Value-Based Allocation of Computing System Resources

Donald C. Loomis

Technical Report #73

December 1974

Department of Information and Computer Science

University of California, Irvine

UNIVERSITY OF CALIFORNIA

Irvine

Value-Based Allocation of Computing System Resources

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Information and Computer Science

by

Donald Clinton Loomis

Committee in charge:

Professor Julian Feldman, Chairman

Professor David J. Farber

Professor Peter Freeman

Professor Fred M. Tonge

1974

The dissertation of Donald Clinton Loomis is approved, and it is acceptable in quality and form for publication on microfilm:

Fred M. Inge

David J. Farber

Peter Freeman

Julian Feldman
Committee Chairman

University of California, Irvine

1974

CONTENTS

CONTENTS

ACKNOWLEDGMENTS

I would like to thank Professor Julian Feldman, my advisor, for his guidance and encouragement in undertaking this project. I appreciate his advice and counsel during this work as well as other parts of my education.

In addition, I would like to thank Professor Fred Tonge for his continuing interest in my studies. Professor Peter Freeman's advice in the early stages of this work was important in getting it underway. I also appreciate Professor David Farber's interest.

Special thanks go to my wife, Helen, for her sacrifice and understanding during this research and for typing this manuscript.

VITA

November 9, 1947 - Born - Lynwood, California

1966-1968 - Programmer, University of California, Irvine

1968-1969 - Programmer, International Business Machines, Los Angeles Scientific Center

1969 - Programmer, Varian Data Machines, Irvine, California

1969 - B.S. in Engineering, University of California, Irvine

1969 - Teaching Assistant, Graduate School of Administration, University of California, Irvine

1970 - Acting Associate Director, Interactive Computer Facility, University of California, Irvine

1970 - M.S. in Administration, University of California, Irvine

1970-1971 - School of Business, Stanford University

1971-1974 - Research Assistant, Department of Information and Computer Science, University of California, Irvine

ABSTRACT OF THE DISSERTATION

Value-Based Allocation of Computing System Resources

by

Donald Clinton Loomis

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 1974

Professor Julian Feldman, Chairman

Allocation of all resources to maximize the total value of the completion times for all jobs in a multiprogrammed computing system is investigated in this study. In traditional multiprogrammed operating systems, scheduling use of the central processor and main memory has been treated separately from allocation of other system resources. This study investigates the benefits of allocating all resources in a single framework using explicitly specified payoff functions.

A model of resource allocation and scheduling forms the basis of the investigation. To aid understanding and designing resource allocation strategies, the model provides for uniform treatment of all resources. Each process is modeled as a series of resource requests and releases. The process requests resources. The operating system must either grant the requests or suspend the process. The

performance of the scheduler is represented by the set of response times produced when scheduling a job mix.

A new resource allocation strategy which overcomes deficiencies of existing schedulers is presented. Explicit specification of the value of jobs as a function of the time taken to complete them allows the use of utility theory evaluations in making resource allocation decisions and provides the system manager better control over the operation of the system.

Dynamic determination of the opportunity costs of resource assignments are used advantageously in making resource allocation decisions. Simulation experiments showed that value-based allocation is feasible. Because value-based scheduling gives the system manager more flexibility in specifying system goals, it is more adaptable to specific requirements than traditional schedulers. When its parameters were set to approximate the value function of a modern multilevel queue scheduler, the value-based scheduler performed as well as the multilevel queue scheduler.

INTRODUCTION

Value-based allocation of a computing system's resources among competing tasks can provide the system manager more control over job completion times, facilitate scheduler design and allow coordinated use of all system resources. Any object which may be explicitly or implicitly assigned to a process, thereby making it unavailable to another process, is a resource. Examples of resources are central processors, main memory, I/O channels, I/O devices, non-reentrant code sections, and data records which may not be accessed while being updated.

Most operating systems and operating system models seek to attain high performance of single resources or pairs of resources individually without giving much consideration to the effect of these policies on the overall system. For example, studies of policies for scheduling disk access requests have considered minimizing average waiting time in the queue and minimizing disk arm movement but generally ignore possible benefits to the whole system of giving priority to particular processes which need to be completed quickly. Where priorities are considered in the scheduling of resources, they do not adequately take into account

external requirements to complete particular tasks at required times as well as requirements to provide effective and balanced use of the system's resources.

The resource allocation strategy developed in this research is a step in overcoming these difficulties. In the model, all resources are treated in a uniform framework. Because the traditionally separate functions of scheduling processes to use the CPU and allocation of other resources are considered together, the terms scheduling and resource allocation are used synonymously in this dissertation.

Organization of the Dissertation

Chapter 1 summarizes the formulation of the problem, proposed solution, and experimental investigations. Chapter 2 introduces the advantages of multiprogramming. It then surveys related resource allocation and scheduling research and development. The difficulties and shortcomings of traditional approaches to resource allocation and scheduling are itemized at the end of Chapter 2. The next chapter formulates the resource allocation problem. The first part of Chapter 3 relates the model to traditional process structure, defines the resource concept, and develops the performance measure. In the final part of Chapter 3 a notation for the model is introduced. This notation allows both a precise specification of the problem and concise statements of algorithms to solve the problem.

The value-based scheduling philosophy is developed in Chapter 4 as a solution to the shortcomings of traditional schedulers. The result of this development is not a single scheduling algorithm but a framework for constructing schedulers. To demonstrate the feasibility of value-based schedulers, an algorithm was developed and implemented in a simulation model. The results of experiments with this resource allocator and performance comparisons with a multilevel queue scheduler are described in Chapter 5. Chapter 6 presents the conclusions drawn from these studies and suggestions for further research in the area.

Chapter 1

AN APPROACH TO RESOURCE ALLOCATION AND SCHEDULING

Three contributions of primary significance have resulted from this investigation:

1. a formulation of the resource allocation and scheduling problem,
2. application of utility theory and decision analysis tools for solving this problem, and
3. a demonstration of the usefulness and flexibility of applying these tools.

The importance of each of the contributions is enhanced by the others. However, the problem formulation is significant separately since other techniques might be applied to its solution (e.g., dynamic programming or other optimization techniques). Casting the resource allocation and scheduling problem into a form amenable to the application of utility theory and decision analysis is the most important contribution of this study. The algorithm which demonstrates the applicability of these tools is of interest itself. It is a scheduler which has efficiency comparable to conventional schedulers but gives the system manager much more flexibility in adjusting the scheduler parameters to indicate the relative values of jobs.

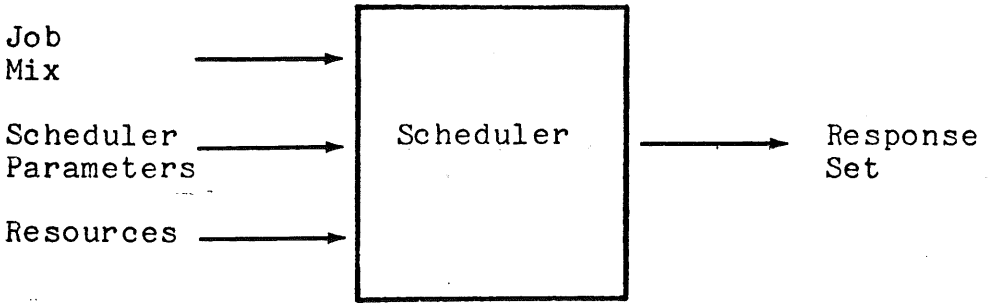
The following sections informally discuss each contribution as a preview to the more detailed descriptions in the body of the dissertation.

FORMULATION OF THE PROBLEM

This formulation of the resource allocation and scheduling problems brings together a number of existing concepts. It also utilizes:

1. the notion of response set as the important criterion for evaluating resource allocation and scheduling algorithms and
2. a uniform framework for considering all resources.

A scheduler has requests as inputs and request completion times as outputs. The requests arise from a job mix. The set of completion times is a response set. The response set for a given job mix depends on the structure of the scheduler, the values of the scheduler parameters, and the quantities of resources available.



Process Structure and Resource Requirements

Each process requires use of varying combinations of the system's resources as it progresses toward completion of the request. Any object which must be explicitly or implicitly assigned to a process, thereby making it unavailable to another process, is a resource. Resources include central processors, main memory, I/O channels, I/O devices, non-reentrant code sections, and data records which cannot be accessed while being updated.

As a consequence of this uniform treatment of all resources, the problems of scheduling use of the CPU and allocating other resources are combined. The scheduler must schedule (allocate) the use of all resources among the jobs.

Scheduler Effectiveness

Evaluation of the response sets for the range of job mixes to be encountered and resources available is the appropriate measure of the quality of a scheduler. Although other measures can be examined (and may be helpful in restructuring or parameterizing a scheduler) they are unimportant as goals themselves.

The evaluation of a response set is subjective. For interactive systems, better response times are generally expected on requests associated with terminal users than on other requests. Often it is satisfactory for response times to be proportional to the size (quantity of resources and

time needed) of a job. Frequently, however, it is desirable to provide much faster response times to small jobs. Except when the response times of one response set completely dominate the response times of another, either might be considered better in some circumstances.

Most conventional schedulers (round robin, strict priority, and multilevel queue with feedback) have been designed to provide a satisfactory response set under specific operating conditions. However, they are not easily adapted to produce different response sets.

Policy-driven schedulers provide more flexibility by allowing response time targets to be specified. However, the policies can over commit the system resources when the system load is heavier than anticipated. The desirability of giving better than target service is not considered by policy-driven schedulers. Also, opportunities to complete jobs in slightly more than the time target which could result in substantial increased system efficiency are not evaluated. Policy-driven schedulers are not applicable to all system resources and do not coordinate allocation of resources.

APPLICATION OF UTILITY THEORY AND DECISION ANALYSIS

Development of the following principles has allowed the application of utility theory and decision analysis to the resource allocation and scheduling problem.

1. Utility functions can represent subjective evaluations of response sets.
2. Scheduling algorithms can use these functions as goals for the production of maximum value response sets.
3. The utility functions can be parameters of the schedulers to allow convenient adaptation of a scheduler to individual utility functions.
4. Decision analysis can be used to produce efficient schedulers which use these functions as goals.

The use of utility theory and decision analysis to design resource allocators has significant advantages over traditional mechanisms:

1. The schedulers give system managers more control over the response sets through parameterization of the utility functions.
2. Expected value analysis of scheduler decisions facilitates the design of rational schedulers.
3. The approach is applicable to all system resources and permits coordinated allocation of the resources.

The feature which distinguishes this approach to resource allocation and scheduling from traditional scheduling algorithms is the explicit use of a value function as a scheduling goal. Providing this function as a parameter to the scheduler allows it to be changed easily.

This provides the system manager a convenient means to specify a personal utility function and thereby obtain a scheduler designed to maximize the utility of the system.

The Utility of Response Sets

The response times for each job determine the overall value or utility of a response set. Generally, the value of the jobs are independent and the total utility of the system is the sum of the utilities of the individual jobs. The dependence of the utility of requests on the time taken to complete them can be provided as a parameter to the scheduler as a function of the form:

Value of job at completion = V(class,size,time)

where

class is the class of service,

size is a function of the resources required and time required, and

time is the elapsed time to complete the request.

The class parameter allows requests to be valued differently on the basis of characteristics recognizable prior to execution (e.g., user name and estimated execution time). The size of the job can be an arbitrary non-decreasing function of the resources required and time they are required. Generally, the function is increasing with the size of the request, since it is usually worth more to

complete a larger job rather than a smaller job of the same class in the same time. However, the function is usually decreasing with time since it is almost always better to finish jobs of the same class and size in shorter rather than longer elapsed time.

A value can be assigned to each partially completed job by evaluating the portion of the job already processed as a completed request. Since the final size of a request is not known until after the job has been completed, the scheduler assumes the next increment of service will complete the job. Thus, to maximize the value of jobs at completion, the scheduler should allocate resources to maximize the value of the partially completed jobs.

Scheduler Decisions

The scheduler must decide how to allocate system resources among requesting jobs. Resource requests may be satisfied by allocating unassigned resources or by preempting resources which have been assigned to jobs. These decisions require an evaluation of the best assignment of the resources.

Jobs which have been assigned all of the resources they require can proceed; jobs which do not have all the resources they require cannot proceed and must be suspended. The value of an executing job will increase as it receives service. On the other hand, the value of a suspended job

will decrease as the elapsed time increases. The rate of these changes can be calculated by examining the partial derivatives of the value function with respect to job size and time. In addition, executing jobs will free the resources they hold sooner. The value of freeing these resources can be calculated by determining the value of alternative use (i.e., the value of executing the processes which need them and have been suspended).

Both the changes in the value of the partially completed jobs and the value of freeing resources must be considered to evaluate the best resource allocation. A simple allocation rule results from assuming that a process which is assigned the resources it requires will increase in value at the calculated rate for a significant period of time and assuming that, when a process releases resources it holds, there will still be the same alternative uses for the resources. Under these circumstances, the processes to be assigned resources should be selected on the basis of the current rate of value increase which will result from their execution and the current rate of value increase which would result from the best alternative use of the resources they already hold.

A more complete evaluation could be made by considering the probability that the jobs may soon request the use of other resources and possibly be suspended. Determining the value of freeing resources based on current alternative uses

will be inaccurate if the demand has changed by the time the resources are released. Using models of program behavior and models for projected aggregate system requirements, probabilities can be used to compute expected values. Thus decisions can be made to maximize the mathematical expectation of the system value.

Summary of Value-Based Resource Allocation

The central features of value-based resource allocation are

1. the parameterization of the scheduler with a specification of the values of jobs as a function of class of service, job size, and elapsed time to complete and
2. a scheduling strategy which attempts to maximize the total value of the jobs processed by the system.

This approach to resource allocation and scheduling permits a very flexible specification of the response set by the system manager. Scheduler design is facilitated by a rational framework for making resource allocation decisions. The technique is applicable to all resources and effects coordinated allocation of the resources. The features which differentiate value-based resource allocation from traditional scheduling techniques are listed in Table 1.

Table 1

Comparison of Scheduling Techniques

Queue

- Order of executions determined by position in queue.
- Processes are entered into queue on basis of service class, quantum expiration, and/or other events.
- Response characteristics are built-in.
- Only slight tuning is possible by varying quantum time.

Policy-Driven

- Response targets are specified by system manager.
- Scheduler evaluates difference between service received and target to determine process most in need of service.
- Policies can over commit system.
- No provision is provided for better or worse than target service.
- Interactions between resources are not considered.

Value-Based

- System manager specifies values of jobs as a function of job size and completion time.
- Scheduler schedules jobs to maximize value of system.
- Allows more flexible response set goals.
- Scheduler design is facilitated by a rational decision framework.
- Coordinates allocation of all system resources.

EXPERIMENTAL DEMONSTRATION

To investigate the feasibility of this approach to resource allocation and scheduling, experiments were conducted using simulation. Allocation of a CPU, pages of main memory, a disk, permission to open and close files, and a swapping channel was considered. Allocations were evaluated according to the basic rule discussed previously every time a resource was requested, a resource was freed, or after 1.2 milliseconds passed without a reevaluation. The possible changes in the values of the jobs and the values of freeing resources were considered in the following steps:

1. Determine current value of running each job from the value functions supplied by the system manager.
2. Adjust the value of running each process by considering the alternative use of the non-preemptable resources held.
3. Allocate free and preemptable resources beginning with highest value process.
4. If swapping channel is free, assign it to make copies of programs in main memory starting with lowest value process so that main memory pages will be preemptable.

The performance of the scheduler was observed under a variety of operating conditions by varying the scripts of the jobs to be scheduled. The job mixes were chosen to

explore both extreme and average mixes. These ranged from completely CPU bound to primarily I/O bound. They also differed in their requests for memory and permission to open and close files. Consequently, the conclusions are valid over a range of operating conditions.

Three sets of value function parameters were chosen to demonstrate the ability of the scheduler to simulate implicit value functions typical of conventional schedulers. A value function designed to give all jobs the same average rate of CPU use and thus response time proportional to the CPU requirements did result in all jobs of a mix receiving equal CPU usage. By specifying a slightly different value function for part of the jobs in a mix, the jobs in one class could be caused to receive service at twice the rate of the jobs in the other class. The third parameterization gave a very high value to the completion of requests requiring less than 1.2 seconds CPU time in a short time. As expected, this gave a significant bias to small interactive terminal requests. This value function is a good approximation to the value function implicit in the design of a conventional multilevel queue scheduler which was also simulated to allow comparison. The experimental value-based scheduler is much more flexible, but was able to produce response sets nearly identical to the multilevel queue scheduler.

Chapter 2

MULTIPROGRAMMED COMPUTER SYSTEMS

The first electronic computers executed a single program at a time. In performing their primary function of calculating ballistic tables, they performed each of the steps of the calculation sequentially until the entire calculation was complete. The entire computer was available to and controlled by this single program.

MULTIPROGRAMMING

For a number of reasons multiprogramming was introduced. In a multiprogramming system, each job is not necessarily completed before others are started: at any time many jobs are partially completed. The facilities of the computer must be shared or multiplexed (switched) among these jobs. The benefits of multiprogramming can be grouped into four categories:

1. completion of a given set of jobs in less time (or more jobs in the same amount of time with a less than proportional increase in hardware),
2. multiple interactive and real-time process control activities,

3. control over the order in which jobs complete (independent of the order in which they began), and
4. efficient use of hardware facilities selected to accommodate a single large job by several small jobs.

I/O Overlap

The possibilities of using multiprogramming to complete several tasks in less time than would be used to do them sequentially can be illustrated by considering the use of a system's central processor and single input-output facilities. Spooling was introduced to take advantage of hardware designs which allow input-output operations to proceed without use of the central processor except briefly after each operation completes to initiate the next.

By having a portion of another program perform these functions, it is possible to have a computing system move data from one input-output device to another while performing a completely independent computation. Because the central processor requirements for initiating the input-output are minimal, the other program will still complete in approximately the same amount of time as it would have if no input-output operations had been going on at the same time. Similarly if the input-output program had been run without other computation, the central processing unit would have been idle most of the time but it would have

taken the same amount of time. By multiprogramming these two jobs, they both can be completed in the amount of time required for one. To have jobs with such completely complementary resource requirements is ideal but not unusual. There are jobs in the real world whose progress is almost entirely limited by the computing system's input-output facilities. Other jobs which do essentially no input-output are completely CPU bound.

