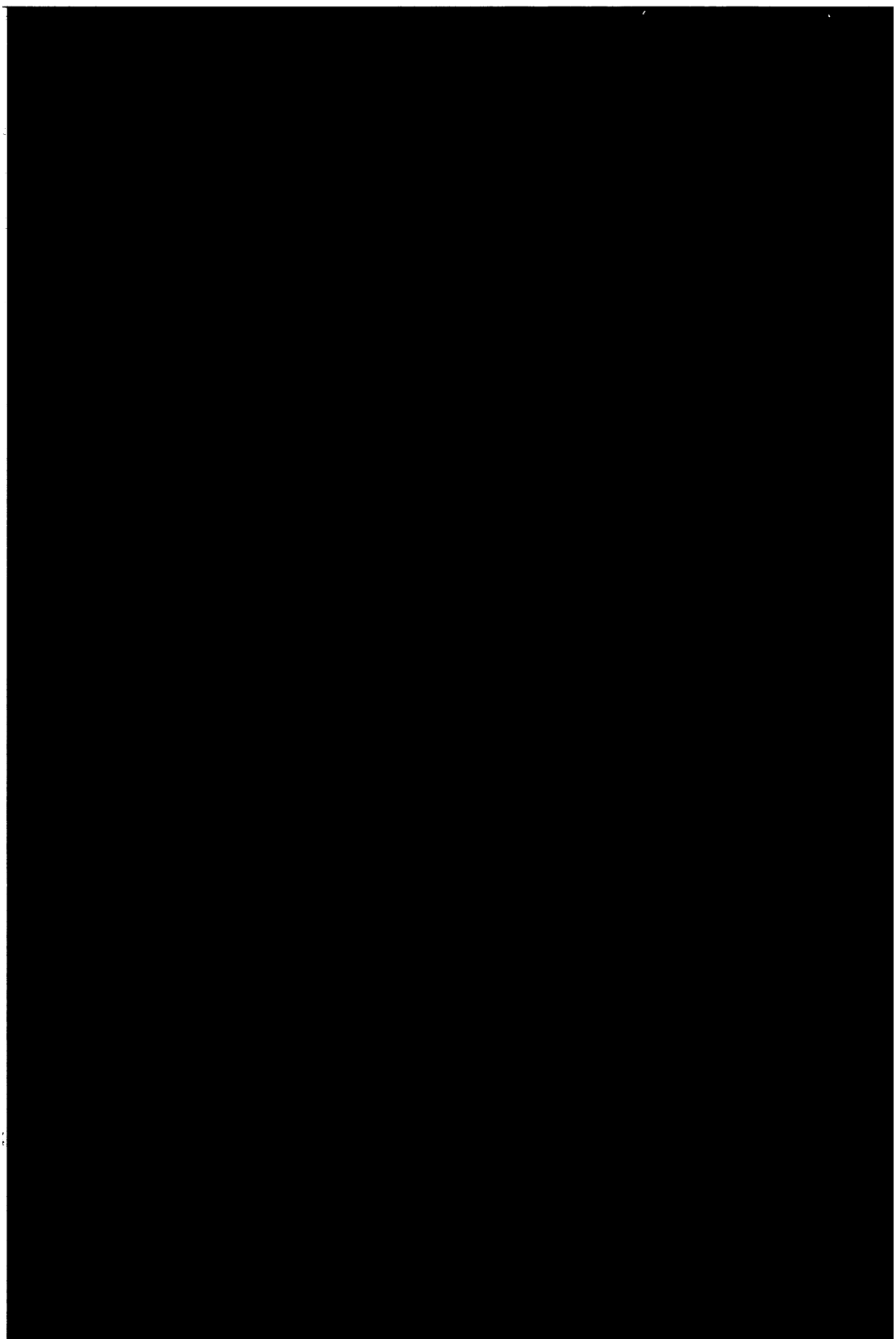


This blank page was inserted to preserve pagination.

ARBITER COMPUTATIONS IN A MULTIPLE-ACCESS COMPUTER SYSTEM

by

HARVEY MICHAEL METTEL

S.B., Massachusetts Institute of Technology
(1967)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 1968

Signature of Author_____
Department of Electrical Engineering, August 19, 1968

Certified by_____
Thesis Supervisor

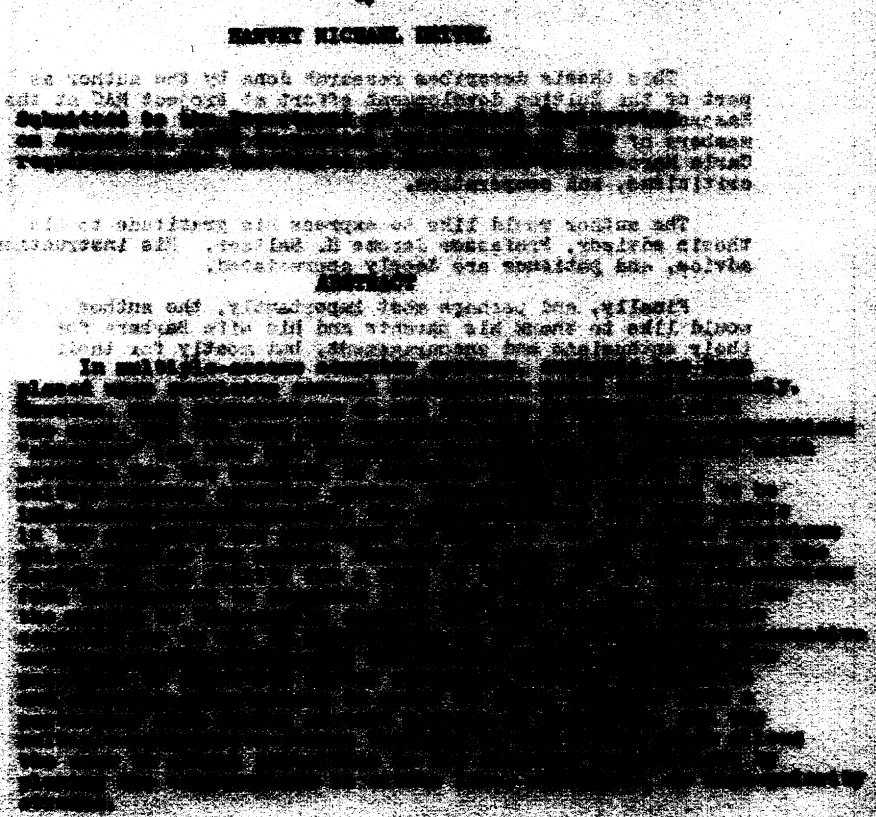
Accepted by_____
Chairman, Departmental Committee on Graduate Students

This empty page was substituted for a blank page in the original document.

[redacted]

by

[redacted]



[redacted]

ACKNOWLEDGEMENT

This thesis describes research done by the author as part of the Multics development effort at Project MAC at the Massachusetts Institute of Technology. Thanks to the many members of the Project MAC research staff, particularly Carla Marceau and Karolyn Martin, for their suggestions, criticisms, and cooperation.

The author would like to express his gratitude to his thesis advisor, Professor Jerome H. Saltzer. His instruction, advice, and patience are deeply appreciated.

Finally, and perhaps most importantly, the author would like to thank his parents and his wife Barbara for their enthusiasm and encouragement, but mostly for their understanding.

H.M.D.

Cambridge, Massachusetts

August, 1968

CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENT	iv
Chapter 1 - INTRODUCTION	1
1.1 Terminology	1
1.2 Background	4
1.3 Organization	6
Chapter 2 - FEATURES OF ABSENTEE COMPUTATIONS	8
2.1 Computation Modes	8
2.2 Features of Absentee Computations	9
2.3 System Features	11
Chapter 3 - OVERVIEW OF THE ABSENTEE MONITOR	15
3.1 Major Sections of the Absentee Monitor	15
3.1.1 Absentee Queue Control	15
3.1.2 Absentee Waiting Queues	17
3.1.3 Absentee Running Queues	17
3.1.4 Absentee Initiation Module	18
3.1.5 Absentee Shelving Module	18

3.2 Relationships between the Absentee Monitor and other Parts of the Multiple-Access Computer System 19

3.2.1 System Control 19

3.2.2 Performance Measurement 19

3.2.3 Load Control 20

3.2.4 Load Control Table 21

3.2.5 Reserver 21

3.2.6 SAVE, RESUME, and QUIT 21

3.2.7 User Commands 22

Chapter 4 - FEATURES OF THE QUEUEING MECHANISM 23

4.1 Necessity for a Queueing Mechanism 23

4.2 Queue Discipline 24

4.3 First-In-First-Out Disciplines 25

4.4 Associating an Ordering with the Running AC's 27

4.5 Computation Streams 28

4.6 Flow of a Computation through a Stream 30

4.7 Multiple-Stream Queueing Mechanism 32

Chapter 5 - LOAD CONTROL 34

5.1 Terminology 34

5.2 Load Control in a Purely Interactive System 36

5.3 Load Trimming Strategies 38

5.4 Load Control in a Single-Stream Purely Absentee System 43

5.5 Load Control in a Multiple-Stream Purely Absentee System 46

5.6 Load Reapportionment in a Multiple-Stream Purely Absentee System 53

5.7 Load Control in a Multiple-Stream Purely Interactive System 59

5.8 Load Control in a Hybrid System with Multiple Interactive and Absentee Streams 64

5.9 Load Reapportionment in a Hybrid System with Multiple Interactive and Absentee Streams 72

Chapter 6 - COMMANDS FOR USE WITH ABSENTEE COMPUTATIONS 79

6.1 Creating an AC 79

6.2 Terminating an AC 80

6.3 Changing the Stream (Priority) of an AC 81

6.4 Converting an IC to an AC 82

6.5 Converting an AC to an IC 82

6.6 Obtaining Status Information for a User's Computations 83

6.7 Requesting Intervention by an IC 84

6.8 Specifying the AC-IC Load Apportionment 84

6.9 Other Commands for Administrative Personnel 85

Chapter 7 - SUMMARY 86

REFERENCES 91

This empty page was substituted for a blank page in the original document.

ILLUSTRATIONS

3.1 Mechanism for Control of Absentee Computations 16

4.1 C.T.S.S. Absentee Queueing Mechanism 25

4.2 Fifo Mechanism with Multiple Running AC's 26

4.3 Queueing Mechanism with Ordered Running AC's 27

4.4 A Computation Stream 29

4.5 Flow of a Computation through a Stream 31

4.6 Multiple Stream Queueing Mechanism 33

5.1 Load Control in a Purely Interactive System 37

5.2 Load Trim-by-Force 39

5.3 Load Trim-by-Attrition 41

5.4 Load Trimming Strategies 42

5.5 Load Control in a Single-Stream Purely Absentee System 44

5.6 Load Control in a Multiple-Stream Purely Absentee System 52

5.7 Load Reapportionment in a Multiple-Stream Purely Absentee System 54

5.8 Queueing Mechanism for a System with Multiple Interactive and Absentee Streams 66

5.9 Summary of Load Control Parameter Definitions for a System with Multiple Interactive and Absentee Streams 67

5.10 Load Control in a System with Multiple Interactive and Absentee Computation Streams 69

5.11 Load Reapportionment in a System with Multiple Interactive and Absentee Computation Streams 73

This empty page was substituted for a blank page in the original document.

CHAPTER 1

Introduction

This thesis presents the detailed design specifications for a mechanism to handle absentee (or background) computations in a multiple-access computer system. The mechanism operates as a package of self-contained modules with a minimum of dependencies upon the environment in which it resides. Thus, it may be inserted into any existing multiple-access computer system which has the proper environmental features.

The work of this thesis is concentrated in several areas. First, those features which are desirable in a system for handling absentee computations are considered. Many of these features exist in current working systems, but several new features are proposed.

Second, the overall design for a new type of absentee mechanism is considered. The functions of each module in the mechanism are discussed, and interfaces between this mechanism and other parts of the multiple-access computer system are defined.

Next, the detailed design is presented for the two major portions of the new absentee handling mechanism. This design is interesting for several reasons:

1 - Absentee computations are supported in a system designed for time-sharing applications.

2 - The amount of absentee usage on the system may be carefully regulated to comprise anywhere from 0% to 100% of the total absentee and interactive system usage.

3 - The portion of system usage assigned to absentee computations may be further subdivided and assigned to absentee computations of various "types". This provides for ease in implementation of priority schemes for determining which computations should currently be serviced by the system.

4 - The apportionment of system usage is made flexible by the absentee handling mechanism to prevent waste of available computing capability.

5 - Computations of a particular "type" are always guaranteed first claim to the portion of system usage assigned to them.

6 - The mechanism may temporarily suspend and then automatically resume an absentee computation thus making such interruption transparent to the computation.

This ability is useful in providing the apportionment flexibility mentioned above.

Finally, an attempt is made to specify a compact set of commands for users and administrative personnel. The commands are designed to provide smooth interaction with the facilities and capabilities of the proposed absentee handling mechanism. In particular, commands are provided to perform certain obvious functions such as creating and terminating absentee computations, and certain functions unique to this application such as converting an interactive computation to absentee and vice versa, and specifying the apportionment of system usage between interactive and absentee computations.

1.1 Terminology

A time-sharing computer system rapidly shares its resources among many users to give each user the illusion that his computation is constantly running. An interactive user controls the operation of his computation by issuing commands (usually in the form of statements typed at a remote teletype terminal) to the system, observing the system's response to each command, and issuing further commands based on previous responses. An absentee user does not have to be

present at a terminal to control his computation; he submits a file of commands which specify the operation of his job. The system enqueues (i.e., maintains an ordered list of) each of these descriptor files as they arrive.

The concept of a computer utility is that of providing accessible computing capability to a large number of users (referred to as the user community) on a twenty-four-hour-per-day, seven-day-per-week basis. It is normal to expect, however, that malfunctions may require that the system undergo an occassional shutdown so that the malfunction may be repaired. To resume normal operation, the system undergoes a startup procedure.

Each user of the system may store information in private files (usually space on secondary storage media reserved for this user). A potential user of filed information must identify himself to the system by typing a secret password thus preventing unauthorized persons from using the system. This identification procedure is referred to as logging in. When a user is finished using the system he logs out to inform the system that he no longer needs its resources.

Unless otherwise specified, files are used to supply input to and receive output from absentee computations since such computations are generally not attached to terminals. However, the user may desire to receive input to his absentee computation from a private magnetic tape in which case his computation may run only if a tape drive is available. The tape drive in this example is referred to as a dedicated resource since it must be specifically assigned to this user for the duration of his computation. To assure that a required dedicated resource is available when it is needed, the user places a reservation (via the system) to use the resource during a specific time period.

Non-time-sharing computers generally handle jobs in a batch processing format; the jobs are submitted in the form of card decks or magnetic tapes at a central computer installation and are processed sequentially, either one at a time, or in the case of recent multi-processing systems, several at a time.

