A PRE-SCHEDULER AND MANAGEMENT MODEL FOR
A CLASS OF COMPUTER-USER SYSTEMS

A THESIS
Presented to
The Faculty of the Division of Graduate
Studies and Research
by
John Marion Hoffman

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in the School of Information and Computer Science

Georgia Institute of Technology
January, 1971
Copyright, 1971
In presenting the dissertation as a partial fulfillment of the requirements for an advanced degree from the Georgia Institute of Technology, I agree that the Library of the Institution shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish from, this dissertation may be granted by the professor under whose direction it was written, or, in his absence, by the Dean of the Graduate Division when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from, or publication of, this dissertation which involves potential financial gain will not be allowed without written permission.
A PRE-SCHEDULER AND MANAGEMENT MODEL FOR
A CLASS OF COMPUTER-USER SYSTEMS

APPROVED:

Chairman

Date approved by Chairman:  Jan. 22, 1971
ACKNOWLEDGMENTS

This study could not have been completed without the valuable assistance of the thesis advisor, Dr. John M. Gwynn, Jr., and the members of the reading committee, Dr. Lynwood A. Johnson and Dr. Lawrence J. Gallaher. Special thanks go to Mr. A. P. Jensen and Dr. E. G. Coffman for their helpful comments. The research on this project was partially supported by grant GN655 of the National Science Foundation.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF ILLUSTRATIONS</td>
<td>vii</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>viii</td>
</tr>
</tbody>
</table>

**Chapter I. INTRODUCTION.**
- The Problem of Scheduling
- Information System Scheduling
- Information System Components and Scheduling
- Fundamentals of a Pre-Scheduler
- Fundamental Assumptions
- Problem Definition
- Research Goals
- Scheduling Theory
- Presentation Development

**Chapter II. FUNDAMENTAL DEFINITIONS AND MEASURES.**
- Purpose
- Preliminary Definitions
- Program Values
- Schedule Measures
- Program Processing Times
- Program Availability, Exclusion, and Classification

**Chapter III. GENERAL COMPUTATIONS.**
- Introduction
- Program Occurrence Completion Time Estimates
- User Loss Estimates
- Resource Compatibility Computations

**Chapter IV. PRE-SCHEDULER SPECIFICATION.**
- The Scheduling Solution
- Pre-scheduling Procedure
- \( Q_3/Q_4 \) Selection Discipline
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>V. TESTING AND EVALUATION</td>
<td>88</td>
</tr>
<tr>
<td>Preliminary Discussion</td>
<td></td>
</tr>
<tr>
<td>Processing Environments</td>
<td></td>
</tr>
<tr>
<td>Experimental Design</td>
<td></td>
</tr>
<tr>
<td>Data Analysis</td>
<td></td>
</tr>
<tr>
<td>VI. CONCLUSIONS AND RECOMMENDATIONS</td>
<td>115</td>
</tr>
<tr>
<td>Conclusions</td>
<td></td>
</tr>
<tr>
<td>Recommendations</td>
<td></td>
</tr>
<tr>
<td>APPENDIX</td>
<td></td>
</tr>
<tr>
<td>A. EXAMPLE OF WORST-CASE TECHNIQUE FAILURE</td>
<td>122</td>
</tr>
<tr>
<td>B. DERIVATION OF USER LOSS FUNCTION</td>
<td>125</td>
</tr>
<tr>
<td>C. DERIVATION OF SYSTEM LOSS FUNCTION</td>
<td>127</td>
</tr>
<tr>
<td>D. EXAMPLE OF IDLE PERIOD OCCURRENCE</td>
<td>130</td>
</tr>
<tr>
<td>E. EVALUATION INPUT DATA</td>
<td>132</td>
</tr>
<tr>
<td>F. INTERPRETATION OF PROCESSING ENVIRONMENTS</td>
<td>139</td>
</tr>
<tr>
<td>G. EVALUATION MEASURES</td>
<td>143</td>
</tr>
<tr>
<td>H. TEST RESULTS</td>
<td>147</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>153</td>
</tr>
<tr>
<td>Literature Cited</td>
<td></td>
</tr>
<tr>
<td>Other References</td>
<td></td>
</tr>
<tr>
<td>VITA</td>
<td>165</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Q₂ Limit Setting for Lowest Losses.</td>
<td>97</td>
</tr>
<tr>
<td>2.</td>
<td>Q₂ Limit Setting for Most Efficient Schedule.</td>
<td>98</td>
</tr>
<tr>
<td>3.</td>
<td>Largest Mix Size and Lowest Loss.</td>
<td>100</td>
</tr>
<tr>
<td>4.</td>
<td>Deadline Effects.</td>
<td>101</td>
</tr>
<tr>
<td>5.</td>
<td>Loss Rate Effects</td>
<td>102</td>
</tr>
<tr>
<td>6.</td>
<td>Nominal System Gain Effects</td>
<td>103</td>
</tr>
<tr>
<td>7.</td>
<td>System Loss Rate Effects</td>
<td>104</td>
</tr>
<tr>
<td>8.</td>
<td>System Reconfiguration Effects</td>
<td>105</td>
</tr>
<tr>
<td>9.</td>
<td>User/System Weight Effects</td>
<td>106</td>
</tr>
<tr>
<td>10.</td>
<td>Queue Discipline Effects</td>
<td>107</td>
</tr>
<tr>
<td>11.</td>
<td>System Configurations</td>
<td>132</td>
</tr>
<tr>
<td>12.</td>
<td>Program Base.</td>
<td>133</td>
</tr>
<tr>
<td>13.</td>
<td>Q₃ Program Compositions and Data Units.</td>
<td>134</td>
</tr>
<tr>
<td>14.</td>
<td>Q₄ Program Compositions and Data Units.</td>
<td>134</td>
</tr>
<tr>
<td>15.</td>
<td>Arrival Times for Q₃</td>
<td>135</td>
</tr>
<tr>
<td>16.</td>
<td>Arrival Times for Q₄</td>
<td>135</td>
</tr>
<tr>
<td>17.</td>
<td>Q₃ User Loss Rates and Deadlines</td>
<td>136</td>
</tr>
<tr>
<td>18.</td>
<td>Q₄ User Loss Rates and Deadlines</td>
<td>136</td>
</tr>
<tr>
<td>19.</td>
<td>Q₃ User Loss Rates for User Loss Change Test</td>
<td>137</td>
</tr>
<tr>
<td>20.</td>
<td>Q₄ User Loss Rates for User Loss Change Test</td>
<td>137</td>
</tr>
<tr>
<td>21.</td>
<td>Q₃ Deadlines for Deadline Analysis</td>
<td>138</td>
</tr>
<tr>
<td>22.</td>
<td>Q₄ Deadlines for Deadline Analysis</td>
<td>138</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>23.</td>
<td>Processor Bound—Loss in $Q_3$—Even Arrivals</td>
<td>148</td>
</tr>
<tr>
<td>24.</td>
<td>Processor Bound—Loss in $Q_3$—Behind Arrivals</td>
<td>148</td>
</tr>
<tr>
<td>25.</td>
<td>Processor Bound—Loss in $Q_4$—Even Arrivals</td>
<td>148</td>
</tr>
<tr>
<td>26.</td>
<td>Processor Bound—Loss in $Q_4$—Behind Arrivals</td>
<td>148</td>
</tr>
<tr>
<td>27.</td>
<td>Processor Bound—Loss Mixed $Q_3/Q_4$—Even Arrivals</td>
<td>149</td>
</tr>
<tr>
<td>29.</td>
<td>I/O Bound—Loss in $Q_3$—Even Arrivals</td>
<td>149</td>
</tr>
<tr>
<td>30.</td>
<td>I/O Bound—Loss in $Q_3$—Behind Arrivals</td>
<td>149</td>
</tr>
<tr>
<td>31.</td>
<td>I/O Bound—Loss in $Q_4$—Even Arrivals</td>
<td>150</td>
</tr>
<tr>
<td>32.</td>
<td>I/O Bound—Loss in $Q_4$—Behind Arrivals</td>
<td>150</td>
</tr>
<tr>
<td>33.</td>
<td>I/O Bound—Loss Mixed $Q_3/Q_4$—Even Arrivals</td>
<td>150</td>
</tr>
<tr>
<td>34.</td>
<td>I/O Bound—Loss Mixed $Q_3/Q_4$—Behind Arrivals</td>
<td>150</td>
</tr>
<tr>
<td>35.</td>
<td>Mixture—Loss in $Q_3$—Even Arrivals</td>
<td>151</td>
</tr>
<tr>
<td>36.</td>
<td>Mixture—Loss in $Q_3$—Behind Arrivals</td>
<td>151</td>
</tr>
<tr>
<td>37.</td>
<td>Mixture—Loss in $Q_4$—Even Arrivals</td>
<td>151</td>
</tr>
<tr>
<td>38.</td>
<td>Mixture—Loss in $Q_4$—Behind Arrivals</td>
<td>151</td>
</tr>
<tr>
<td>40.</td>
<td>Mixture—Loss Mixed $Q_3/Q_4$—Behind Arrivals</td>
<td>152</td>
</tr>
<tr>
<td>Figure</td>
<td>Illustration Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1.</td>
<td>A Pre-Scheduler</td>
<td>9</td>
</tr>
<tr>
<td>2.</td>
<td>Round-Robin Policy Queue System</td>
<td>13</td>
</tr>
<tr>
<td>3.</td>
<td>User Loss Function</td>
<td>17</td>
</tr>
<tr>
<td>4.</td>
<td>System Loss Function</td>
<td>20</td>
</tr>
<tr>
<td>5.</td>
<td>Problem Definition</td>
<td>24</td>
</tr>
<tr>
<td>6.</td>
<td>Interrupt Cycle</td>
<td>48</td>
</tr>
<tr>
<td>7.</td>
<td>Time Reference Collection Re-evaluation</td>
<td>61</td>
</tr>
<tr>
<td>8.</td>
<td>Efficiency per $Q_1/Q_2$ Size for Activity in $Q_3$</td>
<td>109</td>
</tr>
<tr>
<td>9.</td>
<td>Efficiency per $Q_1/Q_2$ Size for Activity in $Q_4$</td>
<td>110</td>
</tr>
<tr>
<td>10.</td>
<td>Loss per $Q_1/Q_2$ Size for Activity in $Q_3$</td>
<td>111</td>
</tr>
<tr>
<td>11.</td>
<td>Loss per $Q_1/Q_2$ Size for Activity in $Q_4$</td>
<td>112</td>
</tr>
<tr>
<td>12.</td>
<td>User Loss Function Derivation</td>
<td>126</td>
</tr>
<tr>
<td>13.</td>
<td>System Loss Function Derivation</td>
<td>129</td>
</tr>
</tbody>
</table>
SUMMARY

A pre-scheduling model based upon computer performance data for systems with basic queue disciplines (classical round-robin, first come-first serve, shortest-job-first, nearest-deadline-first, and least-processing-received-first) was developed for a class of single processor, single queue multiprogrammed systems. The application environment was assumed to be fixed with respect to type and frequency of information requests and operational performance of the associated computer programs. The model was developed using industrial job-shop techniques to identify a worst-case loss situation which was consistent with current and future information requirements and operational policy. From this worst-case basis a procedure was proposed by which various multiprogrammed processing sequences could be constructed and the one which provided the minimum value for a total loss function could be identified. The trial schedules generated in this manner were examined and in some cases improvements could be realized by the application of appropriate environment-dependent heuristics. The proposed procedure was evaluated through the use of a computer program. Not only did the proposed pre-scheduler provide improvements in the user/system performance as measured by the total losses incurred, but also the procedure exhibited the potential of providing management with a tool for system evaluation. To assist in implementation of the pre-scheduler interpretations of storage and retrieval applications were provided to
suggest possible application situations and to indicate how the using community should be structured.
CHAPTER I

INTRODUCTION

The Problem of Scheduling

As a result of the ever increasing costs and complexity of computer based information systems, the computer/user interaction is becoming an obscured and greatly misunderstood process. The purpose of this discussion is to view this interaction, not from the viewpoint of mathematical disciplines, but from the identification of actual interaction parameters as established by the confrontation between system users and the computer system. In particular, the study will be directed toward an investigation of that portion of the interactive process involving the effects of management policy on the planning and control potential of the system—a study of policy control and information system scheduling.

It may be noted that many computer-using organizations have initiated self-appraisals to determine the procedures and policies which will yield the most effective utilization of their system's potential in furthering organizational goals. The introduction of the so-called "third generation" systems and the serious shortage of skilled operating and management personnel have made effective computer utilization a goal that is difficult enough to describe and virtually impossible to achieve (30).
There is much disagreement as to what factors adequately describe the interactive process, what constitutes representative measures, what factors represent meaningful decision variables, and what are relevant relationships among the decision variables which could reflect an improvement in system effectiveness. Much disagreement appears in the description of what measures system effectiveness and what represents an improvement. For example, terms such as *throughput*, *system utilization*, *turn-around-time*, and *waiting time* are often presented as descriptions of the system responsiveness; but unfortunately, their definition are frequently ambiguous, leaving doubt as to their validity as performance measures (58).

Therefore, the primary objective of this research is to provide computer system managers with an effective analysis tool which will assist them in their understanding and evaluating the performance of their user/computer system. To accomplish this goal, it will be suggested that both the user and system will have associated with their performance a loss evaluation procedure. The losses incurred during the processing of user information requests will become the foundation of a pre-scheduling procedure which can predict and monitor the effects of various program and policy situations. Before the details of the pre-scheduling process can be identified, the general scheduling problem as it is related to the user/computer system must be reviewed.

**Information System Scheduling**

Normally, when the problem of computer scheduling is considered, the discussion is centered around the operating system and some component
of it, referred to as a scheduling algorithm. It is the view here that
this is only one portion of the scheduling procedure and that the most
fundamental portion, user/system interaction control, remains ignored
for the most part.

Primary System Scheduler (PSS)

The Primary System Scheduler (PSS) is an element of the com-
puter's operating system. The PSS decisions are normally influenced
or controlled by the operator-determined policies and the remaining
control program routines must then perform as best they can under the
conditions established by the PSS. In particular, there are several
fundamental functions performed by most PSS routines.

1. Program priority conditions are recognized and priority rules
are implemented.

2. Processor time is allocated to each program requesting system
service.

3. Precedence requirements (other than priorities) among programs
are identified and acted upon.

4. Periodic housekeeping functions of the system are scheduled.

5. Precedence requirements of the peripheral devices are identi-
fied.

6. Input/output requests are scheduled for service.

7. The contents of the internal system queues are monitored for
use in making future scheduling decisions.

It should be noted that the functions described above affect those
programs which are present within the computer system. In most cases
these programs are considered for admission to the system based upon such program parameters as arrival time or priority (externally assigned or system generated), and no attention is paid to the effect on performance because of the interactive manner of program processing.

Unfortunately, there are currently no known means for scheduling of programs which not only effectively satisfy the system users but also contribute to a high degree of system utilization. Also, there are no means of utilizing anticipated information requirements in the scheduling process so as to eliminate future program conflicts with respect to demands on system facilities and user priorities.

Procedures have been suggested (6,19,21,23,32,33,45) which address certain aspects of these problems, but by only considering the allocation of or the information transfers between system components, the needs of the system users become forgotten. When the user's needs are considered, the simplifying assumptions made in order to apply existing mathematical procedures reduce the utility of the procedures and the conclusions reached are questionable as to their applicability. Since two goals of this thesis are to include in the pre-scheduling procedure the user as well as the system and to retain as many as possible of the descriptive parameters, these procedures are not applicable.

To provide a better understanding of the scheduling process which involves both the user and computer system and to introduce some of those factors which must be retained in a pre-scheduling procedure, the second portion of the scheduling problem--the environmental control function--will be reviewed.
Environmental Control Function (ECF)

The Environmental Control Function (ECF) involves the screening of program requests according to management and operating policies. It is the contention here that if these policies could be quantitatively described and applied in the screening of the program requests, then an improvement (in terms of measures to be proposed) will be realized. This is to say that the computer system operational environment must be carefully evaluated through the accurate analysis of the true user information requirements and their effect on system performance.

Generally speaking, current operational philosophy holds that the computer system is a tool to be utilized at the user's convenience, and that the system should not reject any valid request or, under normal circumstances, refuse service to any class of users. This philosophy, however, has obvious limitations. It must be recognized that a computing system cannot be all things to all users, nor can it decide who the users should be. A system can, however, under a given operating policy, assist in the specification of a limited, well-defined user-environment to be served. It is this role that the ECF is to play—to act as an interface between the user and the computer system providing information as to the capabilities of the system with respect to its ability to service user programs.

It is this second portion of scheduling which is to be the primary

*As will be seen later, the problem assumptions will permit the reversal of this procedure; i.e., the ECF will also be able to restrict programs which are not compatible with some best operating policy.
subject of this paper. To bring the problem into focus, the major components will be described generally and the particular problem elements of this thesis will be identified.

Information System Components and Scheduling

It is the purpose of this section to describe the relationship among the various components of the total system and how they might relate to the scheduling problem. That is, the concern is not only with the computer system and its related "hardware" and "software," but also with the user community, the nature of the information requests, and the conditions and time frame under which they are submitted for system processing.

At the level of the PSS the conditions could represent the queue handling disciplines, the users could denote the program priorities, and the time frame could represent the various time parameters such as the arrival time, processing time or time of the last processing received. The scheduling function, usually referred to as a resource allocation function, has the requirement to review these various elements and to select the appropriate programs to be processed.

From the view of the ECF, the conditions not only represent the internal status of the computer system, but also the management policies as established for system utilization, stimuli resulting from information requests, and management policy controlling information requirements. The system users are now those who require system service and the time frame can be viewed relative to the workday, the time of last
information request, or the time required for system service. The scheduling function can be interpreted as a selection procedure for program identification. Once identified by the scheduling function, the programs will be submitted for service by the system processor. In addition, the ECF scheduling element can be interpreted as a forecast procedure to reduce user/system conflicts, as a user control function, or as a user/system pre-scheduler. It is the last of these—a user/system pre-scheduler—which is to be the primary concern of this paper. As Scherr (49) has pointed out, such a scheduling process could result in the control of the system users with respect to their system interactions. Hence, denoting the procedure as the ECF appears appropriate.

From this introduction it should be evident that this study will be directed toward the specification of the scheduling function and how it is to be used for user/computer system pre-scheduling. The structure identified by the system users, the different time sets and variety of program bases will not be directly explored as to their characteristics. These will be fixed and the pre-scheduler will be identified within this assumption. The nature of the collection of conditions and especially the role of management will be identified within the development of the pre-scheduler and its measures. The general functions of a pre-scheduler can now be described.

Fundamentals of a Pre-Scheduler

The pre-scheduling process is to be related to the user/computer system as shown in Figure 1. As depicted, the ECF will accept the user information requests and, based upon the current system status, policy
conditions, and operating conditions, make a decision as to what programs are to be submitted for computer system processing. The manner in which the system responds to the user demands will then be reflected in user satisfaction and performance. If the users' information requirements are for some reason not satisfied or if the policy-making components of the system are not pleased with the system's service, then information can cross the policy/operating condition barrier and influence the manner in which the system is to react to future information requests. Another and perhaps a more important result is the influence that policy matters can have on the using community. By establishing new priorities or service periods, the users' information requirements may be altered or submission periods changed with an overall improvement in user/computer system performance.

Policy Data

From Figure 1 notice that the ECF is to provide both the user and policy component with performance data. This data is not that which is usually received from the normal system accounting functions. Statistics on system utilization, the number of device accesses, and processor time utilized leave a good deal to be desired for the establishment of reasonable management practices for a computer system. Management must have better tools to assist in their decision making—tools which reflect system performance relative to the overall objectives of the organization. Therefore, one of the primary goals of this research will be to propose a collection of measures, obtainable from existing accounting and performance data, to be provided by the ECF.
Figure 1. A Pre-Scheduler
Pre-Scheduling

Another function of the ECF is to anticipate information requests from system users and to incorporate this knowledge into the pre-scheduling process (41). There arises the question as to whether service to some requests should be granted, especially if service to other, more important requests must suffer. It will then be a function of the ECF to view current and anticipated user information requirements and to submit for processing only those information requests which have the best chance of providing the highest quality of system service.

To accomplish this, there must be established the various system and user performance measures, a pre-scheduler component for utilizing forecast data, and a component for determining compatibility of program resource requirements. These factors must then become the integral portions of the pre-scheduling procedure to specify what programs are to be presented to the computer system so as to provide the best possible system service. It must be pointed out that the ECF will now block certain user's requests from the system or delay others in favor of those requests which meet the policy objectives. This is in contrast to the usual, current utilization of computer systems.

Now that the problem and general goals have been introduced, the specific problem assumptions and objectives can be presented.

Fundamental Assumptions

It should be evident that the problem as stated cannot be investigated in this discussion for all possible systems and user application environments. Therefore, several assumptions will be made
regarding the system classes to be considered, along with a further set of assumptions to simplify investigation of the fundamental elements of pre-scheduling.

**System Class**

Only those computer systems which have the following hardware limitations and hardware/software capabilities will be considered in this thesis:

1. Remote input/output terminals.
2. The ability to respond to remote information requests in "real time," in addition to satisfying batch processing requests.
3. A single central processing unit with multiprogramming capability.

**Application Environment**

The broad collection of time-shared user applications will be far too general for this discussion. This is due primarily to the large number of user/application variables present. To reduce the problem area and to make it more manageable, the problem will be limited to a subclass of information processing normally referred to as "data processing." Specifically,

1. There will be a known and fixed membership of the information user community.
2. The programs which process the users' information requirements are fixed and are not subject to change by the system users. This requirement includes knowledge of core requirements, peripheral utilization, and input/output channel activity.
3. The processing times are known for all information requests for the run-alone case, and the times for other situations will be functions of the system processing loads and system queue position.