Other tasks alternate their needs for input-output and central processor facilities. While one is performing I/O the other can use the central processor. If the jobs get to a point where they both need to use the I/O facility or both must use the central processor, one must wait. Unless they have requirements which are complimentary and periodic in time, they won't both be able to complete in the time required for one. Furthermore, some overhead is encountered every time a possible reassignment of the processor must be evaluated. However, usually they will be able to complete in less time than would be required to run sequentially.

Timesharing

Terminal oriented timesharing systems utilize multiprogramming to allow users to interact with their jobs as they progress toward solutions to their problems. Each user input, computation, and computer response constitutes a partial completion of that user's job. However, the user

context must be retained by the system between interactions. With multiprogramming, a number of users can retain their working context in a large system. Each user may have access to all or part of the facilities of the system on any request. Those system facilities not in use by the user (because he is thinking and has no request pending or because the facilities are in excess of his need while his request is being serviced) are available to other users.

When there are sporadic and complementary requirements for the use of system facilities, more work can be performed per unit time with multiprogramming than would be possible if each job (user's session) were completed before beginning another.

Urgent Requests

In non-timesharing systems, multiprogramming allows high priority jobs to be introduced into the system and completed quickly even though others were already in progress. Those which were there first can be suspended completely or continue but with a lower priority in the use of the system's facilities than the high priority job. When the high priority job completes, those remaining will again be able to proceed at their previous rate. In contrast, in a system without multiprogramming, either the high priority job could not begin until the current (possibly very long) job completed or the computer operator would have to

terminate the current job and restart it later. The multiprogramming alternative allows preference to be given to jobs without the need to end other jobs. Conversely, even when the arrival of high priority jobs is expected, the computer need not be left idle but low priority jobs may be initiated without fear that the effort will be wasted.

Efficient Alternative Use

The minimum equipment configuration for a computer system must be large enough to accommodate the requirements of the jobs to be run which have the greatest requirements. However, frequently smaller jobs only require a fraction of the resources needed by the largest jobs. Multiprogramming permits partitioning and multiplexing of the resources to allow efficient execution of several small jobs instead of a single large job.

TRADITIONAL APPROACHES TO PROCESS SCHEDULING

The problems of allocating the resources and particularly scheduling use of the central processor in multiprogrammed systems have been studied and reported extensively in the literature. Since most of the developments are summarized in review articles, the following is limited to a survey of the techniques used, an indication of the appropriate review articles, and more

detailed descriptions of work with direct relevance to the model and techniques developed in this dissertation.

Central Processor Schedulers

The development of increasingly sophisticated schedulers are reflected in publications by implementors and proposers of multiprogrammed operating systems. These schedulers utilize the basic techniques of first-come-first-served, shortest job first, round robins, and multilevel queuing with numerous variations in the handling of preemptions, external priorities, and special circumstances. Kleinrock (1968) has summarized the important principles of these schedulers together with the major drawbacks and limitations.

Analytic Models

Analytic models based on probability theory, queuing theory, and Markov chains have been used to investigate scheduling problems. McKinney (1969) and Chang (1970) review the use of these techniques.

Memory Allocation

The use of secondary storage to hold programs while they were not executing with the CPU led to the need for swapping and main memory allocation strategies. As a

result, an efficient system depended on coordinated allocation of the central processor and swapping channel. The requirements are discussed in Denning (1968a, 1968b, 1969, and 1970).

Other Resources

Allocation of almost all other resources except I/O devices has been with first-come-first-served algorithms. In some cases I/O channels have been allocated on a priority basis. Scheduling accesses to disks and drums has received considerable attention. Teorey (1972) discusses disk scheduling; Fuller (1973) discusses drum scheduling.

Deadlocks

Avoidance and detection of deadlocks are surveyed by Coffman, Elphick, and Shoshani (1971) and later by Holt (1972). Deadlocks result when two or more processes hold non-preemptable resources needed by the other(s) and neither can proceed to release the resources needed by the other. All resource allocation algorithms must satisfy the constraints resulting from the possibility of deadlocking.

Two aspects of deadlock studies are significant to this work. First, deadlock research is the only situation where resources have been treated in a uniform framework: since, the central processor and main memory are normally

preemptable they are not considered in deadlock evaluations. Second, the notation which is frequently used to express deadlock problems and solutions lent itself to adaptation and extension for formalizing value-based resource allocation.

POLICY-DRIVEN SCHEDULERS

An analytic approach to policy-driven scheduling is provided by Kleinrock (1970). Two operating system implementations schedule on the basis of policy functions. The Research and Development Center Operating System (R & DC) for the GE 600 described by Bernstein and Sharp (1971) motivated development of value-based scheduling. The IBM 370 OS/VS2 Release 2 system described by Scherr (1973a and 1973b) and IBM (1973a and 1973b) was developed independently and concurrently with value-based resource allocation.

R & DC Operating System

The policy-driven scheduler described by Bernstein and Sharp concentrates on allocation of the central processor and main memory. There are two aspects to the implementation: service policies and scheduling rules.

The policies are based on a resource count function which defines the service received by a process as a weighted sum of its accumulated resource usage. Resource

usage is measured as the time of use (e.g., milliseconds CPU usage) or number of times used (e.g., number disk accesses), depending on the resource. The weighting vector is chosen arbitrarily to control the emphasis on use of each resource. Since the resource count increases as the process receives service, it is a non-decreasing function of the elapsed time from receipt of the request for service.

For each class of service to be provided (e.g., interactive terminal, batch, or spooling) a policy function is parameterized which specifies the minimum acceptable resource count as a function of the elapsed time from receipt of the request for service. Typically, an interactive policy specifies an initial rapidly increasing part of the function to ensure small requests get prompt service and lesser service for greater elapsed time. Batch job policies would be linear indicating no preference to short jobs.

The scheduling rules attempt to keep the resource count for each process greater than the policy specification for the class of job. When a job has received less service than specified so that the resource count becomes less than the policy, the process has a critical need for service.

The processor is always allocated to the process in main memory which has been critical longest, or, if there are no critical processes, the process which will become critical soonest. The processor is reassigned if a time

quantum expires or the process voluntarily relinquishes it.

The swapping rules are designed to minimize the overhead of unnecessary swapping. A process which becomes critical while not in main memory is swapped into memory if either free memory can be found or it can replace a noncritical process. Once in main memory a critical process is never swapped out until it receives enough service to become noncritical. A noncritical process is swapped in only if there is free memory or it can replace a process waiting for terminal input. As a consequence of these rules, critical processes are never swapped out; noncritical processes are only swapped out to make room for a critical process.

IBM 370 OS/VS2 Release 2

The progress of jobs under OS/VS2 Release 2 is also measured as a weighted sum of the resources used. CPU use, number of I/O operations, and the product of CPU use with memory size are considered. Classes of service are allowed. However, in place of parameterizable equations for specifying policies, the IBM system utilizes tables which specify a discrete service rate for each interval in the life of the request. The intervals may be either periods of elapsed time or virtual process time. The inability of policy-driven schedulers to adapt to changes in system load has been reduced by allowing separate tables to be specified

for different workload levels. The system dynamically selects a new table if the workload increases or decreases.

The centralization of many resource allocation algorithms into a single routine is significant. Centralizing the algorithms and data for these decisions potentially allows global resource allocation strategies to be implemented. However, no global framework was adopted: the few situations where allocations interact utilize specialized techniques. The swapping algorithm is influenced by I/O usages to maintain a balance of I/O and CPU bound jobs. The swapping of a job is delayed when it holds certain resources required by another process.

OTHER RELATED RESEARCH

Mahl (1970) has investigated algorithms for maximizing a cost-weighted total of the rates of progress for the jobs in a system. Mahl's algorithms are based on the premise that pricing should be based on marginal costs and the scheduler should maximize the revenue of the system. Value-based scheduling allows a greater range of scheduling goals including priorities based on pricing.

Although not directly related to this research, the work of Grochow (1972) may have importance in the future. Grochow argues that system managers should assess the requirements of the system users by determining the utility of alternative kinds and levels of computing service which

could be provided to them. By measuring the individual utility functions of users, Grochow believes the manager will be aided in both the problem finding and choice aspects of decision making.

Value-based scheduling operates on the premise that the operating system should be based on the system manager's utility functions, not individual user functions, since he is the one who has responsibility for the system. Furthermore, there are no techniques for combining several individual utility functions into a single utility function: the utility of a group of people is not defined. However, at least one of the system manager's goals is to increase the utility of individual users. If the individual user utility functions can be related to the system managers utility function, these two areas of research will become relevant to each other.

Chapter 3

A FORMULATION OF THE RESOURCE ALLOCATION PROBLEM

The definitions and notation presented in this chapter provide a more precise framework for considering the resource allocation problem.

PROCESS STRUCTURE

The terms process, job, and task are used in different systems and different parts of the literature to describe the concept of independent activity. The definitions for these terms vary only slightly and will be used interchangeably here. In many systems, processes are entirely independent except for conflict over the use of system facilities. In other systems, they may be more closely related through initiation, termination, synchronization, and communication primitives implemented in the supervisor program or system hardware. In virtually every system, each process may be identified by a set of state information maintained by the supervisor program controlling that process.

The following additions and restrictions on the standard notion of a process are useful in understanding the

ideas which follow. There are two ways to measure execution time. The difference between the current time and the time of the request for service is the amount of elapsed time the process has been in execution.

ET is time since activity requested

The accumulated amount of time a process has been assigned the system resources it requires is the virtual time of execution.

VT is time the process has been assigned all needed resources

This definition differs from some other definitions of virtual process time based on the accumulated time the process used the central processor. For the global and uniform treatment of resources this new definition is more convenient. If a single process is executing in a system, there can be no conflict over use of system resources and its virtual and elapsed times will be equal. In a multiprogramming system where there is conflict over resources, the virtual time will be the length of time the process would have run if it had been the only process in the system.

If a process is unable to continue because of some condition other than lack of a necessary resource, the process is inactive. The time that a process is inactive is the inactive time.

IT is time process is inactive

An inactive process may be waiting for a response from a terminal users, the passage of some amount of real time, etc. An active process is either running or suspended. A process is running if it has all the resources it currently requires and is therefore continuing execution. If it is unable to continue because it does not have the required resources, it is suspended. The suspended time is the accumulated time the process has been suspended.

ST is time process suspended

Since a process is either inactive, suspended, or running,

$$ET = IT + ST + VT$$

These definitions differ from the traditional definitions of blocked, ready, and running processes (Denning 1971, p. 202) to allow uniform treatment of all resources. In traditional systems which have been centered around central processor scheduling, the traditional terms are useful. However, in this model of resource allocation, a process is in execution when it is doing input-output and does not require use of the central processor. The I/O unit may be thought of as a special purpose processor which is needed for some parts of the calculation while the central processor is needed for other parts. Furthermore, the development of new multiprocessor computer architectures is resulting in a potential for programs which need or are able to use more than one processor at a time.

Resource Requirements

In general, the resource requirements of a process vary over the life of the process. When the process progresses to a point where it needs more resources, it requests the desired resources, from the system supervisor. When the process no longer needs some resources, it releases them to the supervisor. When the process terminates, all the resources it still has are released. When a process requests resources it must have them to continue and must be suspended if they cannot be provided. There are important exceptions to this assumption such as when a program requests all of the available tape drives to do a tape sort; also, when a program asks for but does not expect to be assigned all of a large amount of main memory for use as input-output buffers or in a "free storage" pool.

Furthermore, it is assumed that the process resource requirements are inherent in the program and input data and thus completely determined at the time of the request. Hence, they do not depend on the process's environment, resources available at the time of a request, or job's rate of progress. They could be listed as a function of the process's virtual time. They do not need to be known in advance by the system supervisor but must be predetermined.

Request Size

The size of the request is an arbitrary function based

on the resources required and the virtual time they have been used. This function may be as simple or as complex as required. A simple function could specify request size as the amount of central processor time required to complete the request. Another function could be calculated by first taking the quantity of each resource used multiplied by the virtual time that quantity of resource was required and then taking a weighted sum of the products. The size function could even be non-linear with the quantities of resources used or with time. The only restriction is that the size of each partially complete request must be less than the size of the complete request.

SYSTEM RESOURCES

A system resource is anything which can cause a conflict preventing the simultaneous execution of two or more processes. The exact items which are considered resources vary from system to system but typically include hardware facilities, non-reentrant code, data records which may not be accessed while being updated, etc. Process synchronization primitives may be represented with resources. This will be discussed later.

Any system facility, segment of code, etc. for which there can be no conflict, is not a resource. For example, since reentrant code is sharable, it is not a resource. If an entire data file is assigned to a process before it

updates any records, the individual data records cannot be resources themselves. Resolving the conflicts completely at the file level precludes any conflict over accessing particular records. To be a resource there must be possibility of a conflict between two or more requesting processes.

The resources are the only absolute constraint on the scheduling of processes. Any schedule of resource allocation is feasible if it meets the constraints:

1. A process is always suspended when the resources it requires cannot be assigned to it.
2. The schedule does not result in a deadlock or deadly embrace. A deadlock occurs when two (or more) processes are waiting for non-preemptable resources held by the other. When this happens neither can proceed and release the resources needed by the other.

The deadlock problem has been studied extensively (Coffman 1971, Holt 1972). All the following assumes appropriate provisions are also taken to handle deadlocks.

In a system with a single central processor, the central processor is a unique resource. There is only one unit of central processor which must be multiplexed among all processes. In a classical multiprocessor system there are several processors each of which can be assigned to a process. The number of individually assignable units of a

system resource is the capacity of the resource. In the case of a multiprocessing system the capacity of processors is the number of processors. In a system with a single processor the capacity of processors is one. In a paged memory system the page map allows the physical pages of memory to be treated as identical, separately assignable units of the memory resource. The capacity of main memory is then the number of pages which can be assigned to the processes.

Resource Classes

If one resource may in some cases be acceptable as a substitute for another, they belong to a common resource class. A process may request a resource by class to obtain use of whichever is available. A resource may be a member of more than one class. Classes may be overlapping or one can be a proper subset of another. For example, consider a system which has two line printers with different character sets. Processes which need the unique capabilities of either printer require assignment of a specific resource. Processes which use only the capabilities they have in common can make a request for an assignment from the line printer class of resources. Similar situations exist with processors having different instruction sets, data storage devices, etc. When a process requests use of a resource by class, the system is free to assign any member of that

class.

Conflict over use of a resource is the only obstacle which can prevent processes from running simultaneously. As long as there are no resource conflicts, any number of processes may be running simultaneously. When there is a conflict, one or more of the processes must be suspended while the others continue.

If a resource has been assigned to a process it may be possible to suspend the process and reassign the resource to another process temporarily and then return it to the original process without affecting that process except to delay its progress. If this preemption is possible the resource is preemptable. Some resources are always preemptable. Others are never preemptable. Some may be preemptable only under certain circumstances. For example, a central processor can almost always be taken away and returned later. Exclusive permission to update a file could not be preempted without possibility of destroying the integrity of the file.

SYSTEM PERFORMANCE

Response Time

The amount of real time which elapses between the user request and the response by the computer is the response

time.

RT is ET at request completion

For an interactive user this is the (hopefully) short time from the entry of a command on his terminal until the computer types a reply and prompts for a new command. For batch jobs the size of requests is typically much larger and takes much longer to complete. Nevertheless, the response time is the amount of real time elapsed from the time the job is submitted until the results are output. A user sitting at a terminal or submitting a batch job need not be concerned with the internal scheduling policies of the system. Only the response time is important. Whether the system gives a large amount of service to the process at the beginning of the response interval and a little at the end, none at the beginning and all at the end, or even increments throughout the period is irrelevant to the user. Generally the user is unaware of how the service he receives is distributed over time. He only knows that he requested the performance of a task of some approximate size and it was completed in a certain amount of time. He doesn't really care how it is done but wants a response in a reasonable amount of time.

Response Set

The response times of all jobs taken together constitute the response set, RS.

$$RS \equiv \{RT_1, RT_2, RT_3, \ldots, RT_n\}$$

Scheduling the execution of a set of jobs will result in a response set for that scheduling algorithm. The response set can be compared with requirements or preferences for completion of the jobs to evaluate the efficiency and applicability of the scheduling algorithm. To complete a request, a process will require use of the system resources for some virtual time. The mix of resources needed may vary during this time as the process requests and releases resources.

RESOURCE ALLOCATION NOTATION

To explore resource allocation and scheduling in more depth it is useful to have a more formal notation. The following notation is an extension of the notation used by a number of authors in dealing with the problem of deadlocks which can arise in resource allocation. The first part of this section defines the notation and gives its relation to the activities of resource allocation and scheduling in computing systems. The resource allocation problem is then stated using the notation.

Processes

Let {P1,P2,P3,...,Pn} designate the processes in a computing system. The subscript p is used to denote the

typical element of a set, vector, or array which is indexed by process number. There is an upper bound (possibly quite large) of n processes existing in the system at any point in time. After jobs or requests are completed, their positions are available for new jobs or requests. Hence, an infinite number of jobs or requests are allowed over time but at any point in time at most n may be present in the computing system.

Resources

The set {R1,R2,R3,...,Rm} is the set of resource classes available for allocation to the processes. The typical element of a set, vector, or array which is indexed by resource class number is denoted by the subscript r. A resource class includes one or more identical system resources. In a computing system with two undifferentiated processing units, both are members of the resource class of processing units. If, however, the processors are not identical, they are different resources and belong to separate resource classes such as master processing unit and slave processing unit. Resources belong to the same class if and only if they are identical in capability. Resources of different capabilities may have common capabilities where the capabilities of one class are a superset of the other or where there are different capabilities in each which are not in common. Included in the set of resource classes are all

resources which could potentially be a source of conflict in the progress of any of the processes in the system. Any facility--hardware or logical--over which conflict can occur and suspension of a process result is a resource. The number of resource classes is m.

