltzer and C. T. Clingen. "MULTICS - The First Seven Years," *Proceedings of the Spring Joint Computer Conference*, 40 (1972), pp. 571-583.
- [Crane74] Crane, Michael and Donald I. Iglehart. "Simulating Stable Stochastic Systems II: Markov Chains," *Journal of the ACM*, 21:1 (January 1974), pp. 114-123.
- [Crisman65] Crisman, P. A., ed. *The Compatible Time-Sharing System: A Programmer's Guide*, Second Edition, M. I. T. Press, (1965).
- [Denning76] Denning, Dorothy E. "A Lattice Model of Secure Information Flow," *Communications of the ACM*, 19:5 (May 1976), pp. 236-243.
- [Denning77] Denning, Dorothy E. and Peter J. Denning. "Certification of Programs for Secure Information Flow," *Communications of the ACM*, 20:7 (July 1977), pp. 504-513.
- [Dennis68] Dennis, J. B. "Programming Generality, Parallelism, and Computer Architecture," *Proceedings of IFIP 1968*, North Holland, Amsterdam, (1968), pp. C1-C7.
- [Estrin67] Estrin, G. and L. Kleinrock. "Measures, Models and Measurements for Time-Shared Computer Utilities," *Proceedings of the ACM National Conference*, (1967), pp. 85-96.
- [Fenton74] Fenton, J. S. "Memoryless Subsystems," *The Computer Journal*, 17:2 (May 1974), pp. 143-147.
- [Gallager68] Gallager, Robert G. *Information Theory and Reliable Communication*, John Wiley and Sons, (1968).
- [Gat76] Gat, Israel and Harry J. Saal. "Memoryless Execution: A Programmer's Viewpoint," *Software Practice and Experience*, 6 (1976), pp. 463-471.
- [Gilbert60] Gilbert, E. N. "Capacity of a Burst-Noise Channel," *Bell System Technical Journal*, 39 (September 1960), pp. 1253-1265.
- [Gold69] Gold, Michael M. "Time-Sharing and Batch Processing: An Experimental Comparison of Their Values in a Problem-Solving Situation," *Communications of the ACM*, 12:5 (May 1964), pp. 249-259.
- [Hansen70] Hansen, Per Brinch. "The Nucleus of a Multiprogramming System," *Communications of the ACM*, 13:4 (April 1970), pp. 238-241+.
- [Kleinrock75] Kleinrock, Leonard. *Queueing Systems; Volume 1: Theory*, John Wiley and Sons, 1975.
- [Lampson69] Lampson, B. W. "Dynamic Protection Structures," *Proceedings of the Fall Joint Computer Conference*, 35 (1969), pp. 27-38.
- [Lampson73] Lampson, B. W. "A Note on the Confinement Problem," *Communications of the ACM*, 16:10 (October 1973), pp. 613-615.
- [Lavenberg75] Lavenberg, S. S. and D. R. Slutz. "Introduction to Regenerative Simulation," *IBM Journal of Research and Development*, 19:5 (September 1975), pp. 458-462.
- [Lettieri76] Lettieri, Larry. "Making Their Marks," *Computer Decisions*, 8:1 (January 1976), pp. 28-30.
- [Lipner75] Lipner, Steven B. "A Comment on the Confinement Problem," *Proceedings of the Fifth Symposium on Operating Systems Principles*, The University of Texas at Austin, (November 1975), pp. 192-196.
- [Parnas72] Parnas, D. L. "A Technique for Software Module Specification with Examples," *Communications of the ACM*, 15:5 (May 1972), pp. 330-336.
- [Popek74] Popek, Gerald J. and Charles S. Kline. "Verifiable Secure Operating System Software," *Proceedings of the National Computer Conference*, 43 (1974), pp. 145-151.
- [Ritchie74] Ritchie, Dennis M. and Ken Thompson. "The UNIX Time-Sharing System," *Communications of the ACM*, 17:7 (July 1974), pp. 365-375.
- [Ross70] Ross, Sheldon M. *Applied Probability Models with Optimization Applications*, Holden-Day, (1970).
- [Rotenberg74] Rotenberg, Leo J. "Making Computers Keep Secrets," MIT Project MAC Report MAC-TR-115, Ph. D. Thesis, Cambridge, Massachusetts, (February 1974).
- [Saltzer74] Saltzer, Jerome H. "Protection and Control of Information Sharing in MULTICS," *Communications of the ACM*, 17:7 (July 1974), pp. 388-402.
- [Scherr66] Scherr, Allan L. *An Analysis of Time-Shared Computer Systems*, Research Monograph Number 36, M. I. T. Press, (1966).
- [Schiller75] Schiller, W. L. "The Design and Specification of a Security Kernel for the PDP-11/45," Report ESD-TR-75-69, The MITRE Corporation, (May 1975).
- [Schroeder72A] Schroeder, M. D. "Cooperation of Mutually Suspicious Subsystems in a Computer Utility," MIT Project MAC Report MAC-TR-104, Ph. D. Thesis, Cambridge, Massachusetts, (1972).