4. The future information processing requirements of some system users are identifiable as to user, processing policy and conditions, and time interval for which pre-scheduling is to occur.

5. Information requests will be permitted which are not known in advance as to their arrival for processing, but they must satisfy the other restrictions.

**Operating Systems**

As with possible application situations, not all operating systems can be examined. Since this is a scheduling problem, the element of the operating system which is of primary interest is that which governs the handling of the programs which have been queued and are awaiting processing. There are a variety of policies and disciplines which specify how programs will be sequenced (15). It appears that most sequencing policies can be derived from the following basic disciplines either directly or through some hybrid collection. These include a cycling policy, a sorting policy, or a laissez faire policy. In particular, this paper will consider

1. Classical round-robin queue policy.

2. First come-first serve queue policy.

3. Queue reordering by (a) nearest-deadline-first, (b) shortest-job-first, and (c) least-processing-time-received-first.
It is not necessary to discuss the detailed, internal structure of the operating system for these particulars will be reflected in how well the computer system responds to the user initiated information requests. That is, the performance factors to be proposed by this study should contain sufficient information upon which to establish a prescheduling procedure. The queue discipline that has been implemented for the system under study will perhaps alter the scheduling procedure, but the scheduling parameters will be applicable to all. The particular assumptions relative to the queue disciplines to be studied are introduced below.

Round-Robin Queue Discipline

The fundamental property of the round-robin queue discipline is that there tends to be little or no advantage in processing performance due to the initial program position in the queue. As shown in Figure 2, programs which do not reach completion upon interrupt occurrence are recycled to the end of the queue.

![Figure 2. Round-Robin Policy Queue System](image-url)
The result is that all programs tend to have equal chance for processor service, and the final completion times of the programs become independent of the initial queue position. It is under this assumption that the round-robin pre-scheduler is to be developed.

First Come-First Serve Queue Discipline

The first come-first serve (FCFS) discipline differs from the round-robin in that once in the queue there will tend to be a position advantage for those programs near the head of the queue. This is primarily due to the assumption of requiring, upon interrupt occurrence, that control return to the head of the queue for the next program to receive processor service. If for some reason the first program is not available, then control is passed down the queue until a program is found to be ready for processing. The position advantage will be defined with reference to program processing times and interrupt activity.

A large number of systems utilize this queuing discipline if for no better reason than a tie breaker among programs which have some equal scheduling parameter (e.g., priority). The FCFS procedure will be understood to reference the program position in the processing queue and those near the head of the queue will achieve service prior to those following. In addition, it is assumed that the pre-scheduler has knowledge of the contents of the processing queue and can order programs prior to submission to the queue. Otherwise, the processing queue order will remain static except as it is affected by the normal completion of programs.
1. **Shortest-Job-First (SJF).** The running time remaining will be estimated for those programs in the processing queue, and the queue will be sorted in ascending order by the estimated time.

2. **Nearest-Deadline-First (NDF).** Associated with each program will be a deadline time after which the need for the product increases. The time difference between the deadline time and the current time will be evaluated for all programs in the processing queue and the queue sorted in ascending order according to this difference.

3. **Least-Processing-Time-Received-First (LPR).** This procedure tends to insure that every program receives at least one try for processing. The total amount of processing time received by each program in the processing queue is determined, and the programs are ordered in ascending sequence of this value.

It will be assumed that the various reorderings are to occur only at a mix change; that is, upon the deletion of a program. Between the reorderings the control will be assumed to be the same as in the case of the FCFS situation.

**Values**

The goal is now to specify a measure which will represent the value of a particular processing schedule. The pre-scheduler as it has been introduced is concerned with two factors: the information processing requirements of the system users and how the system responds to these requirements. It is desirable that the system measure be a

---

*Also has been called Shortest-Elapsed-Time-First (SET) in (59).*
function of a user value and a system value, but there is disagreement as to the exact structure and nature of the value assessment (29,31,34, 46,47,48,53,54). For this research a user loss and a system loss are proposed.

The amount that the user can pay for an access to the computer system should perhaps be secondary to losses incurred by him as a result of having to wait for system access and degradation in processing due to other users receiving service. Hence, the user loss as incurred by the system user for having to wait for the computer system to respond is suggested as a foundation for user value evaluation. Identified, where applicable, will be a relative deadline time denoting that time after which the program value will tend to increase at a faster rate. Again this deadline time value is relative to the submission time of the information request.

For the purpose of this discussion, these user/program values or loss functions will be assumed to be linear functions with the provision that the slope may change when the completion time exceeds the deadline time (see Figure 3). This linear assumption appears justified for the following reasons:

1. The condition closely resembles the relatively static priority structure as now established in the operation of many computer systems.

2. The problem generality will not be lost since the proposed procedure is independent of the form of the loss function. That is, as long as the loss is expressed as a function of the completion time, it can be used in the pre-scheduling procedure.
Figure 3. User Loss Function
3. The computation procedure will be simplified and more easily understood by system management.

4. The estimation procedures required in the specification of the loss rates will be greatly simplified.

Under these assumptions there will be retained for each program, operating condition, and system user at most three user loss parameters. These will consist of the deadline time relative to the time of program submission; the loss rate prior to the deadline; and the loss rate after the deadline.

System losses pose a different problem. The performance of the computer system must be measured based upon the actual system response to information requests rather than on an a priori specification by system management. Since this is the situation, specific discussion will be deferred until a later chapter and only a preliminary introduction to the general problem elements will be presented at this time.

A quantity must be defined which will provide an idea of the effect of multiprogramming on system performance and efficiency. In this thesis the quantity selected will be referred to as the system gain, a quantity very similar in its definition to the improvement factor as introduced by Hellerman (33). A value of 1.0 will denote serial processing mode whereas any value higher should denote a multiprogramming improvement. In particular, the value range which will be of interest will be that region between 1.0 and the nominal system gain value achieved by the system. The relationship between the system gain and a system loss value can then be defined in terms of loss rates—one
before a gain of 1.0 and another from 1.0 to the nominal value. This relationship is shown in Figure 4. If the nominal gain is reached, then no system loss is to be assessed. If the gain is less than 1.0, the system is functioning poorly with respect to multiprogramming, and the serial processing case would be superior. In this region, a linear loss function with steep negative slope will define the loss value. For a gain value between 1.0 and the nominal value, the system loss is defined by a linear loss function with a smaller negative slope.

Through the process of pre-scheduling and due to the characteristics of the input processing stream, it is possible that the computer system could enter an idle status. It will be assumed that there will be no penalty assessed for such a period, because many computer usage contracts are written so that the system user pays only for the actual system time used. If an idle penalty were assessed because effective multiprogramming (represented by a high system gain) caused the day's workload to finish early, then there would be less incentive for processing improvements. On the other hand, the occurrence of an idle period could increase the response time experienced by the system users. This can result when the pre-scheduler decides to delay a program on hand in favor of one which has not yet arrived. The occurrence of this delay will be adequately reflected in the increases of the user losses.

Finally, the assessment of an idle period penalty would place an increased importance on the computer system and its "need to keep busy" and would violate the contention that the computer is in reality a servicing facility. Therefore, the losses induced by the occurrence of
Figure 4. System Loss Function
an idle period will be reflected only indirectly through the increases of user losses and system losses.

Program Processing Characteristics

Closely related to the computation of the user losses and system gain is the ability to view the system performance information and to obtain an estimate of the program processing times for various loads placed upon the computer system. Since these factors are measured quantities, their definition will be deferred until the next chapter and only an introduction is presented at this point.

The general methodology of processing time accounting is to be based upon what will be referred to as the "interrupt cycle."* The interrupt cycle is to be viewed as the period which lies between the occurrence of two consecutive processor interrupts. This period is to be divided into two portions: the time spent in processor service and the time spent locked out of processor service due to input/output activity. For the purpose of this discussion it will be assumed that every interrupt cycle is the same (i.e., the processor and lockout periods will not vary in length from one cycle to the next) and the time slice will be longer than the cycle length. This assumption again requires the collection of appropriate running statistics from the system performance histories and its analysis to obtain some average value. It is recognized that this averaging assumption induces a certain amount of error into the results, but the analysis of this is

* (40) and (60) used this concept in analysis of multiprocessing systems. These works failed to provide a method to accurately prorate a single processor's time among programs in the mix.
beyond the scope of the thesis. This utilization of performance data, however, has been suggested as a basis for analysis for methods of obtaining system improvement (6,56).

Based upon a fixed interrupt cycle, the processing descriptors can be evaluated. This procedure can be outlined as follows:

1. Prorate the amount of processor service each program is to receive.

2. Determine the interrupt cycle activity of the entire contents of the mix of programs in the processing queue and adjust the processing effectiveness to account for overhead activity and unused processor time.

3. Determine the elapsed time of the programs in the mix at the instant the first program(s) exits the mix.

The manner in which the first item will be accomplished will be found to depend upon the selection of the queue handling discipline, whereas the last two goals can be realized by a general time accounting procedure.

The resource requirements of each program will be determined through the monitoring of the system accounting and performance data. Peripheral device demands as well as memory requirements can be determined easily from the accounting records maintained by the system. To eliminate memory accounting problems, each program will be permitted to overlay with only itself and must remain within some established maximum allocated space. The specifics will be presented in the next chapter.

**User Demands**

The final assumptions are concerned with the information demands
of the user community. It has been pointed out that the user should be allowed some place in the scheduling process (45). To accomplish this the input stream will be separated into two categories.

First, in order for the pre-scheduler to accurately make a scheduling decision, it must have some idea of the future demands to be placed upon the system. It will be assumed that for the time period to be scheduled (the scheduling interval), certain information requirements such as programs required, system demands, submission time, and number of data units to be processed are known for some users. In addition, other program requests will be permitted to arrive for system service without prior knowledge of their arrival. These programs must still be a member of the known program base, in order to have their processing characteristics available for scheduling decisions.

These assumptions are not unrealistic since the user may be required to specify or to verify an estimate of his need prior to the beginning of the scheduling period. Those who cannot or who have unexpected run requirements are permitted access to the system but will be generally considered after those in the known input stream. The preferential consideration given to the known input should provide sufficient incentive to the system users to plan their processing requirements.

**Problem Definition**

Specifically, the problem is to specify a pre-scheduling interface between the computer system and the using community. Referring to Figure 5 the programs with the known arrival times are to be placed in
$Q_3$ and will be given preferential treatment. Those programs which arrive for service with no \textit{a priori} knowledge of their arrival will be the members of $Q_4$.

![Diagram of program selection and scheduling processes]

\textbf{Figure 5. Problem Definition}

The programs are then selected from these queues for submission to $Q_1$ (the processing queue of the computer) in such a manner as to minimize a loss function based upon user and system losses incurred as programs exit $Q_1$. Because of anticipated excessive scheduling time for large numbers of programs and the actual arrival characteristics of processing requests, the input stream will be processed piecemeal through the intermediate queue $Q_2$. The pre-scheduler will initially fill $Q_2$ to some size limit from $Q_3$ and $Q_4$. As programs are submitted to $Q_1$ from $Q_2$, the number of programs in $Q_2$ is monitored and when a lower limit is reached, $Q_2$ is refilled from $Q_3$ and $Q_4$. The design of the pre-scheduler includes
Q_1 \text{ and } Q_2 \text{ and a selection scheme for obtaining programs from } Q_3 \text{ and } Q_4.

The particular goals of the study of the computer pre-scheduling problem can now be presented.

Research Goals

The specific goals of the study as delimited by the assumptions can be stated as follows:

1. Utilizing system performance and accounting data, describe the construction of the basic components of a pre-scheduler to include (a) how to construct the initial program mix and schedule; (b) how to obtain a decision to delay the processing of a program; (c) how to utilize estimates of future information requirements in the specification of a schedule; and (d) how to reorder a program mix to obtain an expected improvement in a measure of processing performance.

2. Describe procedures (perhaps different) for each of the basic queue disciplines under study.

3. Evaluate the improvements obtained through pre-scheduling versus a system without pre-scheduling.

4. Indicate how well the pre-scheduling procedure responds to changes in management-established policies and how these changes might affect system measures.

5. Specify extensions to this study as to improvements and related areas of study.

Now that the research area and the specific problem have been outlined, the problem will be examined with respect to existing scheduling techniques.
Scheduling Theory

Studies in queuing theory have provided expressions for describing the fundamental attributes of queuing systems. These have been incorporated into procedures which can be used to describe the sequencing of tasks which have priority constraints. Industrial job-shop models have been developed which provide economic interpretations to certain application environments. These procedures, however, have certain limitations with respect to their applicability to the problem presented here. The objective of this section is to explore these limitations.

Queuing Theory

Perhaps the most popular of the mathematical disciplines which have been applied to the investigation of the behavior of time-shared computer systems is queuing theory. One of the first to apply the fundamentals of the queuing techniques to time-shared systems was Kleinrock (36,37). His works were extended and modified in other related studies (1,2,5,8,9,11,12,13,14,16,17,24,27,39,43,59). The major developments in each of these studies required two basic assumptions: (1) task arrival characteristics were described by random variables which were independently distributed according to some (usually Poisson or exponential) distribution, and (2) service performance of the processor was described by similar random variables. With these assumptions and various others concerning queue handling disciplines and priority structure, mathematical expressions were derived to provide descriptions of the basic system performance parameters. These developments,
however, depended upon the initial independence assumptions which cannot reflect and account for the inter-dependencies and interactions which are certainly present among system users and the programs processing their information requests. In addition these studies did not permit deadline constraints except as implied through the specification of the static priority classes.

Related to the queuing models were procedures which used a Markov process to model computer system behavior (10,55). Each of these studies required assumptions similar to those above for the development of expressions for the evaluation of service times, waiting times, and system responsiveness. One of the latest (Chang, 10) utilized an embedded Markov process and developed expressions for the above parameters by applying generating functions and Laplace transformations to several basic expressions. Again to arrive at the final closed form expressions, the initial independence assumptions were required plus the additional requirement of stationary transition probabilities. The prescheduler developed in this research, however, does not require either of these assumptions. Hopefully, this will allow the consideration of less restrictive processing environments.

**Industrial Job-Shop Scheduling**

The techniques of industrial job-shop scheduling have also been applied to the computer scheduling problem. This scheduling discipline removes, in some of its applications, the requirement of independently distributed system parameters, allows for deadline constraints, and provides the machinery for imposing economic constraints on the scheduling problem.
As indicated in the previous sections, there are to be developed measures to reflect schedule performance. To be meaningful to information system management, such measures should have to some extent an economic interpretation. Procedures with such measures have been developed by Burstall (7), Elmaghraby (22), Gere (28), Kortanek and Maxwell (38), Akers and Freidman (3), McNaughton (44), Schild and Freidman (50), Fife (25), Banerjee (4), and are summarized in Conway, Maxwell, and Miller (18). These proposals, however, require relatively static assumptions with respect to arrival points, cost functions, processing times, and deadlines. For example, one or more of the following are required in each of these procedures:

1. Loss functions will be described by a single variable function which is normally linear.

2. Tasks to be serviced are late (i.e., past their deadlines) upon arrival to the system for service.

3. All tasks are assumed available for processing prior to scheduling.

4. The processing times required by the tasks are independent of the time required by other tasks. If there are time dependencies, it will be a function of only those tasks which precede in the scheduling sequence.

5. The completion times are dependent only on the run times of the tasks implemented prior to the current task.

6. No machine may process more than one operation at a time.

7. Each task, once given service by the processor, must run to completion before the processor will service any other task.
Lawler (42) relaxed the requirement of a linear loss function. A deferral cost function was introduced to represent losses incurred by facility users and the only limitations on the function were that it be a monotonically non-decreasing function of the task completion times.

To the experienced reader it should be evident that items 3, 5, and 7 do not describe a multiprogrammed computer system. Since these are fundamental to the job shop scheduling foundation, the procedures cannot be directly applied. Lawler was able to relax the costing restrictions, and it will be possible to couple a portion of his approach with a dynamic programming-like procedure to provide a worst-case cost analysis in the pre-scheduler approximation routines.*

Even though the existing scheduling models, queuing and industrial job-shop, were not directly applicable to the problem under study, they do prove instructive in the identification and classification of parameters which must be monitored and considered in any scheduling process.

**Presentation Development**

The accomplishment of the goals as outlined within the constraints of the proposed problem will now be approached through the specification of a collection of basic definitions which formalize many of the notions introduced. With this as a foundation, calculation procedures are derived and the basic pre-scheduling components specified. Finally, the fundamentals of the pre-scheduler will be incorporated into a computer

---

*See Appendix A for an example of the inadequacy of the serial processing approach.
program and processing situations will be tested to determine possible
effects of management decisions on the pre-scheduling results.
CHAPTER II

FUNDAMENTAL DEFINITIONS AND MEASURES

Purpose

The aim of this chapter is to present formally the pre-scheduling problem by specifying the exact implications of the assumptions, by identifying a collection of measures to be associated with various problem elements, and by developing notation to be used throughout the thesis.

Preliminary Definitions

Essential to the development are several terms introduced in the previous chapter which deserve more precise definitions. These include (a) program mix, (b) program schedule, (c) scheduling interval, (d) program entry to a mix, (e) operational conditions, (f) policy conditions, and (g) resource compatibility.

Mix and Schedule

The first definitions to be presented concern the terms "mix" and "schedule."

Definition 2.1. A program mix (or mix configuration) is an ordered collection of user-requested programs, \( p_i \) (i.e., the members of \( Q_1 \)), which are being serviced by the computer system. Notation: \( \text{MIX} = (p_1, \ldots, p_n) \).
To distinguish one program mix from another, subscripts will be used. For example, MIX_i and MIX_k will denote two program mix configurations. It will also be useful to note that the notation of MIX is to describe the particular contents of Q_l and hence will be used as a variable to describe the contents of Q_l.

Often it is necessary to know if two program mix configurations are "equal." Since a mix is simply a vector whose members are programs, the definition of equality is that of vector equality.

Definition 2.2. MIX_i = MIX_j if and only if MIX_i = (p_1, ..., p_n) and MIX_j = (p'_1, ..., p'_n) and p_k = p'_k for 1 ≤ k ≤ n denote the same user-requested program.

From this definition, for two mix configurations to be the same they must first have the same number of programs, and secondly, the programs must not only be the same but also ordered in the same sequence. A computer schedule can be defined in a similar manner.

Definition 2.3. A schedule, S, is an ordered sequence of mix configurations such that S = (MIX_1, ..., MIX_m) and MIX_i ≠ MIX_{i+1} for 1 ≤ i ≤ m-1.

Notice that two adjacent mix configurations cannot be the same, but the same mix configuration may reoccur many times throughout a schedule in non-adjacent positions. As with a mix configuration, schedule equality can be defined.

Definition 2.4. Schedules S_i and S_j are equal, S_i = S_j, if and only if S_i = (MIX_1, ..., MIX_m), S_j = (MIX'_1, ..., MIX'_m) and MIX_k = MIX'_k for 1 ≤ k ≤ m.
Scheduling Interval

The pre-scheduler has as one of its primary objectives to schedule not only those programs which are currently awaiting processing but also to plan for future information processing demands. How far ahead this planning is to go is denoted by the scheduling interval.

Definition 2.5. The scheduling interval, SI, is an ordered pair (t, t+k) where t is the time at the beginning of the planning interval, t+k is the time at the end of the planning interval (k positive), and the contents of Q are known from t to t+k.

In the previous definition the time is relative to the actual clock time which governs the operation of the total system.

In order to describe the processing and arrival behavior of programs, there must be certain values representing the time parameters of each program. Specifically:

Definition 2.6. The time parameters of a program occurrence, p, are:

a. Arrival time—the clock time that the program p is submitted by user u under condition c for processing. Notation: TA(p,u,c).

b. Entrance time—the clock time that program p submitted by user u under condition c is entered to Q for processing. Notation: TE(p,u,c).

*To be defined formally in the next section,

**The exact specification of the condition is unimportant. All that is relevant is that there do exist factors which not only control the nature of the user request but also control the manner in which the system responds.
c. **Deadline time**--the elapsed time relative to the arrival time as specified by user u denoting the due time for program p under condition c. Notation: $DLTIM(p,u,c)$.

d. **Run time**--the elapsed time from entrance time that it takes program p submitted by user u under condition c to complete processing. Notation: $TRN(p,u,c)$.

e. **System time**--the total system time of program p submitted by user u under condition c is defined as

$$TS(p,u,c) = TRN(p,u,c) + TE(p,u,c) - TA(p,u,c)$$

f. **Total elapsed time**--the total elapsed time of program p submitted by user u under condition c is that amount of time that p has been in $Q$ at a given time t. Notation: $TEL(p,u,c,t)$, with limits of $0 \leq TEL(p,u,c,t) \leq TRN(p,u,c)$.

**Working Set**

In the previous discussion the core-overlaying characteristics of a program were restricted to a prespecified maximum. Any core swapping or memory overlay must take place within this allocated space. Therefore, if there were not sufficient core memory available, no other program would relinquish its memory in favor of another.

The amount of memory required for a program will be referred to as the **working set**. That is,

**Definition 2.7.** The working set of program p, $W_p$, is the fixed memory allocated to and used only by p.
The size of $W_p$ is the subject of various discussions (16, 20, 51, 57), but the efficiencies of memory allocation are not the direct concern of this paper. The pre-scheduler will function based upon those working sets which are provided regardless of their efficiencies.

**Mix Membership**

The status of a program with respect to servicing by the central processing must be represented by membership in one of three mutually exclusive classes: those programs awaiting service, those receiving service, and those in neither. In order to identify uniquely the programs and classes, several comments concerning program identification must be made.

It has already been observed that the same program may be repeated within any queue. Even though these are the same program, each occurrence represents a different information request. It appears that one program occurrence should be distinguished from others by identifying with the program the user requesting service, the time of request, and the number of user-submitted data units\(^*\) to be used in a given processing. To have facility for describing this program identification, the following definition is offered.