The vector $C \equiv (c_1, c_2, \ldots, c_r, \ldots, c_{m-1}, c_m)$ is the capacity of the system. Each element of the vector specifies the number of units of the corresponding resource which are available for allocation to processes before any allocation has occurred. The reserve vector $R \equiv (r_1, r_2, \ldots, r_r, \ldots, r_{m-1}, r_m)$ gives the number of units of the corresponding resource currently available for allocation to processes. As resources are allocated, the number of units allocated are deducted from R. When they are released, the number of units released are added to R. Thus the difference between C and R is the number of resource units currently allocated. The units of measurement for resources depend on the exact nature of the resource and will be different for different resource classes. Typically, the processor resource unit would be the number of processor units; main memory would use number of pages. For other resources the count of the number of processes simultaneously using the resource is the appropriate unit. For non-reentrant code sections the capacity is one process.

Computation Steps

Each process goes through a series of computation steps $S_p \equiv \{S1_p, S2_p, S3_p, \ldots\}$. Each step of the sequence corresponds to a level of resource requirement. The transition from one step to another is associated with either a request for allocation of more resources, the releasing of resources, or both the releasing of some resources and a request for allocation of others.

Resource Requirements

The number of units of a resource required by a process is the demand for the resource by the process. The demand for resource r by process p at step b is d_{prb}. The sequence of demands by process p for all resources and steps is

$$D_p \equiv [d_{prb}].$$

The sequences of resources required by all processes is

$$D \equiv (D_1, D_2, \ldots, D_p, \ldots, D_{n-1}, D_n)$$

At any time t the quantity of resource r assigned to process p is a_{prt}. The state of quantities of all resources assigned to all process is

$$A_t \equiv [a_{prt}].$$

What resources are not assigned to process are in the reserve. This can be expressed:

$$R_t = C - \sum_P A_t.$$

The difference between the quantities of resources requested by a process and the quantities allocated by the system is the <u>want</u>:

$$W_t \equiv [w_{prt}],$$

$$W_t \equiv D - A_t.$$

If $w_{prt} > 0$ for any r then process p is suspended until the requested resources are allocated.

<u>Requirement Intervals</u>

The time interval required by process p to complete step b is i_{pb}. The sequence of time for all steps is

$$I_p \equiv (i_{p1}, i_{p2}, \ldots).$$

Then

$$I \equiv (I_1, I_2, \ldots, I_p, \ldots, I_n)$$

is the time sequences for all processes.

Together D and I specify the resource requirements of the processes. C is the capacity of the system. The allocation sequence for the system is the sequence of assignments

$$A \equiv (A_1, A_2, A_3, \ldots).$$

Where l_t is the length of time for which the assignment is

A_t, the sequence

$$L \equiv (l_1, l_2, l_3, \ldots)$$

is the sequence of allocation lengths.

THE RESOURCE ALLOCATION PROBLEM

The resource allocation and scheduling algorithms of a computing system must determine an A and L such that

$$\sum_p a_{prt} \leq C \quad \text{for all } t.$$

The sequence of ordered pairs (d_{prs}, i_{ps}) over s is a subsequence of the reduced sequence (a_{prt}, l_t) over t for all p. Any A and L which meet the above condition represent a feasible schedule. The resource allocation problem is then to find a feasible schedule which maximizes the value of the response set.

Chapter 4

VALUE-BASED SCHEDULING

System Goals

Every computer has been procured by an individual or organization to fulfill some function. Depending on the situation, the statement of the application may be relatively specific (e.g., maintain the inventory records for a particular product) or be more general (e.g., perform research computing). Furthermore, over a period of time the computer's function may change.

Recognizing the existence of a purpose and consequent value in fulfilling this purpose is important to place the problems of resource allocation in perspective. Associated with the purpose for the computer's existence are one or more goals. The value of the computer depends on how well the computer fulfills the goals. While in some cases the computer may either fulfill the goals satisfactorily or fail to meet the requirements, in most circumstances various degrees of goal fulfillment are possible.

Activities outside the computer as well as inside affect the amount of goal fulfillment. An individual, committee, or complex organizational structure has the responsibility to form and implement plans which will result

in the most value from the use of the computer. For convenience, this individual, committee or organizational structure will be referred to as the system manager. This is appropriate since the system manager is responsible for managing the use of the system. The system manager must make decisions between alternative jobs to which the computer resources may be applied. These decisions may be categorized into three separate but closely related areas.

1. Job submission--selection of which potential jobs to submit
2. Manual selection--selection of which submitted jobs to enter into the computer
3. Automatic resource allocation and scheduling--selection of which entered jobs to give service

Job Submission

Where the origin of the work to be performed is not under the control of the system manager, the interface between the system manager and users or originators of the work is one policy area. For example, a commercial service bureau does not have direct control over requests for computing services from its customers. At the other extreme the system manager may be the user and originate all of the work himself. A situation between these extremes results when an organization designates to one department of the organization responsibility for operating a computer and

providing service to one or more other departments. While the organization manages the computer for its own use there need to be policies governing the relation between the computer management department and user departments.

There are two important aspects of the manager-user relation for resource allocation and scheduling.

1. The user must communicate information to the computer manager which specifies the value of work relative to all other work.
2. The computer manager must provide to the user information which allows the user to decide under what circumstances potential computer applications are cost-effective and should be submitted.

Manual Selection

The second area of resource allocation and scheduling is the manual selection by the system management of the work to be done. The degree of control exercised here can vary greatly. It may involve the evaluation of each job's value and computer resource requirements individually or be implicit in a first-come-first-served policy.

Automatic Resource Allocation and Scheduling

The third area is the automatic resource allocation and scheduling performed by the computer operating system.

After a job has been entered into the computer, barring any manual intervention, all decisions with respect to resource allocation and scheduling will be made automatically by the computer system. The purpose of leading up to the internal computer resource allocation and scheduling algorithm in this way is to illustrate that these algorithms are really a part of the implementation of policies for the system managers. Hence, they ought to implement policies for the system manager which will maximize the value of the computer system.

VALUE OF RESPONSE TIMES

Each possible response set has a value to the system manager. The total value of the response set will depend on the values of the individual job response times. In most cases the values of the individual jobs will be independent and the total value will be the sum of the individual values.

Thus:

$$U=V_1+V_2+V_3+V_4+\ldots$$

where:

U = the total value of the response set

V_i = the value of the response time for job i

and

$$V_i = V_i(RT_i)$$

where:

RT_i=the response time of job i.

The value function described below will allow the system manager to specify a policy by which the system can determine the response time value function for any job. Using these response time value functions and the procedures described later, the system will attempt to allocate resources and schedule jobs so the response set will have a maximum value. The value function V is defined as follows:

V = V(class,size,time)

where:

class = the class of service

size = the size of the job

time = the elapsed runtime to complete the job

This function gives the value of completing a job in the specified class which has the specified resource requirements in the specified time. A more detailed discussion of each parameter follows.

Class of Service

To avoid evaluating each job separately, it is convenient to enable the system manager to specify policies which the operating system can use to categorize jobs into classes of service. Based on information available prior to the start of execution, a job can be assigned to a class of service. The purpose of the class of service designation is to specify groups of jobs to be treated in the same way.

The criterion used may be decided by the system manager. It may include items such as

1. the function of the job,
2. who submitted the job,
3. runtime estimated by user,
4. runtime estimated by system from previous executions of job,
5. maximum running time allowed by user,
6. resource requirements, number tape drives, amount core,
7. the price to be charged, and
8. a deadline after which the job is of no value.

In some circumstances the system manager's policy will not differentiate among jobs based on a priori knowledge and they would all be placed in the same class. In rare instances every job would be assigned to a separate class.

In many circumstances it makes sense to categorize jobs into discrete classes of service. Jobs which will require use of a tape drive are distinct from those that will not. In other situations, while there are differences, discrete classification may be too harsh. For example, while the estimated running time may be a desirable class determinant, ten and eleven minute jobs are not so different. The use of many class categories or, in the extreme, a continuous variable notion of class solves this problem.

The actual resources which the job will require are

almost never known before the job executes. Hence, the actual resource requirements cannot be a factor in assignment to a class of service. Only estimates and maximum limits are available before the job is run. The estimates may be supplied by the user or be derived by the system from previous execution of the job. The estimates may be very good or quite poor but are still only estimates.

Size of Job

The resources required by a job are an important characteristic the system manager may want to use for differentiation among jobs. Since the resource requirements are not known until the job is run these cannot be used to classify jobs into separate classes. However, the resource usage can be determined as the job runs.

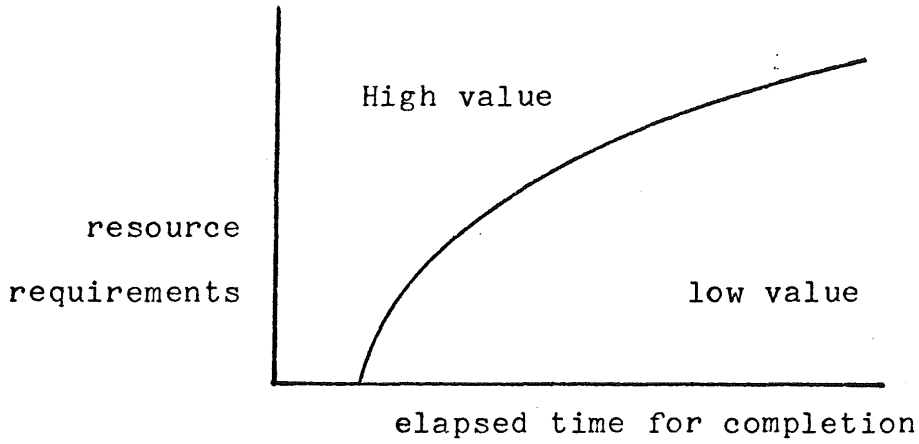
Response Time

As defined previously, response time is the elapsed time from submission of the request until it is completed.

Comparison with Simple Policy Functions

Response policy functions are a special case of value functions. This can be seen by considering for the moment only one class of service, a scalar measure of resource requirements and a single level of system load. If the

function has only two values so that, for each job size, jobs completed within a specified time have a high value and jobs completed in longer time have a low value, then the value function gives the same specification as a policy function. This becomes obvious when the resource requirements are plotted against the elapsed time and choice of value is indicated.



The loci of transition between the high value and low value are the policy curve. A system attempting to fulfill such a policy function would attempt to complete jobs in the high value region if possible; otherwise, in the low value region.

THE RESOURCE ALLOCATION AND SCHEDULING STRATEGY

The next topic is how resource allocation and scheduling can be performed to maximize the value functions. For the purposes of resource allocation and scheduling, each process appears to the system as a series of resource

requests and releases separated by periods of execution with the currently assigned resources. When a process requests a resource, the system must either assign the resource or suspend the process and assign the resource at a later time. When a resource is released, the system must return it to the pool of available resources. In addition, for those resources which are preemptable, the system should preempt the use of a resource and return it later if warranted by the system circumstances. Each opportunity to allocate or preempt a resource is a decision point.

Decisions are classified in the management science literature (Luce and Raiffa 1965, Morris 1964) as either decisions under certainty, risk, or uncertainty according to the following criteria.

1. Certainty--The outcome of each choice is known with certainty.
2. Risk--Some information about the likelihood of each possible outcome for each choice is known.
3. Uncertainty--No information about the likelihood of each possible outcome is available.

Resource allocation decisions can be made as decisions under assumed certainty by ignoring the possibility that future resource requirements may be different than the current requirements. With this short horizon viewpoint, the system can make decisions to maximize the current increase in value of the system. Following a discussion of

decisions under assumed certainty, resource allocations will be evaluated as decisions under risk to consider the effects of changes in requirements over a longer horizon.

DECISIONS UNDER ASSUMED CERTAINTY

To make resource allocation decisions under certainty it is necessary to consider:

1. the available alternative choices,
2. the outcome or state of the world resulting from each choice, and
3. the payoff or value of each outcome.

These data for a decision are sometimes tabulated:

Choice	Outcome	Payoff
alternative-1	state-1	value-1
alternative-2	state-2	value-2
alternative-3	state-3	value-3
alternative-4	state-4	value-4.

Then the choice with the highest payoff is selected. The next step is to look in more detail at the alternative choices, outcomes, and payoffs.

Alternative Choices

Decisions must be made on the allocation of resources. Whenever either a process requests a resource and the resource is available or when a resource is released and

there are one or more processes suspended waiting for that resource, a decision must be made on the allocation of that resource. Assignment of the resource to each of the requesting processes is a separate choice alternative. Another alternative is to assign it into holding and thus keep it available for more important use later. This may be necessary, for example, if it is necessary to wait for additional units of the resource to be released in order to accumulate the quantity of the resource requested by a process.

The other type of decision is the decision to preempt assignment of a resource. This is a choice to withdraw a previous assignment to a process or to holding and then reassign it. Preemption is really a special case of the assignment decision. In preemption the current holder of the resource is contending with requesters. Because there are usually costs associated with the the preemption, the cost of preemption must be considered in evaluating whether to continue the present assignment or to reassign the resource to another process.

While allocation decisions must be made only when a resource is requested or freed, there are no specific events which indicate a preemption or reallocation decision is in order.

Outcomes

The job which is assigned the resource it requested will make progress toward completion and possibly complete. Jobs which are not assigned the resource will remain suspended and not progress toward completion.

When a job completes, it will release the resources it is using. Even if it does not complete, its progress will bring it closer to completion and release of the resources it holds. Although the entire job may not be completed, completion of a step may result in the release of resources. Resources which are released will become available for use by other jobs (if needed). Jobs which are not assigned requested resources will be suspended and continue to hold resources assigned previously (unless preempted) and keep them unavailable for use by other processes.

Payoffs

Knowledge of the function

V(service class,request size,response time),

which gives the value of a completed job, allows calculation of the value of a partially completed job. Let

v(service class,s,t)

where

s = the size of the service received by the partially completed request and

t = the ET of the partially completed request

be the function which gives the value of a partially completed job. When the size of a request is not known in advance of processing the request, the next increment of service may complete the required processing. Since it is trivial to provide this increment of service, the value for a request which is just short of completion is approximately the same as the value of the completed request. Since this is true for requests of all sizes, the value function for a completed request, V, can be used to give the value, v, of a partially complete request.

An increase in the value of the system will result from the progress of the jobs assigned the resource. The rate of this increase will be

$$\sum_{\text{not suspended } p} \frac{\partial V}{\partial t} + \frac{\partial V}{\partial s}\frac{ds}{dt}.$$

There will be a decrease in the value of the system as a result of suspending other processors. The rate will be

$$\sum_{\text{Suspended } p} \frac{\partial V}{\partial t}.$$

Since the term $\frac{\partial V}{\partial t}$ appears in the payoff functions for both suspended and not suspended processes, these will be constant for all outcomes. Thus, for decision purposes only the

$$\sum_{\text{not suspended } p} \frac{\partial V}{\partial s} \frac{\partial s}{\partial t}$$

needs to be considered.

When there is more demand for a resource than the capacity of the resource, some of the requesting processes must be suspended. If additional increments of the resource were available, the productivity of the system could be improved by assigning the additional units of the resource to processes which would remain suspended otherwise. Thus the current opportunity value of a resource is the value of the best assignment of additional units of the resource. The opportunity value can also be considered a marginal value. By considering the value of placing suspended processes into execution, the opportunity value, OV_r, of each resource can be calculated:

$$OV_r = \max_{p,\ w_{prt}>0} \frac{\partial V}{\partial s} \frac{ds}{dt}.$$

The execution of a process which has been assigned a resource brings the process nearer to release of the resource. If a process is suspended without preempting use of the resource, the resource is not in productive use. Thus, in addition to the payoff associated with the increase in the value of the process, there is a payoff associated with earlier freeing of the resource. The value of this earlier freeing is the marginal or opportunity value of the

resource.

It is necessary to be cautious in summing up the rate of change in the value of the progress of a process and the marginal values of the resources it holds to determine the total value of putting the process into execution. This opportunity cost concept suffers from the limited range of applicability of all measurements made at the margin. The opportunity values only approximate the value of freeing the resource. Except when there is only one unit of a resource (e.g., a disk or permission to access a file) the following circumstances can arise. If the requesting process needs more of the resource than will be released, it will still remain suspended. If a large quantity of the resource is released, several processes with differing rates of progress may be able to run.

This definition of the opportunity value allots the full value of the progress for each process to every needed resource. If a suspended process needs more than one of the resources held by a process, attaching the potential rate of increase in value of the suspended process to each resource and then adding the opportunity values together will double count the value of getting the process into execution. It might be appropriate to treat the resource valuation as a shared cost problem and allocate the costs among the required resources. The other possibility is to allocate the full value to each resource since lack of that resource

would prevent any progress. To avoid double counting resource requirements, a process can be valued at the maximum of the opportunity values. This will, in many cases, understate the value of placing into execution a process which has already been assigned needed resources.

Resource Specific Factors

The previous discussions considered the system level factors which affect resource assignment decisions. In addition, there are considerations unique to specific resources. For example, in scheduling use of a moving head disk it can be advantageous to schedule the disk based on the disk locations to be accessed in order to minimize the time wasted in arm movement. The possible increased utilization of the disk must be weighted against the system level factors. Techniques specific to individual resources can be important when evaluated in the context of the system values.

Summary of Decisions under Certainty

To make a scheduler decision under assumed certainty it is necessary to consider the possible resource allocations and the short-term outcomes implied by each choice. In evaluating the payoff of each outcome it is necessary to consider the rate of increase in value of the processes

which do not need to be suspended and the value of their freeing earlier the resources they hold. The rate of increase of the value of a process can be determined from the rate of change with service in the value function supplied by the system manager for the service class, service received, and elapsed time. The value of freeing resources held can be determined from the rates of increase in value for the processes waiting for the resources held. A decision should be made from among the choices which maximizes the rate of increase in the value of the jobs.