- [Schroeder72B] Schroeder, M. D. and J. H. Saltzer. "A Hardware Architecture for Implementing Protection Rings," *Communications of the ACM*, 15:3 (March 1972), pp. 157-170.
- [Spier73] Spier, Michael J. and Thomas N. Hastings and David N. Cutler. "A Storage Mapping Technique for the Implementation of Protective Domains," *Software Practice and Experience*, 4 (1974), pp. 215-230.
- [Turn72] Turn, Rein and Norman Z. Shapiro. "Privacy and Security in Databank Systems - Measures of Effectiveness, Costs, and Protector-Intruder Interactions," *Proceedings of the Fall Joint Computer Conference*, 41 (1972), pp. 435-444.
- [Ware67] Ware, W. H. "Security and Privacy in Computer Systems," *Proceedings of the Spring Joint Computer Conference*, 30 (1967), pp. 279-282.
- [Weissman69] Weissman, C. "Security Controls in the ADEPT-50 Time-Sharing System," *Proceedings of the Fall Joint Computer Conference*, 35 (1969), pp. 119-133.
- [Weissman75] Weissman, Clark. "Secure Computer Operation with Virtual Machine Partitioning," *Proceedings of the National Computer Conference*, 44 (1975), pp. 929-934.
- [Wolfowitz63] Wolfowitz, J. "The Capacity of an Indecomposable Channel," *Sankhya*, Indian Journal of Statistics, Series A, 25 (1963), pp. 101-108.
- [Wolfowitz64] Wolfowitz, J. *Coding Theorems of Information Theory*, Second Edition, Springer-Verlag, (1964).

BIOGRAPHICAL NOTE

Jeffrey Craig Huskamp was born in Louisville, Kentucky on October 17, 1949. He attended public school there, graduating from Atherton High School in June, 1967. He attended Purdue University starting in September, 1967, and received the B. S. degree with highest honors in electrical engineering in June, 1971, and the M. S. degree in computer science in June, 1972. In September, 1974, he entered the Electrical Engineering and Computer Science Department at the University of California, Berkeley.

While at Purdue University, Mr. Huskamp worked on the development of graphic input/output packages for use in teaching circuit analysis to undergraduates in electrical engineering. He also developed a FORTRAN compiler for an IMLAC minicomputer for use in the Computer Aided Design Laboratory in the Mechanical Engineering Department. From July, 1972 to September, 1977, he was employed as a systems programmer at Lawrence Livermore Laboratory. While at the Laboratory, he participated in the RISOS (Research in Secured Operating Systems) Project which studied the security of commercially available operating systems. Mr. Huskamp is presently employed at the Institute for Defense Analyses in Princeton, New Jersey.

Mr. Huskamp is a member of the ACM, Tau Beta Pi, Eta Kappa Nu, Omicron Delta Kappa, Sigma Pi Sigma and Phi Eta Sigma.