The system load on a time-sharing computer system which services both interactive and absentee computations refers to the current demands for service to all computations on the system. When the load decreases more computations may be initiated to push the load back up to peak efficiency

operating levels. At any instant during the operation of the system, requests for service are enqueued on a priority basis (scheduled), and the request at the head of the queues is the first to be serviced.

Priorities are generally assigned to computations to indicate some sort of preferential ordering for service. Admission priorities are used by the system to determine which of several computations attempting to log in should actually be allowed to log in to the system. Scheduling Priorities are used by the system to determine which of several logged in computations should be given service by a processor when that processor becomes available. Admission priorities are generally fixed whereas scheduling priorities are dynamically computed at execution time.

1.2 Background

The concept of providing absentee usage facilities in a time-sharing environment is not new, but the general design principles have not yet been discussed. Two of the earliest successful attempts in this area are:

1 - The Compatible Time-Sharing System (1,15) implemented on the IBM 7094 by M.I.T. in 1962. Absentee capabilities were included in the original design of C.T.S.S. The ability for an interactive user to initiate an absentee computation was added in 1965-1966.

2 - Time Sharing System/360 (4,5) implemented by IBM on its 360/67. Absentee capabilities were included in the original design for which a prototype implementation became available in 1967. Some notable features of the TSS/360 implementation include the ability for a user to interrupt his running interactive computation and convert it to absentee, and the ability for a user to initiate a wide range of bulk input/output operations via commands which may be issued by interactive or absentee computations. These bulk input/output requests are handled as standard absentee computations and are enqueued until the input/output devices needed to service the requests become available.

The absentee handling mechanism designed in this thesis is being implemented as an integral part of the "Multics" system (Multiplexed Information and Computing Service) under development at Project MAC at the Massachusetts Institute of Technology. Multics is being designed and implemented as a general purpose time-sharing operating system for the computer utility. The reader interested in exploring Multics further should consult a group of papers (5,6,7,8,9,10) which were presented at the Fall Joint Computer Conference in 1965. Project MAC Technical Report-30 (4) contains a discussion of the organization of the computer utility and a description of the basic design of the Multics system.

1.3 Organization

Chapter 2 discusses those features which are desirable in a system which supports absentee computations, and the controls that users and system administrative personnel should have over absentee computations.

Chapter 3 presents a block-diagram overview of the absentee handling mechanism, and discusses the functions

performed by each of the modules in the mechanism.

Chapter 4 develops the concept of a computation stream and then illustrates how several such streams may be combined to form a versatile multiple-stream queueing mechanism for absentee computations.

Chapter 5 considers the problems of regulating the system load in a system which supports both interactive and absentee computations. A mechanism for performing the load control function is proposed which utilizes the flexibility of absentee computations to assure that the load remains close to its most efficient operating level. The mechanism allows the computing capability of the system to be allocated in any proportion between interactive and absentee computations, and provides the ability to quickly and smoothly adjust the system to a new load reapportionment.

Chapter 6 presents a set of commands for users and administrative personnel to create, control, and terminate absentee computations.

CHAPTER 2

Features of Absentee Computations

This chapter defines absentee, interactive, and batch computations, and discusses the similarities between absentee and batch computations, the features of absentee computations, and the facilities of the absentee handling mechanism.

2.1 Computation Modes

A user may run his computation in either of two modes, namely interactive or absentee.

An interactive computation (IC) is controlled by a user who enters commands at a remote terminal, receives responses from the system at that terminal, and enters additional commands based on previous responses. The interactive mode affords the user precise control over his computation and allows the user to make major changes of strategy at run time. The interactive mode is particularly useful for program debugging and for implementing programs which "talk" with non-programmer users (administrators, scientists, flight reservation personnel, etc.)

An absentee computation (AC) does not require interaction with the user. The user submits an absentee computation as a file of commands (absentee source file) basically identical to the commands the user would enter if running the same computation interactively. The absentee mode frees the user from having to be present to control his computation and is particularly useful for running checked-out programs and "production" runs.

A batch computation is basically identical to an absentee computation except for the manner in which the file of commands is submitted to the system. An absentee source file is generally submitted via a remote terminal, whereas a batch computation command file is generally submitted in the form of a card deck at the central computer installation.

2.2 Features of Absentee Computations

An absentee computation may be initiated for a user by one of the user's interactive computations, another of the user's absentee computations, or a batch computation submitted by the user.

A user's absentee computations may be terminated by any of that user's interactive or absentee computations.

A user may specify (for his own protection) any running time limit for each of his AC's. If the time limit is exceeded, the system automatically saves the AC so that partial results are not lost. If the user does not specify a time limit for an AC, then a default value is assumed by the system, again for the user's protection.

A computation (IC or AC) of a particular user may obtain status information about any of that user's computations (both IC's and AC's). Detailed information about each of a user's AC's is always available to that user, regardless of whether his AC's are waiting to be run or are currently running.

An interactive user may interrupt his IC at any point and convert it to an AC. This feature is desirable in the case that a user wishes to start his program interactively to make sure that it is working properly, and then convert the program to absentee so that it may continue to completion without the user's attention.

An interactive user may interrupt any one of his AC's at any point and convert it to his current IC. This feature is useful in the case that the user wants to monitor the progress made by his absentee computation, or perhaps make some run-time changes in either the program or its data. In some cases, the user may convert a computation from absentee to interactive to get a higher priority for the computation so that the computation may be completed sooner.

The system administrative personnel may terminate any AC (or IC) which appears to be a "troublemaker".

The system administrative personnel may specify an apportionment of system resources between AC's and IC's

in order to emphasize a particular mode during certain periods of system operation. In effect, this apportionment partitions the system into two distinct sub-systems, one for running AC's and one for running IC's. The AC-IC apportionment may range anywhere from 0%-100% to 100%-0%.

A user may have many running AC's (and many IC's) at one time, but the number can be administratively limited. In the case that a single user has several IC's at various terminals, each of the IC's has equal control over any of the user's AC's.

Input to an AC is normally taken from the appropriate absentee source file. Output from an AC normally goes to a user-specified absentee output file. The user may alternatively specify dedicated resources (in place of files) for AC input/output.

A user requiring dedicated resources for use by any of his AC's should place an advance reservation for the resources.

2.3 System Features

The system enqueues user requests to initiate new AC's so that these AC's may be initiated in the future at a time which the system feels is opportune.

The system may temporarily suspend service to a number of AC's so that an increased load of IC's may be more effectively serviced. Similarly, the system may "bump" a number of IC's so that an increased load of AC's may be serviced. Bumping an IC

involves saving the IC in its current state and automatically logging out the user. To continue a bumped IC the user must log in again and specify that the bumped IC be resumed. After a user has been bumped, he will often find that he cannot log in again immediately. This occurs because the most frequent reason for bumping a user is to decrease the system load and maintain the load at its lower level.

The system may automatically resume service to suspended AC's when the IC load decreases. (Note the asymmetry here. A suspended AC is always automatically resumed by the system while a bumped IC can only be resumed by the interactive user himself. This property is critical to the process of dynamic load balancing discussed in Chapter 5.)

Jobs may be submitted to the system in a batch-processing format as a deck of cards. Such jobs, after undergoing a procedure to validate the identity of their originators, are handled by the absentee mechanism in the same manner as AC's requested by IC's or other AC's.

The system makes shutdown transparent to AC's by suspending any AC's running at shutdown and automatically resuming the suspended AC's at startup time. Any AC's which are sitting in the queues waiting to be run at shutdown, remain enqueued during shutdown and may be initiated by the system after startup.

The apportionment of resources between AC's and IC's is normally done for various long periods of system operation called shifts. However, the demands made by AC's and IC's upon the system over the short term may vary significantly and frequently. The system caters to short term variations in demand by making modifications to the resource apportionment over short periods called integration periods. The Load Control mechanism (see Chapter 5) compares the current demands upon the system with the current shift's resource apportionment, and makes necessary adjustments in the AC and IC loads to insure that good quality service is provided to all running computations regardless of any short-term surges in demand. For example, if the AC-IC apportionment is 25%-75% and the AC demands decrease to only 20%, then Load Control allows enough new IC's to be initiated to bring the IC usage up to 80%. When the short-term variations are of greater magnitude, Load Control may elect not to match a decrease in one mode of usage with an equal increase in the other. Instead, it may match a 20% decrease with only a 10% increase. This provides a damping effect which helps to prevent the current usage of system resources from varying too significantly from the desired shift resource apportionment. See chapter 5 for a more precise discussion of the load balancing operations performed by Load Control.

The system keeps track of all AC's requiring dedicated resources to be initiated. Whenever a reservation made for a particular AC becomes due, the system automatically initiates that AC.

In a system in which many computations are simultaneously competing for service, it is desirable to provide some sort of priority mechanism to allow a user to express the relative importance of his computation. The user is provided with a choice of priority streams in which he may request his AC be run. High priority streams provide service at higher cost, while low priority streams may provide slower service but at reduced cost. If the user does not specify in which priority stream he wants his computation to run, the system automatically inserts the computation into the standard stream. Since absentee computations do not necessarily have to be initiated immediately, a series of waiting queues is provided, one for each priority stream. When Load Control decides that more absentee computations may run in a particular stream, it informs the mechanism which handles the absentee queues to initiate an appropriate number of absentee computations from the waiting queue for that stream.

CHAPTER 3

Overview of the Absentee Monitor

The Absentee Monitor consists of a group of related modules and a series of queues which together are responsible for the enqueueing, initiation, control, and termination of absentee computations. This chapter discusses the major sections of the Absentee Monitor and the interrelationship between the Absentee Monitor and other parts of the multiple-access computer system. Figure 3.1 illustrates the structure of the overall mechanism for handling absentee computations. Chapters 4, 5, 6, and 7 describe the parts of the mechanism in greater detail.

3.1 Major Sections of the Absentee Monitor

The Absentee Monitor consists of three modules and two sets of queues, one queue in each set being assigned to service each priority stream. Chapter 4 discusses the queueing mechanism in detail.

3.1.1 Absentee Queue Control

As new requests to initiate absentee computations are entered by users into the system, the names of the corresponding

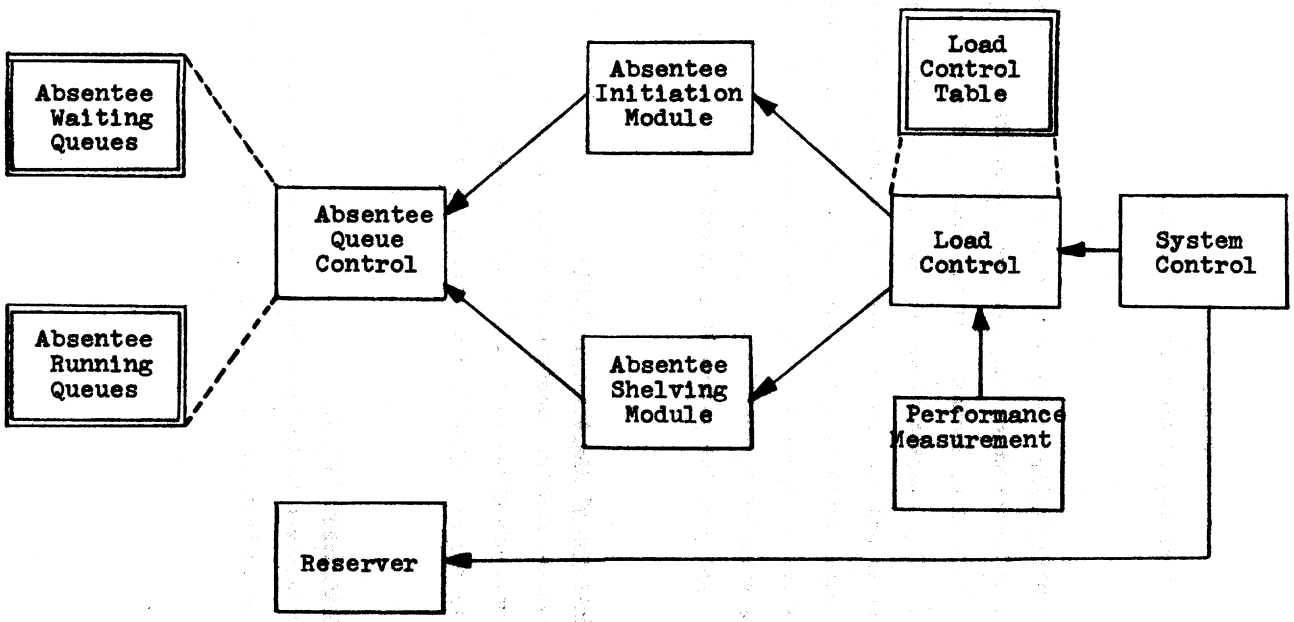


Figure 3.1
Mechanism for Control of
Absentee Computations

absentee source files and absentee output files (and other additional useful information) are placed into queues where they remain until such time as the system decides to initiate additional absentee computations. Absentee Queue Control is the system module responsible for making appropriate entries in the queues, retrieving entries when they are needed by other parts of the absentee mechanism, and deleting entries which are no longer needed.

3.1.2 Absentee Waiting Queues

The Absentee Waiting Queues are a series of queues, one per priority stream, which contain the per computation information for absentee computations waiting to be initiated. A user request to initiate a new absentee computation causes Absentee Queue Control to make an entry for this computation in the appropriate Absentee Waiting Queue.

3.1.3 Absentee Running Queues

The Absentee Running Queues are a series of queues, one per priority stream, which contain the per computation information for each running absentee computation. When an absentee computation is initiated, Absentee Queue Control deletes the corresponding entry from the appropriate Absentee Waiting Queue, and inserts an entry in the appropriate Absentee Running Queue.

3.1.4 Absentee Initiation Module

Whenever more absentee computations may be initiated, Load Control (see Chapter 5 and section 3.2.3 below) informs the Absentee Initiation Module of the number of AC's to initiate. The Absentee Initiation Module decides which AC's to initiate and makes the appropriate calls to initiate them. The Absentee Initiation Module calls Absentee Queue Control to make the appropriate insertions (deletions) in the Absentee Running (Waiting) Queues.

3.1.5 Absentee Shelving Module

Hereafter, the short-term suspension of absentee computations referred to in Chapter 2 is termed "shelving" the absentee computations. The distinction between a shelved computation and a computation which has merely been saved is that a shelved computation is only saved temporarily until such time as the system decides to continue it, whereas a saved computation can only be continued if the user logs in interactively and orders such action. "Unshelving" is the process of resuming a shelved computation and is performed automatically by the Absentee Monitor.

Whenever it becomes necessary to decrease the number of running AC's, Load Control informs the Absentee Shelving Module of how many AC's to shelve. The Absentee Shelving Module decides

which AC's to shelve and makes the necessary calls to shelve them. Absentee Queue Control is called to make the appropriate insertions (deletions) in the Absentee Waiting (Running) Queues. Note that shelving an AC involves placing an entry for it back into the appropriate Absentee Waiting Queue so that the AC again becomes a candidate for initiation by the Absentee Initiation Module.