DECISIONS UNDER RISK

Resource allocation decisions can be evaluated more precisely by treating them as decisions under risk. In this framework the possibilities of various future events affecting the outcomes can be considered. The outcome of the allocation choice is not known with certainty, but some information--based on past history--is available about the likelihood of various outcomes. These decisions are frequently represented in a table with each alternative choice in a row. Each possible outcome is placed in a column with the probability of the outcome. Each entry in the table is the payoff value for the choice (row) and outcome (column).

	outcome-1 probability-1	outcome-2 probability-2	outcome-3 probability-3
choice-1	payoff(1,1)	payoff(1,2)	payoff(1,3)
choice-2	payoff(2,1)	payoff(2,2)	payoff(2,3)
choice-3	payoff(3,1)	payoff(3,2)	payoff(3,3)
choice-4	payoff(4,1)	payoff(4,2)	payoff(4,3)

The usual approach to making decisions under risk is based on maximizing the expected utility for the outcomes of a choice. Utility is a measure of the value of a payoff. The expected utility for a choice is the sum of the utilities of the possible outcomes weighted by their probabilities. When these decisions are made by selecting the choice with the highest expected value, in the long run the average utility of the outcomes of the decisions will be maximized. Thus if the value function specified by the system manager is used as the utility measurement in evaluating decision choices, over a long interval the resource allocation will maximize the value of the system to the manager.

Since similar decisions are made frequently, the goal is long-term system performance, and without information about the future on which better decisions can be made, viewing the choice in this framework is appropriate. This implies choosing at each decision point the alternative with the highest total of value potential outcomes weighted by the estimated probabilities of the outcomes. The complexity of the system is the basis for assuming randomness among the

outcomes consequent of any decision. The frequency of the decisions is the basis of expecting the high system performance goal to be satisfied in the long run. The next step is to look in more detail at the probabilities of each outcome with each choice.

Probabilities of Outcomes

To make decisions which maximize expected utility, a priori estimates of the probabilities of each outcome are needed. Processes which are suspended will, with certainty, make no progress. Thus a probability of 1.00 can be attached to this aspect of the outcome.

Estimating what will happen to a process which is assigned the resources it requires is harder. In particular it is necessary to know the chances in the next interval that it will release resources, request more resources, or just continue execution. Although these events are deterministically specified in the programs, Bayesian probabilities can be attached to them. In this way decisions under uncertainty are treated as decisions under risk. There are several approaches which can be used to estimate the probabilities.

Estimates and limits provided by the user can be a rough indication of how soon a job or step of a job will end. Given the amount of processing complete and the estimated amount, the likelihood of completion in the next

interval can be estimated. Means for users to indicate their expectations in more detail could be provided.

For example, data on the expected duration of the requirement could be provided with every resource request. In some cases compilers or loaders may be able to extract information about the programs they process which will be of use when the programs are run.

User supplied estimates and limits can have a number of biases which will affect their usefulness for resource allocation and scheduling. Being estimates, their accuracy depends not only on the type of job but the skill of the estimator. In addition, the way in which the estimates are used by the system may influence users to purposely bias their estimates. For example, if an estimate is used to determine the class of service for a job this may influence him to make an estimate which causes the job to be assigned to a preferred service class. It is important that such biases be taken into account or eliminated by effective penalties or sanctions.

Another approach to developing probability estimates about a job's future behavior is to infer them from characteristics of the job. The rules for characterizing the jobs and then developing estimates can be based on the management's knowledge about the jobs or measurements made on typical jobs or both. To facilitate this, the probability estimating code for resource allocation and

scheduling can be based on tables specifiable by the system manager.

Statistics on job behavior can be automatically collected by the system and the probability estimating mechanisms altered dynamically. This eliminates the need for intervention by the system. It will also make the system more adaptable to the operation of the specific types of jobs processed by the installation.

In some cases it may be possible to make better probability estimates by looking at the history of the particular job rather than only aggregates of jobs having similar characteristics. If, as a job executes, it becomes apparent that the job is behaving differently than other jobs with similar characteristics, then the probability estimates should be revised to take the specific behavior into account. Data on the behavior of programs could be kept in a file to aid prediction of the behavior of later executions with different data. This of course requires a means of identifying successive executions of the same program (e.g., job name, program name, program location, etc.).

The opportunity values needed to evaluate payoffs and make decisions should be estimates of the opportunity values in the future rather than the values of the best current assignment. These estimates can be derived on the basis of the recent history of the best assignment of additional

units of the resource. While the current best assignment of additional units of the resource may fluctuate through a wide range, a statistic based on recent history will be a much better estimate of the future marginal value. To provide adaptability to changing system loads and still maintain stability, the use of a statistic which exponentially decays the weight of past best assignments is appropriate. The exponentially decayed statistic will be relatively unaffected by unusual and one-time requests for the resource; yet, it will adapt to prolonged changes in system load. The rate of decay will determine the speed of adaptation and sensitivity to unusual requests. It must be chosen to operate at a satisfactory tradeoff point to give adaptability and still be insensitive to one-time demands on the resource.

Summary of Decisions Under Risk

Unlike decisions under assumed certainty where a single outcome for each choice is considered, a decision under risk allows for several possible outcomes with each choice alternative. Based on past behavior of the program and other programs, probabilities can be attached to each of the outcomes for a choice. By summing the rates of increase in system value for each outcome weighted by the probabilities, the expected value of each choice can be computed. The choice which will give the highest mathematical expectation

can then be selected.

THE FLEXIBILITY OF VALUE FUNCTIONS

System implementors have designed numerous scheduling algorithms to satisfy diverse requirements. Most of these algorithms focus on particular situations, use a limited set of goals, and are difficult to change. Although these traditional algorithms vary in their details, they are based on a few scheduling goals.

With appropriate value functions, a value-based scheduler can approximate these goals. Thus, the same basic scheduler can be used in quite diverse situations. Only the value functions must be tailored to the specific circumstances. If the requirements change, the value functions can be changed. The following sections discuss traditional scheduling principles and the implementation of these goals with value functions. These principles can be used individually or jointly in specifying the manager's utility function.

Equal Service

Scheduling to give each job an equal rate of service is frequently used in systems where the resource requirements and urgencies of the jobs are similar. The service rate is frequently based only on use of the central processor but

sometimes other resources are considered. Round robin scheduling is the standard way of providing equal service.

Basing the value function on the ratio of the service received to elapsed time implements this goal. Value functions which attach a higher value of execution to jobs with lower ratios will cause all jobs to receive equal service. This type of value function is demonstrated in Chapter 5.

Declining Rate of Service

To bias a scheduler toward short jobs when the sizes of the jobs are not known, a declining rate of service is implemented. All jobs start with equal priority but after a job has received service, its priority is gradually or abruptly reduced below that of new requests. Thus, large jobs only run for a short time at high priority and then run in the background at lower priority than new requests. This is implemented in a traditional scheduler through multiple queues or adjustments to the dispatching priority. An abrupt change in the rate of service results from a value function which is discontinuous with respect to time. Chapter 5 illustrates a discontinuity after 1200 milliseconds of virtual time have elapsed. Continuous dependence on time can produce more gradual declines in the rate of service.

Classes of Service

As discussed earlier in this chapter, the value functions for all jobs do not need to be identical. Jobs can be classified into discrete classes or on a continuum. The value function for each class can have a very different form with consequent different response characteristics. The following properties can be implemented with appropriate value functions and class assignments.

Priority

Strict priority schedulers always assign requested resources to the highest priority requesting task. Value functions which depend only on the class of the job can implement this type of resource assignment. If required, priority changes can be affected by reassigning jobs to new classes.

Interactive/Batch Distinction

Biases between interactive terminal requests and batch requests can be implemented by assigning these jobs to different classes. Appropriate value functions permit varying the amount of bias toward either type of service. Even strict priority for terminal requests is possible.

Flexible Pricing

In some contexts it is desirable to base the rate of service a job receives on the price the user is willing to pay. Assignment to a class of service on the basis of the price the user is willing to pay makes this possible. Depending on the particular circumstances, this could be a flat payment or a surcharge on other charges for service.

Service Guarantees

Specific rates of service and consequent response times for jobs can be guaranteed to users through classes of service that have a sufficiently high value of execution to attain these rates of service. As with all guarantees of service, it is necessary to avoid over committing the resources available.

Chapter 5

EXPERIMENTAL INVESTIGATIONS

A simple scheduler incorporating value-based resource allocation under assumed certainty principles was simulated to learn more about its behavior. The development and experimentation with this example of value-based resource allocation was conducted for three main purposes. First, the simulation demonstrates the feasibility of the value-based resource allocation. Second, specific aspects of the allocation system have been explored. The third purpose is to allow comparisons of the strategy with a conventional scheduler.

Feasibility Demonstration

A primary constraint on any scheduler is that it not have major defects which result in degenerate, obviously unacceptable behavior. Possible signs of degeneracy are low priority processes receiving more service than high priority processes, some jobs receiving no service or very little service for no good reason. Thrashing and deadlocking are also unacceptable. The algorithm should not be too complex. Since execution of the scheduling program requires use of

computing services, if the calculations performed by the scheduler are too complex, scheduling will consume an unacceptable portion of the systems resources.

Exploration of Mechanisms

The experimental resource allocation mechanisms have been explored by varying the operation of the mechanisms and the environment.

1. Value functions. Varying the value functions has allowed observation of the effect of different value functions.
2. Job mix. Varying the job mix has allowed exploration of adaptability to different types of system load.
3. Resource valuation mechanism. The effect of the resource valuation mechanism has been observed by removing it from the scheduler in some of the experiments.

Comparison with Other Strategies

Comparison of the value-based resource allocation philosophy with traditional approaches to scheduling was the third goal of the experimentation. To do this, a simulation model of a traditionally organized scheduler was implemented so both the value-based and traditional schedulers could be

compared in their ability to allocate exactly the same resources to identical job mixes. The simulation of traditional resource allocation is patterned after the scheduler in the Universal Time-Sharing System (UTS) for the Xerox Sigma 6/7/9 computers (Xerox, January 1972). This is a multilevel queue structured scheduler which has undergone considerable refinement to make it a high quality example of modern general purpose timesharing system schedulers.

Simulation as the Vehicle for Experimentation

Simulation was chosen as the vehicle for experimentation to avoid several problems which would have arisen in experimenting with an actual operating system. Most of the advantages to simulation can be classified as either greater control over operation or ease of implementation.

1. The technique of scripting used in the simulation allows a set of jobs to be defined to have desired resource requirements in advance and then used for successive experiments. Experimentation in the context of a complete operating system would make control of the environment in which the scheduler operates as well as control of the job mix much more difficult.
2. Coding the important aspects of a scheduler for simulation is considerably simpler than coding the

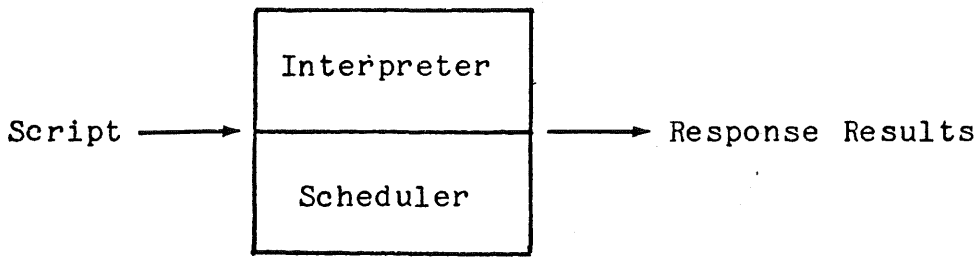
entire scheduler since many housekeeping details which are irrelevant to performance evaluation can be omitted from the simulation. For example, while the saving and loading of the central processor's status and register contents is important when changing the assignment of the central processor in a real scheduler, these activities are irrelevant to evaluating the performance of the scheduler. They can be neglected in a simulation designed to evaluate performance.

3. Provisions for measuring and recording the activities and aspects of performance most relevant to the experiments were easily incorporated into the simulation without interfering with the operation of the simulated system. Making similar measurements on a real system would require considerable care to avoid affecting operation of the system.
4. By using simulation, the need for dedicated use of a computer during the debugging and testing of the experimental scheduler was avoided.
5. Possibly the most important advantage to the use of simulation was freedom in choice of a language in which to code the schedulers.

Structure of the Simulator

The operation of the simulator is based on the formulation of the resource allocation problem in Chapter 3. Jobs are modeled as sequences of requests and releases of system resources. The states of the processes are maintained and their progress measured on the basis of the process structure description. The resource allocation notation, which encompasses the most important variables of the simulation program, describes the system state. The scheduler algorithms, including one which is based on the value-based discussions in Chapter 4, implement different resource allocation decision rules.

The simulator consists of program scripts, a script interpreter, the scheduler to be investigated, and the simulation housekeeper and statistics recorder. A script is the input to the simulation system consisting of the script interpreter and scheduler. The outputs of the simulation are the response results.



To study the effect of the different job mixes new scripts can be written and the other parts left unchanged. To compare different scheduling algorithms, just the

scheduler portion can be replaced.

Program Scripts

The program scripts specify the system load or job mix to be synthesized for the simulation. A simple example of a script will illustrate the basic features of the script definition.

```
        4  Class-1 processes.
        Request 20 pages memory.
Alpha:  Request 1 central processor.
        Run 100 milliseconds.
        Release 1 central processor.
        Think-type 5000 milliseconds.
        Go to Alpha.
```

The first line declares that 4 processes are to execute the script that follows. They are to be scheduled in the first class of service. The second and third lines indicate requests for 20 pages of memory and then use of the CPU. The third line is labeled to be used as a target for a program loop. The specification to run for 100 milliseconds in the fourth line indicates the interval of time that resources are required by the process after they have been assigned. Following this interval the CPU will be released. Five seconds of thinking and typing by the terminal user will ensue. In addition to the passage of time this step is significant in that it signals the beginning of a new

request for service. The last step indicates a return to line 3.

Typically, a script will consist of not just a single program but several programs with varying numbers of processes executing each. Time intervals may be specified as draws from a random distribution of specified parameters. Other resources may also be represented. In addition to the CPU and main memory, these simulations considered disk I/O devices, permission to open/close files, and a high speed swapping channel. The scripts for all experiments are in Appendix I.

Script Interpreter

The script interpreter utilizes the script definitions to simulate process behavior. For each simulated process its progress in completing each step of its script is maintained. Requests for resources are passed to the scheduler for decisions on allocations. Resources which are released are returned to the reserve of resources for assignment to another process by the scheduler.

Scheduler

The scheduler makes decisions on allocations and preemptions based on new requests recognized by the script interpreter, pending requests, previous assignments, and the

total capacity of resources available. When requests are not fulfilled or resources are preempted, the processes needing them must be suspended. The resources consumed in scheduling (scheduler overhead) have been ignored in the simulation on the assumption that all of the schedulers investigated are of similar complexity. Observation of simulation runtimes substantiate this assumption.

The schedulers which were simulated varied in the algorithms used. However, the data available and data structures are the same for each. The data base, kept in APL arrays, contains all of the data needed by any of the schedulers; none of them uses all of the data. More detailed descriptions of the schedulers for the specific scheduling policies will be given later.

Housekeeping and Performance Measurement

The final part of the simulation is maintenance of housekeeping details such as the simulation clock, time of the next event, etc., as well as the recording of statistics for analyzing the performance of the scheduler. To evaluate the response times which would result in a system utilizing the algorithm being simulated, the progress during the interval simulated of the simulated jobs must be measured. Response times could not be measured directly since most of the jobs would not finish during the relatively short simulation interval (60 seconds).

Two types of progress measurements are important. The relative rate of progress for a process is the virtual execution time divided by the real time. Thus the relative rate of progress is the ratio of the progress the process actually made to the progress it could have made if it were the only process in the system and did not have to compete with other jobs for use of the system resources. The total of the individual rates of progress of the processes in a system can be considerably greater than unity since there will be periods while terminal users are thinking or typing and the process will not require service. Also, to the extent that the processes of the resource requirements are complementary, more than one process may be making progress at a time.

A second type of important information collected from the simulation is the percentage of the total available quantity of each resource over time assigned to each process. Most schedulers do not assign the CPU to a process unless it has all of the other resources it needs to make progress. Hence, for a fixed interval of simulation the percentage of time the process is assigned the CPU is a measure of its absolute progress. The percentage of the time other resources are assigned do not necessarily reflect useful assignments since a process may be holding a resource unproductively while it waits for other resources it needs.

Other types of performance statistics which are

frequently examined have not been studied. The interval from receipt of a request until the process has all the resources it needs to begin execution on the request is sometimes measured as an indication of the responsiveness of an operating system. This is useful for tuning particular operating systems but is less useful for comparing different schedulers. As pointed out previously, the performance users see is the interval between the request and the end of processing the request.

The measurements of rate of progress and CPU usage over the entire simulation period reflect the average service received by the process. The variance of the service rate for individual requests is also of interest. Direct measurements of the variances have not been made. However, the variances can be estimated. Demonstrations of the ability of experimental scheduler to allocate accurately on the basis of value functions imply a small variance in performance for all but small requests. For small requests, length of the initial interval until the process has the resources it needs determines this variance. Since this is the delay which impedes the progress for the process, it is reflected in the rate of progress statistic.

Simulated Schedulers

Experiments were actually conducted on four types of schedulers. The first two experimental schedulers combined

a global resource allocation scheme with the simple policy functions developed by Bernstein and Sharp. Difficulty in designing reasonable policies for these schedulers provided the impetus for development of the concept of value-based resource allocation. Following development of this concept, the experimental scheduler based on these principles was written. The other scheduler simulated for comparison is the traditional multilevel queue scheduler. Because the limitations of the first two experimental schedulers resulted in their rejection as less powerful and flexible, the following discussion will concentrate on the experimental value-based example and the traditional multilevel queue scheduler.