**Definition 2.8.** A program occurrence\(^**\) of a program $p$ is identified by the notation $p\{u, TA(p,u,c), D\}$ where user $u$ made the run request of $p$ at time $TA(p,u,c)$ under a condition $c$ and required that $D$ data

\(^*\)This, in effect, specifies the run time of the request.

\(^**\)Where specific occurrence identification is not required for the continuity of the discussion, the notation will be reduced to $p$. 
units be processed.

Within the definition above one program occurrence is equal to another if and only if its descriptive parameters of user, condition, arrival time, and data units are the same. The collection of program occurrences can be classified for scheduling purposes as follows:

Definition 2.9. The program occurrence classes are: (a) a program occurrence is available at some time $t$ if and only if $T_A(p, u, c) \leq t$ and $p(u, T_A(p, u, c), D)$ is not in $Q_1$; and (b) a program occurrence is a mix member if and only if it has been selected as a member of $Q_1$ and $T_A(p, u, c) \leq t$.

Programs which cannot be described as an occurrence are of no interest to the pre-scheduler and are left in the "all others" category. This implies that until a user and arrival time can be identified with a program, either through arrivals to the system for processing or through forecast procedures, the program should be eliminated from scheduling considerations. As it will be shown below, the circumstances which permit determination of those programs which are not to be occurrences can be delimited by the examination of the operating and policy conditions which are in effect.

The Condition Set

The specification of this set of conditions is perhaps the most elusive. The exact membership can be subject to a variety of situations, each being dependent upon the particular operation environment under investigation. What can be presented here, in a relatively general context, are three possible condition categories and how they might
affect the pre-scheduling situation. These categories include policy conditions, operating conditions, and request conditions.

Policy Conditions. The first and the most far reaching is that of policy. Based upon the overall goals and objectives of management, certain directives may be established under which the user group and computer system must perform. Examples of such might include:

1. All batch-type programs will not be processed during real-time operations.

2. Interactive program development will be restricted to certain time periods.

3. Users will be given unlimited and unrestricted system access.

4. The programs should be scheduled to produce highest throughput (programs processed/unit time).

5. The system should stay as busy as possible.

6. Real-time, remote processing will be limited to certain time periods or to certain forms of interactions.

7. User priorities as reflected in the user losses influence the system scheduling routines.

As indicated these are only examples, but there should be an indication as to what could be in the category of policy conditions. The ones specifically applicable to the proposed pre-scheduler include items 4, 5, and 7.

From the examples there appears to be two categories of policy conditions. One type assists in the determination of what are acceptable information requests. Items other than 4, 5, and 7 restrict in
some way the submission of information requests. The other category is related to how programs, once submitted to the pre-scheduler, are to be selected for processing. It is here that items 4, 5, and 7 play an important role. Depending upon which is in effect, the pre-scheduling procedure should respond appropriately. It will not, however, be the specific goal to investigate completely this area, but rather to show that the pre-scheduler can react according to the given condition.

Operating Conditions. Operating conditions are computer system oriented. Generally, these concern system capabilities with respect to available devices and with device responsiveness. Again there are two categories: one restricting programs eligible for processing and one influencing the pre-scheduling process. In the first group could be items such as,

1. A particular peripheral device is not available for use.
2. Core memory is reserved for certain system or user programs.

Such items can be incorporated in the resource compatibility (conflict) procedures and no further consideration need be given to them.

The second group can be described generally by the particular system load. Depending on the system load at the time, there may be programs in $Q_2$ which, when submitted to $Q_1$, will run better (in terms of the system measure) than others in $Q_2$. In fact, if the system load rate is sufficiently large, serial processing of the programs may have been superior to their multiprogramming. Care should be taken by the pre-scheduler to recognize such situations and to make appropriate adjustments in program pre-scheduling. The particular effect that variations
in system load has on schedule results will be investigated as part of the proposed pre-scheduler.

Request Conditions. The final condition class, that of request conditions, is determined primarily by the particular using environment in which information processing is to occur. Because of known situations, it is certain that some programs will never be submitted for processing. For example, information requests to serve financial users may not occur in a military operation, if it is known that the installation has an immediate mission of servicing and flying aircraft. Also if a particular branch of a bank closes earlier than others, then the requests related to customer service may be excluded from the pre-scheduling process.

From the above, the conditions will be used to reduce the programs eligible for processing and to influence the pre-schedule selection of programs for processing. Only the second of these will be considered in pre-scheduler development since it must perform with whatever eligible program collection is presented.

The last of the preliminaries deals with the facility requirements of individual programs.

Resource Compatibility

Every program imposes upon the computer system requirements for certain system facilities. If the facilities are available, then the program can be run. If for some reason the facility is not available (i.e., busy or not functioning), then the program creates a conflict and hence is not resource compatible with either the other programs in Q₁ or
the computer system status at this time. It is the goal here to establish this notion of resource compatibility, as well as pin down several terms introduced in its discussion. First, the system facilities can be categorized with respect to core memory requirements and facility requirements. Memory demands have already been introduced in terms of working sets, but the manner in which peripheral devices are handled has not yet been pursued.

Input/output (I/O) devices can be classified as to those which may be shared among several members of Q1 and those which may be used by only one program per mix. Thus, if I/O device j can be shared by \( d_j \) mix programs, we have

**Definition 2.10.** An I/O device \( j \) is bound if and only if \( d_j = 1 \) and is sharable if and only if \( d_j > 1 \).

Thus from Definition 2.7 and 2.10 the definitions of resource compatibility can be presented.

**Definition 2.11.** Program occurrences \( p, p' \) are resource compatible if and only if

a. \( W_p + W_{p'} \leq MEM \),

b. \( a_j + a_j' \leq d_j \),

where \( MEM \) is the total memory available and \( a_j, a_j' \) are non-negative integers representing the demand by \( p, p' \), respectively, for I/O device \( j \).

From 2.11 it is desirable to introduce the program resource list which will be available for each program occurrence. If there are \( n \) different I/O devices, then
Definition 2.12. The resource list, RL(p), for program occurrence p is an n+1 tuple such that

\[ RL(p) = (W_p, a_1, ..., a_n) \]

where \( a_j \) is an integer such that \( a_j \geq 0 \) and \( a_j \) describes the program requirements for I/O device \( j \).

In the above definition most of the \( a_j \)'s will be 0 or 1 for the peripheral devices but could be other positive integer values for devices such as I/O channels and disk space (segments) utilized.

To simplify the notation define a relation \( RC \) as follows:

\[ RC = \{(p, p')|p, p' \text{ are program occurrences and } \text{Definition 2.11 is satisfied}\} \]

This will be written \( RC^2 \). If \( n \) programs are resource compatible, then \( RC^n \) will be used to denote this condition. The definition will be extended later through the specification of a computation procedure for determining if a collection of \( n \) programs are resource compatible.

This concludes the preliminary definitions. Interest is now directed to the discussion of the system gain factor, the system measure, and the available program classes.

Program Values

Two values have been introduced which are to be associated with each program. These were depicted in Figures 3 and 4 as user loss and
system gain. The purpose here is to provide adequate notation for user loss specification and to define what is meant by system gain.

**User Loss Function**

As shown in Figure 3 there is to be associated with each user/program pair a linear loss function which has a provision for a rate change at the deadline point $DLTIM(p,u,c)$. This will be denoted as follows.

**Definition 2.13.** The *user loss rates* for a given program $p$, user $u$ and conditions $c$ will be written $ULR(p,u,c)$ where

$$ULR(p,u,c) = (R_1, R_2, DLTIM(p,u,c)),$$

$R_1$ is the rate before the deadline, and $R_2$ is the rate after the deadline, $DLTIM(p,u,c)$.

If there is to be no deadline applied, then $R_1 = R_2$ and the deadline can really be any value.

**Definition 2.14.** The *user loss function*, $ULF[p,u,c,TS(p,u,c)]$ is defined as

$$ULF[p,u,c,TS(p,u,c)] = \begin{cases} 
R_1 \times TS(p,u,c), & \text{if there is no deadline or } TRN(p,u,c) + TE(p,u,c) < DLTIM(p,u,c) \\
R_2 \times TS(p,u,c) + DLTIM(p,u,c) \times (R_1 - R_2), & \text{if there is a deadline and } TRN(p,u,c) + TE(p,u,c) > DLTIM(p,u,c)
\end{cases}$$
The latter of these values is derived in Appendix B. There is still the requirement that $R_1$ and $R_2$ are to be specified by the management function of the system.

**System Gain and System Loss**

Unlike the user loss rate which is solely a policy decision, the system losses must be reflected by a measure based upon the actual system performance and not upon some *a priori* specification as a function of deadline time. Measures should not depend upon specific systems or require time consuming monitoring.

Of primary interest is the overall processing degradation experienced due to multiprogramming of user requests and not the idle conditions of equipment. This information can be obtained from system accounting data and does not require additional instrumentation packages. That is, it is not necessary to monitor peripheral or processor performance since all that is required is the start and completion times of a given program run along with the number of data units processed. The multiprogram benefits are to be represented by a gain value over the serial processing case.

When processed serially, a certain amount of time is required for the completion of the program(s) associated with some information request. This run-alone condition (i.e., a mix consisting of only one program) will be used to establish a standard with which to compare the program and system performance.* The multiprogrammed run time per data

---

*Similar to the idea proposed by Hellerman (33).*
unit for the program mix may be larger or smaller than the sum of the individual run alone processing times per data unit. It is this ratio of the run-alone times to the multiprogrammed run time that will define the system gain.

Definition 2.15. The system gain for a MIX, \( SG(MIX) \), is defined as

\[
SG(MIX) = \frac{\text{Sum of the run-alone elapsed times}}{\text{Multiprogrammed run time}}
\]

This factor can be interpreted in the following manner:

1. \( SG(MIX) < 1.0 \) --better to process serially.
2. \( SG(MIX) = 1.0 \) --no improvements expected from multiprogramming.
3. \( SG(MIX) > 1.0 \) --better to multiprogram.

If the system is committed to multiprogramming, it should be obvious that the first case should be avoided as much as possible. This observation will influence greatly the scheduling procedures.

For each computer system and, perhaps, policy condition there will be some nominal value of \( SG \). This is to be defined in terms of the system gain actually achieved for each scheduling interval. If there are \( k \) scheduling intervals during a processing day, then an average gain for day \( j \) can be computed as

\[
\text{ASG}_j = \frac{1}{k} \left( \sum_{i=1}^{k} SG_i \right)
\]
Definition 2.16. The *nominal system gain*, NSG, is the mean of the population of daily ASG.

As shown in Figure 4 the NSG and the SG value of 1.0 establish certain bounds on the system incurred losses. The system losses for a given MIX are computed by the system loss function based upon Figure 4.

Definition 2.17. The *total system loss function* for a time period of length $t$ is defined as follows:

$$
SL(t, SG) = \begin{cases} 
0, & \text{if } SG \geq NSG \\
LTB(SG-NSG)t, & \text{if } 1.0 < SG < NSG \\
(LTA(SG-1.0) + LTB(1.0-NSG))t, & \text{if } SG < 1.0.
\end{cases}
$$

where SG is the actual system gain recorded for the MIX, LTA is the linear loss rate prior to 1.0, and LTB is the rate between 1.0 and NSG.

Note that there is only one system loss function which must apply to the whole user community for the entire scheduling interval; this contrasts to the possibly different user loss functions for different program occurrences processed during the interval.

**Schedule Measures**

Perhaps the most straightforward and intuitive measure to use in the evaluation of a schedule, $S$, is a linear combination of the user and system loss of $S$. If $M$ is this measure and the schedule is

$$
S = (MIX_1, \ldots, MIX_n)
$$
then

**Definition 2.18.** The *measure of a schedule* $S, M(S)$, is defined to be

$$M(S) = \sum_{i=1}^{n} [UL(MIX_i)] + SL(t, SG)b$$

where $a$ and $b$ are weighting factors determined by management to place appropriate emphasis upon the values as established by policy; $t$ is the schedule processing time; $SG$ is the gain for the entire schedule; and $UL(MIX_i)$ is the user loss for $MIX_i$. (See 3.8, page 67.)

From Definition 2.18 the best schedule would be that which has the lowest loss per program processed. Thus,

**Definition 2.19.** If there are $k$ schedules to be considered for some SI, then the *best schedule* is that which satisfies

$$\min_{i \leq j \leq k} M(S_j)$$

Specification of a measure in this form allows for both user and system performance requirements to enter into the decision process. The favoring of one over the other can be accomplished through the appropriate assignment of value to the constants $a$ and $b$ and through the establishment of user and system loss functions.

There are many points of view as to what constitutes a valid schedule measure (52). It is not intended here to determine which is
most representative, but rather to develop the pre-scheduler in terms of the proposed measure and to demonstrate that there can be system improvements realized in terms of that measure through pre-scheduling. The measure does provide management with the total system/user loss for the programs processed and should provide some initial point from which other measures and procedures can be developed.

**Program Processing Times**

One of the fundamental elements of the pre-scheduler is the ability to utilize previous processing performance to provide estimates of processing times of program runs. As with the system gain, the assumption made is that the various processing situations can be based upon the run-alone case.

The basic elements of the time accounting procedure have already been introduced in the preliminary comments. The goal now is to provide specific expressions for prorating the processor service among MIX program occurrences, for determining elapsed times, for evaluating a system load rate, and for identifying the percentage of completion at any given point in time.

**Time Distributions (Round-Robin)**

The interrupt cycle was introduced earlier and was viewed as the time period between two successive interrupts. Also this period was defined to be composed of two sub-intervals, a processor period and a processor lockout period. Schematically, this is shown in Figure 6.
Since program processing characteristics were assumed to be independent of the queue position for the round-robin discipline, it is not necessarily the first member of \( Q_1 \) which governs the amount of service received by each program occurrence. To determine the degree of process-input/output masking which can take place and to evaluate a conservative estimate of the processor service received, it was decided to evaluate the processing parameters relative to what might occur during the period defined by the program occurrence in \( Q_1 \) with the longest cycle length. The fundamental idea is to order the members of \( Q_1 \) in sequence by descending cycle lengths. The first member of the sequence will be set to receive one cycle. Next, compute the number of cycles the second member of the sequence can receive during the lockout period of the first. Remove this processor time received from the lockout period and use the time remaining for a similar computation on the next

\*It is agreed that this is an over simplification of the time accounting for the round-robin discipline, but it does provide a conservative time estimate. Future study into the manner of time allocation is recommended.
program occurrence in the sequence. Continue this process until all members of $Q_1$ have been allocated a portion of the time. The fraction per unit of processing time received by occurrence in the $i$th position in the sequence, $PRR_i$, is found relative to the elapsed time of the cycle of the first member of the sequence. The expression is

$$PRR_i = \frac{C_i \times PR_i}{PR_i + LO_i} \quad (2.1)$$

where $C_i$ is the cycles received, $PR_i$ is the length of processor time per cycle, and $LO_i$ is the length of lockout per cycle. The expressions for the intermediate steps are the same as expressions 2.2 and 2.3 of the next section and will not be developed here.

**Time Distribution (Ordered Queue)**

The manner in which program occurrences in $Q_1$ receive service under the ordering queue disciplines is not unlike that of the round-robin. In particular, the amount of processor service received is not a function of the program occurrence with the longest cycle length, but a function of the time available as defined by the lockout period of the initial, first program occurrence in $Q_1$.

In order to distribute accurately the processor service, it would be necessary to define the exact processing service sequence among all members of $Q_1$. Due to the large number of possibilities which could occur, this would not be a practical undertaking. Therefore, an averaging procedure is proposed such that the running estimates will provide conservative processing averages.
The general idea of the prorating procedure is to examine, for all program occurrences except the first in the queue, the number of interrupt cycles which could occur during the lockout period of the first program occurrence. By considering only the effect of program occurrences which appear earlier in the queue, the masking of lockout periods which occur later with processor periods will not be considered. Thus, the running estimates will be longer than the actual run times, and the pre-scheduler will be able to provide a more conservative decision. As it will be shown below, the number of cycles can be used to prorate the processor activity in a manner similar to the procedure for expression 2.1.

If the contents of $Q_1$ can be defined by $MIX = (p_1, \ldots, p_i, \ldots, p_n)$, then under the queue control assumptions, $p_1$ will always receive one interrupt cycle. While receiving this cycle there is $LO_1$ processor time units available for the remaining members of $Q_1$. The number of cycles $p_2$ can receive in this period, $C_2$, is

$$C_2 = \frac{LO_1}{PR_2 + LO_2}$$

which uses $PR_2 \times C_2$ processor units leaving $LO_1 - PR_2 \times C_2$ for the other members of the queue. The next program occurrence $p_3$ receives $C_3$ cycles which are found by

$$C_3 = \frac{LO_1 - PR_2 \times C_2}{PR_3 + LO_3}$$
using $PR_3 \times C_3$ units and leaving $LO_1 - PR_2 \times C_2 - PR_3 \times C_3$ processor units. This process continues until all program occurrences have been allotted time. Any unused time remaining will be attributed to idle periods in which no program occurrence has completed its lockout period.

The procedure outlined above can be generalized as follows:

\[
C_i = \begin{cases} 
1 & \text{for } i=1 \\
\frac{R_i}{PR_i + LO_i} & \text{for } i=2,3,\ldots,n
\end{cases}
\]  

(2.2)

and

\[
R_i = \begin{cases} 
LO_i & \text{for } i=2 \\
R_{i-1} - C_{i-1} \times PR_{i-1}, & \text{for } i=3,4,\ldots,n
\end{cases}
\]  

(2.3)

The total elapsed time available is $PR_i + LO_i$ of which $C_i \times PR_i$ are used for program occurrence $p_i$, $i=1,2,\ldots,n$. Thus the fraction of each processing unit available for program occurrence $p_i$ is

\[
PRR_i = \frac{C_i \times PR_i}{PR_i + LO_i}
\]  

(2.4)

As in the case of the round-robin, there is to be some of the processor time which must be used to service the switching among the
members of Q. Attention is now turned toward the specification of a system load rate which is to be representative of this overhead factor in the case of the ordered disciplines and the overhead plus possible lockout conflicts in the case of the round-robin.

**System Load Rate**

The more interrupt cycles which must be handled in the course of a MIX processing, the more processor time required to service the transfer of control among the program occurrences. In addition, for the round-robin case, the increase in interrupt cycles increases the chances that no program occurrence will be ready for another processor period when its turn is reached. Thus there is proposed an effective processor rate which is to be based upon the number of cycles defined by a particular mix and which will be used to adjust the amount of time each program occurrence can apply toward the time required for each processor period.

Since the factor $PRR_i$ indicates the processor time $p_i$ is to receive in one unit of processor time and $PR_i$ processor units are required per cycle, the number of cycles for $p_i$ is

$$CY_i = \frac{PRR_i}{PR_i}$$  \hspace{1cm} (2.5)

If this is true for each of the members of the MIX = $(p_1,\ldots,p_n)$, then the total system load rate, $LR(MIX)$, is

$$LR(MIX) = \sum_{i=1}^{n} CY_i$$  \hspace{1cm} (2.6)
Through the course of processing histories it can be observed that for a given LR(MIX), there is a certain portion of the time spent not servicing the program occurrences. Denote this value by \( EPR(LR(MIX)) \). If this quantity is based upon a single unit of processor time, then the value is in the interval zero to one. This says that \( 1.0 - EPR(LR(MIX)) \) time units can be applied to the processor time required by each program occurrence being serviced. Thus, the adjusted processing period for \( p_i \) is

\[
ADJ_i = \frac{PR_i}{1.0 - EPR(LR(MIX))} \tag{2.7}
\]

It is now possible to evaluate the processing elapsed time required for the completion of a program occurrence.

Run Time Evaluation

The run-alone-elapsed-time of a program occurrence \( p_i \), \( RA(p_i) \), can be found to be the cycle length times the number of cycles per data unit multiplied by the number of data units to be processed. This can be expressed

\[
RA(p_i) = (PR_i + LO_i) \times CD_i \times DTA_i \tag{2.8}
\]

where \( CD_i \) is the interrupt cycles required per data unit and \( DTA_i \) is the number of data units to be processed. In this expression all time is chargeable to \( p_i \).
If \( p_i \) is not processed alone, then the elapsed time required for completion can be written

\[
PTR(p_i) = \frac{\text{ADJ}_i \times \text{CD}_i \times \text{DTA}_i}{\text{PRR}_i}
\]  

(2.9)

**Percentage Completion**

The percentage of completion of a program occurrence at some time \( t \) is just the quotient of the number of cycles completed at \( t \) by the total cycles required multiplied by 100. Since the cycles required can be converted to time required through the use of expressions 2.8 and 2.9, the percentage of completion of a program occurrence \( p_i \) with \( t \) time units accumulated in \( Q_1 \) can be written

\[
PC(p_i, LR(MIX), t) = \begin{cases} 
\left(\frac{t}{\text{FA}(p_i)}\right) \times 100, & \text{if MIX} = (p_i) \\
\left(\frac{t}{\text{PTR}(p_i)}\right) \times 100, & \text{if MIX} = (p_1,\ldots,p_n) 
\end{cases}
\]

(2.10)

for \( n > 1 \).

**Program Availability, Exclusion, and Classification**

Regardless of the program occurrences selected from \( Q_2 \) for processing in a given mix configuration, there might be a collection of program occurrences which will be excluded from \( Q_1 \) membership. This exclusion may be due to scheduling choice, to system conflicts, to availability, or to policy decision. The first two categories will now be identified with respect to their role in the pre-scheduling process.
Exclusion by Choice

Let A be a collection of program occurrences. By some means one of these occurrences has been selected to become a member of the mix to occupy Q₁. It may be possible to enter into the mix another program occurrence, but the decision procedure of the pre-scheduler prohibits the action. This is exclusion by choice and it is this collection which the pre-scheduler is attempting to identify. This collection of program occurrences will be denoted by EXC(MIX) where MIX identifies the contents of Q₁.