3.2 Relationships between the Absentee Monitor and other Parts of the Multiple-Access Computer System

This section discusses the environment of the absentee mechanism. Load Control is discussed here instead of in section 3.1 only because its functions are related to both IC's and AC's. However, Load Control is a critical portion of the absentee mechanism and is discussed in detail in Chapter 5.

3.2.1 System Control

System Control is the module which processes command requests from the system administrative personnel. In particular, System Control conveys the AC-IC apportionment information from the System Administrator to Load Control, and the dedicated resource apportionment information to the Reserver.

3.2.2 Performance Measurement

Periodic determination of the current amounts of interactive

and absentee usage being supported by the system is essential to the operation of the load-balancing mechanism. Performance Measurement obtains these usage statistics by observing various system parameters and conveys the information to Load Control.

3.2.3 Load Control

The main functions of the Load Control module are:

1 - to see that the apportionment of resources between interactive and absentee computations remains close to that specified by the System Administrator

2 - to cater to short-term variations in the demands upon the system's resources by dynamically varying the current resource apportionment

3 - to see that the system is neither under- nor over-loaded.

Load Control compares the apportionment information it receives from the System Administrator with the current usage statistics supplied by Performance Measurement. If any significant discrepancies exist between these sets of figures, then Load Control may modify the system load by any of the following means:

1 - call the Absentee Initiation Module to initiate more AC's

2 - call the Absentee Shelving Module to shelve some AC's

3 - increase the maximum number of IC's allowed

4 - decrease the maximum number of IC's allowed

5 - automatically log out some IC's

3.2.4 Load Control Table

In this table, Load Control maintains a list of all logged-in IC's and certain additional information such as the total number of logged-in IC's and the maximum number of IC's which the system currently allows. During the load-balancing operations, Load Control obtains information about AC's from the Absentee Waiting Queues and the Absentee Running Queues and about IC's from the Load Control Table.

3.2.5 Reserver

The Reserver is responsible for scheduling the usage of dedicated resources (for both IC's and AC's). If a user's AC requires dedicated resources, the user must place an advance reservation for the resources. The system automatically initiates the AC when the reservation becomes due.

3.2.6 SAVE, RESUME, and QUIT

These mechanisms are provided in the multiple-access computer system to facilitate certain manipulations of computations useful to both AC's and IC's. QUIT is called to stop the execution of a computation and place the computation into a state in which it may be easily preserved, lost, or continued. SAVE is used to preserve the computation in its current state so that the computation may be continued in the future. RESUME is used to continue a SAVEd computation.

Shelving an AC involves first QUITting the AC, then SAVEing it, deleting the Absentee Running Queue entry for it, creating an absentee source file containing a RESUME command, and placing an entry for this file in the appropriate Absentee Waiting Queue. Unshelving an AC is done by the Absentee Initiation Module in the same fashion as initiating a new AC. However, since the absentee source file for this AC consists of merely a RESUME command to resume a SAVEd file, the saved AC is restarted.

3.2.7 User Commands

Users are provided with a detailed set of commands with which to control the initiation, operation, and termination of absentee computations (see Chapter 5). These commands cause calls to entries in the Absentee Monitor and status information is returned to the user in each case to indicate if the calls are successful.

CHAPTER 4

Features of the Queueing Mechanism

The Queueing Mechanism consists of the Absentee Waiting Queues, Absentee Running Queues, and the Absentee Queue Control module. This chapter describes the structures of the various queues and the operation of Absentee Queue Control.

4.1 Necessity for a Queueing Mechanism

During the operation of the system, user requests to initiate new AC's may arrive faster than the new AC's can be initiated. One reason for this is that the maximum number of AC's which may run simultaneously may be limited, thus necessitating the placement of waiting requests into a waiting line or queue. Actually, a queueing mechanism can be avoided if it is felt by the system designers that if there currently is no room for more AC's, then the user should retry his request at a later time. This, in effect, is the method used in the case of new interactive users. If the new IC cannot be initiated the user must wait until a later time and then reattempt to log in. The reason for this choice is obvious. Suppose, for example, that an interactive user tries to log in and that the system cannot handle any more interactive users now. Suppose that the system

then proceeds to enqueue this user's request in a waiting line with other requests for IC's. Clearly, there is no way to tell how long it will be before the new request may be serviced. Thus, the user might sit faithfully at his console for several minutes or perhaps several hours before his request may be serviced. From the human factors standpoint which is so critical in the design considerations for a multiple-access computer system, such an occurrence is not tolerable.

However, since a user does not have to be present to run his absentee computation, it is clear that requests for AC's may be enqueued for future initiation without any inconvenience for the user. As a matter of fact, in this way the user is assured that his AC will be initiated at the earliest possible time. As will become clear in Chapter 5, this feature of absentee computations is most critical to the design and operation of the load-balancing mechanism.

4.2 Queue Discipline

The order in which requests to initiate new AC's are serviced need not necessarily be the same as the order in which these requests arrive. The method of choosing the next AC to be initiated from the queues is referred to as the queue discipline. Among the more common queue disciplines are first-in-first-out (fifo) which selects entries on a first-come-first-served basis, and last-in-first-out (lifo) which selects entries on a most-current-first-served basis.

The queue discipline chosen for the Absentee Waiting Queues and Absentee Running Queues utilizes both fifo and lifo disciplines in a slightly modified fashion.

4.3 First-In-First-Out Disciplines

The C.T.S.S. system uses a fifo discipline in which at most one absentee computation may run at a time. Once an AC has begun to run it must be run to completion or until it is automatically logged out. No provision is made to allow the running computation to be temporarily suspended and then resumed. Absentee usage is never too significant a portion of total system usage since there is usually a single running absentee computation and as many as 30 running interactive computations. Hence, the suspension of the single absentee computation is not really a versatile tool in terms of allowing more interactive usage to occur. Figure 4.1 illustrates the C.T.S.S. queueing mechanism for absentee computations.

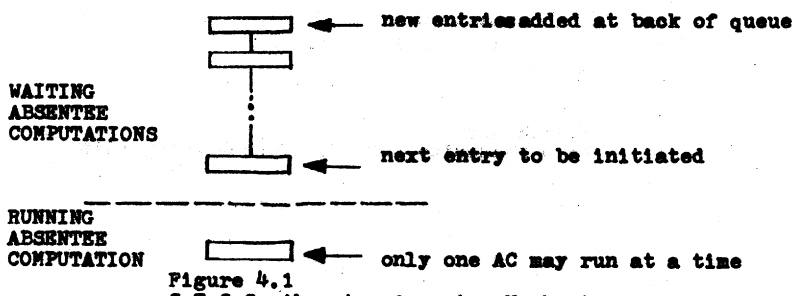


Figure 4.1
C.T.S.S. Absentee Queueing Mechanism

An obvious extension to the C.T.S.S. queueing mechanism is to allow many AC's to run at one time. This introduces some interesting load considerations since it might result in a system with a poor interactive response if the number of running AC's becomes large. This problem is discussed in detail in Chapter 5. Figure 4.2 illustrates a queueing mechanism which allows for many running AC's at one time.

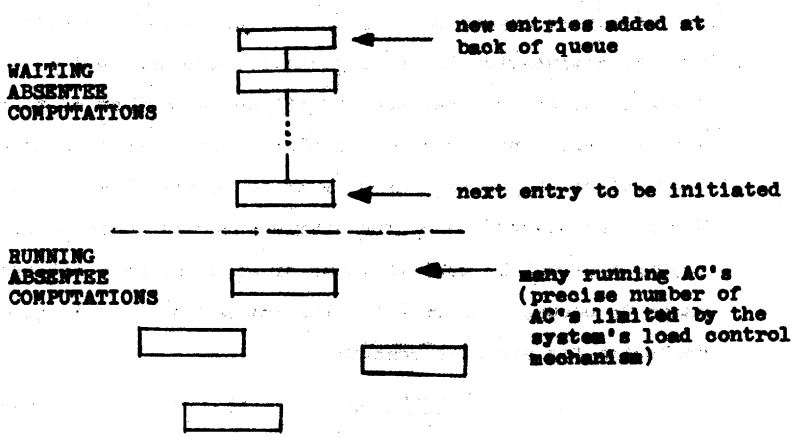


Figure 4.2
Fifo Mechanism with Multiple Running AC's

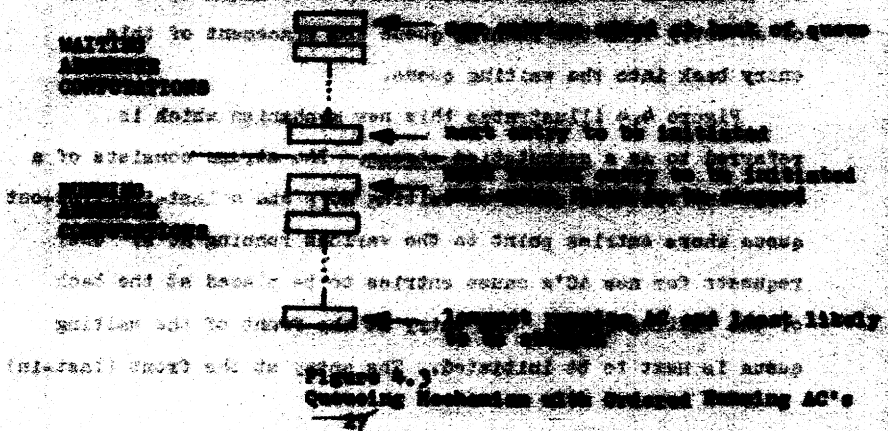


Figure 4.3
Queuing Mechanism with Ordered Running AC's

Note that the structure of the mechanism of Figure 4.3 allows arbitrary criteria to be used in deciding the ordering of the entries in the running queue. The lifo discipline is particularly useful for the extension made to this structure in the next section.

4.5 Computation Streams

Now let us consider the idea of shelving an absentee computation. Since a user is generally not present to control his absentee computation, the user does not suffer any inconvenience if his computation is temporarily suspended and automatically resumed. As has been mentioned previously, this property of AC's facilitates the design of the load balancing mechanism presented in Chapter 5. In this section, the mechanism of Figure 4.3 is extended to allow the removal of an entry from the running queue and placement of this entry back into the waiting queue.

Figure 4.4 illustrates this new mechanism which is referred to as a computation stream. The stream consists of a first-in-first-out queue of waiting AC's and a last-in-first-out queue whose entries point to the various running AC's. User requests for new AC's cause entries to be placed at the back of the waiting queue. The entry at the front of the waiting queue is next to be initiated. The entry at the front (last-in)

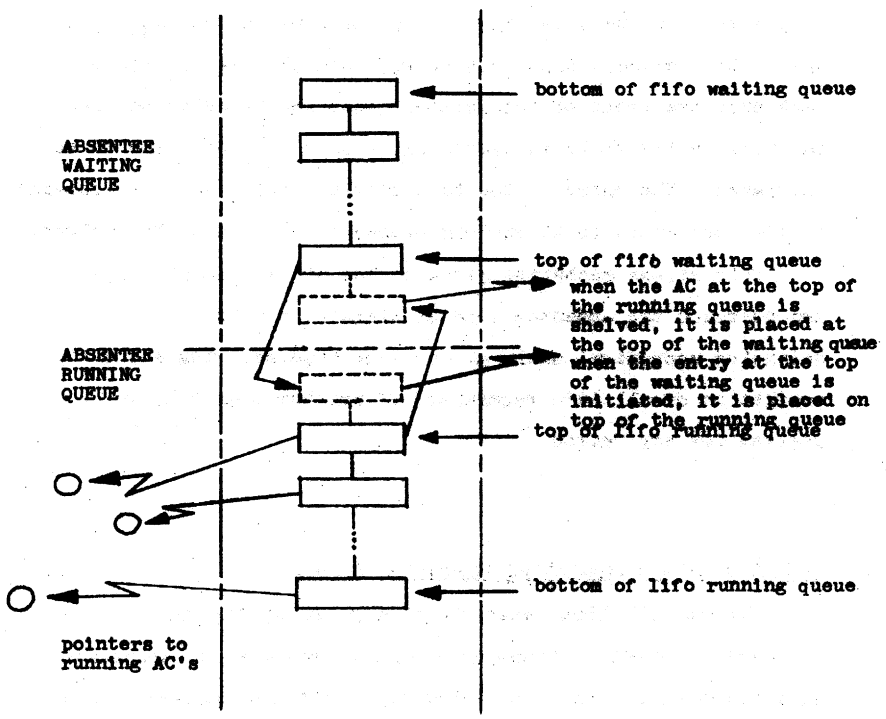


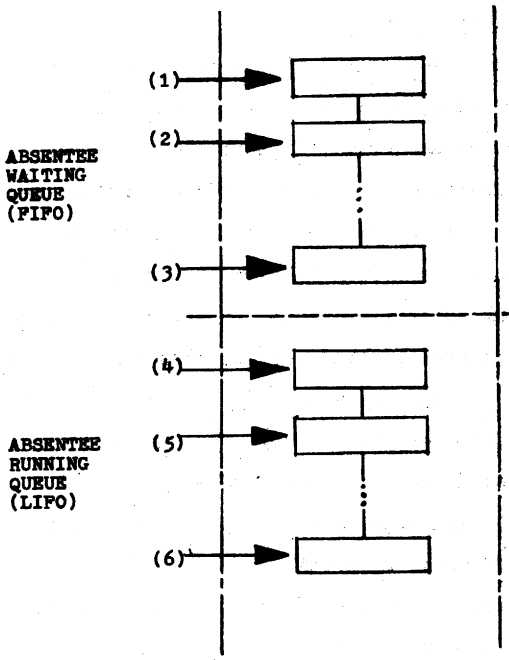
Figure 4.4
A Computation Stream

of the running queue is the first to be shelved when any AC's are to be shelved. Also this entry, when shelved, is placed back onto the front of the waiting queue and hence it becomes the first entry to be initiated again when more AC's are to be initiated. The entry at the back of the running queue (first-in) is the last entry to be shelved whenever AC's are to be shelved.

This stream mechanism gives us the ability to increase and decrease the absentee load as dictated by the load balancing mechanism, while at the same time assuring automatic completion of all AC's regardless of whether they are ever shelved for any reason.

4.6. Flow of a Computation through a Stream

Figure 4.5 illustrates the flow of an AC through a computation stream. Since an AC may be shelved and unshelved many times as it runs, the entry for the AC may pass back and forth through the waiting and running queues until it eventually reaches completion while residing in the running queue. Note that it is possible for an AC to leave the stream while it is in the waiting queue. This happens, for example, if the user decides to terminate the AC. Chapter 6 discusses user control of AC's in detail.