Experimental Value-Based Example

The experimental value-based scheduler evaluates resource assignments in the system every time a process requests a resource, releases a resource, or after a period of running when no resources have been requested or released. The first step in evaluating the assignments is the determination of the current value of running each process from the value functions supplied by the system manager. The value functions used in these experiments are given in Table 2. More detailed descriptions of the various value functions used in each of the experiments is included with the descriptions of the particular experiments. Each

Table 2

Experimental Value Functions

I. Identical Value Functions--Equal Service

Value of executing job

$$= \frac{\text{real time}}{\text{CPU assigned time}}$$

II. Two Classes of Service

Value of executing job

$$= \frac{2 \cdot \text{real time}}{\text{CPU assigned time}} \quad \text{if class 1 job}$$

$$= \frac{\text{real time}}{\text{CPU assigned time}} \quad \text{if class 2 job}$$

III. Activity-Biased Function--Multilevel Queue Approximation

Value of executing job

$$= 1000 + \frac{\text{real time}}{\text{CPU assigned time}} \quad \text{if CPU assigned time} < 1200 \text{ ms.}$$

$$= \frac{\text{real time}}{\text{CPU assigned time}} \quad \text{otherwise}$$

Table 3

Experimental Value-Based Scheduler Algorithm

(Appendix III contains a more detailed description of this algorithm.)

Executed every time a resource is requested, a resource is freed, or no resource has been requested or freed for one quantum.

V1. Calculate value of execution of each process. This is rate of increase in value function with respect to service (job size) as function of job class, service received, and elapsed time.

V2. Adjust values by value of resources held. Values of resources held are determined from the values of processes waiting for these resources.

V3. Sort processes by value.

V4. Assign free and preemptable resources to requesting processes in descending order of value.

V5. If the swapping channel is not still free, go to V8.

V6. Select suspended process with lowest value which has not been copied out of main memory. If none, select a running process when there is need for memory which exceeds the loss from suspending the running process. If there is nothing to swap out, go to V8.

V7. Initiate copy of selected process out of memory.

V8. Exit to job processing.

of the values is then adjusted to take into account the additional value attributable to the possibility that the process will release resources it holds and make them available to other processes which are suspended waiting for the resources. In the experimental scheduler the adjusted value has been calculated as the maximum of the value of running the process and the value of running any process which is waiting for a non-preemptable resource the process holds.

A set of resource allocations then needs to be found to maximize the sum of the adjusted values of the processes assigned all the resources they require. A procedure for obtaining an allocation which approximates this is to allocate to one process at a time in order of descending adjusted value. In most cases this will be almost as good. If only part of the requests for a process can be satisfied, these resources are placed in holding until the rest become available.

If a resource can be preempted from a process with a lower adjusted value, this is done. The central processor is always preemptable. Pages in main memory are preemptable only if there is a current copy of their contents on the swapping device. After a process has had its pages preempted, the swapping channel resource is required (in addition to main memory page and other requirements) to copy the information back in. When not in use to copy

information into main memory, the swapping channel is used to create current copies of pages on the swapping device starting with the process having the lowest adjusted value for running. Table 3 outlines the steps of the algorithm.

Consider, for example, three jobs with the following resource assignments and wants.

	Assigned	Want
Job 1	memory, open	CPU
Job 2	memory	CPU
Job 3		memory, swap channel open, CPU

The scheduler will first calculate the value of running each job from the value function specified by the system manager. Assume these to be 2, 4, and 6 for jobs 1, 2, and 3 respectively. Since job 1 holds the nonpreemptable permission to open files and this resource is needed by job 3, job 1 will have an adjusted running value of 6. Therefore, it will be allocated use of the CPU. If job 2 is not already copied out, the swap channel will be assigned to copy it out. In this example, holding permission to open files causes job 1 to receive extra service until it releases this critical resource. Following the release of the critical resource, it will receive a less than normal share of service until its average rate of service declines and its value of execution rises as high as the values of execution for other jobs in the system.

Multilevel Queue Scheduler

The multilevel queue scheduler which was implemented for comparison with the value-based example is considerably more complicated and contains the essential details of the UTS scheduler. However, since it was intended only to be representative of the class of high quality multilevel schedulers and not just UTS, it was not statistically validated against the UTS scheduler. An informal examination showed its performance with various job mixes to be consistent with UTS performance. Only an overview of its operation will be given here. Additional details are contained in Appendix IV and the UTS system documentation.

The basic strategy for a multilevel queue scheduler is to enter jobs into first-in-first-out queues based on the occurrence of events relevant to the scheduling activity. In general, all processes which have just received a request from a user terminal are placed in a queue with first priority for use of the CPU. Processes which become unblocked because their terminal output buffer emptied have next priority and processes which have just completed other types of I/O have the next priority. Lowest priority is given to jobs which have already had a quantum of service (typically 500 to 2000 milliseconds CPU usage) since the last request was made. A process which needs the CPU but has been swapped out will be swapped in if jobs not currently needing the CPU or lower priority jobs also

waiting for the CPU can be found for it to replace. Exceptions to the basic rules are made in special circumstances such as keeping jobs waiting to do I/O in memory and reducing the likelihood that a job with exclusive permission to open or close a file will be swapped out. Table 4 outlines the steps of this algorithm.

VIABILITY OF SCHEDULING USING VALUE FUNCTIONS

In the initial tests of the experimental scheduler, identical value functions were used for all jobs so each job would receive an approximately equal share of central processor time.

The value function has been defined:

$$\text{Value of executing job} = \frac{\text{real time}}{\text{CPU assigned time}}.$$

Jobs which have been in the system longer than average for the amount of CPU assigned have higher than average value of execution and will receive better than average service. Jobs which have received more than average use of the CPU for the time they have been in the system will have lower value and receive less service. Thus the value function should cause all jobs to receive approximately equal use of the CPU. A special problem occurs in computing this function immediately following a user request when the process has had no CPU assigned time and the value of the

Table 4

Multilevel Queue Scheduler Algorithm

(Appendix IV contains a more detailed description of this algorithm.)

Executed on the occurrence of events associated with requesting or freeing specific resources and completion of one quantum of CPU use.

Q1. Enter processes at appropriate queue level on the basis of the event which has occurred.

Q2. If the swapping channel is not free, go to Q7.

Q3. Search down queue for first process which needs to be swapped into main memory. If none, go to Q7.

Q4. If the needed pages are free initiate the swap in and go to Q7.

Q5. Search up the bottom of the queue for a process or processes which can be swapped out of memory to make enough room. If none, go to Q7.

Q6. Initiate swap out and swap in.

Q7. Select the first process in core from the top of the queue and allocate the CPU and other resources required.

Q8. Exit to job processing.

function is infinite. Setting a small minimum value to be used as the CPU assigned time avoids this initial problem while still causing the execution value to be relatively high.

Each of the simulation runs reported in the following is based on 60 seconds of simulated time. The figures include data gathered during starting transients before the system reached a steady state. While start up transients should be eliminated from the results of most simulations, they are of interest in some of the following cases. Except as specifically noted, the start up transients last approximately two to ten seconds and do not significantly affect the overall measurements.

Single Value Functions

The results from running several identical CPU bound jobs with the experimental scheduler are shown in Table 5. Column three, percentage of CPU usage, which is a measure of the absolute progress of each job, shows each job received approximately the same share of CPU use and thus achieved the same progress. The last column shows for each job the rate of progress as a fraction of the rate of progress which would have resulted if the job were the only job in the system. As expected, since the resource requirements are not complementary, the total rounds off to unity with each job receiving an equal share. The fourth column (%Page)

shows the average percentage of the total available pages assigned to the jobs.

A slightly more complicated situation results when jobs doing I/O are also included. Table 6 summarizes the results of a job mix with four compute bound jobs and also four jobs doing I/O. The third column shows that, as specified by the value function, each job receives equal use of the CPU. However, because of the complementary requirements for resources, the total rate of progress for all jobs exceeds one.

In the last test of this value function, which causes each job to receive an equal share of the CPU, several interactive jobs which request and receive input from terminals are also included. During the time that one of these processes is requesting terminal input, it is not in contention for use of the CPU. Receipt of the input constitutes a new request for service and the job begins contention on par with all other jobs. Hence, the total percentage of CPU usage by the interactive jobs is different for each job and is less than that of non-interactive jobs. This is shown in Table 7.

Value Functions Giving Class Preference

We now examine the consequences of using value functions which do not simply cause the CPU usage to be distributed equally among the processes. First, the effect

Table 5

Value-Based Scheduler with Identical Value Functions
Value Function I
Script A

Job	Type	%CPU	%Page	Progress
1	CPU bound	12.4	7.5	.125
2	CPU bound	12.5	7.8	.125
3	CPU bound	12.5	7.8	.125
4	CPU bound	12.5	7.6	.125
5	CPU bound	12.5	7.7	.125
6	CPU bound	12.5	7.9	.125
7	CPU bound	12.5	7.8	.125
8	CPU bound	12.5	6.8	.125
Total		99.9	61.0	1.000

Table 6

Value-Based Scheduler with Identical Value Functions
Value Function I
Script B

Job	Type	%CPU	%Page	%Disk	Progress
1	CPU bound	11.8	6.2		.118
2	CPU bound	11.9	6.1		.119
3	CPU bound	11.9	6.6		.119
4	CPU bound	11.8	9.2		.118
5	I/O	11.8	9.2	11.1	.229
6	I/O	11.9	9.3	11.5	.234
7	I/O	11.8	8.7	11.9	.237
8	I/O	11.8	8.8	12.0	.238
Total		94.7	61.8	46.6	1.416

Table 7

Value-Based Scheduler with Identical Value Functions
Value Function I
Script C

Job	Type	%CPU	%Page	%Disk	Progress
1	CPU bound	11.2	6.1		.112
2	CPU bound	11.2	6.2		.112
3	CPU bound	11.2	6.2		.112
4	CPU bound	11.3	6.1		.113
5	I/O	11.2	8.6	11.3	.226
6	I/O	11.2	8.4	11.3	.224
7	I/O	11.2	8.3	11.0	.223
8	I/O	11.2	8.5	11.4	.227
9	Interactive	1.8	1.5		.125
10	Interactive	2.2	1.9		.138
Total		93.7	61.8	45.1	1.612

of changes in the value functions to implement different classes of service will be investigated. Half of the jobs (the first half) were arbitrarily assigned to a class of service with a value function designed to cause them to receive twice as much use of the CPU:

$$\text{Value of executing of job} = \frac{2 \cdot \text{real time}}{\text{CPU assigned time}}.$$

A summary of the results of these two classes of service with all CPU bound jobs is shown in Table 8. Similarly, a mix of CPU bound jobs and jobs with I/O are shown in Table 9. In both cases about twice as much service is received by those jobs with value functions designed to result in twice the CPU usage of the other jobs.

Activity-Biased Value Functions

In large timesharing systems it is usually desirable to give better service to processes associated with interactive terminals than batch jobs. When fulfilling each request from a terminal requires only a short period of resource usage, preference can be given to these requests with relatively little effect on service to longer batch jobs. Terminal users are thus given very quick response to their small requests. However, long terminal requests cannot be shown the same amount of preference without degrading the response to small requests. The usual solution to this dilemma in multilevel queue schedulers is to place processes

Table 8

Value-Based Scheduler with Two Classes of Service
Value Function II
Script D

Job	Type	Class	%CPU	%Page	Progress
1	CPU bound	1	16.6	9.7	.166
2	CPU bound	1	16.6	9.0	.166
3	CPU bound	1	16.6	9.1	.166
4	CPU bound	1	16.7	8.6	.167
5	CPU bound	2	8.4	6.5	.084
6	CPU bound	2	8.3	6.2	.084
7	CPU bound	2	8.3	6.4	.083
8	CPU bound	2	8.4	5.5	.084
Total			100.0	61.0	1.000

Table 9

Value-Based Scheduler with Two Classes of Service
Value Function II
Script E

Job	Type	Class	%CPU	%Page	%Disk	Progress
1	CPU bound	1	15.8	8.2		.158
2	CPU bound	1	15.8	7.8		.158
3	I/O	1	15.8	11.4	15.3	.311
4	I/O	1	15.8	11.6	7.5	.320
5	CPU bound	2	7.9	4.7		.079
6	CPU bound	2	8.0	4.8		.080
7	I/O	2	8.0	6.5	16.1	.155
8	I/O	2	7.9	6.7	7.6	.155
Total			95.0	61.8	46.5	1.415

in a high priority queue at the beginning of a request. If the request cannot be completed in a fixed time period (quantum), the process is placed in a lower priority queue.

In the model of the multilevel queue scheduler the quantum has been set at 1200 milliseconds. The approximate effect of this is to give all requests which have received less than 1200 milliseconds first priority and other jobs second priority. The value function of the value-based scheduler can also be designed to give higher priority to jobs which have not yet received one quantum of CPU service by assigning a higher value to execution of these jobs. Since 1000 is a large number relative to normal values for executing jobs, a function with a discontinuity after one quantum of CPU usage can be formed:

Value of executing job

$$= 1000 + \frac{\text{real time}}{\text{CPU assigned time}} \quad \text{if CPU assigned time} < 1200 \text{ ms}$$

$$= \frac{\text{real time}}{\text{CPU assigned time}} \quad \text{otherwise.}$$

This specifies a value function which approximates the value function implicit in the multilevel queue scheduler.

Tables 10 and 11 contrast the effect of value functions without initial higher execution value (Table 10) and with initial high execution value (Table 11) on the same job mix. The rate of progress in the last column for interactive processes is greater in Table 11 where the initial service was given a higher value. Except during the initial startup

period, where none of the processes had received a quantum of CPU usage and all had similar values for execution, the interactive jobs had much higher values of execution and were given preference when they needed use of the CPU.

These experiments and others not reported confirm the ability of a simple scheduler to use value functions in performing resource allocations. The value functions can be changed easily, without rewriting the scheduler, to implement different policies for the system management.

GLOBAL RESOURCE ALLOCATION

Value functions and global or centralized allocation of resources were explored together because the preliminary analysis indicated they were improvements in resource allocation which have the potential to work well together. It is possible to conceive of a traditional scheduler which schedules use of the CPU using priorities based on value functions while other resources are allocated on another basis such as first-come-first-served. Also, resource allocation could be centralized without being based on value functions. Global resource allocation in the IBM OS/370 VS2 Release 2 system is described by Scherr (1970b). However, some conclusions can be drawn from the simulation experiments about global allocation in the context of value-based scheduling.

Discussion of each item will follow a listing of the

Table 10

Value-Based Scheduler without Interactive Bias
Value Function I
Script F

Job	Type	%CPU	%Page	Progress
1	CPU bound	22.5	13.3	.225
2	CPU bound	22.5	14.0	.225
3	CPU bound	22.5	12.8	.225
4	CPU bound	22.4	13.4	.224
5	Interactive	2.7	2.3	.218
6	Interactive	2.5	2.0	.215
7	Interactive	2.3	2.0	.213
8	Interactive	2.5	2.0	.214
Total		99.9	61.8	1.759

Table 11

Value-Based Scheduler with Interactive Bias
Value Function III
Script F

Job	Type	%CPU	%Page	Progress
1	CPU bound	22.9	14.1	.229
2	CPU bound	23.1	13.4	.231
3	CPU bound	22.5	15.4	.225
4	CPU bound	22.7	15.1	.227
5	Interactive	2.3	1.1	.483
6	Interactive	2.2	1.0	.468
7	Interactive	1.8	.9	.474
8	Interactive	2.3	.9	.698
Total		99.9	61.8	3.035

points. First, simple global resource allocation strategies can be used with value functions to do scheduling and resource allocation. Second, the problems of allocating all system resources including CPU's, memory, and I/O devices are basically the same and can be handled in a single framework. Third, global resource allocation can simplify allocations of the various resources to avoid wasteful, inconsistent allocations.

Value-Based Allocation

The quality of local resource allocation policies to implement value-based allocation has not been evaluated. However, the simulation experiments show global allocation can be used satisfactorily for value-based scheduling. The results described in the previous sections using the simple experimental scheduler guided by value functions and doing centralized resource allocation demonstrate the tractability of global resource allocation in a value-based context.

Uniform Treatment of Resources

The operation of the experimental scheduler also shows the feasibility of considering the CPU, memory, I/O devices, and all other resources in the same framework. With the exception of the use of the swapping channel to make copies of main memory pages onto the swapping device when it would

otherwise be idle, the experimental scheduler uses the same algorithms and code to allocate all resources. Tables were used to indicate capacity of the resources, whether they were preemptable, etc. While the use of exactly the same algorithms does work, it is probably extreme. To obtain the most efficient utilization of individual resources, a system should probably be structured so specialized policies can be utilized within a global framework.

Consistent Resource Allocations

Intuitively, the use of centralized resource allocation where the scheduler can coordinate the use of all the resources should result in better use of the resources than resource allocation decisions made separately and independently. For example, it makes little sense to allocate part of the resources requested by a process if other resources it needs are not available. Similarily, if a process already holds preemptable resources it is frequently desirable to take them away while the process waits for new requests to be filled.

The consideration given to the value of the resources held by a process in adjusting the value of the execution of that process is a global allocation policy whose benefit can easily be demonstrated. Although not a calculation of the true expected value of releasing the resources, this adjustment will give preference to processes with inherently

low value while they hold resources needed by high value processes so that the resources will be released and become available to the high value processes. Experiments where this provision of the scheduler was eliminated resulted in situations with low value processes holding non-preemptable resources (exclusive right to open files) but not having enough value to cause preemption of the memory held by higher value processes waiting for permission to open files. Since the lower value process could not get the memory it needed to continue execution and release the right to open files, deadlock resulted. With the adjustment of a process's value to take into account the resources it holds, this process would temporarily have enough value to cause preemption and would be assigned the memory it needs.

EFFICIENCY OF VALUE-BASED RESOURCE ALLOCATION

The different circumstances for which schedulers were designed and the ways in which they operate make evaluation of their performances difficult. While one scheduler may be intended to favor one type of job, another is intended to favor another type of job. Two other schedulers may have been designed for different points of compromise between those two extremes. Moreover, the performance of schedulers is frequently dependent on the job mix being scheduled.

However, despite the conceptual difficulty of the problem, the flexibility in setting the value functions in

the value-based scheduler allows some meaningful comparisons to be made with the multilevel queue scheduler. As described previously, value functions can be used which approximate the implicit value functions in the multilevel queue scheduler. For comparison with the multilevel queue scheduler, the value function described in the section on Activity-Biased Functions (value function III) was used.