Definition 2.20. \( EXC(MIX) = \{ p | p \in Q₂ \text{ and is excluded from MIX by scheduling choice} \} \).

Resource Exclusion

Resource exclusion differs from exclusion by choice in that for a given mix there are not sufficient system facilities for some of the program occurrences. That is, the members of the class of resource excluded program occurrences are those with which the mix in question is not resource compatible.

Definition 2.21. \( EXR(MIX) = \{ p | p \in Q₂, \text{ MIX = (p₁,...,pₙ) and (p₁,...,pₙ,p) \notin RC^{n+1} } \} \).

Available Set

In Definition 2.9 it was defined what was meant by an "available program occurrence." It will simplify the specification of the pre-scheduler if the collection of available program occurrences at a given time can be identified with a notation of its own. Therefore, the set of available program occurrences is defined as follows:
Definition 2.22. $AV(t) = \{ p | p \in Q_2 \text{ and } p \text{ is available} \}$.

Eligible Set

The final definition in this series will link several together to identify at time $t$ and some mix, those program occurrences which are eligible to become members of the mix. It is these program occurrences which can be considered by the pre-scheduler. The eligible set of program occurrences at time $t$ for a mix is

Definition 2.23. $EL(t, \text{MIX}) = \{ p | p \in AV(t) \text{ and } p \notin EXR(\text{MIX}) \}$.

This concludes the identification and definition of the more fundamental concepts related to the pre-scheduling process. These will now be incorporated into a collection of computational elements essential to the pre-scheduler operation.
CHAPTER III

GENERAL COMPUTATIONS

Introduction

In each of the queue handling disciplines under study, the proposed pre-scheduling procedure involves one or more basic computation procedures. These include program completion time estimates, \( Q \), runout computations, and a worst-case loss estimate. To simplify the specific pre-scheduler development, these will be introduced here.

Program Occurrence Completion Time Estimates

The run time assumptions as presented in terms of the system load rate and prorate factor do not exactly specify just how the completion times of the program occurrences are to be estimated if there are changes in the composition of \( Q \). The purpose is to develop a general schema describing how this dynamic process can be monitored and meaningful completion times can be evaluated. The method of obtaining program occurrence running times will be described in terms of a time reference collection, a time reference function, and a next exit function.

Relative Elapsed Time

Before examining these functions, it is imperative that the notion of relative elapsed time be fully understood. For a given mix each program occurrence will have achieved a certain percentage of completion as defined by expression 2.10.
**Definition 3.1.** The *relative elapsed time* for a program occurrence in Q is that amount of time accumulated which will reflect the correct percentage of completion when compared to the estimated total run time.

This allows the percentage of completion, not the time accounting, to be the controlling factor.

**Time Elapsed Function**

In order to update the relative elapsed time, there must be some way of obtaining the relative elapsed time of a program occurrence given a percentage of completion. This can be defined in terms of the percentage of completion function developed in the previous chapter.

**Definition 3.2.** The *relative elapsed time function* for program occurrence p in a MIX with a percentage of completion f is that value of t for which \( \text{PC}(p, LR(MIX), t) = f \) is true. Notation: \( \text{TE}(p, LR(MIX), f) = t \).

This can be obtained directly for the appropriate portion of expression 2.10. If LR(MIX) is the value of a single program occurrence mix, then the run-alone portion of expression 2.10 is used; otherwise, the more general computation applies.

**Time Reference Collection**

The purpose of the *time reference collection* is to provide to the pre-scheduler the elapsed time of each program occurrence in Q and this is to be relative to the current estimated time required for completion. If the system load rate and prorate factors never change during the processing of a program occurrence, the member of the time reference collection will always be the same as the actual elapsed time. As system
factors change due to program occurrence additions and deletions, the relative elapsed times must also reflect a change. Therefore,

**Definition 3.3.** The *time reference collection* for a MIX which defines the contents of $Q_1$ is

$$TR(MIX) = \{t \in p \in MIX \text{ and } t_p \text{ is the relative elapsed time of } p\}$$

The members of $TR(MIX)$ are computed based upon the program occurrences which define the composition of the MIX. This computation is defined by a time reference function.

**Time Reference Function**

The *time reference function* allows for the adjustment of the relative elapsed times due to the addition or deletion of program occurrences in $Q_1$. This is to account for changes, if any, in the system load and prorate factors. For each program occurrence in the MIX, the function must take into consideration current $TR(MIX_1)$ values, the old $MIX_1$ system load and prorate factors, the new $MIX_2$ system load and prorate factors, and the actual elapsed time since the last application of the function. If $LR(MIX_1)$ is the current load rate and $LR(MIX_2)$ is the new, then

**Lemma 1.** For each program occurrence $p \in MIX_2$

$$TRF(p, t, LR(MIX_1), LR(MIX_2)) = TE[p, LR(MIX_2), PC(p, LR(MIX_1), t + t)]$$

*The prorate values are not used directly but are required in the computation of the appropriate load rates.*
where \( t \) is the elapsed time since the last TRF evaluation and 
\( t_P \epsilon \text{TR} (\text{MIX}) \).

Proof. Referring to Figure 7, \( PC(p, LR(\text{MIX}), t_P + t) \) provides the new value of the percentage of completion for the old \( \text{MIX}_1 \) with an increase of \( t \) time units over the previous evaluation. The function \( TE \) of Definition 3.2 is used with the new percentage of completion value and the parameters for the new \( \text{MIX}_2 \) to find the new relative elapsed time, \( \tilde{t}_P \).

The diagram indicates an increase in the running times, but a similar discussion would apply for a decrease. Now based upon the time reference function and the time reference set, a next exit function can be defined.

**Next Exit Function**

The next program occurrence to exit \( \text{Q}_1 \) due to an end of job can be computed by an analysis of the difference between the estimated run times and the current relative elapsed time. For each program occurrence \( p \) in the \( \text{MIX} \) currently defining \( \text{Q}_1 \), the relative completion time is given by \( TE(p, LR(\text{MIX}), 100.0) \). Since the amount of processing time received is given by \( t_P \epsilon \text{TR(MIX)} \), the time remaining to completion is

\[
TC(p, LR(\text{MIX})) = TE(p, LR(\text{MIX}), 100.0) - t_P
\]

(3.1)

*For a given \( \text{MIX} \) each program occurrence defines a run time function based upon expression 2.10. These are shown as linear but could be any monotonic, nondecreasing function.*
Figure 7. Time Reference Collection Re-evaluation
Hence, a next exit function should provide the smallest elapsed time remaining.

**Lemma 2.** The *next exit function* for a MIX defining the contents of \( Q \) is

\[
\text{NE}[\text{TR}(\text{MIX}), \text{LR}(\text{MIX})] = \min_{p \in \text{MIX}} \left\{ \text{TC}[p, \text{LR}(\text{MIX})] \right\}
\]

**Proof.** From Definition 3.2, \( \text{TE}(p, \text{LR}(\text{MIX}), 100.0) \) is the amount of elapsed time required for a program occurrence to complete its processing in the MIX. For each \( p \in \text{MIX} \) there is a \( t \in \text{TR}(\text{MIX}) \) denoting the elapsed time accumulated toward completion. The program occurrence(s) for which 3.1 achieves a minimum are those with the shortest expected elapsed time remaining to completion.

**Next Exit Set**

The next exit function of Lemma 2 will define a non-empty set consisting of the program occurrences which will exit \( Q \) at the time defined by the function's application. Thus,

**Definition 3.4.** For a given application of the next exit function to the MIX, the *next exit set* is

\[
\text{NES}(\text{MIX}) = \{ p | p \in \text{MIX} \text{ and } \text{TC}(p, \text{LR}(\text{MIX})) = \text{NE}(\text{TR}(\text{MIX}), \text{LR}(\text{MIX})) \}
\]

**New Base Mix**

If the program occurrences which are in \( \text{NES}(\text{MIX}) \) are removed from \( Q \), then the result is a new base \( Q \) configuration. It is this base
configuration which will be used to build the next mix which is to
define the contents of Q₁. For notational convenience the new base
mix will be denoted by

\[ BMIX₁ = MIX₁ - NES(MIX₁), \] (3.2)

"-" denoting the deletion of all members of \( NES(MIX₁) \) from the vector
\( MIX₁ \) and the preservation of the order of the remaining members in
\( BMIX₁ \).

**Mix Elapsed Time**

In order to compute the total run time of a program occurrence,
the elapsed time that it spends in each mix configuration is required.
With the assumption that the system is operative throughout the entire
time period under investigation the definition can be written

**Definition 3.5.** If \( t_{i+1} \) denotes the actual time that \( MIX_{i+1} \) is
derived from \( MIX₁ \), then the *relative elapsed time* of \( MIX₁ \), \( RET₁ \), is

\[ RET₁ = t_{i+1} - t_i \quad \text{for } i=0,1,2,... \]

and \( t₀ = 0 \).

Since the content of \( Q₁ \) is changed only with program occurrence
exits, the value of \( RET₁ \) is found to be

\[ RET₁ = NE(IR(MIX₁),LR(MIX₁)). \] (3.3)
Program Occurrence Run Times

The actual run time for a program occurrence can now be evaluated. Recall that by run time is meant the total time that the program occurrence is in $Q_1$. If it were possible to identify which mix configurations the occurrence was in, then it would be possible to evaluate the running time. The program occurrence once entered into $Q_1$ will remain there until its completion. Thus, there will be an initial $MIX_k$ and a final $MIX_n$. The run time can now be defined in terms of the previous definition.

**Definition 3.6.** The run time for program occurrence $p$ is

$$TRN(p,u,c) = \sum_{i=k}^{n} RET_i$$

where $p$ enters $Q_1$ in $MIX_k$ and exits $Q_1$ in $MIX_n$.

Schedule Run Time

Finally, the schedule run time can be found in terms of the relative elapsed times of the mix configurations which define the schedule.

**Lemma 3.** The running time for a schedule $S = (MIX_1, \ldots, MIX_{n-1}, MIX_n)$ is

$$RT(S) = \sum_{i=1}^{n-1} RET_i + RO(MIX_n)$$

where
Proof. From Definition 3.5 and expression 3.3 the factor \( \sum_{i=1}^{n-1} \beta_{i,1} \) is the elapsed time required for the processing of the schedule elements \( \text{MIX}_1 \) through \( \text{MIX}_{n-1} \). The final schedule element \( \text{MIX}_n \) may have several program occurrences as members and their run times must be evaluated. Since there are no new program occurrences to be added, the recursive structure of the function RO will evaluate the run time of the successive program exits and will remove these from \( Q_1 \) prior to the next stage evaluation of the function. The requirement of the time reference function is implied before the next application of RO, thus adjusting the elapsed times accumulated to reflect the change in the contents of \( Q_1 \). When \( Q_1 \) is empty, the RO function evaluates to 0 and the process terminates.

It is the value \( RT(S) \) which is applied in the evaluation of the system gain (Definition 2.15) and system loss (Definition 2.17).

**Schedule User Loss Evaluation**

Related to the schedule run time is the *schedule user loss*. At each change from \( \text{MIX}_i \) to \( \text{MIX}_{i+1} \) program occurrences will be completed and will exit the system. According to Definition 2.14 these times of mix configuration change can be used to compute the user losses incurred. Therefore, it is now possible to link Definitions 2.14, 3.4, 3.5, 3.6,
and Lemma 3 in such a way so as to define the user loss evaluation procedure.

Suppose the schedule is

$$S = (\text{MIX}_1, \ldots, \text{MIX}_{n-1}, \text{MIX}_n)$$ \hspace{1cm} (3.4)

and \text{MIX}_1 is begun at the time denoted by \text{STRT}. The collection of program occurrences to complete their processing in \text{MIX}_i, \text{i} \neq n is denoted by \text{NES(\text{MIX}_i)}. The total elapsed schedule run time through \text{MIX}_i is given by

$$\text{RT}(\text{MIX}_1, \ldots, \text{MIX}_i) = \sum_{j=1}^{i} \text{RET}_j \text{ for } i \neq n$$ \hspace{1cm} (3.5)

The clock time at the end of \text{MIX}_i, \text{CL(\text{MIX}_i)}, is given by

$$\text{CL(\text{MIX}_i)} = \text{STRT} + \text{RT}(\text{MIX}_1, \ldots, \text{MIX}_i)$$ \hspace{1cm} (3.6)

From Definition 2.14, the total system time for each program occurrence in \text{NES(\text{MIX}_i)} must be used in the user loss evaluation. From Definition 2.6e this quantity is the completion time of the program occurrence less the arrival time. Thus, for all program occurrences \text{p} in \text{NES(\text{MIX}_i)}

$$\text{TS(p,u,c)} = \text{CL(\text{MIX}_i)} - \text{TA(p,u,c)}$$ \hspace{1cm} (3.7)

*The recursive portion of the function is not required since the last member of the schedule is excluded and runout times of \text{MIX}_i are not required.*
The user loss for MIX\textsubscript{i} for i≠n can be expressed by

\[ UL(MIX\textsubscript{i}) = \sum_{p\in NES(MIX\textsubscript{i})} ULF(p,u,c,TS(p,u,c)) \] \hfill (3.8)

The user loss for the final schedule component MIX\textsubscript{n}, requires the computation of the runout costs of the program occurrences in MIX\textsubscript{n}. As with the runout time, this value can be expressed recursively.

\[ ROC(MIX\textsubscript{n}) = \begin{cases} \sum_{p\in NES(MIX\textsubscript{n})} ULF(p,u,c,CL(MIX\textsubscript{n-1})) + & \\
NE(TR(MIX\textsubscript{n}),LR(MIX\textsubscript{n})) - & TA(p,u,c) + ROC(MIX\textsubscript{n-1}-NES(MIX\textsubscript{n})), \text{ if MIX}\textsubscript{n} \text{ is not empty} \\
0 & \text{, if MIX}\textsubscript{n} \text{ is empty} \end{cases} \] \hfill (3.9)

Expression 3.9 bases the total system time of a program occurrence upon the runout time accumulated and the clock time of the previous mix configuration. The clock time is updated by the application of the NE function, and it is applied at the next stage of the recursion. When all program occurrences have completed, expression 3.9 evaluates to zero and the process is terminated.

The total user loss for the schedule given by expression 3.4 is

\[ TUL(S) = \left\{ \sum_{i=1}^{n-1} UL(MIX\textsubscript{i}) + ROC(MIX\textsubscript{n}) \right\} \times a \] \hfill (3.10)
where \( a \) is the user loss weighting factor of Definition 2.18. Also from the same definition, the schedule measure is

\[
M(S) = TUL(S) + SL(RT(S),SG) \times b
\]  
(3.11)

**User Loss Estimates**

The definition of the pre-scheduler requires an estimate of the user losses and system losses incurred through the processing of a collection of program occurrences. This estimation process should not require examination of a large number of alternatives or retention of a quantity of intermediate results. Finally, the number of calculations should be minimized. These restrictions, along with the desire of allowing non-linear loss functions with a deadline capability, make the selection of a procedure difficult.

It is suggested that the evaluation should identify those program occurrences which, when processed in serial fashion, can be delayed with the least cost. Once this identification has been completed, the sequence of occurrences with the least losses can be identified. To derive this optimal sequence the procedure suggested by Lawler (42) could be used. Unfortunately, this method of scheduling is expensive; e.g., to schedule \( n \) items requires on the order of \( n^2 \) calculations with a storage of \( 2^n \) intermediate results. This is far too costly a procedure to be used many times in an operational environment. Therefore, the following alternative is proposed.
Using the serial, run-alone processing times select that program occurrence with the least user loss when processed after all others. Delete the selected occurrence from those to be scheduled and repeat the procedure until all have been placed in the processing sequence. This selection process will not be optimal, but it is economical in that no intermediate results other than identification of the program occurrence with the least cost need be retained. Also the number of calculations required is at most on the order of $n^2$ for the sequencing of $n$ items. Once the losses for the sequence as specified by this procedure have been computed, they will be used as a benchmark for the future scheduling decisions made by the pre-scheduler. This initial sequence identification will be referred to as the worst case sequence.

**Worst-Case Procedure**

The basic nature of the scheduling problem being studied requires some knowledge of future events in order to make the proper scheduling decisions. Due to the many possible choices of facility sharing which could occur, it is not practical to consider all possibilities at each decision point. What is beneficial is to examine what has been referred to as the worst-case and to select the situation which would avoid most of the adverse scheduling situations.

By specifying the worst-case procedure as a serial processing situation, the effects of multiprogramming and facility sharings do not have to be examined. The program occurrence processing behavior becomes independent of the other occurrences to be scheduled, and the losses involved become only a function of those program occurrences which
precede it. Therefore, the selection of the least-cost-program-
occurrence-last can be accomplished by a straightforward technique.

Suppose that \( Q_2 = \{p_1, \ldots, p_n\} \) are the program occurrences to be
sequenced. The program occurrence with the least user loss when run
last must satisfy at least

\[
\min_{p_i \in Q_2} \left[ \text{ULF}(p_i, u_i, c_i, \sum_{i \neq j} \text{TE}(p_j, LR((p_j)), 100.0)) + \right.
\]

\[
\left. \text{TE}(p_i, LR((p_i)), 100.0) \right] \]

Once \( p_i \) has been selected \( Q_2 \) is reduced by \( p_i \), written \( Q_2 - p_i \), and the
procedure is repeated until all program occurrences have been placed in
sequence. The recursive expression for the total user loss of the
selected sequence is given by

\[
\text{UL}(Q_2) = \min_{p_i \in Q_2} \left[ \text{ULF}(p_i, u_i, c_i, \sum_{i \neq j} \text{TE}(p_j, LR((p_j)), 100.0)) + \right.
\]

\[
\left. \text{TE}(p_i, LR((p_i)), 100.0) + \text{STRT} - \text{TA}(p_i, u_i, c_i) \right] + \text{UL}(Q_2 - p_i) \]

where \( \text{STRT} \) is the starting time of the first member of the sequence.
The single element argument for the load rate function is to imply a
single element mix configuration and hence a serial processing situ­
ation.

Since the actual running time and the serial running time are to
be the same, from Definition 2.15 the system gain will be one. Thus
from Definition 2.17 the system loss for the worst-case sequence will be

\[ SL(Q_2) = SL(t,1.0) = LTB(1.0-NSG) \times t \]  \hspace{1cm} (3.14)

where

\[ t = \sum_{i=1}^{n} \frac{TE\left[p_{LR}(p_i)\right],100.0}{\text{100.0}} \]

Therefore, the worst-case loss is found from 3.13, 3.14, and Definition 2.18 as follows:

\[ WCM(Q_2) = a \times UL(Q_2) + b \times SL(Q_2) \]  \hspace{1cm} (3.15)

with the constants a and b assigned according to the current operating policy.

If there are currently program occurrences in \( Q_1 \) being processed, then they must be processed to completion prior to the application of the worst-case procedure. The only modification is that expression 3.13 must have a starting time value, \( STRT \), set to include the runout time from \( Q_1 \) of those unfinished program occurrences.

If ties occur in the selection of which member of \( Q_2 \) is to be placed last, an observation should prove of assistance. The running time is used in the evaluation of the total system time for the cost computations. If the tie occurs and that program occurrence with the longest running time is selected to break the tie, then the sum of the
running times of the remaining, unscheduled program occurrences will be smaller than if the reverse decision were made. Since the user loss functions were assumed to be monotonic and a non-decreasing function of the total system time, this reduction in time for the remaining occurrences could lead to a reduction in system time, and hence a potential reduction in user losses. Therefore, if there is a tie among several program occurrences in the sequence selection defined by the worst-case procedure, the program occurrence among those which have tied with the longest running time will be selected to assume the position last in the sequence. *

Finally, it is possible that some of the program occurrences in \( Q_2 \) have not yet arrived for processing and there could be idle periods in the worst-case sequence. ** If there are idle periods, then one attempt will be made to fill the period with a program occurrence later in the worst-case sequence. Those occurrences later in the sequence which have arrived and are available during an idle period will be placed in the period and the losses evaluated. If there is a reduction in the total losses, the new sequence will be selected as the worst-case sequence and the next idle period, if any, is examined in a similar manner. If there is not a cost reduction, then the sequence is returned to the original and the next idle period is examined. This continues

---

* This technique will not necessarily guarantee loss reductions in all cases, but it proved helpful under the linear loss assumptions.

** An example of a situation with idle periods induced by delaying the processing of an available program occurrence is presented in Appendix D.
until all potential idle periods have been tested. The final sequence is then identified as the worst-case sequence and the user/system losses associated with the sequence is used as the benchmark for all other scheduling decisions.

**Resource Compatibility Computations**

Definition 2.11 introduced the foundation upon which the resource compatibility of program occurrences can be identified. This definition indicated those program occurrences which are pairwise resource compatible (i.e., members of the relation $RC^2$). This can now be extended to a computation procedure which will determine if there are adequate resources to permit the processing of $n$ program occurrences.

**Lemma 4.** Program occurrences $p_1, p_2, \ldots, p_n$ are resource compatible if and only if

\[ a. \sum_{i=1}^{n} W_i \leq MEM \]

\[ b. \sum_{i=1}^{n} a_{ij} \leq d_j \]

where $W_p$ is the working set requirements of $p_i$, $MEM$ is the total memory available, $a_{ij}$ is the requirements of $p_i$ for device $j$, and $d_j$ is the number of device $j$ available for use by the program occurrences.