WAITING QUEUE FLOW

(1) AC enters the stream

(1)→(2) AC moves toward the
(2)→(3) front of the waiting queue as others AC's before it are initiated

(3)→(2) AC moves toward the
(2)→(1) rear of the waiting queue as some running AC's are shelved

FLOW BETWEEN QUEUES

(3)→(4) AC is initiated (or unshelved)

(4)→(3) AC is shelved

RUNNING QUEUE FLOW

(4)→(5) AC moves towards the
(5)→(6) rear of the running queue as other AC's initiated before it are completed

(6)→(5) AC moves towards the
(5)→(4) front of the running queue as other AC's initiated after it are shelved

(4) An AC may be completed
(5) regardless of its
(6) position in the running queue. The AC's entry is deleted and all other AC entries maintain their same relative positions in the running queue.

Figure 4.5
Flow of a Computation through a Stream

4.7 Multiple Stream Queueing Mechanism

It is apparent from the discussion of section 4.5 that a stream-type queueing mechanism coupled with a load control mechanism which orders the shelving and unshelving of AC's is a useful means of controlling the amount of absentee usage supported by the system. An even greater degree of control over the absentee usage is made available by utilizing a multiple stream queueing mechanism. Thus, it may be advantageous to differentiate between various types of absentee computations (such as might be done in the implementation of a priority scheme) and such a differentiation could be made by associating AC's of each type with a distinct stream. Then, the load control mechanism could control the usage in each stream individually. These operations are described in detail in Chapter 5.

Figure 4.6 illustrates a mutiple stream queueing mechanism. Note that no reference has been made so far to a means of ordering interactive computations or differentiating between various types of IC's. This has been so because our primary concern has been considerations related to absentee computations. The discussion of Chapter 5 includes several such considerations of IC's.

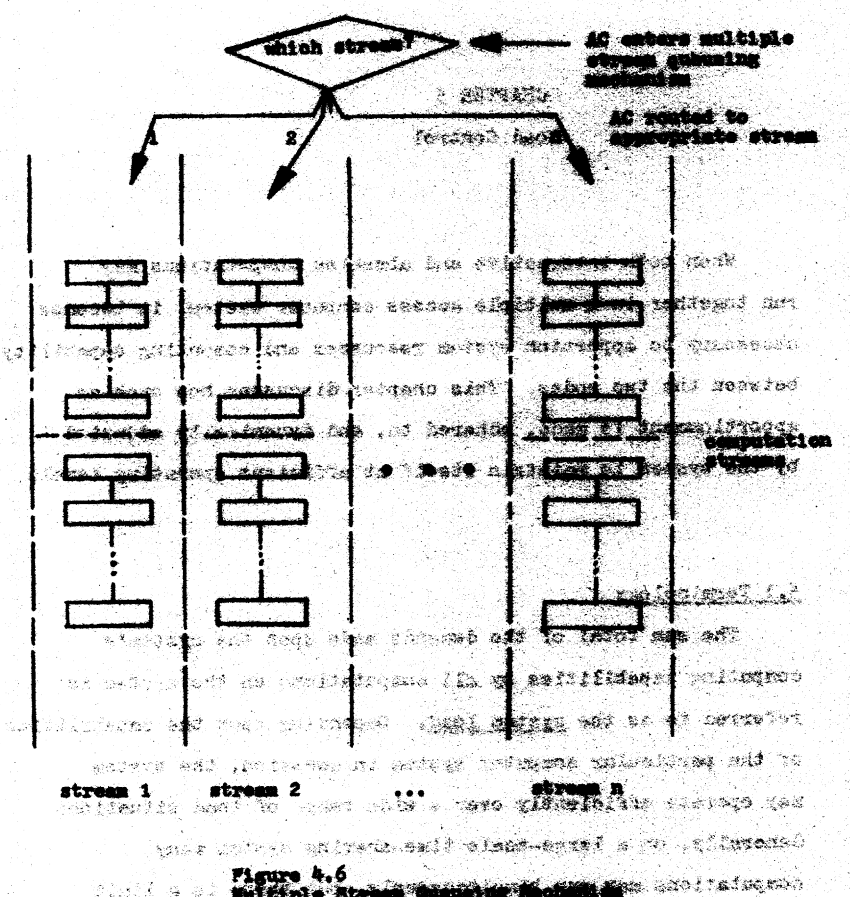


Figure 4.6
Multiple Stream Queuing Mechanism

CHAPTER 5
Load Control

When both interactive and absentee computations may run together in a multiple access computer system, it becomes necessary to apportion system resources and computing capability between the two modes. This chapter discusses how such an apportionment is made, adhered to, and dynamically adjusted by the system to maintain itself at efficient operating levels.

5.1 Terminology

The sum total of the demands made upon the system's computing capabilities by all computations on the system is referred to as the system load. Depending upon the capabilities of the particular computer system in question, the system may operate efficiently over a wide range of load situations. Generally, on a large-scale time-sharing system many computations may run simultaneously, but there is a limit

to the number of computations which the system can support without becoming over loaded.

As a measure of system load we use the number of running computations. The stream load in a computation stream is the number of running computations in that stream. The stream backup in a computation stream is the number of computations in the waiting queue for that stream. Stream backup is a measure of potential stream load. The system's load configuration is a summary of the stream load and stream backup for each of the system's computation streams.

For any particular multiple access computer system, the most efficient load configuration (i.e., the load configuration which results in the most useful computation) is difficult to predict while the system is under development. After the system becomes operational, however, efficient load configurations readily become apparent. A multiple access computer system is said to be properly-loaded if it is operating near its most efficient load configuration, over-loaded if there is less useful computation being performed than when the system is in its most efficient load configuration, and under-loaded if the addition of more computations would result in an increased amount of useful computation. These terms may also be used to describe the load in a computation stream. In particular, a stream which normally services seven computations is over-loaded

if nine computations are currently running in that stream, under-loaded with five running computations, and properly-loaded with seven running computations.

The load control problem is to maintain the system in a properly-loaded state. If a system is operating in an under-loaded state, the load control problem is to increase the number of running computations. If a system is operating in an over-loaded state, the load control problem is to decrease the number of running computations. If a system is operating in a properly-loaded state, the load control problem is to maintain this state.

5.2 Load Control in a Purely Interactive System

Consider the load control problem in a system dedicated to servicing only interactive computations. The system load, L, is equal to the number of running IC's. The system resources are available to the various IC's. It is only necessary to limit the number of IC's to some maximum, N_1, to prevent the system from becoming over-loaded (N_1 IC's therefore corresponds to a properly-loaded state). The System Administrator specifies an initial value of N_1 at startup time and may alter N_1 at any time during system operation.

Figure 5.1 shows a simple load control mechanism for a

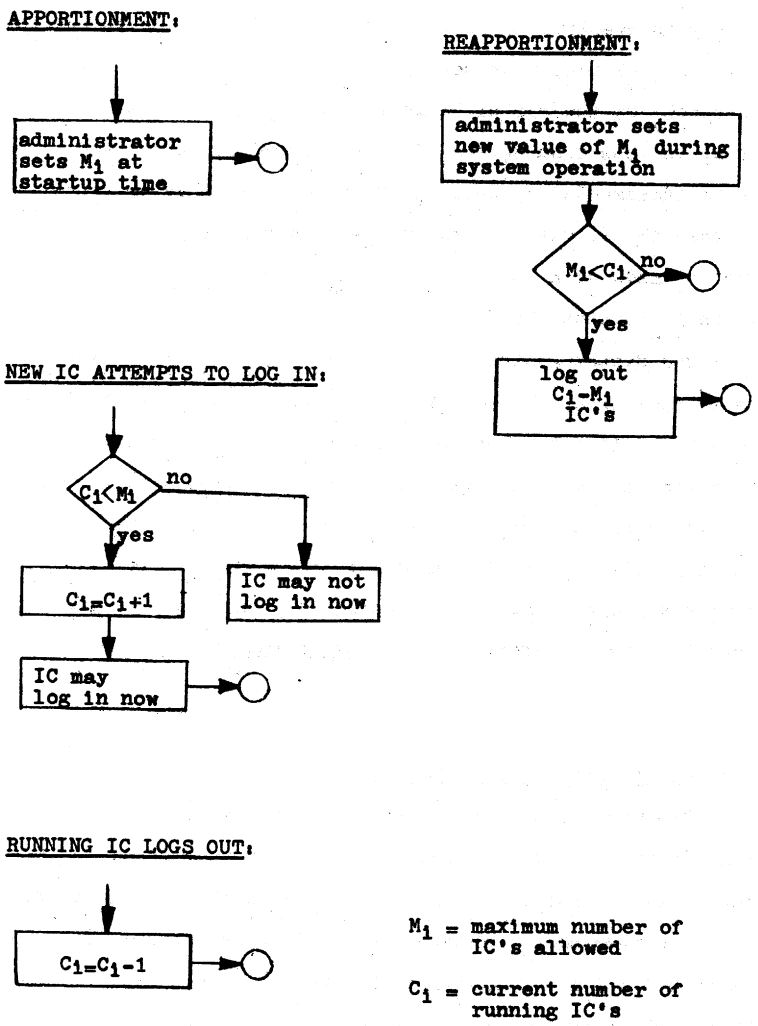


Figure 5.1 Load Control in a Purely Interactive System

purely interactive system. When a new IC attempts to log in Load Control checks to see if the number of IC's, C_i, currently on the system is less than the allowed maximum. If C_i is less than M_i, the IC is allowed to log in and C_i is incremented by one. If C_i is greater than or equal to M_i, then the IC may not log in now; a new attempt to log in must be made at a later time. When a running IC logs out C_i is decremented by one.

The System Administrator may reset M_i while the system is in operation. Load Control checks to see if M_i is less than C_i. If not, then no adjustments in the current IC load are needed. However, if M_i is less than C_i the system automatically assumes an over-loaded state and $C_i - M_i$ running IC's must be logged out to bring the IC load down to a properly-loaded state. Decreasing the number of running computations is referred to as load trimming.

5.3 Load Trimming Strategies

The most direct way to trim the IC load is to immediately log out the necessary number of IC's. (Note that it is therefore desirable to associate some ordering with the IC's in order to have a criterion for choosing which IC's to log out first. See section 5.7 for a discussion of interactive computation streams.) Such a strategy is referred to as a trim-by-force and is illustrated in Figure 5.2.

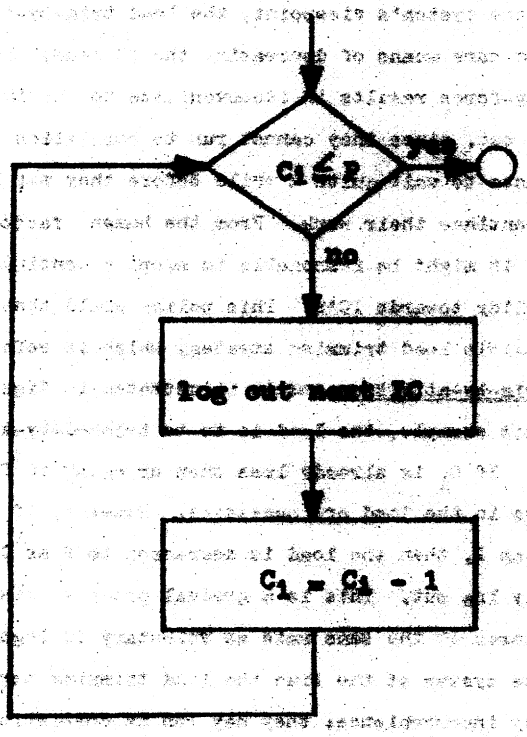


Figure 5.2 Load Trim-by-Force

In this example, the load is to be trimmed to P IC's. If C_i is less than or equal to P then no adjustments in the load are needed. However, if C_i is greater than P, then IC's are automatically logged out one-by-one until C_i equals P.

From the system's viewpoint, the load trim-by-force is a quick and sure means of decreasing the IC load. However, the trim-by-force results in inconvenience to the IC's which are logged out, since they cannot run to completion now, and they may have to wait quite a while before they may log in again to continue their work. From the human factors standpoint it might be reasonable to adopt a continuous service policy towards IC's. This policy would then require a more flexible load trimming strategy which is referred to as load trim-by-attrition and is illustrated in Figure 5.3.

In this example, the load is to be trimmed-by-attrition to P IC's. If C_i is already less than or equal to P, then no adjustments in the load are necessary. However, if C_i is greater than P, then the load is decreased to P as C_i-P IC's voluntarily log out. This is a gradual process, since the load decreases at the same rate as voluntary IC logouts. The IC's on the system at the time the load trimming begins do not undergo any inconvenience; they may run to completion and log out when they are done. However, once an IC logs out, it may not log in again until C_i becomes less than P. The

LOAD TRIM-BY-ATTRITION TO P IC's:

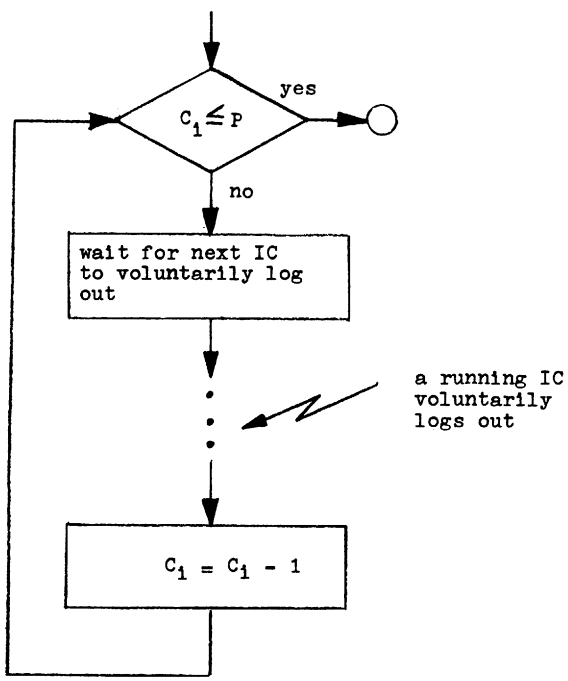
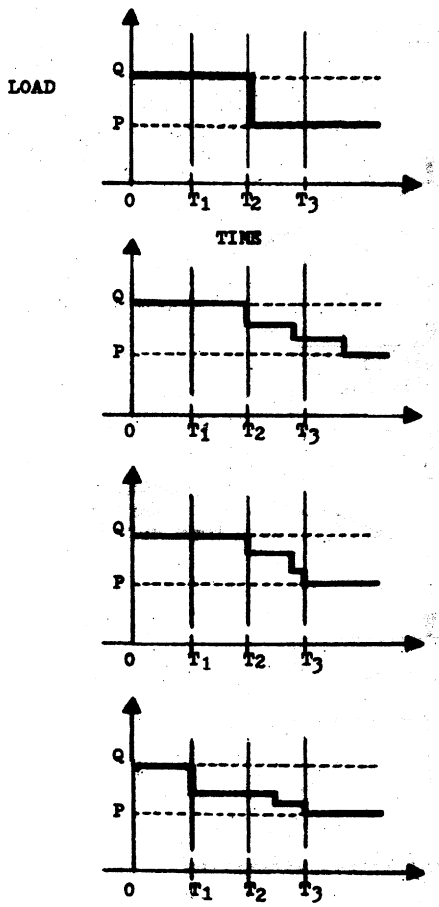


Figure 5.3 Load Trim-by-Attrition



1 - Trim-by-Force at time T_2 which automatically logs out enough computations at T_2 to bring the load down to P.