Comparisons with only these value function do not consider the difference in flexibility to specify response sets. However, if the experimental scheduler can perform as well as the multilevel queue scheduler in the domain of operation for which the multilevel queue scheduler was designed and tailored, the flexibility of the value-based scheduler implies it is more powerful.

Domain of Experimentation

The important external variables which determine scheduler performance were identified in Chapter 1 as

1. job mix,
2. scheduler parameters, and
3. resources available.

In the previous experiments the job mixes and scheduler parameters were varied to demonstrate the adaptability of the scheduler to different job mixes and the flexibility of the value function specifications.

As discussed previously, reasonable comparison of the

experimental and multilevel queue schedulers necessitates fixing the scheduler parameters so the schedulers have similar goals. For these comparisons the quantities of resources have been held constant and the job mixes varied. The quantity of a resource available is a measure which is relative to the requirements of the jobs (e. g., the number of jobs which can reside in main memory at one time). Thus, varying both the job mix and resources available are unnecessary.

There are several important factors to be considered in varying the job mix:

1. frequency of requests for service,
2. number of jobs which fit in main memory at a time,
3. frequency of I/O, and
4. use of other resources.

The experiments test average and extreme conditions. Demand for service includes both continuous and interactive requests. In most cases three jobs will fit in main memory at a time; however, with script G, one or two jobs can take up all available memory. Both CPU bound and highly I/O bound jobs have been included in the mixes. Scripts H and I include requests for permission to open a file as well as the basic CPU, memory, and disk resources. The experiments are intended to demonstrate operation under typical circumstances and do not exhaustively explore all posible operating conditions.

Biases in the Comparisons

In making the comparison it is also important to take into account the difference in the amount of effort which has gone into development of the multilevel queue and value-based schedulers. The multilevel queue simulation is modeled after a production operating system which has benefited from substantial investment in design and refinement. On the other hand, the value-based simulation has not received this attention.

Comparison Results

Comparisons were made on a number of job mixes of which three are reported here. Each simulation was repeated four times with different random numbers. In numbering the tables of results, replications which differ only by the random numbers have been given different letter suffixes. For example, Tables 12-A, 12-B, 12-C, and 12-D show the results of four repetitions. In Tables 12 and 13 the results of a mix of four interactive jobs, one compute bound job, and four jobs also doing I/O are shown. The CPU usage (a measure of total computation) with both schedulers is comparable. The value-based scheduler gave slightly more total CPU usage. It gave slightly lower rates of progress.

The next mix of jobs which includes conflict over the resource giving exclusive right to open or close files is summarized in Tables 14 and 15. The first four processes

Table 12-A

Value-Based Scheduler
Value Function III
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	5.8	7.2		.457
2	Interactive	5.8	8.6		.367
3	Interactive	5.8	8.9		.410
4	Interactive	5.8	5.2		.557
5	CPU bound	9.7	4.7		.097
6	I/O	10.3	9.9	10.0	.203
7	I/O	10.1	9.6	10.0	.202
8	I/O	6.8	8.0	7.1	.139
9	I/O	7.5	7.4	7.7	.152
Total		67.7	69.5	34.9	2.584

Table 12-B

Value-Based Scheduler
Value Function III
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	5.0	10.0		.381
2	Interactive	5.8	5.8		.465
3	Interactive	5.8	7.0		.399
4	Interactive	5.8	4.6		.750
5	CPU bound	9.3	4.7		.093
6	I/O	9.8	10.7	10.3	.200
7	I/O	8.2	8.3	8.3	.165
8	I/O	8.3	8.8	8.7	.171
9	I/O	8.5	7.9	8.6	.171
Total		67.9	66.6	35.9	2.795

Table 12-C

Value-Based Scheduler
Value Function III
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	6.6	8.5		.407
2	Interactive	5.8	6.4		.432
3	Interactive	5.0	5.9		.519
4	Interactive	5.8	7.7		.384
5	CPU bound	7.7	4.4		.077
6	I/O	11.2	11.8	11.3	.224
7	I/O	8.0	7.6	7.9	.159
8	I/O	8.0	8.1	7.7	.157
9	I/O	8.0	7.7	8.2	.162
Total		66.1	68.2	35.1	2.521

Table 12-D

Value-Based Scheduler
Value Function III
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	5.8	8.7		.670
2	Interactive	5.0	5.0		.631
3	Interactive	5.8	8.5		.440
4	Interactive	5.0	6.1		.411
5	CPU bound	11.1	5.4		.111
6	I/O	7.6	7.7	7.7	.153
7	I/O	7.6	8.0	7.6	.152
8	I/O	11.7	11.4	11.8	.235
9	I/O	7.5	7.5	7.8	.153
Total		67.1	68.2	34.8	2.956

Table 13-A

Multilevel Queue Scheduler
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	5.5	4.9		.625
2	Interactive	5.0	4.4		.566
3	Interactive	5.8	4.7		.520
4	Interactive	5.8	5.8		.337
5	CPU bound	9.9	5.7		.099
6	I/O	7.9	8.7	8.4	.163
7	I/O	7.9	9.2	8.9	.168
8	I/O	7.9	9.1	8.3	.163
9	I/O	7.9	7.6	7.7	.156
Total		63.7	60.1	33.2	2.797

Table 13-B

Multilevel Queue Scheduler
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	5.0	5.3		.636
2	Interactive	5.8	4.1		.732
3	Interactive	5.0	4.3		.265
4	Interactive	5.8	4.9		.767
5	CPU bound	7.9	4.3		.079
6	I/O	7.9	8.8	8.4	.163
7	I/O	7.9	9.0	8.1	.160
8	I/O	7.9	8.7	8.0	.159
9	I/O	7.9	8.9	7.7	.156
Total		61.3	58.4	32.1	3.117

Table 13-C

Multilevel Queue Scheduler
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	5.8	4.1		.378
2	Interactive	5.8	5.3		.798
3	Interactive	5.8	4.8		.325
4	Interactive	5.0	4.6		.425
5	CPU bound	10.0	4.9		.099
6	I/O	8.0	8.4	8.1	.161
7	I/O	8.0	8.6	7.6	.155
8	I/O	8.0	9.8	8.1	.161
9	I/O	8.0	8.6	7.9	.158
Total		64.4	59.0	31.7	2.660

Table 13-D

Multilevel Queue Scheduler
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	5.7	4.4		.838
2	Interactive	5.7	3.1		.733
3	Interactive	5.7	4.2		.796
4	Interactive	4.9	6.0		.297
5	CPU bound	9.8	5.2		.098
6	I/O	7.9	9.0	8.2	.161
7	I/O	7.9	8.0	8.5	.164
8	I/O	7.9	7.9	7.5	.154
9	I/O	7.9	9.2	7.8	.156
Total		63.3	57.0	32.1	3.437

simply alternate I/O access with central processor use. The next four alternate I/O access and CPU use but repeatedly request exclusive control of file opening and hold this resources for five I/O accesses before releasing it. Thus, the second group of processes model jobs opening files and doing the I/O necessary to accomplish this and the first group of jobs simply perform I/O on the same disk. The last process is completely compute bound.

The total progress of all jobs as well as the total CPU usage is similar. However, the variance among jobs is lower with the experimental scheduler. The different utilization of the resource for file opening is somewhat interesting since the average progress of the jobs opening files in both systems is nearly the same. Apparently, the time that processes holding this resource were suspended while waiting for other resources was longer with the multilevel queue scheduler. As a result, the total time the resource was assigned was longer.

In the last example, reported in Tables 16 and 17, four jobs opening and closing files and two compute bound jobs have been simulated. The first two file opening jobs are interactive while the others are not and have lower priority after their first quantum is expended. The much greater rate of progress for the interactive file opening processes with the value-based scheduler is the most obvious difference in these results. This occurs because the

Table 14-A

Value-Based Scheduler
Value Function III
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	9.7	7.8	9.7		.193
2	I/O	9.6	8.0	9.2		.188
3	I/O	9.6	8.0	8.7		.183
4	I/O	9.5	7.3	9.1		.186
5	I/O for open	9.4	5.8	9.1	22.5	.184
6	I/O for open	10.0	6.0	10.1	24.3	.201
7	I/O for open	9.4	6.4	8.9	20.8	.183
8	I/O for open	9.4	6.7	9.6	24.2	.190
9	CPU bound	9.7	6.1			.097
Total		86.1	61.8	74.2	91.9	1.602

Table 14-B

Value-Based Scheduler
Value Function III
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	9.5	7.1	9.2		.187
2	I/O	9.7	7.4	9.3		.190
3	I/O	9.5	7.7	9.1		.186
4	I/O	9.5	6.8	9.2		.187
5	I/O for open	9.3	6.6	9.7	21.6	.190
6	I/O for open	9.5	6.7	10.2	25.4	.197
7	I/O for open	9.3	6.2	9.0	23.2	.184
8	I/O for open	9.3	7.6	9.3	21.6	.187
9	CPU bound	9.8	5.8			.098
Total		85.5	61.8	75.1	91.8	1.606

Table 14-C

Value-Based Scheduler
Value Function III
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	9.5	7.9	9.6		.191
2	I/O	9.7	7.8	9.2		.188
3	I/O	9.5	8.0	9.3		.188
4	I/O	9.7	7.4	9.0		.187
5	I/O for open	9.5	5.9	9.6	22.2	.191
6	I/O for open	9.4	6.0	9.3	23.5	.187
7	I/O for open	10.1	6.8	10.2	23.4	.202
8	I/O for open	9.4	5.9	8.8	23.0	.182
9	CPU bound	9.7	6.0			.097
Total		86.3	61.8	74.9	92.1	1.613

Table 14-D

Value-Based Scheduler
Value Function III
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	9.7	7.4	9.9		.196
2	I/O	9.5	7.6	9.9		.194
3	I/O	9.7	7.4	9.6		.192
4	I/O	9.7	7.9	9.2		.189
5	I/O for open	9.3	5.7	9.2	24.6	.186
6	I/O for open	9.3	6.7	9.1	21.8	.185
7	I/O for open	10.0	6.3	10.6	25.8	.206
8	I/O for open	9.3	6.0	8.6	20.5	.179
9	CPU bound	9.6	6.8			.096
Total		86.2	61.8	76.2	92.6	1.623

Table 15-A

Multilevel Queue Scheduler
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	10.8	8.0	11.0		.218
2	I/O	13.8	9.4	13.4		.272
3	I/O	9.9	6.9	9.7		.196
4	I/O	11.9	8.2	12.3		.242
5	I/O for open	7.7	6.2	7.7	25.7	.154
6	I/O for open	10.2	7.8	10.1	29.6	.202
7	I/O for open	7.7	6.1	7.6	22.2	.153
8	I/O for open	7.2	6.1	7.3	21.7	.144
9	CPU bound	8.0	3.0			.080
Total		87.1	61.8	79.1	99.1	1.661

Table 15-B

Multilevel Queue Scheduler
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	15.7	11.3	15.6		.313
2	I/O	7.9	5.1	7.7		.155
3	I/O	11.8	8.6	12.6		.244
4	I/O	11.1	7.4	11.3		.224
5	I/O for open	7.6	6.6	7.1	22.3	.147
6	I/O for open	8.5	6.4	8.7	25.9	.172
7	I/O for open	7.6	5.8	7.1	20.0	.148
8	I/O for open	9.4	7.7	9.2	29.8	.187
9	CPU bound	7.9	2.9			.079
Total		87.6	61.8	79.3	98.0	1.669

Table 15-C

Multilevel Queue Scheduler
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	17.8	11.6	17.9		.357
2	I/O	6.2	4.4	6.0		.122
3	I/O	11.9	7.9	11.5		.234
4	I/O	11.9	7.2	12.2		.241
5	I/O for open	5.2	4.7	5.2	16.4	.103
6	I/O for open	9.9	9.6	9.6	35.8	.194
7	I/O for open	5.2	4.7	4.9	17.2	.101
8	I/O for open	9.4	8.8	9.3	28.3	.186
9	CPU bound	10.0	2.9			.100
Total		87.3	61.8	76.6	97.7	1.638

Table 15-D

Multilevel Queue Scheduler
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	10.9	7.8	11.0		.219
2	I/O	13.9	9.6	13.5		.274
3	I/O	11.9	8.5	12.6		.245
4	I/O	9.9	6.9	9.5		.194
5	I/O for open	7.8	6.6	7.4	23.3	.153
6	I/O for open	8.5	6.3	8.7	26.2	.172
7	I/O for open	7.7	6.6	8.4	25.0	.161
8	I/O for open	8.5	6.5	8.6	24.0	.171
9	CPU bound	8.8	3.1			.088
Total		88.0	61.8	79.6	98.5	1.677

value-based scheduler does not simply allocate use of the file opening resource on a first-come-first-served basis as the multilevel queue does. However, the improvement in the progress rate for the interactive jobs was greater than the decrease in the progress rate of the other two jobs. Apparently, better use was made of the resource for file opening. Presumably, this is due to the global resource allocation strategy of giving preference to low value jobs which are holding critical resources.

A comparison of the times for running the experimental and multilevel queue simulations indicates the amount of scheduling overhead for the two strategies is roughly equal. Since the simulators were coded in APL, the execution times are not representative of the execution times for routines implemented in assembly language or high system implementation languages. The interpretive APL implementation results in slower execution; the relative efficiency of many APL primitives is quite different from other language implementations. Furthermore, the data structures and code for the simulated schedulers are not designed for efficiency but to facilitate experimentation and to utilize the specialized APL primitives. Due to these factors, the run time for either simulation is thirty to forty times the length of time simulated. However, comparison of execution times between the two is probably not biased significantly.

Table 16-A

Value-Based Scheduler
Value Function III
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	9.2	6.4	8.4	16.7	.523
2	Interactive open	8.6	7.7	7.6	14.2	.428
3	Open	5.8	9.0	6.1	25.1	.119
4	Open	6.0	8.6	5.6	20.7	.116
5	Interactive CPU bound	34.0	14.5			.384
6	Interactive CPU bound	32.8	15.5			.374
Total		96.4	61.8	27.7	76.7	1.944

Table 16-B

Value-Based Scheduler
Value Function III
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	9.4	7.1	8.8	17.4	.453
2	Interactive open	8.3	7.0	7.9	14.5	.464
3	Open	5.9	8.7	5.9	24.0	.118
4	Open	5.8	8.4	5.6	22.1	.115
5	Interactive CPU bound	31.4	15.0			.381
6	Interactive CPU bound	35.3	15.6			.416
Total		96.2	61.8	28.2	78.0	1.947

Table 16-C

Value-Based Scheduler
Value Function III
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	8.3	6.1	7.4	17.1	.441
2	Interactive open	10.0	7.5	8.9	20.8	.478
3	Open	5.2	7.2	4.7	17.2	.099
4	Open	5.2	9.0	4.8	18.4	.100
5	Interactive CPU bound	37.0	15.8			.432
6	Interactive CPU bound	31.5	16.1			.368
Total		97.1	61.8	25.7	73.5	1.918

Table 16-D

Value-Based Scheduler
Value Function III
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	7.5	5.3	6.8	13.1	.434
2	Interactive open	10.4	8.0	9.2	18.3	.490
3	Open	5.2	7.4	5.0	15.0	.102
4	Open	5.8	10.6	6.3	23.6	.121
5	Interactive CPU bound	33.1	15.9			.383
6	Interactive CPU bound	33.1	14.5			.389
Total		95.1	61.8	27.2	70.1	1.919

Table 17-A

Multilevel Queue Scheduler
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	5.8	10.9	5.0	20.7	.185
2	Interactive open	5.7	10.9	5.2	15.4	.170
3	Open	7.7	7.8	7.0	30.2	.147
4	Open	6.8	8.7	7.1	33.4	.138
5	Interactive CPU bound	33.9	12.3			.394
6	Interactive CPU bound	32.9	11.4			.369
Total		92.7	61.8	24.3	99.6	1.403

Table 17-B

Multilevel Queue Scheduler
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	5.8	13.3	5.2	30.1	.149
2	Interactive open	5.5	10.6	5.3	17.1	.182
3	Open	6.8	6.7	6.8	29.2	.137
4	Open	5.2	5.7	4.7	23.3	.099
5	Interactive CPU bound	36.0	13.0			.416
6	Interactive CPU bound	33.0	12.5			.379
Total		92.2	61.8	22.0	99.6	1.362

Table 17-C

Multilevel Queue Scheduler
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	6.1	15.1	5.3	30.8	.140
2	Interactive open	5.5	12.3	4.8	24.9	.149
3	Open	6.8	5.2	6.5	22.2	.133
4	Open	5.2	5.9	4.8	21.7	.100
5	Interactive CPU bound	37.0	11.1			.435
6	Interactive CPU bound	31.5	12.1			.360
Total		92.2	61.8	21.4	99.7	1.317

Table 17-D

Multilevel Queue Scheduler
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	5.7	13.6	5.1	19.8	.148
2	Interactive open	5.4	14.0	5.2	29.6	.133
3	Open	5.8	4.7	6.1	20.1	.119
4	Open	6.0	6.9	6.1	29.6	.121
5	Interactive CPU bound	35.3	11.4			.421
6	Interactive CPU bound	34.4	11.4			.392
Total		92.7	61.8	22.5	99.1	1.334

Random Scheduler

Another evaluation of the value-based scheduler can be made by comparing it with a hypothetical scheduler that makes random allocation decisions. A number of algorithms qualify as making random decisions. This comparison uses an algorithm which only allocates resources when they are requested and always allocates a resource if it has been requested. However, the priorities of the processes are selected randomly every time a resource allocation decision is made. This is not completely random since it attempts to keep all resources in use. Instead of taking previous resource usage and type of job into account, it decides among conflicting requests randomly.

The experimental value-based scheduler resolves these conflicts by considering requests for allocations in an order determined from the value functions for the processes. Modifying the value-based scheduler given in Appendix III by changing steps VBS3 through VBS6 to select randomly an order for considering the processes produced the random scheduler.

The results of simulations using the same scripts as the comparisons between the value-based and multilevel queue scheduler are given in Tables 18, 19, and 20. These results differ from both value-based and multilevel queue in several ways. Although the total CPU utilization is similar, the distribution among the same type of jobs is much less even. The rates of progress for jobs are lower with the random

scheduler. Also, it has no bias toward interactive jobs.