**Proof.** All program occurrences must be pairwise compatible. If parts $a$ and $b$ are true for $n > 2$, then certainly they are true for $n = 2$ and Definition 2.11 is satisfied. Also, if the conditions of the lemma are met, no program occurrence will lack the required system facilities.
for its processing. Hence, every program occurrence will be a member of $RC^n$, and resource compatibility is established. If one of the lemma conditions is not satisfied, then for some program occurrence, there will not be adequate memory or a peripheral device will be in use, and the collection of $n$ program occurrences cannot be resource compatible.

With this last specification, the discussion of the general computations is completed. These elements can now be linked together for the specification of the pre-scheduling procedure.
CHAPTER IV

PRE-SCHEDULER SPECIFICATION

The Scheduling Solution

Classically the solution to a scheduling problem consists of two fundamental elements: (1) the selecting of the tasks to be serviced and (2) the ordering of the selected tasks. The solution to the information system pre-scheduling problem is no different, but the character and behavior of multiprogrammed systems and their users make it impossible to carry the similarity much farther.

Specifically, there is already a program scheduling component (sometimes referred to as a resource allocation component) identified with the operating system which controls many of the computer system's actions. It is this component which dictates much of the system's behavior—especially the manner in which programs enter and exit the processing queue, $Q_1$. As the initial assumptions required, once a program occurrence becomes a MIX element and enters $Q_1$, the pre-scheduler can only monitor its progress and can only indirectly alter its queue position and running characteristics through the addition (or not adding) new program occurrences.

The queue disciplines of round-robin and first come-first serve will place new program occurrences, which are to be added to $Q_1$, at the end of $Q_1$. The pre-scheduler will be constructed so that it will have to examine the addition of at most one program occurrence at each
decision point. Therefore, no explicit ordering examination is required. Once in $Q_1$, no reordering by the pre-scheduler is permitted, and the problem of pre-scheduling is, for these two cases, that of schedule membership determination.

The other queue disciplines which sort the members of $Q_1$ at every mix configuration change are no different, with respect to the pre-scheduling problem, than the first two disciplines. The membership determination is the same, but the position of the new members of $Q_1$ are determined, not by the pre-scheduler, but by the sorting discipline which governs the operation of the queue. For each new addition, the pre-scheduler will determine what position the new arrival will occupy and then make the decision whether to add or not.

This brief introduction should indicate that the pre-scheduler is to consist of a basic program selection procedure with the capability of $Q_1$ sorting for those disciplines which require it. In addition, there is a selection discipline which describes the piecemeal process of obtaining program occurrences from the queues $Q_3$ and $Q_4$. Since these are outside the boundaries of the actual pre-scheduler, the manner of their handling will not be considered a part of the pre-scheduler. It will be noted, however, that the $Q_3/Q_4$-selection discipline will have a definite effect upon the pre-scheduling decisions.

The pre-scheduling procedure and the manner in which it selects program occurrences from its queue, $Q_2$, for processing can now be presented.
Pre-Scheduling Procedure

The program selection procedure is basically the same whether \( Q_1 \) is or is not empty. In fact, the empty condition is just a special case of the more general second condition. Therefore, only the more general procedure will be specified. Also, since the selection procedure is similar for all queue ordering disciplines, the discussion will concern the round-robin and first come-first serve which place new \( Q_1 \) program occurrences at the end of \( Q_1 \). The other disciplines will be considered by noting the point at which the sorting procedures are to be applied.

Base Mix and Eligible Program Set

At a given point in time, \( t \), a program occurrence is to be added to \( Q_1 \) from \( Q_2 \). This point must coincide with program occurrence exits from \( Q_1 \) or \( Q_1 \) must be empty (i.e., there is an idle period). In either case the contents can be referred to as the base mix and will either contain no members or those defined by expressions 3.2.

Once the base mix has been determined, the contents of \( Q_2 \) are sequenced according to the worst-case procedure with a starting time of \( t \). If \( Q_1 \) is described by the empty base mix, then the time \( t \) is adequate; otherwise, the start time of the procedure must be increased by the runout time of the base mix which is evaluated by RO of Lemma 3. This sequencing will provide an approximation of a limiting, worst-case-to-be-permitted processing order of those unscheduled program occurrences in \( Q_2 \). Now that \( Q_2 \) has been ordered, then members of the eligible set can be identified in accordance with Definition 2.23. The special cases of how to use the base mix and eligible set are now
described prior to the development of the general mix construction procedure.

Special Cases

If either the base mix and/or the eligible set are empty, then some special actions must be taken to either establish the next base mix or to establish the next eligible set. These conditions are now presented.

If at time t the base mix for the schedule component MIX₁ is empty, then some program occurrence must be selected as an initial Q₁ member. Through this selection a base mix configuration is established and the scheduling process can continue. If the eligible set is not empty, then the base mix, BMIX₁, is set to contain the first member of the eligible set. This permits entry into Q₁ that program occurrence whose delay would hurt the users the most. There is no consideration of how well this program can or will mix with the other members of Q₂. The contents of Q₂ are then adjusted to reflect the Q₁ addition.

If BMIX₁ is non-empty but the eligible set at time t, EL(t,BMIX₁), has been exhausted, no new program occurrences can be added to Q₁. If this situation occurs, the base mix automatically becomes MIX₁ and at least one program occurrence must exit before any member of Q₂ can be considered to enter Q₁. If it should happen that both Q₁ and Q₂ are

---

*It should be noted that the eligible set for an empty Q₁ consists of all those members of Q₂ for which the arrival time to the system is less than or equal to t. A resource compatibility check is not required in this case.*
empty, then the pre-scheduling process terminates for all program occurrences to be scheduled have been exhausted.

Finally, if $BMIX_i$ and $EL(t, BMIX_i)$ are empty and $Q_2$ is not, then an idle period has occurred. One of the previous two conditions can be created by adjusting the time, $t$, to the arrival time of the first program occurrence in $Q_2$. This will insure at least one member of the eligible set and hence at least one member of a base mix.

Once there is established a non-empty base mix and a non-empty eligible set, it is now necessary to examine the construction of potential mix configurations and to select one as the next description of the contents of $Q_1$.

MIX Construction

Suppose that $MIX_i$ has just been completed at time $t$, and the base mix for the next mix configuration has been established. The goal is now to select members of $Q_2$ to add to the base mix to form new trial mix configurations and then to select one of the trial alternatives as the next mix for $Q_1$. For the purpose of conversation assume that the next mix configuration will be $MIX_{i+1}$.

The general procedure can be outlined as follows:

1. Add from the eligible set each program occurrence to the largest trial mix, evaluate the schedule measure, and then remove the added program occurrence.

---

\*It is possible that one or more of the special case situations could have occurred after $MIX_i$. If so, then $t$ would be adjusted to the completion of the mix configuration prior to $MIX_{i+1}$.\*
2. Select as the trial mix, for each possible $Q_1$ size, that mix which had the lowest measure value.

3. Identify the new eligible set based upon the new largest trial mix.

4. Continue the above process until the eligible set is empty or the maximum size of $Q_1$ has been reached.

5. Of those trial mix configurations constructed, select that with the lowest measure as the contents of $Q_1$, thus defining $MIX_{i+1}$ in the schedule.

This procedure can be developed in terms of the definitions and computational procedures in the earlier discussions.

Let $TMIX_j$ denote the trial mix of size $j$. Since the base mix is always to be one of the scheduling alternatives, it will become the first and smallest trial mix. If $BMIX_{i+1} = (p_1, \ldots, p_K)$, then

$$TMIX_k = BMIX_{i+1} = (p_1, \ldots, p_K) \quad (4.1)$$

It is now upon $TMIX_k$ that attempts to construct a mix configuration of size $k+1$ are made. Since $EL(t, TMIX_k)$ may have several members, there will be a cost evaluation for each of the eligible program occurrences. Let the cost of the $r$th alternative for a trial mix size of $k+1$ be denoted by $ALT(k+1, r)$. If there are $z$ different evaluations, then the trial mix of size $k+1$ is the program occurrence collection for which

$$\min_{1 \leq r \leq z} ALT(k+1, r) \quad (4.2)$$
occurs. Since the new trial will be one larger than previous ones, we can write \( \text{TMI}_{k+1} = \text{TMI}_k \oplus p_j \) for some \( p_j \) in \( EL(t, \text{TMI}_k) \). The contents of the eligible set as well as \( Q_2 \) can then be reduced by this program occurrence and the process continued until the eligible set becomes empty or the largest trial mix fills \( Q_1 \).

If there are \( y \) trial mix situations established, then the one to become \( \text{MIX}_{i+1} \) is that which satisfies

\[
\min \left( \min_{0 \leq j \leq y} \text{ALT}(k+j, r) \right)
\]  

(4.3)

What remains is the specification of how the trial costs are to be evaluated. This evaluation must provide a schedule measure in the form of Definition 2.18 and utilize the expressions which have been developed to monitor the system performance.

**Schedule Evaluation**

In keeping with the conservative worst-case concept, the schedule evaluation of the processing sequence with one of the trial mix configurations as the last pre-scheduled member will be viewed as follows. If the trial mix under investigation is in position \( i+1 \) in the scheduling sequence, then the user loss will be that already incurred in \( \text{MIX}_i \) through \( \text{MIX}_{i+1} \), the runout costs of \( \text{MIX}_{i+1} \), and the worst-case sequence user loss for the unscheduled members of \( Q_2 \). The system losses can be found from the running times of the total schedule and the run alone.

\[\text{®} \] denotes the addition of \( p_j \) to the trial mix and the place in the sequence is determined by the queue discipline in use.
times of the individual program occurrences involved.

Let

\[ \text{BASCST} = \sum_{j=1}^{i} \text{UL}(\text{MIX}_j) \]  \hspace{1cm} (4.4)

represent the user losses already accumulated in the first \( i \) members of the schedule. If \( \text{T MIX}_k \) for some \( k \) is to be evaluated in the \( i+1 \) position of the schedule, then the user loss is the runout cost as given by expression 3.9. The starting time for the running of the remaining members in the worst-case sequence (those program occurrences remaining in \( Q_2 \)) is the clock time as given by expression 3.6. The worst-case cost can then be evaluated by 3.13. If \( Q_2 \) represents the contents of \( Q_2 \), then the total user losses for the entire schedule sequence, \( S \), is

\[ \text{TUL}(S) = \text{BASCST} + \text{ROC}(\text{T MIX}_k) + \text{UL}(Q_2) \]  \hspace{1cm} (4.5)

To evaluate the system losses, the run times must be evaluated for the entire proposed schedule \( \mathcal{S} \). If \( \text{TOTIM} \) is the time elapsed through \( \text{MIX}_1 \), then from expression 3.5,

\[ \text{TOTIM} = \text{RT}((\text{MIX}_1, \ldots, \text{MIX}_i)) = \sum_{j=1}^{i} \text{RET}_j \]  \hspace{1cm} (4.5)

The total time of the proposed schedule is now found from Lemma 3, Definition 3.2, and expression 4.6.
\[ TOTIM = TOTIM + RO(TMIX_K) + \sum_{p \in \mathcal{P}_2} TE(p, LR((p)), 100.0) \quad (4.7) \]

Let the run-alone time be denoted by RUNAL. Since there is to be no multiprogramming of the program occurrences in the worst-case sequence, this will be the sum of the individual processing times.

\[ RUNAL = \sum_{p \in \mathcal{P}} TE(p, LR((p)), 100.0) \quad (4.8) \]

From definition 2.15, the system gain is defined as

\[ SG(\bar{S}) = \frac{RUNAL}{TOTIM} \quad (4.9) \]

From Definition 2.17 the system loss can be found. Thus in keeping with the specification of the system measure, the schedule loss with TMIX_K in the i+1 position of the schedule with a worst-case processing of the unscheduled program occurrences is

\[ M(\bar{S}) = a \times TUL(\bar{S}) + b \times SL(TOTIM, SG(\bar{S})) \quad (4.10) \]

Thus, if this evaluation is for the jth alternative of a trial mix of size k, expression 4.10 can be used to satisfy expression 4.3. That is,

\[ ALT(k,j) = M(\bar{S}) \quad (4.11) \]

This concludes the specification of the proposed pre-scheduling
procedure. The discussion turns now to how program occurrences are moved from the external queues, $Q_3$ and $Q_4$, into the pre-scheduler queue, $Q_2$.

**$Q_3/Q_4$-Selection Disciplines**

The pre-scheduler accepts its input from the external queues $Q_3$ and $Q_4$. $Q_3$ has been designated to contain a sequence of known information request arrivals while $Q_4$ consists of information requests which arrive with no previous warning. At various points in the pre-scheduling process these queues will be examined and program occurrences will be selected to become members of $Q_2$. The purpose of this section is to explain this selection procedure.

**Normal $Q_3$ Handling**

The contents of $Q_3$ will be those information requests which are to be given preferential treatment with respect to $Q_2$ membership. This preference is intended to favor those processing demands which are regular occurrences in the processing stream or are important enough to be planned in advance. Therefore, these program occurrences should enter the scheduling process as soon as possible to insure their consideration for processing in accordance with the user loss objectives.

When $Q_2$ has fewer than $CTRLQ2^*$ program occurrences remaining, then an attempt will be made to select program occurrences from $Q_3$. If $LIMQ2$ is the maximum number of program occurrences that can be a member of $Q_2$, then with $Q2SZ$ program occurrences in $Q_2$ at most $LIMQ2 - Q2SZ$ can

---

*This is the lower limit of the size of $Q_2$. \*
be added from $Q_3$. Preliminary results have indicated that this can
ignore $Q_4$ arrivals and cause the overlooking of a high loss case.
Hence, at most $\text{LIMQ}2 - (Q2SZ+1)$ program occurrences will be initially
selected from $Q_3$. This will be accomplished as follows:

1. Identify all program occurrences which are available and
add to $Q_2$ not more than $\text{LIMQ}2 - (Q2SZ+1)$. Add in the sequence as speci-
fied by the arrival time.

2. If there is still room in $Q_2$ (i.e., less than $\text{LIMQ}2-1$ mem-
bers), compute the serial run time of those program occurrences just
added to $Q_2$ and reflect the potential multiprogrammed gain by dividing
this run time by the nominal system gain ($\text{NSG}$).

3. Add to $Q_2$, without exceeding the established limit, as many
program occurrences as possible which arrive during this adjusted run
time interval.

$Q_4$ Selection

After adding as many program occurrences from $Q_3$ as possible in
the manner as specified above, all program occurrences which have
arrived in $Q_4$ are considered for addition to $Q_2$ on a first come-first
serve basis. The remaining positions of $Q_2$ will be filled until either
the limit of $Q_2$ is reached or $Q_4$ is empty. From the manner in which $Q_3$
was initially handled, there will always be room for at least one member
of $Q_4$.

If this process still does not fill $Q_2$ to its limit, then atten-
tion is returned to the program occurrences remaining in $Q_3$. 
If \( Q_2 \) still has not been filled to its limit, then program occurrences are selected on a first come-first serve base from \( Q_3 \) until \( Q_2 \) is filled or \( Q_3 \) is empty. This is done to insure that as many processing requests as possible are in the pre-scheduling decision process as soon as possible.

**Idle Periods**

Many times in the course of pre-scheduling idle periods occur. These can be caused either by processing delays induced by the pre-scheduler or due to no program occurrences in an available status. If either situation occurs and \( Q2SZ \) falls below \( CTRLQ2 \), then the previously described \( Q_2 \) updating procedures are carried out. On the other hand, idle periods can occur before the lower limit on \( Q_2 \) is reached. There is no need to examine \( Q_3 \) for it has already been given preferential treatment. To insure that information requests which are in \( Q_4 \) will not be overlooked, an attempt is made to fill \( Q_2 \) on a first come-first serve basis from those members which have arrived in \( Q_4 \).

The \( Q_2 \) addition procedures as described above were all based upon a first come-first serve criterion as the program occurrences appeared in their respective queues. Once the \( Q_2 \) fill procedure has been completed, the pre-scheduler resumes control of the process and pre-scheduling continues.

This concludes the specification of the proposed pre-scheduling procedure and the associated queue handling routines. The remainder of the paper has been devoted to the discussion of an experiment in which
the fundamentals of the procedure were implemented through a computer program.
CHAPTER V

TESTING AND EVALUATION

Preliminary Discussion

In order to verify that the proposed pre-scheduling procedure does have the potential of improving the computer system performance as specified by the total loss measure, the pre-scheduler with the proposed time accounting scheme was programmed in Burroughs Extended Algol. It was assumed that the time accounting procedures were actually representative of the processing behavior of program occurrences. With this pre-scheduler and processing assumption, several processing situations were analyzed to determine what factors could affect and influence the outcomes of the scheduling decisions. The purpose of this chapter is to describe first the processing situations analyzed and then to examine the pre-scheduling results.

General Experiment Description

The pre-scheduler analysis was divided into three phases. The first was to establish a collection of processing environments describing the contents of queues $Q_3$ and $Q_4$ and to determine the nature of the piecemeal process through the variation of the limits placed upon the pre-scheduler queue, $Q_2$.

Next, an investigation was made into what were the effects that changes in management-established factors had in the pre-scheduling results. These included monitoring of results for changes in user loss
functions, deadlines, user/system weighting constants, system recon-
figurations, system loss rates, and nominal system gain. The major
purpose of this investigation was to show that the proposed pre-
scheduler could be used as a management tool in that it would be sensi-
tive to changes in a management decision.

Finally, two large processing situations were processed to
demonstrate that the pre-scheduling philosophy could be applied over
higher volumes of processing requests and longer scheduling intervals.
In these cases a faster time accounting procedure was employed; the
processing time per data unit for each program was assumed to be a
linear function of the total interrupt rate. This was done to speed
up the processing but did not alter the basic pre-scheduling procedure,
since only time accounting was changed. The scheduling results were,
however, only as accurate as the run time estimates.

The purpose of the remainder of this chapter is to describe each
of the three testing situations and the basic methodology of the experi-
ments. Once this has been accomplished the results will be summarized
with indications as to what could be concluded or inferred. It is re-
emphasized that the results go no farther than data analyzed, but will
demonstrate the applicability of the pre-scheduler as a management
evaluation and analysis tool.

*The specifics concerning input data, detailed output, and vari-
ous classical measures of system performance have been included in
various appendices. The appropriate ones will be identified throughout
the discussion.
Processing Environments

The analysis of the piecemeal process and management factors was centered on a common data base while the large processing situation was identified with a different one. The next few sections will be devoted to the specification of these data bases. Since it is in the contents of $Q_3$ and $Q_4$ which, in fact, do specify the program occurrences to be processed, most of the discussion will be centered upon their compositions. Since there are two basic data base configurations, the presentation will be similarly partitioned.

Piecemeal and Management Process Base

The contents of $Q_3$ and $Q_4$ which describe the processing stream must refer to some collection of programs. For the studies conducted, the program base was designed to consist of ten programs; five were processor bound and five were I/O bound. From these the various contents of $Q_3$ and $Q_4$ were selected to represent three program classes, three user loss configurations, and two arrival time conditions.

Program Classes. The program classes refer to which programs of the program base were selected to be members of $Q_3$ and $Q_4$. The three considered were: I/O bound, processor bound, and a mixture. In all cases 25 programs were in the $Q_3/Q_4$ system: 14 in $Q_3$ and 11 in $Q_4$. In the I/O bound case, 17 were selected from those programs in the

* The specific input descriptions are provided in the tables of Appendix E.

** Possible interpretations are presented in Appendix F to provide the reader some intuitive idea of potential application situations.
program base in which the lockout period was longer than the processor period. For the processor bound case, 18 were selected from those basic programs in which the processor period was longer than the lockout period. Finally in the mixed situation, 14 were processor oriented and the remaining 11 were I/O bound.

User Loss Configurations. Since the user losses play an important role in the pre-scheduling process, three different loss situations were specified as to the overall loss rates of the program occurrences which were in $Q_3$ and $Q_4$. In the first situation, all loss rates associated with the program occurrences in $Q_3$ were at least a multiplicative factor of ten larger than those in $Q_4$. In the second case, the reverse was true and $Q_4$ contained all the larger, by at least a factor of ten, user loss rates. Finally, a randomly selected mixture of user loss rates was established for the third case with no one queue favored over the other.

Arrival Time Classification. The arrival rate at which program occurrences were eligible for $Q_2$ membership (i.e., become available) can influence the pre-scheduling process. If the arrivals were to be sparsely spaced, then the system would not have the potential to multiprogram as many program occurrences. If, however, the arrival rate exceeds the ability of the system to handle them, then potentially more multiprogramming could occur.

For this discussion two sequences were considered. In the sequence referred to in the tables of Appendix E as "Behind" the arrival rate was approximately ten times faster than the sequence labeled "Even"
for the members of $Q_3$ and approximately three times that for $Q_4$.\footnote{The "Even" and "Behind" refers to how the average arrival rate compares with the average system time. In the first, the average arrival rate is much closer to the processing rate than in the second.}

Long Term Processing

The fundamental goal here was to specify a larger processing situation than considered above and to demonstrate that the proposed pre-scheduling procedure did have the potential of providing a processing improvement in a longer and more complex situation.\footnote{Due to the volume of data required, the input sequences were not included in the paper.} A program base of 25 programs was extracted from the processing histories of a base level U. S. Air Force data processing installation. Also from these processing statistics an input of 259 program occurrences was identified with 200 placed in $Q_3$. The user loss rates were randomly assigned\footnote{The current operational policy of the Air Force has not yet required the assignment of such loss functions; hence, no data were available.} and the system configuration represented the Burroughs B-3500 system on which the original programs were processed. The pre-scheduler was processed for various $Q_2$ limits. The contents of $Q_3$ and $Q_4$ were swapped and the pre-scheduling repeated. The only statistics gathered were total loss, schedule efficiency and $Q_1/Q_2$ size.