2 - Trim-by-Attrition starting at time T_2 and finishing whenever enough computations have voluntarily logged out to bring the load down to P.

3 - Trim-by-Attrition starting at time T_2 with the restriction that if enough computations have not voluntarily logged out by T_3 a trim-by-force will be applied at T_3 to bring the load down to P.

4 - Trim-by-Attrition starting earlier than time T_2 so that there is a better chance for the load to be down to P by T_3. A trim-by-force is applied at T_3 if more than P computations remain to bring the load down to P.

Figure 5.4 Load Trimming Strategies

trim-by-attrition strategy does have one significant drawback. If the IC's on the system do not log out in a short time, they can remain on the system as long as they would like to and thus prevent the load from being trimmed. In such a case it might be reasonable to impose some time limit on the voluntary logouts, and if the load has not been trimmed to P IC's by that time, then a trim-by-force could be used to complete the load trimming. If it is required that the IC load be trimmed to P by a certain time, then the load trimming could be initiated in advance of this time. Figure 5.4 illustrates the various load trimming strategies discussed in this section.

5.4 Load Control in a Single-Stream Purely Absentee System

Now consider the load control problem in a system dedicated to servicing only absentee computations. The system load, L, is equal to the number of running AC's. The System Administrator specifies the maximum number of running AC's, M_a, at startup time and may alter M_a at any time during system operation.

Figure 5.5 shows a load control mechanism for a purely absentee system utilizing the computation stream concept discussed in Chapter 4. When a user enters a request for an AC, Load Control checks to see if the number of running AC's, C_{ar},

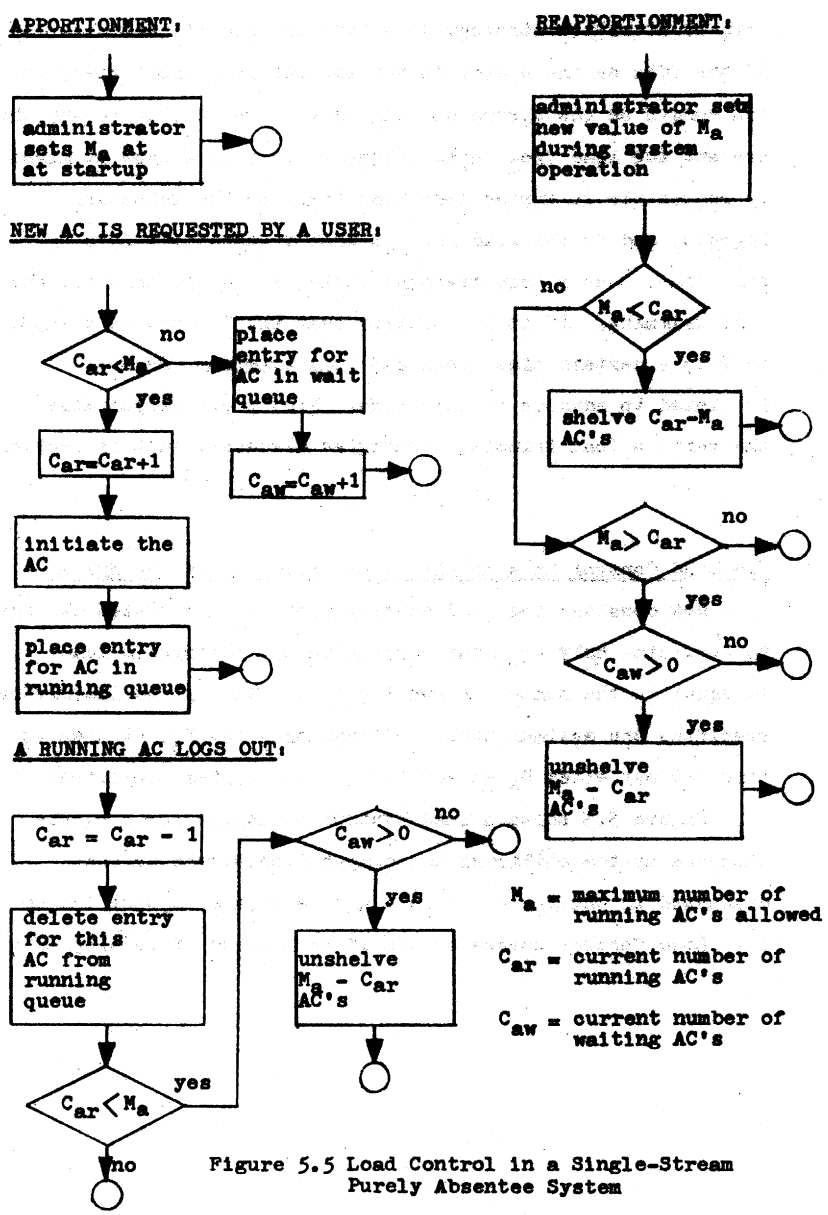


Figure 5.5 Load Control in a Single-Stream Purely Absentee System

is less than the allowed maximum. If C_{ar} is not less than M_a then the AC cannot be initiated now and must wait in the computation stream's waiting queue. The number of AC's waiting to be initiated, C_{aw}, is incremented by one. If C_{ar} is less than M_a then the AC can be initiated immediately. C_{ar} is incremented by one and an entry for this AC is placed into the computation stream's running queue.

When a running AC logs out C_{ar} is decremented by one and the running queue entry for the AC is deleted. If this causes C_{ar} to fall below M_a then more AC's may be initiated. Load Control checks to see if any AC's are waiting to be initiated and if so initiates enough AC's to bring the AC load back up to M_a.

The System Administrator may increase or decrease M_a while the system is in operation. If M_a is decreased so that it becomes less than C_{ar} then Load Control orders $C_{ar}-M_a$ AC's shelved. If M_a is increased to a value greater than C_{ar} then Load Control orders M_a-C_{ar} AC's unshelved. Note that if the AC load has to be decreased (i.e., M_a is set to a value less than C_{ar}) then it is reasonable to immediately shelve the necessary number of AC's. The method of load trim-by-attrition need not be used since the shelved AC's do not undergo any inconvenience in the sense that a bumped IC does. Thus the AC load is only trimmed by a load trim-by-force.

5.5 Load Control in a Multiple-Stream Purely Absentee System

Now let us extend the discussion of Section 5.3 to include the load control problem in a multiple-stream purely absentee system. In particular, consider a system comprised of n absentee streams. The following definitions will be useful in the discussion:

$M_{a(i)}$ = maximum number of AC's which may run in stream i at one time

$C_{ar(i)}$ = current number of AC's running in stream i

$C_{aw(i)}$ = current number of AC's waiting to be run in stream i

$M = \sum_{i=1}^{n} M_{a(i)}$ = maximum number of AC's which may run on the entire system at one time

$R = \sum_{i=1}^{n} C_{ar(i)}$ = current number of AC's running on the entire system

$W = \sum_{i=1}^{n} C_{aw(i)}$ = current number of AC's waiting to be run on the system

The System Administrator apportions resources by specifying $M_{a(i)}$ for each stream. The sum of these, M, therefore represents what the Administrator considers to be the maximum number of AC's which may run at one time and still keep the system operating in a properly-loaded state. In effect the Administrator specifies that M slots are available for

use by AC's. At any time a particular slot is either empty or in use. Since M running AC's represents a properly-loaded system, the load control problem is to maintain the system so that either M or fewer slots are in use; if more than M AC's are running Load Control must trim the load to M. If less than M AC's are running Load Control must check to see if any AC's are waiting and if so initiate enough AC's to get the load back up to M. The functions to be performed seem clear, but the fact that more than one stream is involved introduces some complications. For example, it is possible that one stream could be properly-loaded and all the other streams could be empty. If there are AC's waiting to be run in the properly-loaded stream, they must wait until running AC's in that stream log out before they may be initiated. This is obviously a waste of system resources since the "computing power" is available to handle more AC's and it is not being used. One solution to this problem might be to allow the waiting AC's to run in other streams. This is satisfactory until such time as new AC's arrive and request to be initiated into streams which might be full of AC's from other streams. Should these new AC's also be placed into streams in which they too do not belong? In this section a strategy

is developed to prevent this sort of chaos while at the same time assuring that system computing power does not go to waste if there are waiting AC's which could use that power.

Recalling some definitions given in Section 5.1 a particular stream is properly-loaded if $C_{ar(i)}=M_{a(i)}$, under-loaded if $C_{ar(i)}<M_{a(i)}$, and over-loaded if $C_{ar(i)}>M_{a(i)}$. The load in a particular stream is said to be <u>balanced</u> if the stream is properly-loaded, or if the stream is under-loaded and no AC's are waiting to be run in that stream.

One solution to the load control problem in a multiple-stream system might be to balance the load in each stream independently of any considerations involving the other streams. This would result in a system in which

$$C_{ar(i)} \leq M_{a(i)} \quad \text{for } i = 1, 2, \ldots, n$$

and hence it would always be true that

$$R = \sum_{i=1}^{n} C_{ar(i)} \leq M$$

The ideal situation is $R = M$. However, the above <u>component-wise-balanced</u> system may have $R < M$ even while there are some waiting AC's (i.e., the situation presented in the beginning of this section).

To see how the situation in which $R<M$ and $W\neq 0$ can be

avoided, let us trace the buildup of the load on a multiple-stream system starting with no running or waiting AC's. As requests to initiate new AC's arrive, Load Control observes that the current load in each stream is less than the allowed maximum and therefore allows the AC's to be initiated. After a while, however, one of the streams will eventually become properly-loaded. Suppose that another AC requests to be placed into the properly-loaded stream. Load Control checks the load in this stream and discovers that the stream is full. Load Control can then check the loads in the other streams to see if there are any available slots. If there is an available slot, Load Control allows this new AC to be initiated. This, of course, causes a properly-loaded stream to become over-loaded. However, the system as a whole is not over-loaded and therefore by initiating this new AC into an already properly-loaded stream we are preventing usable resources from going to waste.

Load Control may continue to allow new AC's to be initiated until R becomes equal to M. Once this occurs the addition of another running AC would cause a genuine over-loaded situation. Now Load Control must first check to see if the new AC wishes to be initiated into a stream which is either properly-loaded or over-loaded. If this is the case then the

new AC is placed into the waiting queue for that stream to avoid over-loading the system. If, however, the stream into which this AC wishes to be placed is under-loaded, then some other stream must be over-loaded causing all slots to be in use. The only reason that this other stream was allowed to use more than its maximum number of slots was to prevent available resources from being wasted. Now, however, there is legitimate demand for these resources and they should be given back to the AC which is specifically requesting them. This poses no real problem to Load Control. It is merely necessary to shelve one AC from the over-loaded stream so that the needed slot becomes available in which to initiate the new AC. This procedure is followed as additional AC's request to be initiated until finally the system reaches a load configuration in which every stream is properly-loaded and there may or may not be AC's waiting to be initiated in any of the streams. This situation is referred to as the ideal load configuration for obvious reasons.

Now consider the load control functions which must be performed when a running AC logs out. Since we know that R was always kept less than or equal to M as the load was building up, it must be true that a slot is made available (i.e., it can not be the case that the system went from

one over-loaded state to a less over-loaded state). Load Control must now decide which waiting AC (if there is any) to initiate into the newly available slot. First preference for the slot goes of course to any AC's waiting in that particular stream. Note that if the scheme proposed in this section for initiating new AC's is used, then it is not possible for an AC to be waiting in a stream which is under-loaded. Therefore if no AC is waiting in the stream in which an AC just logged out, then Load Control must initiate one of the AC's waiting in any of the properly-loaded or over-loaded streams.

Thus we have arrived at a scheme for initiating and shelving AC's which assures that available slots never go to waste if there is demand for them, while at the same time we have developed a mechanism which guarantees an AC first priority in claiming slots allocated to the stream in which that AC wishes to run. Figure 5.6 illustrates the load control operations discussed in this section. The reapportionment function is treated separately in the next section because of its complexity in a multiple-stream system.

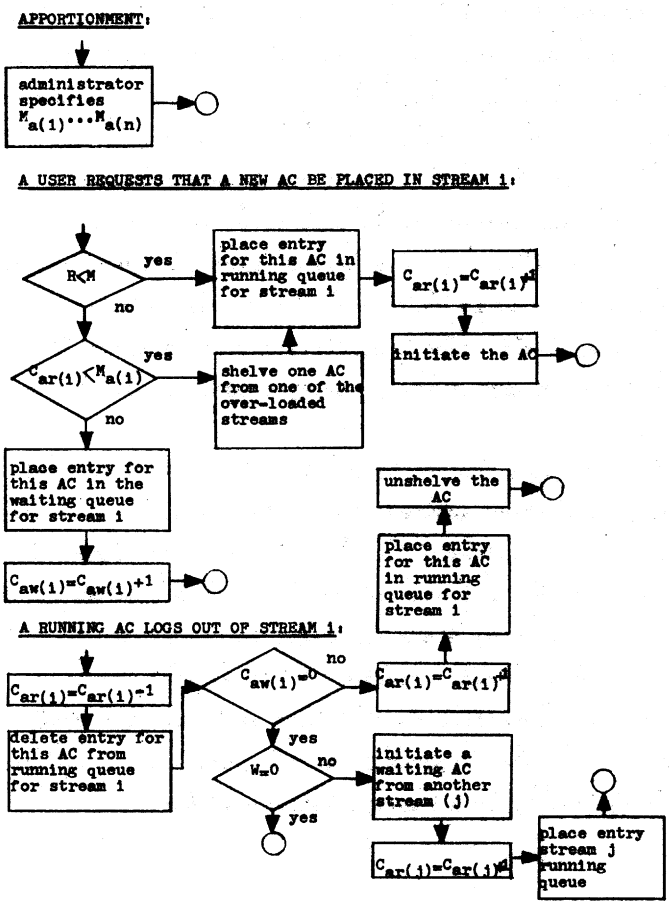


Figure 5.6 Load Control in a Multiple-Stream Purely Absentee System

5.6 Load Reapportionment in a Multiple-Stream Purely Absentee System

In Section 5.5 requests for new AC's and logouts of running AC's were handled in an orderly fashion to prevent the overall system from becoming overloaded. The scheme presented never requires the system to become temporarily overloaded while in the process of adjusting to its new load configuration. When the system load is reapportioned by the System Administrator, however, each stream as well as the entire system might become overloaded and thus it is necessary to provide a mechanism for quickly adjusting the overall system to a properly-loaded state. Besides the load trimming which may be necessary, it is also possible that the load balance within the streams may become severely distorted by a reapportionment. In this section a strategy is presented for quickly and smoothly readjusting the system load configuration after a reapportionment. The load control operations necessary for reapportionment are illustrated in Figure 5.7.