Summary of Comparisons

The experimental value-based scheduler performance and multilevel queue scheduler performance are quite similar over the range of performance goals which could be tested. Both were superior to the random scheduler. The value-based scheduler allows flexible specification of many other response targets besides those implicit in the multilevel queue scheduler.

Table 18-A

Random Scheduler
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	3.3	5.5		.074
2	Interactive	2.5	6.1		.049
3	Interactive	3.1	5.7		.060
4	Interactive	3.3	5.5		.106
5	CPU bound	34.4	17.0		.344
6	I/O	.5	3.2	.5	.010
7	I/O	6.9	8.5	7.1	.140
8	I/O	5.2	7.3	5.2	.104
9	I/O	6.6	6.6	6.6	.132
Total		65.9	66.5	19.4	1.019

Table 18-B

Random Scheduler
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	3.3	6.1		.090
2	Interactive	3.3	4.3		.086
3	Interactive	3.3	6.4		.081
4	Interactive	3.3	4.2		.068
5	CPU bound	15.4	9.5		.154
6	I/O	14.0	14.4	14.0	.280
7	I/O	.5	4.2	.7	.012
8	I/O	2.6	5.6	2.8	.054
9	I/O	13.8	14.1	13.9	.277
Total		59.6	68.9	31.3	1.102

Table 18-C

Random Scheduler
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	2.5	3.1		.048
2	Interactive	3.3	4.9		.063
3	Interactive	1.7	2.4		.028
4	Interactive	2.5	6.1		.051
5	CPU bound	64.3	27.9		.643
6	I/O	1.4	3.7	1.6	.030
7	I/O	.4	3.3	.5	.010
8	I/O	.6	2.3	.8	.014
9	I/O	7.7	8.9	8.0	.156
Total		84.3	62.6	10.9	1.043

Table 18-D

Random Scheduler
Script G

Job	Type	%CPU	%Page	%Disk	Progress
1	Interactive	3.3	4.4		.076
2	Interactive	3.3	6.5		.092
3	Interactive	2.5	4.0		.053
4	Interactive	3.3	4.9		.069
5	CPU bound	34.0	17.1		.340
6	I/O	3.7	5.9	3.6	.073
7	I/O	12.5	12.7	12.6	.251
8	I/O	2.3	5.1	2.4	.047
9	I/O	4.0	6.2	4.3	.083
Total		69.0	66.9	22.9	1.084

Table 19-A

Random Scheduler
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	11.0	9.7	10.9		.219
2	I/O	11.2	9.4	11.7		.229
3	I/O	9.3	8.7	9.1		.184
4	I/O	10.8	9.4	10.6		.214
5	I/O for open	3.4	3.6	3.5	34.3	.070
6	I/O for open	.7	.6	.4	1.3	.011
7	I/O for open	6.8	6.0	7.1	56.3	.140
8	I/O for open	1.8	1.3	1.5	4.7	.034
9	CPU bound	32.0	13.3			.320
Total		87.2	62.0	54.9	96.7	1.421

Table 19-B

Random Scheduler
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	10.3	9.6	11.1		.214
2	I/O	11.0	10.2	11.4		.224
3	I/O	10.1	8.5	10.2		.203
4	I/O	10.2	8.2	10.4		.206
5	I/O for open	1.8	2.0	1.6	11.9	.035
6	I/O for open	1.0	1.5	1.0	4.3	.020
7	I/O for open	3.5	3.9	3.2	11.6	.067
8	I/O for open	5.0	4.9	4.9	44.8	.099
9	CPU bound	34.1	13.2			.341
Total		87.0	62.1	53.8	92.6	1.409

Table 19-C

Random Scheduler
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	10.0	8.8	9.8		.198
2	I/O	10.8	9.1	10.1		.210
3	I/O	13.0	10.9	13.2		.262
4	I/O	8.8	8.3	8.8		.176
5	I/O for open	1.0	.7	.8	1.8	.018
6	I/O for open	2.7	2.7	2.8	30.0	.055
7	I/O for open	1.0	1.6	.8	18.2	.018
8	I/O for open	5.2	4.5	5.1	43.4	.103
9	CPU bound	36.7	14.4			.367
Total		89.3	61.1	51.5	93.5	1.407

Table 19-D

Random Scheduler
Script H

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	I/O	9.7	8.6	9.7		.194
2	I/O	10.7	9.2	10.6		.213
3	I/O	11.9	9.9	11.1		.230
4	I/O	11.3	9.1	11.4		.226
5	I/O for open	7.7	5.8	7.8	42.8	.155
6	I/O for open	2.7	2.9	2.7	34.8	.053
7	I/O for open	1.0	1.5	1.0	7.5	.020
8	I/O for open	1.0	1.2	.9	7.7	.019
9	CPU bound	33.9	13.8			.339
Total		89.8	61.9	55.3	92.9	1.449

Table 20-A

Random Scheduler
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	5.6	8.9	5.2	18.6	.178
2	Interactive open	4.0	7.7	3.4	18.2	.097
3	Open	10.1	8.3	9.9	37.1	.199
4	Open	4.3	4.3	4.0	18.3	.083
5	Interactive CPU bound	33.4	16.0			.383
6	Interactive CPU bound	33.2	16.5			.381
Total		90.6	61.8	22.4	92.2	1.321

Table 20-B

Random Scheduler
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	2.9	5.6	2.7	14.7	.067
2	Interactive open	2.5	6.0	2.2	16.6	.057
3	Open	10.2	11.1	10.4	48.1	.206
4	Open	1.8	2.3	1.7	9.4	.035
5	Interactive CPU bound	38.5	18.2			.453
6	Interactive CPU bound	39.3	18.4			.464
Total		95.2	61.6	17.0	88.8	1.282

Table 20-C

Random Scheduler
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	3.8	8.6	3.3	18.3	.118
2	Interactive open	4.0	8.6	3.6	13.5	.126
3	Open	7.6	7.3	7.7	33.4	.153
4	Open	6.8	5.6	6.2	27.2	.130
5	Interactive CPU bound	30.8	15.7			.373
6	Interactive CPU bound	37.3	16.0			.435
Total		90.4	61.8	20.8	92.4	1.335

Table 20-D

Random Scheduler
Script I

Job	Type	%CPU	%Page	%Disk	%Open	Progress
1	Interactive open	3.8	5.7	3.6	16.8	.081
2	Interactive open	3.8	5.9	3.3	20.1	.077
3	Open	5.2	6.0	5.2	25.4	.104
4	Open	6.8	7.1	6.8	29.4	.137
5	Interactive CPU bound	39.3	18.6			.468
6	Interactive CPU bound	35.8	17.8			.423
Total		94.8	61.2	19.0	91.7	1.290

Chapter 6

CONCLUSIONS

This investigation has resulted in a number of findings which are of interest for both the design of operating systems and further research on resource allocation and scheduling. This chapter summarizes these ideas and describes some areas for additional research.

Problem Formulation

A model of resource allocation in multiprogrammed computer systems forms the basis of this approach. This model relates the progress of processes to the resources required by the processes. A conventional, general notion of a process has been adopted. However, the concepts of resources and progress of processes take on specific meanings in the model.

The important aspect of a process which is recognized in the model is the sequential nature of the requests and releases of resources. When a request cannot be fulfilled, the process must be suspended and the process can not make progress. A resource is a facility or privilege for which there are constraints on its availability for assignment to

a process. The principal constraint which has been examined is a restriction on the number of processes which may simultaneously be assigned the resource.

As a consequence of these definitions, a more general approach can be taken to the problem of allocating resources or scheduling the processes. For example, this viewpoint shows that, while usually considered separately, the same type of constraints apply to scheduling the use of the CPU and main memory resources as other resources. Traditionally, completely different algorithms have been used for different resources. As a result, the slight amount of coordination between the algorithms has been through specialized techniques. Formulating the resource allocation in a uniform framework allows a single, general coordination mechanism.

Value-Based Scheduling

The value-based scheduling strategy provides an improved means for utilizing the system manager's goals in the decisions made by the resource allocation and scheduling programs. This is achieved by allowing the system manager to tell the system how to calculate the relation between the time taken to complete the job and the value of completing the job in that time for every job submitted. The scheduler then attempts to maximize the total value of all jobs submitted for processing.

Several aspects of the manager's control over the system operation are different from other systems. First, the manager has much more flexibility in specifying the rates of service jobs will receive. In most systems the rates of service that jobs with different characteristics will receive were built into the system at the time the scheduler was designed. Only slight adjustments such as a change of the quantum time can be made easily. Schedulers based on external priorities, such as strict priority class schedulers, cannot take other factors into account. In almost all other schedulers there is little flexibility in adjusting the response set.

Tailoring a value-based scheduler to the requirements of a specific installation is easier since the response to be given jobs is specified directly. Unlike other types of schedulers where the responses are a consequence of juggling internal scheduler parameters, there is a direct relationship between scheduler performance and the values specified by the system manager: the value-based scheduler seeks to attain responses specified by the system manager directly.

As a consequence of this direct control and the flexibility in specifying rates of services for jobs, the value-based scheduler can facilitate interfacing the internal computer scheduling policies with extra-computer pricing and administrative control policies. Various

schemes of flexible pricing, dynamic partitioning, guaranteed response, etc. can be accommodated. With the right choice of value functions, the response characteristic of a variety of conventional schedulers can be approximated.

Significant or slight changes can be made easily to the value functions. This is one of the most important virtues of the value-based scheduler. It is natural to implement the value functions in tables or easily changed routines separate from the rest of the scheduler. Where the need to recode a substantial part of the scheduler of a conventional system would make changes impractical, the changes could be made easily in a value-based system.

The use of value functions provides a convenient means of combining process priorities with efficient use of system resources. The more complete information about the value of execution of each process allows tradeoffs to be made between efficient use of the system resources and getting specific timely required work done. Since the value of process execution is always based on both the elapsed time and total service received instead of short-term rate of service, individual decisions made to gain efficient use of a resource will only delay a process temporarily. Any delay will cause it to have higher value and get preferential treatment in later decisions. This preferential treatment will continue until it has received a sufficient average rate of service.

Global, Centralized Scheduling

Global scheduling allows decisions about allocation of one resource to be made with the additional information about requirements in the system for other resources. Thus, a coordinated allocation of all resources is possible. IBM has developed, independently, a scheduler in which most resource allocation is centralized to allow the use of some rather specialized techniques to coordinate the use of the resources. In contrast, the global allocation used with the value-based scheduler is based on a uniform treatment of the resources with dynamic assignment of values to the resources to effect coordination.

In the context of the value-based scheduler, centralized allocation is beneficial for several reasons. The technique of dynamically attaching values to resource facilitates consistent decisions on all resources. Separate allocation mechanisms would result in conflicting, inefficient allocations.

Value-based scheduling can be implemented most easily with centralized resource allocation since information about the allocations of system resources are needed to calculate the expected values. Centralized control of the resources facilitates access to this information.

The use of the same or similar code for allocating several resources is less complex than the use of completely different strategies. Where special characteristics of the

resource are not important in its allocation, the same code can be shared.

Summary of Findings

There are several important results of this study.

1. The model of the resource allocation and scheduling problem forms the basis of this investigation. The uniform treatment of all resources in the model aids in understanding and designing resource allocation strategies. Each process is modeled as a series of resource requests and releases. The operating system must either grant the requests or suspend the process. The performance of the scheduler is the set of response times produced when scheduling a job mix.
2. The value-based resource allocation strategy overcomes deficiencies of existing schedulers. Explicit specification of the value of jobs as a function of the time taken to complete them allows the use of utility theory evaluations in making resource allocation decisions and provides the system manager better control over operation of the system. Dynamic determinations of the opportunity costs of resource assignments can be used advantageously in making resource allocation decisions.

3. Simulation experiments showed that value-based allocation is feasible. When its parameters were set to approximate the value functions of a modern multilevel queue scheduler, the value-based scheduler performed as well as the multilevel queue scheduler.

SUGGESTIONS FOR FURTHER RESEARCH

These studies of resource allocation can be extended in several directions. The rest of this chapter itemizes some opportunities for additional investigation.

Production System

The development of a complete operating system which utilizes value-based resource allocation would allow more complete evaluation of its performance. Although simulation has been used to study the effect of various job mixes, only a limited number of processes could be simulated and only for relatively short periods of time. Performance on a real job mix over an extended period of time should be measured in a production system.

Implementation of a system would also allow experimentation with the effect of different value functions in accomplishing the system manager's goals. The difficulty and payoffs of tailoring the system by specifying value

functions need to be assessed. An important part of this evaluation is to determine the forms of value functions which are useful to the system manager in specific organizational contexts.

Expected Value Decision Analysis

Additional study into ways of estimating both the probabilities and payoffs of outcomes from decisions made by the resource allocator would be useful. Several possible indicators were pointed out in the section on decisions under risk in Chapter 4. Experimental determination of the reliability of these indicators or the determination of better ones is needed.

Additional study is needed on incorporation of resource specific allocation policies into the value analysis framework. Techniques for balancing efficient use of individual resources with overall system efficiency need to be developed.

Better means of coordinating allocations when a process has outstanding requests for several resources would be useful. This situation can arise if a process requests several resources at the same time or when resources have been preempted from the process. Practical algorithms for assuring all the resources will become available at the same time without unproductive holding of resources while waiting for others to become available will result in better system

performance. Similarly, knowing when to preempt resources while waiting for others to become available is also important.

Cooperating Processes

Process coordination using P and V operations on semaphores or equivalent synchronization primitives can be treated in the same manner as other types of resource allocation problems using this resource allocation model. A P operation corresponds to a resource request which may result in the process being blocked. A V operation is the release of a resource. The capacity of the resource is the initial value of the semaphore.

The standard example of a producer process generating data for a consumer process illustrates the use of P and V operations to represent requests for real resources--the buffers in the buffer pool. Other semaphores can be used with imaginary resources such as permission or exclusive right to access certain data. In the context of this model, the effect on system performance of the interrelationships between cooperating processes could be analyzed. The amount of unproductive resource assignment which results from complex resource requirements needs to be assessed.

The examination of I/O devices as sources and sinks of data is a related area for study. For example, when a process is considered a producer and an output device a

consumer the situation is similar to the two cooperating processes in the previous example. If the output device is slow compared to the rate the process produces the data, the process often is unproductively assigned a substantial amount of resources while it is waiting for the I/O device to catch up. A solution to this inefficiency was developed long ago by decoupling the production and consumption of the data through spooling. However, other inefficiencies similar to this are potential candidates for investigation from a resource allocation viewpoint.

Demand Paging

The presence of page-organized memory address translation hardware on a computer facilitates the management of a process's address space and allows the pages of real memory to be used interchangeably.

In these studies, the schedulers always allocated all of a process's requirement for memory at one time (i.e., the process was either entirely in core or swapped out). In a virtual memory or demand paging system the scheduler needs to allocate only part of the requirement. If the program references data which is not in main memory, a fault occurs and the scheduler must swap in the required data to an unassigned page of memory or replace the data of an assigned page.

From a resource allocation viewpoint, the scheduler can

dynamically vary the amount of memory assigned and consequently vary requirements for use of the swapping channel to bring in needed pages. If all the requested memory has been allocated, all the data can be kept in memory and faults will not occur. With a small allocation of memory, page faults will be very frequent and the frequent use of the swapping channel will be required. Also, the page faults result in delaying the process's progress and unproductive assignment of the other resources it holds while the page swap is being performed. The resource allocation model could be used to evaluate an optimum allocation of real memory and prefetch strategy on the basis of the dynamic value to the system of memory pages, the swapping channel, other resources held, and the value of the process's progress.

BIBLIOGRAPHY

Adiri, Igal. "A Dynamic Time-Sharing Priority Queue." Journal of the Association for Computing Machinery, Vol. 18, No. 4, October 1971, pp. 603-610.

Balas, E. "Project Scheduling with Resource Constraints." Applications of Mathematical Programming Techniques. Ed. E. M. L. Beale. London: English University Press, 1970.

Bernstein, A. J. and J. C. Sharp. "A Policy-Driven Scheduler for a Time-Sharing System." Communications of the Association for Computing Machinery, Vol. 14, No. 2, February 1971, pp. 74-78.

Chamberlin, D. D., H. P. Schlaeppi, and I. Wladawsky. "Experimental Study of Deadline Scheduling for Interactive Systems." IBM Journal of Research and Development, Vol. 17, No. 3, May 1973, pp. 263-269.

Chang, W. "Single-Server Queuing Processes in Computing Systems." IBM Systems Journal, Vol. 9, No. 1, 1970, pp. 36-71.

Chao, S. K. and W. W. Chu. "A Cost-Priority Scheduling System." Proceedings of the Second International Conference on Computer Communication, Stockholm, August 12-14 1974, pp. 395-400.

Coffman, E. G., Jr. "Analysis of Two Time-Sharing Algorithms Designed for Limited Swapping." Journal of the Association for Computing Machinery, Vol. 15, No. 3, July 1968, pp. 341-353.

Coffman, E. G., Jr., M. J. Elphick, and A. Shoshani. "System Deadlocks." Computing Surveys, Vol. 3, No. 2, June 1971, pp. 67-78.

Coffman, E. G., Jr. and L. Kleinrock. "Computer Scheduling Methods and their Countermeasures." Proceedings of the 1968 Spring Joint Computer Conference, Vol. 32, pp. 11-21.

Coffman, E. G., Jr. and L. Kleinrock. "Feedback Queueing Models for Time-Shared Systems." Journal of the Association for Computing Machinery, October 1968, pp. 549-576.

Cosine Committee. "An Undergraduate Course on Operating System Principles." Washington, D. C.: Commission on Education, National Academy of Engineering, June 1971.

Denning, P. J. "Equipment Configuration in Balanced Computer Systems." IEEE Transactions on Electronic Computers, Vol. C-18, November 1969, pp. 1008-1012.

Denning, P. J. "Resource Allocation in Multiprocess Computer Systems." Project MAC Report MAC-TR-50 (Ph.D. Thesis). Cambridge, Massachusetts: Massachusetts Institute of Technology, May 1968a.