Experiment Design

To determine if the pre-scheduler could respond to changes in management policy and to demonstrate that the pre-scheduler does have the potential of being an effective management tool, several processing
situations were established using the basic data in Appendix E. Six situations were investigated: (a) The role of \( Q_2 \) limits, (b) user loss rates/deadline variations, (c) system loss effects, (d) alterations of user/system loss weights, (e) \( Q_1 \) discipline changes, and (f) system reconfiguration effects. All test runs except those for e were made with the first come-first serve \( Q_1 \) discipline since the time accounting was perhaps the most accurate.

**\( Q_2 \) Limitation**

The maximum size of \( Q_2 \) along with how often program occurrences are selected from the \( Q_3/Q_4 \) system can have a definite influence on both the efficiency of the pre-scheduling procedure and the value of system measure obtained. To determine exactly what these effects might be, 72 processing situations were established as follows. For each program base, the three cost situations were processed for the two different time sequences. Each data base established by the above procedure was processed for \( Q_2 \) maximum/minimum limits of 5/4, 6/4, 7/5, and 8/6. The system loss rates \( LTA \) and \( LTB \) were set at 100 and 10, respectively, with a nominal system gain of 1.0. The system configuration used was that denoted as "Basic" in Table 11.

For each test run several factors were noted. These included the system gain, the total loss, the average mix size, the average number of program occurrences in \( Q_1 \) and \( Q_2 \) at mix configuration change, the system

---

*Preliminary pre-scheduler runs with the basic program occurrence set indicated that one of these values would most likely yield the best results.*
through-put, and the schedule efficiency. *(These will be referred to as the "Basic Measures"). The purpose was to note if any trends might occur with the change in input conditions and to give an indication as to the proper $Q_2$ limits.

**User Loss Rates/Deadline Variations**

The management function can govern the manner in which the user loss rates and deadlines are to be established. To determine what effects across-the-board changes could have on the pre-scheduling outcomes, user loss rates and deadlines were altered for the I/O bound, $Q_3/Q_4$ loss rate mixture, and behind input stream.

In the first case tested the user loss rates were set to 10 and 100 and the two deadline conditions were established as in Tables 21 and 22 of Appendix E. This was reversed by setting all deadlines to 0.5 hours after program occurrence arrival. The two loss rate situations established are given in Tables 19 and 20 of Appendix E. In both cases the "Basic" system configuration was used, and the maximum/minimum $Q_2$ limits were set at $5/4$. The same base measures as before were noted.

**System Loss Effects**

Using the same input situation as in the user loss test but without changing the original deadlines and loss rates, eight processing runs were made to determine what effects, if any, resulted from changing the loss rate between the gain values of 1.0 and NSG. To insure that a larger degree of multiprogramming took place, the resource compatibility conditions were "expanded" as shown in the second portion of Table 11, Appendix E.

*These values are explained in detail in Appendix G.*
In the first four runs the system loss rates were set at 100 and 10 and the nominal system gain varied. The values used were 1.0, 1.3, 1.5, 1.7. In the second four runs, the nominal system gain was set at 1.5, and the values of 10, 50, 100, and 200 were assigned to the LTB system loss rate. For the values 100 and 200, LTA was set to the same value as LTB. Both run sets were processed with $Q_2$ limits of $5/4$. The measures were noted as before.

System Configuration Changes

As a by-product of the runs with the system loss rates, the effect on system performance of an increase in system facilities can be observed. These were noted and comparisons were made.

User/System Loss Weights

In the statement of the scheduler measure (Definition 2.18) there are two weighting constants. The relative value of these two parameters can certainly alter the pre-scheduling decisions. Again using the I/O bound $Q_3/Q_4$, Behind situation, four runs were made with the following user/system weight values: $4/1, 2/1, 1/1, 1/2$. In each the nominal gain was set to 1.30, and $Q_2$ limits were set to $5/4$. To insure system loading, the expanded system configuration was used. The same basic system performance measures were recorded.

Queue Discipline Variation

Since all the test runs thus far were made with the first come-first serve $Q_1$ discipline, three of the input situations were selected to run the other queue disciplines. They were (a) I/O-Processor, $Q_4$ loss, Behind; (b) Processor bound, $Q_3/Q_4$ loss mixture, Even; and
(c) I/O bound, $Q_2$ loss. Even. There was no special reason for this selection except that at least one of every condition established for the input set was included. The purpose of this test was to gather the performance measures and to demonstrate that the pre-scheduler could be used to analyze various priority schemes. In each case the basic system configuration was used as described in Table 11 of Appendix E. All but the first case were processed with $Q_2$ limits of 5/4. The limits 6/4 were used in the first case to facilitate the decision process.

Data Analysis

Presented in this section are the results of the various situations analyzed. The detailed results for all but the long term cases are given in Appendix H, and only the summaries needed to justify some results are included here.

Pre-Scheduling Improvement

When each collection of program occurrences was pre-scheduled, the benchmark as explained in Appendix G was also accomplished. This was done to provide some basis of comparison. The average percentage of improvement of the smallest pre-schedule losses over the benchmark losses was 68.3 per cent. When the most efficient schedule was examined, the average percentage of improvement was 59.3 per cent.

From these figures and this data set two observations can be made. First, there appears to be a significant improvement obtained through the use of the pre-scheduler. Second, the increasing of the $Q_2$ limits to obtain the smallest losses would not probably be worth the additional costs of pre-scheduler operation.
Piecemeal Processing

There are two possible factors which could govern the decision of what the $Q_2$ limits are to be. That is, should the pre-scheduler attempt to achieve minimum total loss or should the most efficient pre-schedule be obtained? Which of these selected certainly depends upon the objectives of the system management.

From Table 1 it can be observed that the maximum/minimum $Q_2$ limits which yielded the lowest losses were the 8/6 combination except in the case of the situations in which the largest costs were centered in $Q_4$. This was an understandable occurrence. With smaller limits fewer members of $Q_3$ would be included in the pre-scheduler queue, the examination of the contents of $Q_4$ would occur more frequently, and future $Q_3$ members would have less an influence in the pre-scheduling decisions.

Table 1. $Q_2$ Limit Setting for Lowest Losses

<table>
<thead>
<tr>
<th>Program Class</th>
<th>$Q_3$</th>
<th>$Q_4$</th>
<th>$Q_3/Q_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Even</td>
<td>Behind</td>
<td>Even</td>
</tr>
<tr>
<td>Processor</td>
<td>8/6</td>
<td>8/6</td>
<td>7/5</td>
</tr>
<tr>
<td>Mixture</td>
<td>8/6</td>
<td>8/6</td>
<td>7/5</td>
</tr>
<tr>
<td>I/O</td>
<td>8/6</td>
<td>8/6</td>
<td>8/6</td>
</tr>
</tbody>
</table>

Almost the reverse was true for the $Q_2$ limits which established the most efficient pre-schedule. These results are shown in Table 2.
Table 2. $Q_2$ Limit Setting for Most Efficient Schedule

<table>
<thead>
<tr>
<th>Program Class</th>
<th>$Q_3$ Even</th>
<th>$Q_3$ Behind</th>
<th>$Q_4$ Even</th>
<th>$Q_4$ Behind</th>
<th>$Q_3/Q_4$ Even</th>
<th>$Q_3/Q_4$ Behind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>5/4</td>
<td>8/5</td>
<td>7/5</td>
<td>6/4</td>
<td>7/5</td>
<td>8/6</td>
</tr>
<tr>
<td>Mixture</td>
<td>5/4</td>
<td>6/4</td>
<td>7/5</td>
<td>6/4</td>
<td>5/4</td>
<td>7/5</td>
</tr>
<tr>
<td>I/O</td>
<td>5/4</td>
<td>7/5</td>
<td>7/5</td>
<td>5/4</td>
<td>6/4</td>
<td>5/4</td>
</tr>
</tbody>
</table>

In the more even input stream situations except the $Q_4$ loss case, the $Q_2$ limits were smaller than 8/6. This was understandable in that the extra program occurrences in $Q_2$ which had not yet arrived added little except extra computations to loss reductions. When the arrivals became more frequent, the processing did not keep up as well with the arrivals, and it was necessary to have more program occurrences participating in the pre-scheduling decision.

The exception was again the situation in which the losses were centered in $Q_4$. To keep the losses at a minimum as many members of $Q_4$ as possible must be in the pre-scheduling decision. With the slower arrivals this required a larger maximum on $Q_2$. With the increase in arrival rate, however, the frequency with which $Q_3$ and $Q_4$ were examined became critical. Therefore, the smaller limits were required.

From this analysis, the following are suggested as possible guidelines to the setting of $Q_2$ limits for controlling the piecemeal process:

1. If the goal is to achieve the lowest losses and $Q_4$ loss rates are not expected to be larger than $Q_3$, set the maximum/minimum limits as large as economically possible.
2. If most of the losses are to be centered in \( Q_h \), then the limits on \( Q_2 \) will be expected to be somewhat smaller than in the other cases.

3. If the processor can keep up with the arrival sequence and the losses are not centered in \( Q_h \), then efficient scheduling can be accomplished with a reasonably small \( Q_2 \) limit setting.

4. If the arrivals occur such that a backlog of program occurrences occur and the losses are not centered in \( Q_h \), then a larger \( Q_2 \) limit is required.

These suggestions can only be stated generally, for exact specifications must depend upon the particular processing environment encountered. These, however, should provide a position from which a given situation could be investigated.

**Throughput, Average Mix Size, and Average System Time**

Three of the contemporary measures of computer system performance include system throughput, the average number of programs being serviced by the system, and the average time spent by each request from submission until its completion (i.e., the average system time). It is the purpose here to note that for the 72 situations analyzed and the proposed performance measure, that utilization of these three factors do not necessarily lead to the lowest total losses.

From the tables of Appendix H it was observed that the highest throughput coincided with the lowest losses in only three cases: I/O Processor Mixture, Losses in \( Q_3 \), Behind; Processor bound, Losses in \( Q_3 \), Behind; and I/O bound, Losses in \( Q_h \), Even. This would seem to imply
that there would be a good chance that the overall management and operational goals would not be met for these input sequences if the throughput was the only basis of schedule selection.

The results indicated that using a larger mix size would provide a better selection procedure than the throughput. Referring to Table 3 it can be seen that in five situations the larger average mix size resulted in the lowest loss. These situations can perhaps be explained.

Table 3. Largest Mix Size and Lowest Loss

<table>
<thead>
<tr>
<th>Program Class</th>
<th>Losses</th>
<th>Arrival Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Q_3/Q_4</td>
<td>Behind</td>
</tr>
<tr>
<td>I/O</td>
<td>Q_4</td>
<td>Behind</td>
</tr>
<tr>
<td>Processor</td>
<td>Q_4</td>
<td>Even</td>
</tr>
<tr>
<td>Processor</td>
<td>Q_3</td>
<td>Behind</td>
</tr>
<tr>
<td>Mixture</td>
<td>Q_3</td>
<td>Behind</td>
</tr>
</tbody>
</table>

Notice that all cases but one have the processor service rate behind the arrival rates. In the other the losses were in Q_4. In every situation it would be desirable to examine frequently the contents of Q_3 and Q_4 to be able to account for new arrivals. Regardless of the Q_2 limits, the more program occurrences entering Q_3 from Q_2, the quicker the lower Q_1 limit would be reached. Hence, the input queues would be examined more frequently.

Finally from Appendix H notice that of the runs made, the smallest losses generally occurred with the smallest average system time.
The exceptions were found with the I/O bound data set. This indicated that there would be some advantage to delay the processing of some program occurrences in favor of others. Notice that in the processor bound case, a situation in which it would be difficult in mask processor--I/O operations, the lowest costs and shortest system time occurred in the same $Q_2$ limit cases. Thus, it was suggested that when the ability to carry on many operations simultaneously was reduced, the pre-scheduler could also minimize the average system time of the programs.

User Loss Rates/Deadline Variations

The results for the deadline variations with equal user loss are shown in Figure 4. The second column reflects deadlines which were four times those of the first (see Appendix E, Tables 21 and 22).

Table 4. Deadline Effects

<table>
<thead>
<tr>
<th></th>
<th>Deadline 1</th>
<th>Deadline 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG</td>
<td>1.04</td>
<td>1.06</td>
</tr>
<tr>
<td>Losses</td>
<td>88.980</td>
<td>195.173</td>
</tr>
<tr>
<td>AVMXSZ</td>
<td>1.20</td>
<td>1.32</td>
</tr>
<tr>
<td>AVSYT</td>
<td>0.08636</td>
<td>0.15979</td>
</tr>
<tr>
<td>TP</td>
<td>35.52</td>
<td>36.30</td>
</tr>
<tr>
<td>EFF</td>
<td>0.49698</td>
<td>0.28912</td>
</tr>
<tr>
<td>%IPRV</td>
<td>86.8</td>
<td>61.5</td>
</tr>
<tr>
<td>Cost Calculations</td>
<td>1193</td>
<td>1077</td>
</tr>
</tbody>
</table>

*The notation is explained in Appendix H.*
The outcomes of the across-the-board user loss changes are shown in Table 5. The columns correspond to loss rates shown in Appendix E, Tables 19 and 20, respectively.

<table>
<thead>
<tr>
<th></th>
<th>Loss Rate 1</th>
<th>Loss Rate 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG</td>
<td>1.06</td>
<td>1.06</td>
</tr>
<tr>
<td>Losses</td>
<td>254.344</td>
<td>1017.375</td>
</tr>
<tr>
<td>AVMXSZ</td>
<td>1.40</td>
<td>1.40</td>
</tr>
<tr>
<td>AVSYT</td>
<td>0.30753</td>
<td>0.30753</td>
</tr>
<tr>
<td>EFF</td>
<td>0.04654</td>
<td>0.18618</td>
</tr>
<tr>
<td>%IPRV</td>
<td>17.05</td>
<td>17.05</td>
</tr>
</tbody>
</table>

The immediate observation was that across-the-board rate changes did not alter the processing situations. All that resulted was that the losses and related measures were changed as a function of the rate changes. The schedules generated and the pre-scheduling activity required were the same. Hence, if management desired to alter user loss rates, then it appears that this must be accomplished on a selective, individual basis.

Deadline changes with all loss rates equal did appear, in the case tested, to provide the management function with a tool to improve performance. With the smaller deadlines the pre-scheduling procedure became more selective. Notice the increase in cost calculations denoting the checking of more alternatives. The result was the considerable reduction in losses and average system time for the case with shorter
deadlines but only at the expense of system efficiency which was denoted by decreases in average mix size and throughput.

From these results it appears that the establishment of meaningful task deadlines should be of primary concerns of system management. After this has been accomplished, then the loss rates could be set with caution. Care should be taken to insure a variety of loss rates to insure that the pre-scheduler has as many factors as possible in its decision process.

System Loss Effects

As was noted in the specification of the experiment, the system configuration was expanded for these investigations. The basic results are shown in Tables 6 and 7.

Table 6. Nominal System Gain Effects

<table>
<thead>
<tr>
<th>Nominal System Gain</th>
<th>1.0</th>
<th>1.3</th>
<th>1.5</th>
<th>1.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Losses</td>
<td>133.947</td>
<td>133.947</td>
<td>133.63314</td>
<td>136.19732</td>
</tr>
<tr>
<td>AVMXSZ</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
</tr>
<tr>
<td>AVSYT</td>
<td>0.07717</td>
<td>0.07717</td>
<td>0.08528</td>
<td>0.08528</td>
</tr>
<tr>
<td>TP</td>
<td>48.33</td>
<td>48.33</td>
<td>49.07</td>
<td>49.07</td>
</tr>
<tr>
<td>EFF</td>
<td>0.63342</td>
<td>0.63342</td>
<td>0.69411</td>
<td>0.69302</td>
</tr>
<tr>
<td>%IPRV</td>
<td>87.7</td>
<td>87.7</td>
<td>87.7</td>
<td>87.6</td>
</tr>
</tbody>
</table>
Table 7. System Loss Rate Effects

<table>
<thead>
<tr>
<th>LTB Values--Nominal Gain 1.5</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
</tr>
<tr>
<td>Losses</td>
<td>133.533</td>
<td>135.950</td>
<td>136.950</td>
<td>136.950</td>
</tr>
<tr>
<td>AVMXSZ</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
</tr>
<tr>
<td>AVSYT</td>
<td>0.08528</td>
<td>0.08528</td>
<td>0.08528</td>
<td>0.08528</td>
</tr>
<tr>
<td>TP</td>
<td>49.07</td>
<td>49.07</td>
<td>49.07</td>
<td>49.07</td>
</tr>
<tr>
<td>EFF</td>
<td>0.59411</td>
<td>0.71640</td>
<td>0.73120</td>
<td>0.76135</td>
</tr>
<tr>
<td>%IPRV</td>
<td>87.4</td>
<td>87.5</td>
<td>87.7</td>
<td>88.6</td>
</tr>
</tbody>
</table>

As with the user loss rates, the results for the system loss rates indicate that it was perhaps the setting of the nominal gain value which had more effects on the outcomes than changes in the loss ratio. Increases in the nominal gain increased the system gain and throughput at the expense of user-oriented values of system time. There appeared to be a limiting value on the benefits which could be achieved. This implied that after a certain setting, increasing values more would not overcome limitations imposed by the arrival sequence and resource requirements.

System Reconfiguration

The potential of the pre-scheduler to evaluate possible system reconfigurations can be easily seen by examining the table below. The first column shows the I/O bound, Q/q loss mixture, behind arrival sequence processed with the basic system configuration. The second column of Table 8 shows the results for the expanded system.
Table 8. System Reconfiguration Effects

<table>
<thead>
<tr>
<th></th>
<th>Basic System</th>
<th>Expanded System</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG</td>
<td>1.10</td>
<td>1.41</td>
</tr>
<tr>
<td>Losses</td>
<td>971.130</td>
<td>133.947</td>
</tr>
<tr>
<td>AVMXSZ</td>
<td>1.44</td>
<td>1.60</td>
</tr>
<tr>
<td>AVSYT</td>
<td>0.29359</td>
<td>0.07717</td>
</tr>
<tr>
<td>TP</td>
<td>37.58</td>
<td>48.33</td>
</tr>
<tr>
<td>EFF</td>
<td>0.15506</td>
<td>0.63342</td>
</tr>
<tr>
<td>%IPRV</td>
<td>13.6</td>
<td>87.5</td>
</tr>
</tbody>
</table>

It is obvious that if the processing times of the program occurrences do not change with the system reconfiguration, then considerable advantages both to the system and the users can be realized. Management would now have the use of the pre-scheduler to determine possible advantages which could result from system reconfiguration and could evaluate if the increased costs would be justified by improved overall performance.

User/System Loss Weights

The final factors which could be established by the management function of the system were the weighting factors. It would be expected that the higher the user weight relative to the system weight, the more the individual user would be favored. Similarly, if the relative value of the system weight exceeded the user weight, then system-related parameters should be favored. The results of the experiment appear in Table 9.
Table 9. User/System Weight Effects

<table>
<thead>
<tr>
<th>User/System Values--Nominal Gain 1.3</th>
<th>4/1</th>
<th>2/1</th>
<th>1/1</th>
<th>1/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG</td>
<td>1.00</td>
<td>1.00</td>
<td>1.41</td>
<td>1.41</td>
</tr>
<tr>
<td>Losses</td>
<td>3026.335</td>
<td>349.371</td>
<td>133.947</td>
<td>133.947</td>
</tr>
<tr>
<td>AVMXSZ</td>
<td>1.0</td>
<td>1.08</td>
<td>1.60</td>
<td>1.60</td>
</tr>
<tr>
<td>AVSYT</td>
<td>0.27478</td>
<td>0.11782</td>
<td>0.07717</td>
<td>0.07717</td>
</tr>
<tr>
<td>TP</td>
<td>34.24</td>
<td>34.38</td>
<td>48.33</td>
<td>48.33</td>
</tr>
<tr>
<td>EFF</td>
<td>1.18298</td>
<td>1.53367</td>
<td>0.63438</td>
<td>0.63438</td>
</tr>
<tr>
<td>%IPRV</td>
<td>30.20</td>
<td>84.00</td>
<td>87.50</td>
<td>87.50</td>
</tr>
</tbody>
</table>

With the user weights high, individual users with large loss rates were favored by being given complete control of the system. In the 4/1 case the program occurrences were processed serially since any delay in their processing caused by multiprogramming would induce costs which could not be offset by any multiprogrammed gains. As the weights were lowered to more even values, the gain increased and all users as well as the system were benefited. Notice that with a nominal gain of 1.3, changing of system weights to high values did not affect the outcomes. The reason for this was there were no system losses incurred due to the gain exceeding the nominal value.

It was concluded that if the system management wanted to dedicate the system to the preferential users, then the user weight should be the dominating factor. If the good of all users and the maximum system

*This appears contrary to the results in Table 5 where a limiting situation was reached with one-to-one weighting values. Notice that in Table 5 multiprogramming reduced the overall user losses.
activity were desired, then the system weight should be the maximum of the two values.

Queue Discipline Variations

The data obtained from the pre-scheduling runs to evaluate the different queue priority disciplines is shown in Table 10.