To effect the reapportionment, the System Administrator specifies new values of $M_{a(1)}$ through $M_{a(n)}$. Load Control first checks to see if the overall system has assumed an overloaded state ($R > M$). If it has then R-M AC's are

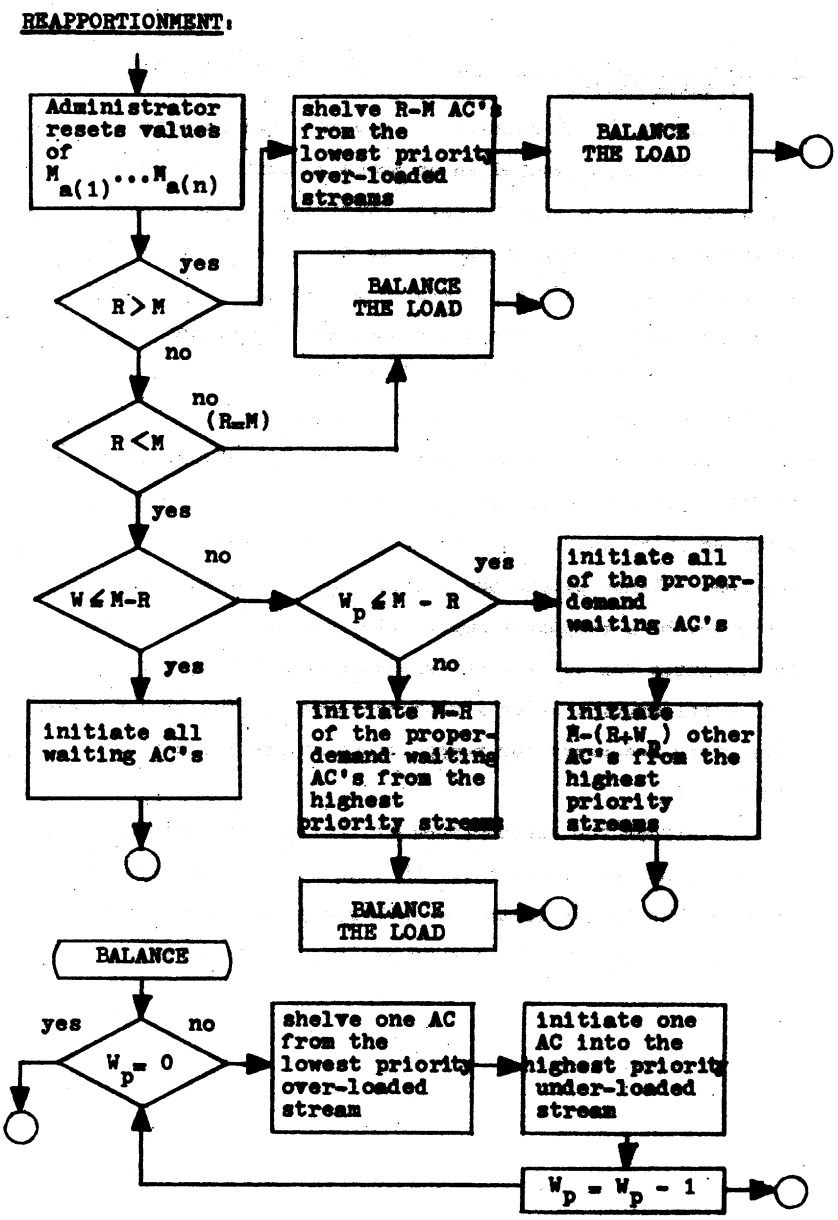


Figure 5.7 Load Reapportionment in a Multiple-Stream Purely Absentee System

immediately shelved from some of the over-loaded streams. Then Load Control proceeds to balance the remainder of the load, an operation which is discussed later in this section. The question arises here as to how to choose AC's to be shelved. Several criteria are useful in making this decision, but perhaps the most significant is to assume that there is some ordering among the various streams (i.e., assume each stream corresponds to a different priority in a priority scheme). Thus the first AC to be shelved is one of the AC's in the over-loaded stream of lowest priority. Similarly, a strategy to use for selecting AC's to be initiated might be to initiate an AC waiting in the stream of highest priority. Note that within the stream itself there is never any ambiguity as to which AC should be selected for shelving or initiation. The entry at the front of the waiting queue is always the next to be initiated; the entry last-in to the running queue is next to be shelved.

If the reapportionment causes the overall system to become properly-loaded (R=M) then it is still possible that some adjustments have to be made and the balancing mechanism is invoked.

If reapportionment causes the overall system to become under-loaded (R<M) then the situation becomes more complex

depending upon how many AC's are waiting to be initiated and whether or not these AC's are waiting in under-loaded streams. If the total number of waiting AC's in all streams is less than or equal to the under-load (M-R) then all the waiting AC's may be initiated and the reapportionment operations are complete. If the number of waiting AC's is greater than the under-load then we must consider just how many of these waiting AC's are really entitled to be run in their respective streams. More specifically, we are interested in the number of AC's which are waiting to be run in under-loaded streams, and of these we are only interested in the first $M_{a(i)} - C_{ar(i)}$ waiting AC's in each stream (since initiating more than this amount would cause the stream to become over-loaded). We refer to the number of AC's satisfying these conditions as the proper-demand, and define

$$\text{proper-demand} = W_p = \sum_{\substack{\text{all} \\ \text{under-loaded} \\ \text{streams}}} \left[\text{minimum}(M_{a(i)} - C_{ar(i)} \, , \, C_{aw(i)}) \right]$$

If the proper-demand is less than the under-load Load Control immediately initiates all of the proper-demand waiting AC's. Now R is still less than M, but every one of the AC's entitled

to be running is running. Load Control now initiates $M - (R + W_p)$ additional AC's to bring the load up to M the reapportionment is complete. If W_p is greater than the under-load Load Control immediately initiates M-R of the proper-demand waiting AC's, but the reapportionment is not yet complete since more proper-demand AC's are still waiting, and hence the load balancing mechanism must be invoked.

From the preceding discussion it is clear that the load balancing mechanism is called upon whenever Load Control has ascertained that R=M, but there may still be some proper-demand waiting AC's which must replace AC's running in over-loaded streams. The balancing mechanism checks to see if there are indeed any proper-demand waiting AC's. If not the reapportionment is complete. If there are then Load Control shelves one AC from the lowest priority over-loaded stream and initiates one AC into the highest-priority underloaded stream. This procedure continues until there are no additional proper-demand waiting AC's.

To summarize sections 5.5 and 5.6, a mechanism has been presented to perform efficiently the load control operations required in a multiple-stream purely absentee system. The

mechanism has the following significant characteristics:

1 - The System Administrator can control the amount of absentee usage in each computation stream by specifying the maximum number of AC's which may run in that stream at one time.

2 - The mechanism prevents waste of available slots by allowing properly-loaded streams to become over-loaded as long as there is no demand for these slots from AC's in the streams to which the slots belong.

3 - The mechanism guarantees an AC first claim to slots in its own stream. If all slots on the system are in use when an AC requests one of its rightful slots then an AC is shelved from one of the over-loaded streams.

4 - The System Administrator is provided with the ability to reapportion the loads in the various streams at any time during system operation.

5 - The mechanism is constructed in such a way as to comply quickly and efficiciently with reapportionment requests from the System Administrator. If a reapportionment causes the system to assume an over-loaded state, the over-load is corrected quickly

so that the system operates with an over-load for the shortest possible time, and then any balancing which must be done to assure AC's first claim to their own stream's slots is done on a "shelve-one-initiate-one" basis to keep the system operating with all slots in use.

5.7 Load Control in a Multiple-Stream Purely Interactive System

Now consider a system devoted to servicing only interactive computations. One improvement which might be made to the scheme in Section 5.2 is to consider that there is some ordering associated with the running IC's which indicates the next IC to be logged out in the event that such action is indeed necessary. By analogy to the case of absentee computations we define here the notion of an interactive computation stream. Furthering the analogy we assume that it is desirable for some reason (such as a priority scheme) to differentiate between various types of IC's. Thus we arrive at a multiple-stream mechanism for handling interactive computations similar to the absentee mechanism described in Sections 5.5 and 5.6. In this section we consider the

modifications which must be made to adapt the absentee mechanism to the handling of interactive computations.

Note immediately that there are no waiting queues in the interactive streams. This is the case because a user is only present for as long as it takes for his logging in attempt. Thus it is not meaningful to consider interactive demand at this level.

Next consider the problem of attempting to fill all interactive slots if the demand for them exists. If the system load operates at a level such that no stream ever becomes properly-loaded then the load control operations are identical to those in the absentee mechanism. However, suppose that one stream does become properly-loaded while some other streams remain under-loaded. If a new IC should request to be initiated into the properly-loaded stream, Load Control has only two choices. Either it can initiate the IC and over-load the stream, or it can refuse to initiate the IC (i.e., the IC cannot be placed into a waiting queue for an indefinite period until a slot becomes available). If initiation is refused then waste occurs because an available slot goes unused. If the IC is initiated then this waste is prevented, but another problem arises. What happens when the overall system becomes properly-loaded (i.e., no more

available interactive slots), and a user attempts to log into an under-loaded stream? In the case of the absentee mechanism it was possible to make a slot available immediately by shelving one AC from an over-loaded stream. In the interactive case, however, making a slot available would necessitate logging out an IC from an over-loaded stream. This is contrary to the continuous service policy discussed in Section 5.2. Several alternatives are available here, but unfortunately none of them is as neat as the shelving of an AC:

1 - Do not allow any streams to become over-loaded. Control the load in each stream independently of the load in any other stream. This results in a component-wise-balanced system. The obvious disadvantage is that slots available in under-loaded streams can never be used by IC's from over-loaded streams thus causing waste of available slots.

2 - Allow overflow in all streams. Initiate new IC's into whichever stream they request as long as slots are available in any of the stream. Once all slots are in use allow no additional IC's until slots again become available. This method assures that

available slots never go to waste, but has the disadvantage that IC's do not get first claim to slots in the streams in which the IC's specify they would like to run.

3 - Allow overflow in all streams. Guarantee completion to all IC's once initiated. If a new IC requests to be initiated into an under-loaded stream then initiate the IC into that stream and trim the load in an overloaded stream by attrition. This method prevents waste of available slots and assures IC's first claim to slots in the streams in which the IC's specifiy they would like to run. The disadvantage is that the trim-by-attrition might result in a slow trim and hence the overall system might be forced to operate in an overloaded state for some time.

Each of these methods has its advantages and disadvantages, but none of them is a "perfect" solution. Our inability to arrive at such a solution here is attributable to the fact that there is no action which may be performed on IC's to correspond to the shelving of AC's. Thus a particular Load Control implementation might choose one of these schemes (or perhaps others) depending upon the particular problems at that installation. If we assume that the most desirable

properties are preventing waste of available slots and guaranteeing IC's first claim to the slots in the streams in which these IC's wish to run, then we arrive at another scheme (which is still, incidentally, not a perfect solution) which is similar to the third scheme above, but prevents the overall system from becoming overloaded:

4 - Allow overflow in all streams, however, if an IC is initiated into a properly-loaded or over-loaded stream then the IC is given second-class status and is warned that his computation is likely to be logged out if the overall system becomes properly-loaded and an IC demands a slot in an under-loaded stream. This method does not assure IC's continuous service, but still allows an IC to get on and use a slot for as long as the slot is not in demand. Since the IC is warned of its second-class status it knows that it is likely to be logged out and hence it can take advantage of being logged in to get a small job done. Of course, as other IC's log out of this over-loaded stream, the IC we are considering may eventually be able to be removed from the over-load portion of the stream. At this time the system could inform the IC that it is no longer of second-class status.

We will assume that this last method is used for the purpose of any further discussions in this thesis. Again, other methods might be more desirable in particular installations.

Now consider the reapportionment problem in a multiple-stream purely interactive system. Since we are allowing streams to become over-loaded as long as the overall system does not become over-loaded, reapportionment can be handled rather straightforwardly. If the total number of running IC's is less than or equal to the maximum allowed, then no action need be taken (note that there are no waiting IC's to be considered as in the absentee case). If the total number of running IC's is greater than the allowed maximum then Load Control logs out IC's from the lowest priority over-loaded streams until the number of running IC's is equal to the maximum allowed. To aid the IC's which are to be logged out Load Control might warn them a few minutes in advance to allow them to "clean up" any details before being forced off the system.

5.8 Load Control in a Hybrid System with Multiple Interactive and Absentee Streams

In this section we combine the mechanisms developed

for handling multiple-stream absentee systems and multiple-stream interactive systems to form a mechanism for handling mixed interactive/absentee systems. (see Figures 5.10 and 5.11).

Figure 5.8 illustrates a multiple-stream queueing mechanism for handling a mixed interactive/absentee system. For the purposes of this discussion we assume that there are n absentee streams and m interactive streams. Note that there are no waiting queues for IC's. This is consistent with our previous discussion of interactive streams.

Figure 5.9 summarizes the parameters used by Load Control in making its various load balancing decisions. The reader is urged to familiarize himself with the definitions of these parameters before proceeding with the following discussion.