Denning, P. J. "Third Generation Computer Systems." Computing Surveys, Vol. 3, No. 4, December 1971, pp. 175-216.

Denning, P. J. "Virtual Memory." Computing Surveys, Vol. 2, No. 2, September 1970, pp. 153-189.

Denning, P. J. "The Working Set Model for Program Behavior." Communications of the Association for Computing Machinery, Vol. 11, No. 5, May 1968b, pp. 323-333.

Dijkstra, E. W. "Co-operating Sequential Processes." Programming Languages. Ed. F. Genuys. London: Academic Press, 1968, pp. 43-112.

Elmaghraby, S. E. The Design of Production Systems. New York: Reinhold, 1966.

Fuller, S. H. "Summary of Minimal-Total-Processing-Time Drum and Disk Scheduling Disciplines." Operating Systems Review: Fourth Symposium on Operating System Principles, Vol. 7, No. 4, October 1973, pp. 54-57.

Grochow, J. M. "A Utility Theoretic Approach to Evaluation of a Time-Sharing System." Statistical Computer Performance Evaluation. Ed. W. Freiberger. New York: Academic Press, 1972, pp. 25-53.

Hellerman, H. "Some Principles of Time-Sharing Scheduler Strategies." IBM Systems Journal, Vol. 8, No. 2, 1969.

Holt, R. C. "Some Deadlock Properties of Computer Systems." Computing Surveys, Vol. 4, No. 3, September 1972, pp. 179-196.

IBM Corporation. "Introduction to OS/VS2 Release 2." Poughkeepsie, New York: IBM Corporation, February 1973a.

IBM Corporation. "OS/VS 2 Planning Guide for Release 2." Poughkeepsie, New York: IBM Corporation, March 1973b.

Kleinrock, L. "A Continuum of Time-Sharing Scheduling Algorithms." Proceedings of the 1970 Spring Joint Computer Conference, Vol. 36, pp. 453-458.

Kleinrock, L. and R. R. Muntz. "Processor Sharing Queueing Models of Mixed Scheduling Disciplines for Time Shared Systems." Journal of the Association for Computing Machinery, Vol. 19, No. 3, July 1972, pp. 464-482.

Luce, R. D. and H. Raiffa. Games and Decisions. New York: Wiley, 1965.

Mahl, R. "An Analytical Approach to Computer Systems Scheduling." Computer Science Report UTECH-CSC-70-100 (Ph.D Thesis). Salt Lake, Utah: University of Utah, June 1970.

Marshall, B. S. "Dynamic Calculation of Dispatching Priorities under OS/360 MVT." Datamation, Vol. 15, No. 8, August 1969, pp. 93-97.

Manrak, S. A. "Simulation Analysis of a Pay-for-Priority Scheme for the IBM 360/75." Technical Report UIUCDCS-R-73-605, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, August 1973.

McKinney, J. M. "A Survey of Analytical Time-Sharing Models." Computing Surveys, Vol. 1, No. 2, June 1969.

McNaughton, Robert. "Scheduling with Deadlines and Loss Functions." Management Science, Vol. 6, No. 1, October 1959, pp. 1-12.

Michel, J. A. and E. G. Coffman, Jr. "Synthesis of a Feedback Queuing Discipline for Computer Operation." Journal of the Association for Computing Machinery, Vol. 21, No. 2, April 1974, pp. 329-339.

Morris, W. T. The Analysis of Management Decisions. Homewood, Illinois: Irwin, 1964.

Nicholson, T. A. J. Optimization in Industry, Volume II, Industrial Applications. Chicago: Aldine-Atherdon, 1971.

Pass, E. M. and J. Gwynn. "An Adaptive Microscheduler for a Multiprogrammed Computer System." Proceedings of the ACM 1973 Annual Conference, pp. 327-331.

Raiffa, H. Decision Analysis. Reading, Massachusetts: Addision-Wesley, 1968.

Ryder, K. D. "A Heuristicc Approach to Task Dispatching." IBM Systems Journal, Vol. 9, No. 3, 1970.

Scherr, A. L. "The Design of IBM OS/VS2 Release 2." Proceedings of the 1973 National Computer Conference and Exposition, Vol. 42, June 1973a, pp. 387-394.

Scherr, A. L. "Functional Structure of IBM Storage Operating Systems Part II: OS/VS2-2 Concepts and Philosophies." IBM Systems Journal, Vol. 12, No. 4, 1973b, pp. 382-400.

Sevcik, K. C. "Scheduling for Minimum Total Loss Using Service Time Distributions." Journal of the Association for Computing Machinery, Vol. 21, No. 1, January 1974, pp. 66-75.

Teorey, T. J. "Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems." Proceedings of the 1972 Fall Joint Computer Conference, Vol. 41, pp. 1-11.

Watson, R. W. Timesharing System Design Concepts. New York: McGraw-Hill, 1970.

Xerox Corporation. Xerox APL Language and Operations Reference Manual. El Segundo, California: Xerox Corporation, September 1972.

Xerox Corporation. Xerox Universal Time-Sharing System System Management Reference Manual. El Segundo, California: Xerox Corporation, January 1972.

Xerox Corporation. Xerox Universal Time-Sharing System Time-Sharing Reference Manual. El Segundo, California: Xerox Corporation, February 1971.

Appendix I

SIMULATION SCRIPTS

The scripts in this appendix were used as system loads in the simulation studies. The following resources were available.

Resource	Capacity	Preemptable
CPU	1	always
main memory pages	78	if copied to swapping device
disk	1	no
permission to open/close files	1	no
swapping channel	1	no

A range of time indicates use of a random draw from a uniform distribution between the specified limits.

SCRIPT A

```
        8 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L1:     Run 30,000 milliseconds.
        Go to L1.
```

SCRIPT B

```
        4 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L1:     Run 30,000 milliseconds.
        Go to L1.

        4 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L2:     Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Go to L2.
```

SCRIPT C

```
        4 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L1:     Run 30,000 milliseconds.
        Go to L1.

        4 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L2:     Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Go to L2.

        2 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
```

```
        Run 1 millisecond.
L3:     Run 100 milliseconds.
        Release 1 CPU.
        Think-type 1 to 8000 milliseconds.
        Request 1 CPU.
        Go to L3.
```

SCRIPT D

```
        4 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L1:     Run 30,000 milliseconds.
        Go to L1.

        4 class-2 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L2:     Run 30,000 milliseconds.
        Go to L2.
```

SCRIPT E

```
        2 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L1:     Run 30,000 milliseconds.
        Go to L1.

        2 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L2:     Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Go to L2.
```

```
        2 class-2 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L3:     Run 30,000 milliseconds.
        Go to L3.

        2 class-2 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L4:     Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Go to L4.
```

SCRIPT F

```
        4 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L1:     Run 30,000 milliseconds.
        Go to L1.

        4 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L2:     Run 100 milliseconds.
        Release 1 CPU.
        Run 1 to 80,000 milliseconds.
        Request 1 CPU.
        Go to L2.
```

SCRIPT G

```
        4 class-1 processes.
        Request 30 pages.
        Request 1 CPU.
        Run 1 millisecond.
```

```
L1:     Release 1 CPU.
        Think-type 5001 to 10000 milliseconds.
        Request 1 CPU.
        Run 500 milliseconds.
        Go to L1.


        1 class-1 process.
        Request 40 pages.
        Request 1 CPU.
L2:     Run 30,000 milliseconds.
        Go to L2.


        4 class-1 processes.
        Request 40 pages.
        Request 1 CPU.
        Run 1 millisecond.
L3:     Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Go to L3.
```

SCRIPT H

```
        4 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L1:     Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Go to L1.


        4 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L2:     Run 100 milliseconds.
        Request 1 open.
        Release 1 CPU.
        Request 1 disk.
```

```
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 150 milliseconds.
        Release 1 disk.
        Release 1 open.
        Request 1 CPU.
        Go to L2.


        1 class-1 process.
        Request 20 pages.
        Request 1 CPU.
L3:     Run 30,000 milliseconds.
        Go to L3.
```

SCRIPT I

```
        2 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L1:     Run 100 milliseconds.
        Request 1 open.
        Run 1 millisecond.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
```

```
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
        Release 1 disk.
        Release 1 open.
        Request 1 CPU.
        Run 50 milliseconds.
        Release 1 CPU.
        Think-type 1 to 8000 milliseconds.
        Request 1 CPU.
        Go to L1.


        2 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L2:     Run 100 milliseconds.
        Request 1 open.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
```

```
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
        Release 1 disk.
        Request 1 CPU.
        Run 100 milliseconds.
        Release 1 CPU.
        Request 1 disk.
        Run 50 to 100 milliseconds.
        Release 1 disk.
        Release 1 open.
        Request 1 CPU.
        Go to L2.

        2 class-1 processes.
        Request 20 pages.
        Request 1 CPU.
        Run 1 millisecond.
L3:     Run 1 to 1400 milliseconds.
        Release 1 CPU.
        Think-type 300 milliseconds.
        Request 1 CPU.
        Go to L3.
```

Appendix II

SCRIPT INTERPRETER

Variable Definitions

ASSIGNED[R,P] = Number of resource units assigned. Indexed by resource and process.

CAPACITY[R] = Total number of resource units in system. Indexed by resource.

CODE[I,J] = Script code. First index selects instruction. Second index selects instruction fields:

- 1 = type resource to be released (if any) before process step
- 2 = quantity of resource to be released
- 3 = type resource to be requested (if any) before process step
- 4 = quantity of resource to be requested
- 5 = index (address) of next instruction
- 6 = minimum time for this process step in milliseconds
- 7 = range of time for process step for uniform distribution
- 8 = activity type for process step: inactive (thinking and typing) or active.

ENTERED[P] = Real time when process last changed STATES. Indexed by process.

FUTUREWANT[R,P] = Number of resource units which will be needed to proceed after WANT has been specified.

IA[P] = Pointer to script instruction for current process step. Indexed by process.

INTERVAL[P] = Milliseconds remaining in current process step. Indexed by process.

M = Number of resources.

N = Number of processes.

NEWWANT[R,P] = Number of resources requested for next process step. Indexed by resource and process.

PREEMPTABLE[R,P] = Logical values specifying whether or not current resource assignment is preemptable.

QUANTUMREMAINING[P] = Milliseconds of time remaining in current quantum allotment. Indexed by process.

REALTIME = Milliseconds of real time simulated.

RELEASED[R,P] = Number of resource units released at end of process step. Indexed by resource and process.

RESERVE[R] = Number of currently available resource units. Indexed by resource.

STATE[P] = Current state (queue) for process. Indexed by process. Indicates one of the following multilevel queue states:

current-user,
request-received,
special-compute,
I/O-complete,
compute-interrupted,
compute,
I/O-in-progress,
waiting-to-open/close,
waiting-for-request, and
waiting-for-request-outswapped.

SWAPTIME = Milliseconds until swap complete.

TIMESLICE = Milliseconds until timeslice expires.

TOTALINACTIVETIME[P] = Milliseconds process has been inactive since simulation began.

TOTALUSE[R,P] = Milliseconds process has been assigned resources since simulation began. Indexed by resource and process.

TOTALVIRTUALTIME[P] = Milliseconds process has been active since simulation began. Indexed by process.

USE[R,P] = Milliseconds process has been assigned resources since it was inactive. Indexed by resource and process.

VIRTUALTIME[P] = Milliseconds process has been active since it was inactive. Indexed by process.

WANT[R,P] = Number of resource units needed to proceed. Indexed by resource and process.

SCRIPT INTERPRETER & HOUSEKEEPING ALGORITHM

Simulation Starts Here

I1. Set currently available resources (RESERVE) to total available resources (CAPACITY).

I2. Point instruction address (IA) to beginning of appropriate script for each process. Set STATE to special-compute state for all processes.

I3. Zero USE, TOTALUSE, REALTIME, VIRTUALTIME, TOTALVIRTUALTIME, WANT, FUTUREWANT, ENTERED.

I4. Set PREEMPTABLE to indicate resources which are always preemptable are preemptable and all others are non-preemptable.

I5. Go to I12.

I6. Advance REALTIME by minimum of
a) SWAPTIME if swap in progress,
b) TIMESLICE, or
c) INTERVAL for active processes.

I7. Reduce INTERVAL for active processes and SWAPTIME by time increment.

I8. Update USE and TOTALUSE by adding ASSIGNED multiplied by time increment.

I9. Advance VIRTUALTIME and TOTALVIRTUALTIME of active processes by time interval.

I10. Advance TOTALINACTIVETIME of inactive processes by time interval.

I11. If REALTIME greater than simulation period, print TOTALUSE, TOTALVIRTUALTIME, and TOTALINACTIVETIME.

I12. Zero RELEASED and NEWWANT.

I13. Do step I14 for each active process P. Then, go to I19.

I14. While the time to the end of the next step (INTERVAL[P]) is non-zero, do steps I15 through

I18.

I15. If a resource is to be released, place quantity into RELEASED array.

I16. If a resource is to be requested, place quantity into NEWWANT array.

I17. Set time for step (fixed time or random draw from uniform distribution) into INTERVAL[P].

I18. Set IA to address of next script instruction.

I17. Subtract RELEASED from ASSIGNED.

I18. Add total released resources to RESERVE.

I19. Add NEWWANT array to WANT array.

I20. Go to scheduler being tested.

Appendix III

VALUE-BASED SCHEDULER

(See Appendix II for script interpreter and variable definitions)

Enter from Interpreter

VBS1. If swapping completed, then
a) Set memory pages preemptable,
b) Deassign swapping channel,
c) Set swapping channel available, and
d) Add FUTUREWANT for process to WANT.

VBS2. Set currently available resources to sum of currently available resources and assigned, preemptable resources.

VBS3. Calculate value of running each process from rate of change in value with respect to service for class of service, current service received, and elapsed time. (See functions in text.)

VBS4. Calculate opportunity value of each resource as maximum value of processes which have been suspended and need the resource.

VBS5. Calculate the adjusted value of running each process as the maximum of the value of running and the maximum value of non-preemptable resources held.

VBS6. Do steps VBS7 through VBS9 for each process, P, in order of descending adjusted value.

VBS7. If assigned resources are preemptable and not available, deallocate them and increase want; Otherwise, decrease quantity available. If memory pages were deallocated, move want for other preemptable resources into future want and set swapping channel wanted.

VBS8. If all resources wanted are available, then assign them, decrease available, zero want, and do not do step VBS9.

VBS9. If pages of memory are assigned and not preemptable, set the swapout candidate to the current process number.

VBS10. If the swap channel is not free, go to VBS11. If there is a running job which holds only memory and preemptable resources, there is a job waiting for memory, and the adjusted value of the suspended job exceeds the adjusted value of the running job by the value which would be lost by suspending the running job to copy it out, suspend the running job and make it the swapout candidate. If there is a swapout candidate, initiate swapout: assign swap channel and set swap channel unavailable.

VBS11. Set pages for process using CPU and I/O devices non-preemptable.

VBS12. If a new swap was started, set swaptime to transfer time for number of memory pages allocated (.83 milliseconds per page) + uniform random rotational latency (0 to 34 milliseconds).

VBS13. Set TIMESLICE to 1200 milliseconds.

VBS14. Go to interpreter step I6.

Appendix IV

MULTILEVEL QUEUE SCHEDULER

(See Appendix II for script interpreter and variable definitions)

Enter from Interpreter

MQS1. Assign process releasing disk (if any) to I/O complete state. Assign process completing inactive period (if any) to request-received state.

MQS2. Assign current user of CPU (if any) to new state:

a) Requested disk to I/O-in-progress state
b) Became inactive to wait-for-request state
c) Requested open/close and unavailable to waiting-to-open/close state
d) If neither a, b, or c and less than 40 milliseconds remain in quantum allotment to compute state
e) If neither a, b, c, or d and quantum has expired or both at least 20 milliseconds of quantum allotment has been used and there are other processes in request-received or special-compute state to compute-interrupted state.

MQS3. If the open/close resource is available, assign it to the first process if any in the open/close state. Place the process in the special compute state.

MQS4. If the swap out of a process waiting for a request (inactive) was completed, place the process in the wait-for-request-outswapped state.

MQS5. Assign a new quantum allotment (1200 milliseconds) to processes with less than 40 milliseconds remaining in their old quantum allotment, processes in the request-received state, and processes in the special-compute state.

MQS6. If the swap channel is not free, go to MQS12.

MQS7. If a swap has been completed, free the swap channel, zero the WANT for memory pages swapped in, set the pages swapped in ASSIGNED, zero ASSIGNED for the pages swapped out, and adjust the RESERVE for the difference between the number of pages swapped out and swapped in.

MQS8. Select a candidate to swap in from the states (in order)

request-received,
special-compute,
I/O-complete,
compute-interrupted, and
compute.

If more than one process is in the same state, choose the process which has been in the state longest. If none, go to MQS12.

MQS9. If required pages are free, go to MQS11.

MQS10. Look for a single process which can be swapped out to provide enough free pages. This process must have received at least 40 milliseconds CPU use and not be opening or closing a file. The process is selected by searching in order the states

waiting-for-request and
waiting-to-open/close

and choosing the process which entered the state most recently. If no single process can be found, multiple processes are selected by searching in order the states

waiting-for-request,
waiting-to-open/close,
compute,
compute-interrupted,
I/O-complete,
special-compute, and
request-received

starting with the process which entered the state most recently. If the swap in candidate is encountered before enough pages can be found go to MQS12.

MQS11. Initiate swap out (if any) and swap in. Calculate SWAPTIME from transfer time for number of pages to swap (.83 milliseconds per page) + uniform random latency (0 to 34 milliseconds). Set pages WANTED for pages being swapped out.

MQS12. If the disk is available, assign it to the proccess which has been in the I/O-in-progress state longest. Reduce RESERVE. Reduce WANT.

MQS13. Assign the CPU by searching the following states in order and choosing the process which has been in the state longest.

current-user
request-received
special-compute
I/O-complete
compute-interrupted
compute

If a process is found, set ASSIGNED, reduce RESERVE, reduce WANT, and place the process in the current user state. Set TIMESLICE to remaining portion of quantum for this process

MQS14. Go to interpreter step I6.