Table 10. Queue Discipline Effects

<table>
<thead>
<tr>
<th>Run</th>
<th>Discipline</th>
<th>SG</th>
<th>Loss</th>
<th>AVSYT (Hours)</th>
<th>TP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FCFS</td>
<td>1.02</td>
<td>329.57</td>
<td>0.12451</td>
<td>34.34</td>
</tr>
<tr>
<td></td>
<td>SJF</td>
<td>1.02</td>
<td>353.93</td>
<td>0.12772</td>
<td>34.31</td>
</tr>
<tr>
<td></td>
<td>NDF</td>
<td>1.02</td>
<td>322.98</td>
<td>0.13501</td>
<td>34.27</td>
</tr>
<tr>
<td></td>
<td>LPF</td>
<td>1.02</td>
<td>330.22</td>
<td>0.13447</td>
<td>34.36</td>
</tr>
<tr>
<td></td>
<td>RR</td>
<td>1.02</td>
<td>337.40</td>
<td>0.13603</td>
<td>34.14</td>
</tr>
<tr>
<td>2</td>
<td>FCFS</td>
<td>1.30</td>
<td>982.80</td>
<td>0.29741</td>
<td>30.77</td>
</tr>
<tr>
<td></td>
<td>SJF</td>
<td>1.29</td>
<td>982.72</td>
<td>0.29744</td>
<td>30.69</td>
</tr>
<tr>
<td></td>
<td>NDF</td>
<td>1.29</td>
<td>982.72</td>
<td>0.29744</td>
<td>30.69</td>
</tr>
<tr>
<td></td>
<td>LPF</td>
<td>1.05</td>
<td>596.58</td>
<td>0.21929</td>
<td>24.84</td>
</tr>
<tr>
<td></td>
<td>RR</td>
<td>1.22</td>
<td>548.02</td>
<td>0.19797</td>
<td>29.03</td>
</tr>
<tr>
<td>3</td>
<td>FCFS</td>
<td>1.23</td>
<td>397.73</td>
<td>0.18563</td>
<td>42.17</td>
</tr>
<tr>
<td></td>
<td>SJF</td>
<td>1.04</td>
<td>553.976</td>
<td>0.26300</td>
<td>35.76</td>
</tr>
<tr>
<td></td>
<td>NDF</td>
<td>1.04</td>
<td>553.976</td>
<td>0.26300</td>
<td>35.76</td>
</tr>
<tr>
<td></td>
<td>LPF</td>
<td>1.19</td>
<td>442.679</td>
<td>0.20002</td>
<td>40.71</td>
</tr>
<tr>
<td></td>
<td>RR</td>
<td>1.26</td>
<td>397.810</td>
<td>0.18515</td>
<td>43.05</td>
</tr>
</tbody>
</table>

* The run numbers correspond as follows:
  (1) I/O-processor mixture, Q_0 loss, behind;
  (2) processor bound, Q_3/Q_4 loss mixture, even;
  (3) I/O bound, Q_3 loss, even.

** The notation used here was developed in Chapter I.
The goal here was not to determine the best queue discipline but to demonstrate that the pre-scheduler could respond in the different situations and could provide management some indication of what the effects of the different disciplines would be.

From the loss figures it was obvious that some of the disciplines for the given data bases were by far superior to others. In every case, the worst loss situation was for the shortest-job-first discipline. Running a close second was the nearest-deadline-first discipline. The least-processing-received-first and the first come-first serve were far better than the other reordering disciplines. In every case the more even time distribution of the round-robin discipline provided either the smallest or next to the smallest losses.

These can only be observations. Firm conclusions could only be made after the pre-scheduling of many different situations. It is the conjecture that in practice the decision as to the best discipline will depend upon the individual application environments and general, far reaching decisions may not be available. In any case, the pre-scheduling results are only as good as the running time estimates of the program occurrences in the job stream.

Long Term Processing

The efficiency results of the pre-scheduling runs with the larger program base and job streams are presented in Figures 8 through 11.\(^\dagger\) In both cases analyzed, Figures 8 and 9 show several low efficiency values.

\(^\dagger\)The connecting of the points in these figures was to facilitate the reader and was not intended to imply continuity.
Figure 8. Efficiency per $Q_1/Q_2$ Size for Activity in $Q_3$
Figure 9. Efficiency per $Q_1/Q_2$ Size for Activity in $Q_4$. 
Figure 10. Loss per $Q_1/Q_2$ Size for Activity in $Q_3$
Figure 11. Loss per $Q_1/Q_2$ Size for Activity in $Q_4$. 
These occurred when the minimum $Q_2$ limit was much lower than the maximum limit. The striking difference between the two cases tested was the value of the average $Q_1/Q_2$ size where the most efficient schedule occurred. When most of the activity was in $Q_3$, the larger $Q_1/Q_2$ sizes provided generally larger efficiency results with the largest efficiency rating occurring with a size of 6.9. When the activity was centered in $Q_4$, the most efficient pre-schedule resulted with the smallest $Q_1/Q_2$ analyzed. The reasons for this were as before. The smaller $Q_2$ limits provide for smaller $Q_1/Q_2$ sizes and a more frequent examination of the $Q_3/Q_4$ system. Since at least one program occurrence, if there was one available, was always selected from $Q_4$ on each examination, there was a higher level of program occurrences drawn from $Q_4$ with the smaller $Q_2$ limits than when the limits were higher. With a small $Q_2$ maximum, the future $Q_3$ activity would influence the pre-scheduler less for there would be fewer of them in the scheduling decision process. The result was a favoring of the program occurrences in $Q_4$.

Figures 10 and 11 show the total loss values as a function of $Q_1/Q_2$ size. Notice that the smaller losses were obtained for the larger queue sizes when the majority of the activity was in $Q_3$. With the processing load placed in $Q_4$, the best loss was achieved for a smaller $Q_1/Q_2$ size (i.e., approximately 6.5 instead of 7.3). This continued to demonstrate that the smaller values favored the $Q_4$ activity.

This concludes the discussion of the test results. Again it should be understood that no general conclusions can be reached from the testing and analysis with respect to the pre-scheduler behavior in
all processing environments. Only implications could be made for the
data tested and these must be verified through future studies involving
many different processing situations. It was felt, however, that these
results have demonstrated that the pre-scheduler can be used as not only
a cost saving device but also as an effective management tool.
CHAPTER VI
CONCLUSIONS AND RECOMMENDATIONS

Conclusions

The purpose of this presentation was twofold: (1) to develop a pre-scheduling procedure which could provide an improvement in the overall user/computer system performance; and (2) to demonstrate that the proposed procedure could be used as a management tool to assist in the evaluation of system operational policies.

In the accomplishment of these basic objectives:

1. The user/computer system parameters were identified and defined.

2. A basic time accounting scheme was proposed for the queue disciplines under study.

3. Expressions were developed to permit a dynamic approach to system performance and progress analysis.

4. A pre-scheduling procedure was specified with the necessary modifications for the various queue disciplines.

5. A piecemeal input stream process was postulated and described.

6. Several basic experiments were performed to provide insights into the pre-scheduling process and to test the ability of the pre-scheduler to respond to management decisions.

The outcomes of the experiments suggested the following with respect to the capabilities of the pre-scheduling procedure.
1. The use of the pre-scheduling procedure did have the potential, with the correct selection of the $Q_2$ size limits, of providing a considerable improvement in system performance as measured by the proposed total loss value.

2. The methodology of the pre-scheduler extended the capabilities of classical job-shop scheduling techniques to account for variable task completion rates and to permit the scheduling outcomes to depend upon all tasks in the scheduling sequence and not just those which preceded in the task sequence.

3. The values of the $Q_2$ limits which resulted in the most efficient pre-schedule sequence were smaller than those values which provided the minimum total losses. This suggests that trying to obtain the minimal total loss would not be justified by the increased processing time required.

4. To obtain the lowest total losses, the $Q_2$ limits will generally be larger when the majority of the user losses were centered in the preferential queue, $Q_3$, than when they were in $Q_4$.

5. An increase in the number of arrivals per unit time required an increase in the $Q_2$ size limits, except when the losses were centered in $Q_4$. The $Q_4$ loss cases required smaller $Q_2$ limits, which seemed to allow more frequent program occurrence submissions from $Q_4$.

6. The classical measurement factors of throughput and average mix size did not appear to be good indicators for identifying that scheduling sequence which would produce the smallest total loss.

7. The selection of the $Q_2$ size limits which produced the
smallest total losses generally produced a pre-schedule with the smallest system time. The exceptions occurred in the I/O bound situations in which there appeared to be some advantage of delaying some program occurrences in favor of others. Since the system gain was generally high in these cases, it was felt that the delays were induced in favor of selecting program occurrences for $Q_1$ membership which could mask the processor-input/output operations to provide, in the long run, a total loss reduction.

8. The pre-scheduling procedure could not anticipate the potential ability of program occurrences to multiprogram efficiently when initial $Q_1$ membership selections were made. The result was that the schedule sequences produced by the proposed procedure were, in some instances, sub-optimal.

9. The pre-scheduling procedure provided, in the instances tested, an indication of the effects on the system performance resulting from changes in user deadlines, system reconfigurations, nominal system gain variations, changes in system loss rates, and variations in user/system loss weights.

10. In the cases tested, general increases in user loss rates had no effect on the pre-scheduling outcomes. It appears that if management wishes to influence the pre-scheduling results through user loss rate changes, then the rate changes must be carefully made on an individual user basis.

11. The pre-scheduler was able to produce schedules for the various queue disciplines used for the system queue, $Q_1$. For a given input
stream, the pre-scheduler would have provided the necessary information to select that discipline satisfying the operational goals of the system management.

12. The pre-scheduler was able to produce an improvement in the total losses, when compared with the benchmark, for a larger collection of program occurrences, which were to be scheduled over a long period of time. When the processing activity was shifted to $Q_4$, the behavior of the pre-scheduler was similar to that of the smaller cases tested.

13. There was an indication that there would be limits imposed by the characteristics of the input stream upon the benefits obtained through system expansions or system loss rate changes.

14. The individual system users could be favored by setting the user loss weighting factor much larger than the system weighting factor. This would cause the individual mix sizes to be reduced permitting the sharing of the computer facilities among fewer program occurrences at any given time. The lower limit on the mix size was defined by the serial processing sequence in which each mix was of size one. Even though individual users were favored, the losses were generally high and the total processing situation involving all users was degraded.

15. From the data gathered it appeared that the setting of the nominal system gain value had a greater influence upon the achieved system gain value than did the setting of the system loss rates. The reason was that after the nominal gain was reached, there were no losses attributed to the system.
Recommendations

The user/computer system descriptions and the pre-scheduler development have provided a foundation upon which several additional studies can be based.

1. The results of the testing was dependent upon the accuracy of the time accounting data used for the processing time estimates of the program occurrences. Additional studies need to be made into the nature of the micro-time allocation procedures of single processor, multiprogrammed computer systems so as to provide the pre-scheduler with more accurate running time estimates.

2. An alternative to exact and accurate time data would be to measure the uncertainty associated with each schedule developed by the pre-scheduler. This could be in the form of confidence limits which would assist in data selection to insure the desired scheduling results.

3. Studies must be made into the proper allocation and assessment of the user loss functions. Guidelines must be developed to assist management in the assigning of the rates so as to achieve the desired operational objectives.

4. The pre-scheduler was specified in such a way that no core swapping among the program occurrences was permitted. The procedure should be expanded in order to determine when and if any advantages could be realized through core sharing and to assist in the determination of the extent to which it is to occur.

5. Program occurrences were assumed to be processed to
completion once they were submitted to \( Q_1 \) for processing. Due to the piecemeal processing and the potential of high cost program occurrences in \( Q_4 \), it may be to the overall advantage of the system and its users to "put a program to sleep," process the new program occurrence, and then resume processing on the original program occurrence. The pre-scheduler should be expanded to consider this possibility which is generally referred to as pre-empt/resume scheduling.

6. Various processing situations defining the contents of the \( Q_3/Q_4 \) system should be identified as to the program classes involved, the nature of the arrival sequence, and the user loss conditions. These should be studied and management guidelines established for the setting of the weighting factors, the nominal system gain, the user loss rates and the system loss rates.

7. Studies should be made into the benefits, if any, by allowing negative losses for any multiprogrammed gains achieved over the nominal system gain value.

8. As it has been pointed out the pre-scheduler cannot anticipate future multiprogrammed benefits (or the lack of them) when the selection of the initial \( Q_1 \) member made. Studies should be made to develop a procedure for efficiently and effectively predicting the overall multiprogrammed effects caused by this initial selection, and this procedure should be added to the pre-scheduling processing.

9. Once some of the management factors have been solved, the final test of the proposed procedure should be made by implementing it on some existing system and by evaluating the changes in the system's performance.
APPENDIX A

EXAMPLE OF WORST-CASE TECHNIQUE FAILURE

The purpose of this example is to demonstrate that the sequence which can be generated by the worst-case procedure is not necessarily that sequence which will lead to a successful scheduling sequence within a multiprogramming situation. The initial sequencing procedure is that of the worst-case situation as described in Chapter III.

Suppose there are four program occurrences with the following basic characteristics:

<table>
<thead>
<tr>
<th>Program Number</th>
<th>Deadline</th>
<th>User Loss Rate Before</th>
<th>User Loss Rate After</th>
<th>Run Alone Time (Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.58</td>
<td>11.38</td>
<td>24.39</td>
<td>0.069</td>
</tr>
<tr>
<td>2</td>
<td>5.50</td>
<td>27.50</td>
<td>44.78</td>
<td>0.573</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>31.25</td>
<td>43.60</td>
<td>0.122</td>
</tr>
<tr>
<td>4</td>
<td>2.82</td>
<td>2.17</td>
<td>6.15</td>
<td>0.324</td>
</tr>
</tbody>
</table>

The arrival sequence is set at

<table>
<thead>
<tr>
<th>Program Number</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8.23</td>
</tr>
<tr>
<td>4</td>
<td>8.24</td>
</tr>
<tr>
<td>1</td>
<td>8.77</td>
</tr>
<tr>
<td>3</td>
<td>8.91</td>
</tr>
</tbody>
</table>
Scheduling is begun with the arrival of the last program occurrence. This allows all arrivals to compete for system service.

The worst-case sequence produced \((2,3,1,4)\) with a processing time required of 1.09 and a total loss of 22.23 units. The prescheduling would automatically select 2 as the first member of the first mix configuration. This selection will produce the processing schedule,

\[
S_1 = [(2),(3,1),(1),(4)]
\]

with a processing time of 1.06 hours and a loss of 21.96 units. Notice that this provided an improvement over the serial, run alone situation.

Suppose, however, a sequence of \((1,2,3,4)\) were taken as the serial processing sequence and it is this sequence which is to be used to construct the processing schedule. The results would have been a schedule

\[
S_2 = [(1,3),(1,4),(4),(2)]
\]

with a processing time of 1.06 hours and total loss of 17.66 units. The improvement results from the selection of program occurrence 1 as the initial member of the first mix configuration permitting a better mix among the tasks to be scheduled. It is this condition that a task independent sequencing technique cannot recognize without resorting to a combinatorial argument.
Even though the pre-scheduler did not provide the better of the alternatives, it did account for a mixing of the program occurrences and a sharing of the system's facilities to reflect improved conditions over the serial processing situation.

It is not to be understood that the technique used in this thesis and example reflects the behavior of all job-shop procedures. This is far from true. It does, however, require the fundamental assumptions employed in most job-shop techniques: (1) run characteristics are independent of the other jobs and (2) the scheduling decision is a function of only those tasks preceding in the scheduling sequence. In the example, the processing behavior depended upon the other tasks to be scheduled and how they were mixed and an improved scheduling sequence could be obtained by considering those tasks which followed in the processing sequence. Hence, an industrial job-shop procedure should be used only as a guessing procedure to be improved upon by other techniques.
APPENDIX B

DERIVATION OF USER LOSS FUNCTION

Referring to Figure 12, if a program completes prior to deadline only one rate, $R_1$, applies. Similarly, if there is a deadline and the program submission is after this time, only the rate $R_2$ applies. This accounts for the first portion of Definition 2.14. The complications arise from a program being processed over two rates.

Assuming that the deadline $DLTIM(p,u,c)$ is interpreted relative to the arrival of the program occurrence, the following is an applicable argument. Let $TS(p,u,c) = t$ and $t > DLTIM(p,u,c)$. The value of user loss cannot be expressed by the function $R_2 \times t$ since the loss value at $DLTIM(p,u,c)$ is not zero. In particular the value is expressible in terms of the rate $R_1$. The problem is to express a linear function through $(DLTIM(p,u,c), R_1 \times DLTIM(p,u,c))$ with slope $R_2$. Hence,

$$R_1 \times DLTIM(p,u,c) = R_2 \times DLTIM(p,u,c) + b \quad (B.1)$$

and

$$b = (R_1 - R_2) \times DLTIM(p,u,c) \quad (B.2)$$

which gives the loss function

$$v = ULF(p,u,c,TS(p,u,c)) = R_2 \times TS(p,u,c) + DLTIM(p,u,c) \times (R_1 - R_2) \quad (B.3)$$
Figure 12. User Loss Function Derivation
APPENDIX C

DERIVATION OF SYSTEM LOSS FUNCTION

Referring to Figure 13, all that is necessary is to determine the values of \( b \) and \( c \). The rates \( LTA \) and \( LTB \) are assumed to be determined through management policies.

For a range of \( SG \) between 1.0 and \( NSG \) we have

\[
SL = LTB(SG) + c \quad (C.1)
\]

Using the point \((NSG,0)\),

\[
0 = LTB(NSG) + c \quad (C.2)
\]

\[
c = (-LTB)(NSG) \quad (C.3)
\]

Thus if \( 1.0 \leq SG < NSG \)

\[
SL = LTB(SG) - (LTB)(NSG) \quad (C.4)
\]

\[
= LTB(SG-NSG)
\]

Using \( C.4 \), the value \( d \) of the point \((1.0,d)\) is found to be

\[
d = LTB(1.0-NSG) \quad (C.5)
\]
Since the line whose equation is

\[ SL = LTA(SG) + b \]  \hspace{1cm} (C.6)

must also pass through \((1.0,d)\), we have from C.5 and C.6

\[ SL = LTB(1.0-NSG) = LTA(1.0) + b \]

and

\[ b = LTB(1.0-NSG) - LTA \]  \hspace{1cm} (C.7)

Hence for \(SG < 1.0\), we have

\[ SL = LTA(SG) + LTB(1.0-NSG) - LTA \]  \hspace{1cm} (C.8)

\[ = LTA(SG-1.0) + LTB(1.0-NSG); \]

Therefore, C.4 and C.8 represent the expressions for evaluation of system losses.
Figure 13. System Loss Function Derivation
APPENDIX D

EXAMPLE OF IDLE PERIOD OCCURRENCE

This appendix is provided to demonstrate that it is possible to generate a worst-case processing sequence which contains idle periods even though there are program occurrences available which could be processed.

Consider the following program occurrences.

<table>
<thead>
<tr>
<th>Program Number</th>
<th>Deadline</th>
<th>User Loss Rate Before</th>
<th>User Loss Rate After</th>
<th>Run Alone Time (Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.50</td>
<td>5.55</td>
<td>26.87</td>
<td>0.62</td>
</tr>
<tr>
<td>2</td>
<td>2.89</td>
<td>11.98</td>
<td>33.62</td>
<td>0.51</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40.77</td>
<td>41.77</td>
<td>0.53</td>
</tr>
<tr>
<td>4</td>
<td>5.51</td>
<td>41.35</td>
<td>46.59</td>
<td>0.60</td>
</tr>
<tr>
<td>5</td>
<td>2.96</td>
<td>0.48</td>
<td>11.33</td>
<td>0.66</td>
</tr>
<tr>
<td>6</td>
<td>6.49</td>
<td>14.98</td>
<td>32.37</td>
<td>0.90</td>
</tr>
</tbody>
</table>

The arrival sequence is set at

<table>
<thead>
<tr>
<th>Program Number</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.07</td>
</tr>
<tr>
<td>3</td>
<td>9.08</td>
</tr>
<tr>
<td>2</td>
<td>9.12</td>
</tr>
<tr>
<td>6</td>
<td>9.28</td>
</tr>
<tr>
<td>5</td>
<td>9.67</td>
</tr>
<tr>
<td>4</td>
<td>9.68</td>
</tr>
</tbody>
</table>
The worst-case procedure produces the following sequence when pre-scheduling is begun with the arrival of program occurrence 3 at 9.08.

<table>
<thead>
<tr>
<th>Program Number</th>
<th>Start Time</th>
<th>Stop Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9.08</td>
<td>9.62</td>
</tr>
<tr>
<td>4</td>
<td>9.68</td>
<td>10.28</td>
</tr>
<tr>
<td>6</td>
<td>10.28</td>
<td>11.18</td>
</tr>
<tr>
<td>2</td>
<td>11.18</td>
<td>11.68</td>
</tr>
<tr>
<td>1</td>
<td>11.68</td>
<td>12.51</td>
</tr>
<tr>
<td>5</td>
<td>12.31</td>
<td>12.97</td>
</tr>
</tbody>
</table>

The cost of this sequence is 144.80. To have placed the available occurrences in the sequence upon completion of 3 would have yielded the following results.

<table>
<thead>
<tr>
<th>Program Occurrence Sequence</th>
<th>Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,5,4,6,2,1</td>
<td>190.70</td>
</tr>
<tr>
<td>3,2,4,6,1,5</td>
<td>150.60</td>
</tr>
<tr>
<td>3,1,4,6,2,5</td>
<td>158.20</td>
</tr>
<tr>
<td>3,6,4,2,1,5</td>
<td>167.40</td>
</tr>
</tbody>
</table>

From the above results it is obvious that the concept of always keeping the system busy is the best way to satisfy the users may not always be true.
APPENDIX E

EVALUATION INPUT DATA

Included in the tables of this appendix are the data elements used as input in the testing of pre-scheduler response to changes in Q₂ size limits and management factors. One table outlines the program base, another the system configuration used, and the others the various management variations tested. The reference of occurrence number is to the relative position in the input queues. The notation of Q₃, Q₄ and Q₃/Q₄ is used to denote in which the costs are centered. As explained earlier, "Even" and "Behind" refer to the arrival intervals as compared to the processing rate. To obtain the run input data for Q₃ and Q₄ membership, select from the appropriate tables the program number, arrival time, user loss rates, data units, and deadlines.