Previously, we have considered systems which supported either IC's or AC's but not both. In these systems the apportionment made by the System Administrator was said to divide the system into a certain number of slots, each of which was capable of handling one running computation. It is worthwhile to note here that we have assumed in this division that each slot is, in some sense, of equal size. Thus we have also assumed that regardless of the characteristics of the computations using the slots, the actual demands placed upon the system by these computations is directly proportional to

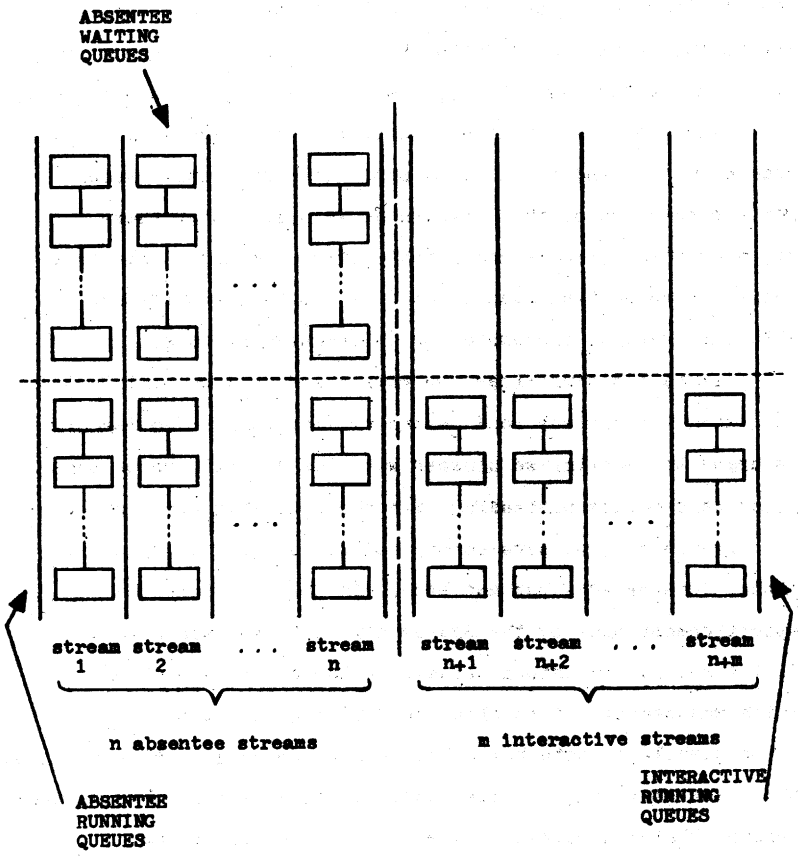


Figure 5.8 Queueing Mechanism for a System with Multiple Interactive and Absentee Streams

$C_{ar(j)}$ = current number of running AC's in stream j

$C_{aw(j)}$ = current number of waiting AC's in stream j

$N_{a(j)}$ = maximum number of running AC's allowed in stream j

$C_{ir(j)}$ = current number of IC's running in stream j

$N_{i(j)}$ = maximum number of running IC's allowed in stream j

$W = \sum_{j=1}^{n} C_{aw(j)}$ = total number of waiting AC's on the system

$R_A = \sum_{j=1}^{n} C_{ar(j)}$ = total number of running AC's on the system

$N_A = \sum_{j=1}^{n} N_{a(j)}$ = maximum number of running AC's allowed on the system

$R_I = \sum_{j=n+1}^{n+m} C_{ir(j)}$ = total number of running IC's on the system

$N_I = \sum_{j=n+1}^{n+m} N_{i(j)}$ = maximum number of running IC's allowed on the system

$N_T = N_A + N_I$ = total number of computations allowed on the system

$W_p = \sum_{\substack{\text{all} \\ \text{under-loaded} \\ \text{AC streams}}} \left[\text{minimum} \left(N_{a(j)} - C_{ar(j)} - C_{aw(j)} \right) \right]$ = number of proper-demand waiting AC's

Figure 5.9 Summary of Load Control Parameter Definitions for a System with Multiple Interactive and Noninteractive Systems

the number of slots in use. This is obviously a simplifying assumption which can be avoided by providing a module capable of deciding just how "big" the slot need be to effectively service a particular computation. With such a module available we could proceed to define an atomic-slot as the unit of computation size measurement. Computation streams could then be envisioned to consist of atomic-slots, and each computation requesting to be run in a particular stream would be granted an appropriate number of atomic-slots in that stream. A stream would then be considered properly-loaded if all of its atomic-slots were in use.

This problem was discussed here because we must again make a simplifying assumption, namely that absentee slots are the same "size" as interactive slots. From the discussion above we can envision ways of avoiding this assumption, too.

The System Administrator apportions system computing power by specifying the maximum number of computations which may run in each of the absentee and interactive streams. Reapportionment may also be done and is discussed later in this section.

When an IC attempts to log in to stream j ($n+1 \leq j \leq n+m$) Load Control checks to see if the total number of running computations on the system is less than the maximum number allowed. If

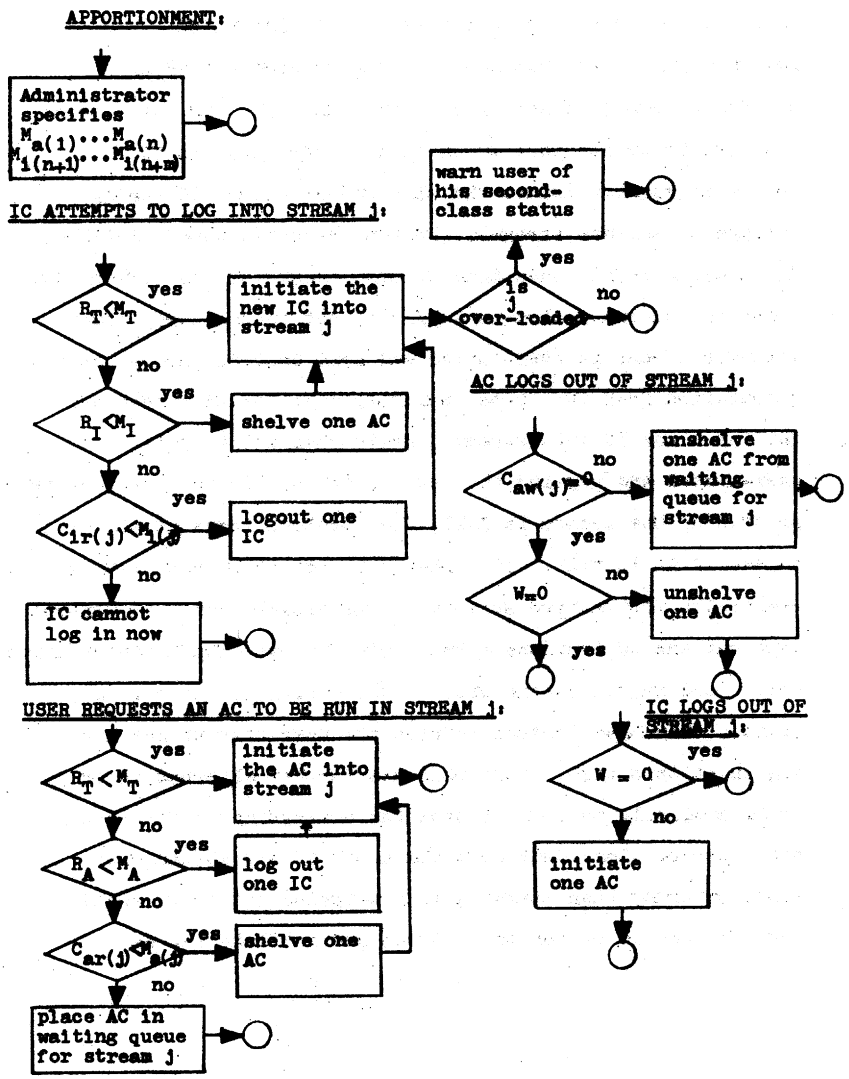


Figure 5.10 Load Control in a System with Multiple Interactive and Absentee Computation Streams

R_T is indeed less than M_T then the IC is initiated immediately. Note that Load Control proceeds to check whether stream j is now over-loaded, and if it is over-loaded the user is warned that there is a possibility his IC may be logged out (since the slot taken by his IC really belongs to another stream). If the system is now full (i.e., R_T is equal to M_T) then Load Control checks if the interactive "portion" of the system is full. If not then clearly some absentee stream is over-loaded and Load Control shelves one computation from the lowest priority over-loaded stream and initiates the IC into stream j again checking if j is over-loaded and warning the user if it is. If the interactive portion of the system is full, Load Control looks at stream j to see if it is full. If stream j is not full then clearly one of the interactive streams is over-loaded and Load Control logs out one IC from the lowest priority over-loaded interactive stream and initiates the new IC into stream j. If stream j is full then Load Control informs the user that he may not log in at this time. Note that one further alternative is still available to Load Control for attempting to initiate this IC now. Load Control could check if any stream with lower priority than stream j is overloaded and if so could then log one IC out of this stream and initiate the new IC.

This alternative is avoided here mostly because of our efforts to provide continuous service (if at all possible) to running IC's.

When a new AC requests to be run in stream j ($1 \leq j \leq n$) Load Control checks to see if the system is full.and, if not, initiates the AC immediately. Note that no warning need be given to an AC if stream j is overloaded. If the system is full Load Control checks if the absentee portion of the system is full. If not, then clearly some interactive stream is over-loaded and Load Control logs out an IC from the lowest priority over-loaded interactive stream and initiates the AC into stream j. If the absentee portion of the system is full Load Control checks if stream j is full. If not, then clearly some other absentee stream is over-loaded and Load Control shelves one AC from the lowest priority over-loaded stream and initiates the new AC into stream j. If stream j is full, then the AC may not be initiated now and Load Control places the AC in the waiting queue for stream j.

Note that this discussion and the diagrams of Figure 5.10 have been simplified by the omission of some of the queue manipulation details prevalent in previous discussions. Thus when a computation is said to be initiated or shelved in this discussion it is meant to be implicit here that these manipulations are performed when appropriate.

When an IC logs out of stream j Load Control checks if any AC's are waiting. If there are none then no operations are performed. If there are AC's waiting then Load Control initiates one AC.

When an AC logs out of stream j Load Control checks if any AC's are waiting in stream j. If there are then Load Control initiates one AC into stream j. If no AC's are waiting in stream j Load Control checks if AC's are waiting in any other absentee streams. If there are then one AC is initiated.

Note that in a system in which the load builds up under the control of the above mechanism the overall system never becomes over-loaded and no AC is ever placed into a waiting queue if slots are available in the stream in which the AC wishes to run. Reapportionment, however, can cause both of these conditions to occur and the methods of alleviating these problems will now be discussed.

5.9 Load Reapportionment in a Hybrid System with Multiple Interactive and Absentee Streams

Figure 5.11 illustrates the operations which must be performed by Load Control to smoothly effect a load reapportionment ordered by the System Administrator. The

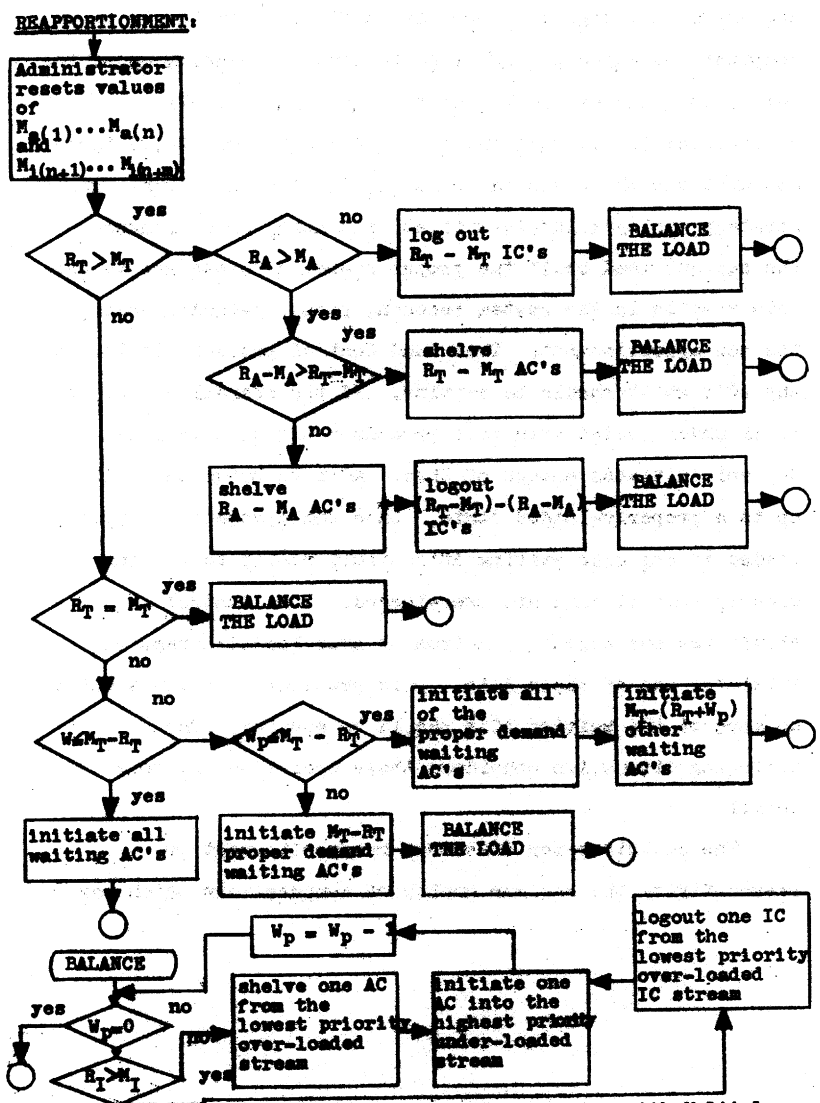


Figure 5.11 Load Reapportionment in a System with Multiple Interactive and Absentee Computation Streams

mechanism is designed to provide continuous service to those computations which are entitled to continue running under the new apportionment, while, at the same time, quickly eliminating (i.e., shelving AC's and logging out IC's) those computations which should no longer be allowed to run. The strategy employed involves first increasing or decreasing the current load until the proper number of slots are in use. This results in the system becoming properly-loaded under the new apportionment. Then Load Control proceeds to locate any AC's which should be running. If the overall system is still under-loaded when this procedure begins then Load Control initiates enough of these AC's to bring the system up to a properly-loaded state. Once the system is properly-loaded if any more waiting AC's should really be running then clearly some streams are over-loaded. Load Control eliminates one computation from an over-loaded stream and initiates one of these AC's. This procedure continues until no more of the waiting AC's should be running. The following discussion considers these operations in more detail.

The Administrator effects a reapportionment by respecifying the maximum number of computations which may

run in each stream in the system. Load Control first checks if the total number of running computations is greater than the maximum allowed under the new apportionment. If R_T is greater than M_T then Load Control must eliminate computations from over-loaded streams to get the load down to M_T. If the number of running AC's, R_A, is less than or equal to the maximum number of AC's allowed, M_A, then the total over-load is made of IC's. Load Control logs out R_T-M_T of the IC's in over-loaded interactive streams (beginning with the lowest priority streams). Once these IC's have been logged out the system is properly-loaded. However, it is still possible that the reapportionment caused some of the waiting AC's to become proper-demand waiting AC's and hence Load Control invokes the balancing mechanism to initiate all proper-demand waiting AC's and log out enough over-load computations to keep the system from becoming over-loaded.

If R_A is greater than M_A then some AC's must be shelved. In particular, if the absentee over-load is greater than the total system over-load then R_T-M_T AC's are shelved. If the absentee over-load is not greater than the total system over-load then all over-load AC's are shelved and then enough over-load IC's are logged out to bring the total system load down to M_T. In each of the above two cases the load-balancing mechanism is invoked after the eliminations.

Up to this point we have been considering cases in which the overall system is over-loaded. If after the reapportionment the overall system becomes properly loaded ($R_T=M_T$) then it is still possible that some AC's have become proper-demand waiting AC's and hence the load balancing mechanism is invoked.

If the reapportionment causes the overall system to become under-loaded then Load Control must first bring the system up to a proper-loaded state if there is enough AC demand. If the number of waiting AC's is less than or equal to the system under-load, then all waiting AC's are initiated and we are done. If there are more waiting AC's than the system under-load then Load Control checks how many of these AC's are proper-demand waiting AC's. If the number of proper-demand waiting AC's is greater than the under-load then Load Control initiates enough of these to get the system up to a properly-loaded state and then invokes the load balancing mechanism to initiate all of the remaining proper-demand waiting AC's. If the number of proper-demand waiting AC's is less than or equal to the under-load then Load Control initiates all of the proper-demand waiting AC's and then initiates enough of the remaining AC's to bring the overall system up to a properly-loaded state and we are done.

The load balancing mechanism is only invoked when the overall system load is equal to the maximum load allowed under the new apportionment. The mechanism checks if there are any proper-demand waiting AC's. If not then we are done. If there are proper-demand waiting AC's then clearly some stream is over-loaded. We know at this point that all slots are in use. Thus it must be true that either AC's are using all AC slots and IC's are using all IC slots, or one of the modes is using more than its allocated number of slots. If both modes are using their allocated slots and there are proper-demand waiting AC's then clearly some AC's are running in over-loaded streams. Load Control shelves one of these over-load AC's and initiates one proper-demand waiting AC and repeats this process until all proper-demand waiting AC's have been initiated. If AC's are using more than their allocated number of slots then the balancing procedure is the same as if both modes are using their allocated slots. However, if IC's are using more than their allocated slots then Load Control logs out one over-load IC and initiates one proper-demand waiting AC. This process continues until either the number of running IC's is equal to the maximum number of IC's allowed or all proper-demand waiting AC's are initiated. If the first condition is satisfied first then

there may still be some proper-demand waiting AC's. Load Control handles this by shelving one over-load AC and initiating one proper-demand waiting AC and continuing this process until all proper-demand waiting AC's have been initiated.

CHAPTER 6

Commands for use with Absentee Computations

Users of the computer system communicate with the system by issuing commands (usually in the form of typewritten statements) at remote terminals. This chapter discusses a set of commands used by system users and administrative personnel to create, monitor, and terminate absentee computations. The discussion here is less detailed than in Chapters 4 and 5; it is included to illustrate what functions might usefully be controlled at the command level.

6.1 Creating an AC

Creating an AC involves two functions. First, the identification of the user must be validated to prevent unauthorized access to the system. Second, the user must inform the system of the absentee source file which is to be used for input to the computation, and the absentee output file which is to receive output from the computation. Additional parameters are supplied to specify in which stream the user wishes his AC to be run, the time limit to be placed on the running AC to prevent waste if the AC

develops problems while the user is not present, and perhaps the user may wish to give a date and time before which his AC should not be run (useful if it is known that data needed by the AC will not be available until that time).

The CREATE-ABS command is provided for users to create AC's. CREATE-ABS may be used by an IC or an AC belonging to the user creating the new AC. Since the computation must already be logged in there is no identity validation necessary. CREATE-ABS results in a call to Load Control which either initiates the computation immediately or places it into the waiting queue for the specified stream depending upon the current system load.

6.2 Terminating an AC

Every computation, upon completion must undergo an orderly logging out procedure to remove the computation from the system and take care of certain "cleanup" problems. In addition, it is sometimes desirable to be able to bring a computation to an early end (such as when the user discovers he has left an AC running with bad input data).

The TERM-ABS command is provided to perform both the normal-end and early-end functions for AC's. The user specifies the computation-identification of the AC to be

terminated. Load Control is called upon to search the queueing mechanism to see if the AC is waiting to be run, running, or no longer on the system (either it is done or the computation-identification was incorrect). If the AC is waiting to be run then the entry for the AC in the waiting queue is deleted. If the AC is currently running then it is stopped immediately and logged out. If the AC is not on the system then the user is so informed.

6.3 Changing the Stream (Priority) of an AC

When the system is heavily loaded the user may find that his AC's take longer to run to completion. To speed up the processing the user may wish to place the AC into a higher priority stream (for which he may be charged more but will get better service).

The CHANGE-STREAM command is provided to remove an AC from one stream and place it into another. The user specifies the computation-identification of his IC, the stream it is currently in, and the stream into which it is to be placed. Note that the CHANGE-STREAM command is also useful for switching IC's from one interactive stream to another.

6.4 Converting an IC to an AC

A user may wish to run a large computation as absentee but in order to be sure that he has set the computation up properly he may want to run it interactively for a while. Once the computation gets going successfully (perhaps the user notes that the proper output is being generated) then the user may convert this running computation to absentee.

The CONVERT command is provided to enable a user to switch a running IC to an AC. The user first presses the "QUIT" button at his terminal to stop the computation so that the CONVERT command may be typed.

6.5 Converting an AC to an IC

The user may wish to monitor the progress of one of his AC's for a while to make sure that it is running smoothly, or perhaps to user would like to make some changes in the absentee source file or other data supplied to the AC.

The CAPTURE command is provided to allow the user to capture control of one of his AC's so that it can be controlled from the user's terminal. Note that the user may wish to finish the computation interactively or he may wish to issue a CONVERT command to allow the computation to finish as absentee.

Note that by using CAPTURE and CONVERT the user may actually control several computations at one time from a single terminal. This is particularly convenient for computations which may need only minor intervention.

6.6 Obtaining Status Information for a User's Computations

A user may have many computations running at one time and may have many absentee computations in the waiting queues waiting to be run. The user may want to monitor the progress of these computations and find out if it might be necessary to intervene (via CAPTURE and CONVERT) with some of them to correct any error conditions which might exist. Also the user may find that some computations are running too slowly and thus it may be desirable to issue a CHANGE-STREAM command.

The STATUS command is provided to give the user information about his various computations on the system. STATUS may be used either to find out about a specific computation, a group of computations, or about all of this user's computations. Information is returned to the user indicating how much time each computation has used, what dedicated resources are being used by each computation, when each computation was initiated, etc.

6.7 Requesting Intervention by an IC

Normally, if an AC develops problems while it is running it cannot be run to completion because it needs information which is unavailable to it in the absence of its owner. However, if a user who submits an AC happens to also be running interactively when such trouble occurs then it is possible that the user will be able to supply the necessary information (or corrections) to the AC so that it may run to completion.

The INTERVENE command is provided to aid an interactive user in specifying that he is available to aid his AC's if trouble develops. Sometimes the nature of the interactive user's work would make it undesirable to be interrupted by a call for help by an AC and in such a case INTERVENE would not be issued by the IC. An interactive user uses CAPTURE and CONVERT to effect an intervention.

6.8 Specifying the AC-IC Load Apportionment

The System Administrator must specify the apportionment of system computing power between the various computation streams on the system.

The LOAD-SPEC command is provided to allow the

System Administrator to make a load apportionment or reapportionment. The initial load apportionment is performed at system startup time and reapportionments may be done whenever necessary. If for a particular application the apportionments should be the same for certain regular periods (shifts) then the System Administrator may specify apportionments for each of these shifts and Load Control will keep these available. Whenever the time for a new shift arises then Load Control will dynamically reapportion the system in the manner discussed in Chapter 5.

6.9 Other Commands for Administrative Personnel

Administrative personnel may find the STATUS and TERM-ABS commands useful. STATUS may be used to obtain status information for any computation on the entire system and TERM-ABS may be used to terminate an AC which appears to be causing problems (such as tying up certain resources).

CHAPTER 7
Summary

The work described in this thesis was concentrated in two areas; a general discussion about the characteristics of absentee computations, and the design of a mechanism for handling absentee computations in a multiple-access computer system.

Perhaps the most significant contributions of the thesis are the concepts of shelving and unshelving absentee computations, the concepts of absentee and interactive computation streams, the design of the multiple-stream queueing mechanism, and the design of the load control mechanism for hybrid multiple-stream interactive/absentee systems.

The design of the combined queueing and load control mechanisms has the following significant characteristics:

1 - The System Administrator may apportion the computing capability of the system between interactive and absentee computations in any proportion whatever. This allows the system to be 100% interactive,

100% absentee, or any intermediate combination of the two modes.

2 - The computation stream concept allows computations of different "types" to run in different streams. One such differentiation might be a priority scheme in which each stream contains all the computations of a particular priority.

3 - Absentee streams have the property that running computations may be temporarily suspended and restarted (shelved and unshelved) several times as they flow through the stream. This property is one of the keys to the success of the load control mechanism.

4 - The multiple-stream mechanism has the property that the load in each stream is individually controlled.

5 - The multiple-stream mechanism maintains a precise ordering among all computations whether they be interactive or absentee and waiting or running. For example, in a priority scheme the computation streams are ordered by their respective priorities. Within each computation stream waiting computations are ordered by virtue of their position within the waiting queue (first-in-first-out discipline is used in this work for choosing the next entry), and

running computations are ordered by virtue of their position within the running queue (last-in-first-out discipline is used in this work). Thus, if at any time the load control mechanism wishes to eliminate or initiate a computation, the choice of which computation to eliminate or which to initiate is determined by the ordering described above. Thus the load control mechanism is made more efficient than it would be if the above choice was not always predetermined.

6 - The load control mechanism prevents waste of available computation slots by allowing streams to become overloaded if slots in other streams are unused. At the same time, the mechanism assures computations in a particular stream first claim to slots which have been specifically allocated to that stream. Thus stream i can become over-loaded by using available slots in stream j. However, if the demand builds up again in j, then the over-load stream i computation must relinquish the usurped slot and is either shelved if it is absentee or is logged out if it is interactive.

7 - Finally, the load control mechanism effects load reapportionments quickly and smoothly. If a computation running before the reapportionment should also run after the reapportionment, load control carefully avoids either shelving or logging out the computation. Initiation of waiting computations and elimination of running computations is done quickly because the ordering described in (5) above makes the selection such computations trivial.

It is worthwhile to note here that there are two obvious levels at which load control decisions can be made, namely the admission level and the scheduling level. At the admission level decisions are made regarding which AC's and IC's shall be allowed to log in to the system. At the scheduling level decisions are made regarding which of the logged in AC's and IC's shall be the next to be given a processor when one becomes available. The mechanism designed in this work operates at the admission level only. Once this load control mechanism allows a computation to log into the system, the computation must then fend for itself in the

competition for processors. At this higher level decisions must be made on the basis of less specific information and must be intended to be enforced over longer periods of time. The apportionment we speak of would probably be in force for at least several hours at a time, and the load control mechanism we propose might be reasonably certain to assure that actual usage closely approximates the apportionment in the average over such a long period.

Recalling the simplifying assumptions made in Chapter 5, namely that each slot is the same size regardless of the particular traits of the computation using the slot, we see that perhaps it would be useful to have our load control mechanism receive information from the scheduling level. Such information combined with an atomic-slot mechanism as discussed in Chapter 5 would help to provide much more precise control over the system load than the mechanism proposed in this work. The design of such a mechanism is suggested for those interested in pursuing research in this area.

REFERENCES

Abbreviations used in the references:

AFIPS	American Federation of Information Processing Societies
FJCC	Fall Joint Computer Conference
SJCC	Spring Joint Computer Conference
ACM	Association for Computing Machinery

References:

(1) Crisman, P.A., editor, The Compatible Time-Sharing System: a programmer's guide, second edition, M.I.T. Press, Cambridge, Mass., 1965, section AH.1.03.

(2) IBM System/360 Time Sharing System, "Concepts and Facilities," IBM Systems Reference Library, S/360-20, C28-2003-0, 1966.

(3) IBM System/360 Time Sharing System, "Command Language User's Guide," IBM Systems Reference Library, S/360-36, C28-2001-0, 1966.

(4) Saltzer, J.H., "Traffic Control in a Multiplexed Computer System," MAC-TR-30, Project MAC, 545 Technology Square, Cambridge, Massachusetts, 1966.

(5) Corbato, F.J., and Vyssotsky, V.A., "Introduction and Overview of the Multics System," AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 185-196.

(6) Glaser, E.L., et al., "System Design of a Computer for Time Sharing Application," AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 192-202.

(7) Vyssotsky, V.A., et al., "Structure of the Multics Supervisor," AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 203-212.

(8) Daley, R.C., and Neumann, P.G., "A General-Purpose File System for Secondary Storage," AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 213-229.

(9) Ossana, J.F., et al., "Communications and Input/Output Switching in a Multiplex Computing System," AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 231-241.

(10) David, E.E., Jr., and Fano, R.M., "Some Thoughts About the Social Implications of Accessible Computing," AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 243-247.

(11) Corbato, F.J., et al., "An Experimental Time-Sharing System," AFIPS Conf. Proc. 21 (1962 SJCC), National Press, Palo Alto, Calif., 1962, pp. 335-344.

CS-TR Scanning Project
Document Control Form

Date : 3/27/96

Report # LCS-TR-52

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 104 (111 images)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:
☒ DOD Form ☒ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number): FOLLOW PAGES NUMBERED, i, vii, AND ix

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

IMAGE MAP: (1-104) i (TITLE PAGE), UNNUMBERED BLANK, iii-vii, UNNUMBERED BLANK, ix, UNNUMBERED BLANK, 1, 1A, 1B, 2-92 (105-111) SCANCONTROL, COVER, FUNDING AGENT, DOD, TRGTS (3)

Scanning Agent Signoff:
Date Received: 3/27/96 Date Scanned: 4/5/96 Date Returned: 4/11/96

Scanning Agent Signature: Michael N. Cook

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Massachusetts Institute of Technology
Project MAC

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
None

3. REPORT TITLE
Absentee Computations in a Multiple-Access Computer System

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Masters Thesis, Department of Electrical Engineering, August 1968

5. AUTHOR(S) *(Last name, first name, initial)*
Deitel, Harvey M.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
August 1968	104	11

8a. CONTRACT OR GRANT NO.
Office of Naval Research, Nonr-4102(01)
b. PROJECT NO.
NR 048-189
c.
RR 003-09-01
d.

9a. ORIGINATOR'S REPORT NUMBER(S)
MAC-TR-52

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES
Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES
None

12. SPONSORING MILITARY ACTIVITY
Advanced Research Projects Agency
3D-200 Pentagon
Washington, D.C. 20301

13. ABSTRACT

In multiple-access computer systems, emphasis is placed upon servicing several interactive users simultaneously. However, many computations do not require user interaction, and the user may therefore want to run these computations "absentee" (or, user not present). A mechanism is presented which provides for the handling of absentee computations in a multiple-access computer system. The design is intended to be implementation-independent. Some novel features of the system's design are: a user can switch computations from interactive to absentee (and vice versa), the system can temporarily suspend and then continue absentee computations to aid in maintaining an efficient absentee-interactive workload on the system, system administrative personnel can apportion system resources between interactive and absentee computations in order to place emphasis upon a particular mode during certain periods of operation, and the system's multiple-computation-stream facility allows the user to attach priorities to his absentee computations by placing the computations in either low-, standard-, or high-priority streams.

14. KEY WORDS

Absentee computations
Computers
Interactive computations
Machine-aided cognition
Multiple-access computers
On-line computer systems
Real-time computers
Time-sharing
Time-shared computers

DD FORM 1 NOV 68 1473 (M.I.T.)

UNCLASSIFIED
Security Classification

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 04/09/1996

M.I.T. Libraries
Document Services