Table 11. System Configurations

<table>
<thead>
<tr>
<th></th>
<th>Basic</th>
<th>Expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>Printer</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Punch</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Reader</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Tape</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Disk (Segments)</td>
<td>2,000</td>
<td>2,000</td>
</tr>
<tr>
<td>Remote</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Namespace</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Table 12. Program Base

<table>
<thead>
<tr>
<th>Program Number</th>
<th>Core</th>
<th>Resource Requirements</th>
<th>Time Requirements</th>
<th>Cycles/</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Printer</td>
<td>Punch</td>
<td>Reader</td>
</tr>
<tr>
<td>1</td>
<td>20,000</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>25,000</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>10,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>30,000</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>40,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>25,000</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>50,000</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>15,000</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>20,000</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>20,000</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 13. $Q_3$ Program Compositions and Data Units

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>Program Numbers</th>
<th>Data Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I/O Bound</td>
<td>Processor Bound</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

*The Data Units are included here for they do not change.

Table 14. $Q_4$ Program Composition and Data Units

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>Program Numbers</th>
<th>Data Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I/O Bound</td>
<td>Processor Bound</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 15. Arrival Times for Q₃

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>Arrival Time</th>
<th>Behind</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.00</td>
<td>8.00</td>
</tr>
<tr>
<td>2</td>
<td>8.00</td>
<td>8.00</td>
</tr>
<tr>
<td>3</td>
<td>8.03</td>
<td>8.01</td>
</tr>
<tr>
<td>4</td>
<td>8.07</td>
<td>8.02</td>
</tr>
<tr>
<td>5</td>
<td>8.11</td>
<td>8.03</td>
</tr>
<tr>
<td>6</td>
<td>8.12</td>
<td>8.03</td>
</tr>
<tr>
<td>7</td>
<td>8.12</td>
<td>8.04</td>
</tr>
<tr>
<td>8</td>
<td>8.16</td>
<td>8.05</td>
</tr>
<tr>
<td>9</td>
<td>8.20</td>
<td>8.06</td>
</tr>
<tr>
<td>10</td>
<td>8.22</td>
<td>8.07</td>
</tr>
<tr>
<td>11</td>
<td>8.26</td>
<td>8.09</td>
</tr>
<tr>
<td>12</td>
<td>8.27</td>
<td>8.10</td>
</tr>
<tr>
<td>13</td>
<td>8.31</td>
<td>8.11</td>
</tr>
<tr>
<td>14</td>
<td>8.35</td>
<td>8.12</td>
</tr>
</tbody>
</table>

Average Time Between Arrivals 0.0250 Hours 0.0086 Hours

Table 16. Arrival Times for Q₄

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>Arrival Time</th>
<th>Behind</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.01</td>
<td>8.01</td>
</tr>
<tr>
<td>2</td>
<td>8.03</td>
<td>8.02</td>
</tr>
<tr>
<td>3</td>
<td>8.07</td>
<td>8.03</td>
</tr>
<tr>
<td>4</td>
<td>8.10</td>
<td>8.04</td>
</tr>
<tr>
<td>5</td>
<td>8.14</td>
<td>8.05</td>
</tr>
<tr>
<td>6</td>
<td>8.17</td>
<td>8.07</td>
</tr>
<tr>
<td>7</td>
<td>8.21</td>
<td>8.08</td>
</tr>
<tr>
<td>8</td>
<td>8.25</td>
<td>8.09</td>
</tr>
<tr>
<td>9</td>
<td>8.28</td>
<td>8.10</td>
</tr>
<tr>
<td>10</td>
<td>8.31</td>
<td>8.12</td>
</tr>
<tr>
<td>11</td>
<td>8.35</td>
<td>8.14</td>
</tr>
</tbody>
</table>

Average Time Between Arrivals 0.0309 Hours 0.0118 Hours
Table 17. $Q_3$ User Loss Rates and Deadlines

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>$Q_3$</th>
<th>$Q_4$</th>
<th>$Q_3/Q_4$</th>
<th>Deadline (Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R2</td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>500</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>150</td>
<td>200</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>150</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>200</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>150</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>150</td>
<td>200</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>200</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>200</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>125</td>
<td>200</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>300</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>125</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>150</td>
<td>300</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>100</td>
<td>300</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>14</td>
<td>100</td>
<td>200</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 18. $Q_4$ User Loss Rates and Deadlines

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>$Q_3$</th>
<th>$Q_4$</th>
<th>$Q_3/Q_4$</th>
<th>Deadline (Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R2</td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>15</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>30</td>
<td>150</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>20</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>50</td>
<td>300</td>
<td>500</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>30</td>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>30</td>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>15</td>
<td>150</td>
<td>500</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>15</td>
<td>150</td>
<td>200</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>30</td>
<td>150</td>
<td>300</td>
</tr>
</tbody>
</table>
Table 19. \(Q_3\) User Loss Rates for User Loss Change Test

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>Test 1</th>
<th>Test 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>200</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>6</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>7</td>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>200</td>
</tr>
<tr>
<td>9</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>11</td>
<td>200</td>
<td>250</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>600</td>
</tr>
<tr>
<td>13</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>14</td>
<td>200</td>
<td>400</td>
</tr>
</tbody>
</table>

All deadlines set to 0.5 hours.

Table 20. \(Q_4\) User Loss Rates for User Loss Change Test

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>Test 1</th>
<th>Test 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>600</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>600</td>
<td>1000</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>10</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>600</td>
</tr>
</tbody>
</table>

All deadlines set to 0.5 hours.
Table 21. \( Q_3 \) Deadlines for Deadline Analysis

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>Deadlines (Hours)</th>
<th>Test 1</th>
<th>Test 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.00</td>
<td>0.500</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.20</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.02</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.00</td>
<td>0.500</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.20</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.10</td>
<td>0.025</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.20</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.02</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.20</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.20</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.10</td>
<td>0.025</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1.00</td>
<td>0.250</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.20</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.02</td>
<td>0.005</td>
<td></td>
</tr>
</tbody>
</table>

User Loss Rate \( R_1 = 10 \) \( R_2 = 100 \)

Table 22. \( Q_4 \) Deadlines for Deadline Analysis

<table>
<thead>
<tr>
<th>Program Occurrence</th>
<th>Deadlines (Hours)</th>
<th>Test 1</th>
<th>Test 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>0.250</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.20</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.20</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>0.250</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.02</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.02</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.02</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2.00</td>
<td>0.500</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.02</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.02</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.10</td>
<td>0.025</td>
<td></td>
</tr>
</tbody>
</table>

User Loss Rate \( R_1 = 10 \) \( R_2 = 100 \)
APPENDIX F

INTERPRETATIONS OF PROCESSING ENVIRONMENTS

Introduction

To provide the reader with an idea of potential application environments in which the pre-scheduler might be used, the following discussion has been included. The general situation is first outlined and then the loss and $Q_3/Q_4$ assignments are outlined. Examples are presented for each of the three program classes used in the $Q_3/Q_4$ testing phase (i.e., I/O bound, processor bound, and a mixture of the first two).

Input/Output Bound

This type of processing situation could represent a document storage and retrieval system in which the majority of the programs require the searching of large volumes of data with little processing required. Also new data is to be added to update existing files.

For example, the using community is to consist of the following categories of users: (a) library staff, (b) faculty and graduate students, (c) undergraduates, and (d) off campus users. It is the primary responsibility of the library staff to keep the files as current as possible for the hierarchy of users as shown above. Hence, in terms of the $Q_3/Q_4$ treatment and user losses, the library staff would be assigned the highest losses with shortest deadlines whereas the
off-campus users would be assigned minimal losses and likely permitted no deadline establishments. Since $Q_3$ members are established to have preference over those in $Q_4$, it could be that only the library staff and faculty could schedule (plan in advance) system service and all other users would be required to enter $Q_4$ to compete for $Q_2$ membership on a first come-first serve basis.

Another example of a primarily file-oriented, I/O bound system could be a medical system. Again, there would be similar user classes with a possible change in the hierarchy of losses. That is, the classes could be: (a) operating room support, (b) the doctors using the system for on-the-spot diagnostic purposes, (c) the library maintenance staff, (d) the doctor using the system for educational purposes, and (e) the medical student for educational purposes. In this case the first two classes are assigned to $Q_3$. As soon as it is known that an operation is to occur, processing histories could establish a basis for entering program occurrences in $Q_3$ to indicate a need for support. To insure the system availability the assigning of losses high enough to these types of requests will always override other uses of the system. In the meantime, the system could be serving other requests of file maintenance and educational needs.

**Processor Bound**

This type of activity could be representative of a scientific processing situation. In particular, consider a master-slave computer system in which the slave functions only as an input/output device. All data required for the computation process would be passed to the
the master system, the computations performed, and the results returned to slave system. If the master system has sufficient core storage for holding all data required for processing and data transfer between the two systems is at data channel speeds, then little time will be expected to be spent in lockout periods.

With this type of system description, an application environment could be established as follows. Suppose that the system is to be used as a monitor and control mechanism for some real time processes. In normal mode, data is collected by the slave system and passed at periodic intervals to the master for analysis. If adjustments are required to the process being monitored, then the master system initiates the appropriate action. Finally, while the control and monitor data are in the master system, certain summary operations can be performed for later analysis by the system technicians and engineers.

This type of application environment has defined three user classes. The high loss will be assigned to the control adjustment applications. The monitor function will be next and followed by the summary operations. The first two types of operation will be given the preferential $Q_3$ membership, whereas the summary situations will be given to $Q_4$.

**Mixture of I/O-Processor Bound**

In the previous situation suppose that during certain periods of operation the monitoring and control functions are not to be required as frequently and some of the system's time could be used for other applications. A fourth class of users could be introduced to include the
technical and engineering personnel who wish to implement certain analysis programs based upon the summary data collected. Since files are to be involved, the frequency with which interaction of the two systems occur increases. Thus, there are introduced some I/O bound tasks.

If the personnel have previously informed the system personnel of their processing needs, these requests will be placed in $Q_3$; otherwise, they will have to enter $Q_4$ and await their turn.
APPENDIX G

EVALUATION MEASURES

In order to discuss how well the pre-scheduler performed, there must be established some measures upon which comparisons can be based. These are to include the decision measure as specified in Definition 2.18 and expression 3.11 as well as those of throughput, average system time, average mix size, percentage of improvement, and schedule efficiency. These will now be established.

Throughput

The definition of throughput used here is perhaps the most common. It is to be the average number of programs processed per unit time of processing.

\[ TP = \frac{\text{total program occurrences processed}}{RT(S)} \]

where \( RT(S) \) is the total schedule time from Lemma 3.

Average System Time

The average system time is intended to represent how long each program occurrence was in the system. That is

\[ \text{AVSYT} = \frac{\sum (\text{occurrence exit} - \text{occurrence arrival})}{\text{total program occurrences}} \]
Average MIX Size

This value of average $Q_1$ size is intended to give an indication of how many program occurrences can be expected to be in $Q_1$. It is required that this value be somewhat larger than one if there is to be any multiprogramming advantages as reflected by the system gain. If $MIXS_i$ is the size of $MIX_i$ in the schedule and there are $n$ mix configurations required to define the schedule, then

$$AVMXS = \frac{\sum_{i=1}^{n} MIXS_i}{n}$$

$Q_1/Q_2$ Size

Another measure defines the average number of program occurrences within the pre-scheduler boundary at each mix configuration change. If $Q2SZ_i$ is the size of $Q_2$ at $MIX_i$ then

$$Q_1/Q_2 \text{ Size} = \frac{\sum_{i=1}^{n} (Q2SZ_i + MIXS_i)}{n}$$

is the expected number of program occurrences within the pre-scheduler queues when there are $n$ mix configurations defining the schedule.

Schedule Efficiency and Percentage of Improvement

The final measures have to do with the pre-scheduler efficiency. If the cost savings produced by the pre-scheduler when compared to some benchmark are small relative to the system activity required to produce the saving, then the pre-scheduler is costing too much to run. The
purpose of schedule efficiency is to monitor the effects of the number of program occurrences in the pre-scheduler queues on the scheduling outcome. The percentage of improvement is intended to give some indication of how the pre-scheduler decisions compared to some test case.

The basis of pre-scheduler system requirements is to be the frequency with which the user loss function, ULF, is required. The reasoning is the pre-scheduler cannot make decisions without relying upon this function. If the benchmark processing sequence has a total user/system loss of BC and the pre-scheduling losses are specified by Definition 2.18 and expression 3.11, then the schedule efficiency is

$$\text{EFF} = \frac{BC - M(S)}{\text{number of ULF evaluations}}$$

The percentage of improvement is another comparison with the benchmark procedure. It is defined as

$$\%\text{IPRV} = \frac{BC - M(S)}{BC} \times 100$$

**Benchmark Procedure**

The benchmark referred to throughout the paper is to be established as the first come-first serve submission from the combined Q3/Q4 queues to Q1. A program occurrence will not be added to Q1 only when it is not resource compatible. The processing losses can be found using the same expression as those in the pre-scheduling procedure. By using this for a basis of comparison, the benefits due to the pre-scheduling
decision can be examined for the only basic difference in the pre-
scheduler and benchmark is the benchmark does not contain any decision
element.
APPENDIX H

TEST RESULTS

Included here are the detailed results of the processing runs made with the data of Appendix E to determine the nature of the piece-meal process. The notation used is explained as follows:

- **SG** - System Gain.
- **EFF** - Pre-Schedule Efficiency.
- **%IPRV** - Percentage Improvement of Pre-Schedule versus the Benchmark.
- **AVMXSZ** - Average Mix Size (i.e., number of program occurrences in \( Q_1 \)).
- **TP** - Throughput.
- **AVSYT** - Average System Time.

\( Q_1/Q_2 \) Size - Average Number of Program Occurrences in the \( Q_1/Q_2 \) System at Mix Change.

The calculations necessary are explained in Appendix G. The numbers in parentheses with the \( Q_1/Q_2 \) Size are the maximum/minimum \( Q_2 \) limits used.
Table 23. Processor Bound—Loss in $Q_3$—Even Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2174 (5/4)</td>
<td>1.09</td>
<td>567,496</td>
<td>0.23929</td>
<td>40.70</td>
<td>1.80</td>
<td>25.92</td>
<td>0.25488</td>
</tr>
<tr>
<td>5.8696 (6/4)</td>
<td>1.09</td>
<td>608,453</td>
<td>0.17459</td>
<td>38.40</td>
<td>1.96</td>
<td>25.87</td>
<td>0.30881</td>
</tr>
<tr>
<td>6.1250 (7/5)</td>
<td>1.02</td>
<td>401,663</td>
<td>0.23079</td>
<td>58.80</td>
<td>1.52</td>
<td>24.09</td>
<td>0.19641</td>
</tr>
<tr>
<td>6.8333 (8/6)</td>
<td>1.02</td>
<td>400,628</td>
<td>0.17070</td>
<td>58.80</td>
<td>1.52</td>
<td>24.14</td>
<td>0.19565</td>
</tr>
</tbody>
</table>

Table 24. Processor Bound—Loss in $Q_3$—Behind Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0833 (5/4)</td>
<td>1.26</td>
<td>1109,405</td>
<td>0.19837</td>
<td>19.90</td>
<td>1.64</td>
<td>29.92</td>
<td>0.42384</td>
</tr>
<tr>
<td>5.4583 (6/4)</td>
<td>1.30</td>
<td>1040,314</td>
<td>0.17873</td>
<td>24.90</td>
<td>1.68</td>
<td>30.94</td>
<td>0.40784</td>
</tr>
<tr>
<td>6.0833 (7/5)</td>
<td>1.26</td>
<td>862,261</td>
<td>0.17852</td>
<td>37.70</td>
<td>1.52</td>
<td>29.85</td>
<td>0.35537</td>
</tr>
<tr>
<td>6.7500 (8/5)</td>
<td>1.31</td>
<td>528,024</td>
<td>0.21237</td>
<td>61.80</td>
<td>1.60</td>
<td>31.00</td>
<td>0.26453</td>
</tr>
</tbody>
</table>

Table 25. Processor Bound—Loss in $Q_4$—Even Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0000 (5/4)</td>
<td>1.01</td>
<td>1081,118</td>
<td>0.00139</td>
<td>0.183</td>
<td>1.76</td>
<td>24.06</td>
<td>0.34013</td>
</tr>
<tr>
<td>5.5833 (6/4)</td>
<td>1.01</td>
<td>857,002</td>
<td>0.08783</td>
<td>17.40</td>
<td>1.50</td>
<td>24.06</td>
<td>0.30626</td>
</tr>
<tr>
<td>5.6667 (7/5)</td>
<td>1.25</td>
<td>609,862</td>
<td>0.26062</td>
<td>43.70</td>
<td>1.32</td>
<td>29.73</td>
<td>0.22676</td>
</tr>
<tr>
<td>6.6667 (8/6)</td>
<td>1.26</td>
<td>681,525</td>
<td>0.14181</td>
<td>37.00</td>
<td>1.76</td>
<td>29.91</td>
<td>0.25133</td>
</tr>
</tbody>
</table>

Table 26. Processor Bound—Loss in $Q_4$—Behind Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0000 (5/4)</td>
<td>1.01</td>
<td>1364,168</td>
<td>0.12152</td>
<td>11.20</td>
<td>1.76</td>
<td>24.06</td>
<td>0.42053</td>
</tr>
<tr>
<td>5.2917 (6/4)</td>
<td>1.13</td>
<td>547,313</td>
<td>0.43246</td>
<td>63.80</td>
<td>1.36</td>
<td>25.82</td>
<td>0.17524</td>
</tr>
<tr>
<td>6.1257 (7/5)</td>
<td>1.02</td>
<td>243,510</td>
<td>0.36305</td>
<td>84.00</td>
<td>1.36</td>
<td>24.09</td>
<td>0.13749</td>
</tr>
<tr>
<td>6.5318 (8/6)</td>
<td>1.02</td>
<td>362,324</td>
<td>0.24216</td>
<td>74.40</td>
<td>1.37</td>
<td>24.01</td>
<td>0.13732</td>
</tr>
</tbody>
</table>
### Table 27. Processor Bound--Loss Mixed Q₃/Q₄--Even Arrivals

<table>
<thead>
<tr>
<th>Q₁/Q₂ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9583 (5/4)</td>
<td>1.30</td>
<td>982.804</td>
<td>0.12413</td>
<td>12.12</td>
<td>1.68</td>
<td>30.77</td>
<td>0.29748</td>
</tr>
<tr>
<td>5.1667 (6/4)</td>
<td>1.03</td>
<td>583.997</td>
<td>0.36633</td>
<td>48.80</td>
<td>1.40</td>
<td>24.44</td>
<td>0.21881</td>
</tr>
<tr>
<td>5.7083 (7/5)</td>
<td>1.10</td>
<td>259.837</td>
<td>0.51412</td>
<td>77.00</td>
<td>1.48</td>
<td>26.20</td>
<td>0.13584</td>
</tr>
<tr>
<td>6.4583 (8/6)</td>
<td>1.30</td>
<td>55.223</td>
<td>0.47521</td>
<td>95.00</td>
<td>1.36</td>
<td>30.74</td>
<td>0.08703</td>
</tr>
</tbody>
</table>

### Table 28. Processor Bound--Loss Mixed Q₃/Q₄--Behind Arrivals

<table>
<thead>
<tr>
<th>Q₁/Q₂ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7083 (5/4)</td>
<td>1.14</td>
<td>989.846</td>
<td>0.49966</td>
<td>42.10</td>
<td>1.52</td>
<td>27.06</td>
<td>0.30819</td>
</tr>
<tr>
<td>5.2500 (6/4)</td>
<td>1.14</td>
<td>956.335</td>
<td>0.36883</td>
<td>43.00</td>
<td>1.56</td>
<td>27.13</td>
<td>0.30370</td>
</tr>
<tr>
<td>5.9583 (7/5)</td>
<td>1.03</td>
<td>545.144</td>
<td>0.48625</td>
<td>67.80</td>
<td>1.52</td>
<td>24.48</td>
<td>0.24480</td>
</tr>
<tr>
<td>6.6250 (8/6)</td>
<td>1.11</td>
<td>64.932</td>
<td>0.54795</td>
<td>96.00</td>
<td>1.64</td>
<td>26.44</td>
<td>0.09902</td>
</tr>
</tbody>
</table>

### Table 29. I/O Bound--Loss in Q₃--Even Arrivals

<table>
<thead>
<tr>
<th>Q₁/Q₂ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7917 (5/4)</td>
<td>1.23</td>
<td>397.733</td>
<td>0.20740</td>
<td>33.15</td>
<td>1.60</td>
<td>42.17</td>
<td>0.18563</td>
</tr>
<tr>
<td>5.0000 (6/4)</td>
<td>1.18</td>
<td>399.839</td>
<td>0.15260</td>
<td>33.33</td>
<td>1.60</td>
<td>40.56</td>
<td>0.19507</td>
</tr>
<tr>
<td>6.0833 (7/5)</td>
<td>1.01</td>
<td>471.792</td>
<td>0.05558</td>
<td>24.40</td>
<td>1.35</td>
<td>34.63</td>
<td>0.24745</td>
</tr>
<tr>
<td>7.0000 (8/6)</td>
<td>1.18</td>
<td>396.409</td>
<td>0.05738</td>
<td>32.50</td>
<td>1.40</td>
<td>40.34</td>
<td>0.19626</td>
</tr>
</tbody>
</table>

### Table 30. I/O Bound--Loss in Q₃--Behind Arrivals

<table>
<thead>
<tr>
<th>Q₁/Q₂ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8333 (5/4)</td>
<td>1.07</td>
<td>666.477</td>
<td>0.24415</td>
<td>27.80</td>
<td>1.40</td>
<td>36.71</td>
<td>0.27415</td>
</tr>
<tr>
<td>5.5000 (6/4)</td>
<td>1.07</td>
<td>543.463</td>
<td>0.28237</td>
<td>41.10</td>
<td>1.64</td>
<td>36.62</td>
<td>0.25528</td>
</tr>
<tr>
<td>6.0417 (7/5)</td>
<td>1.05</td>
<td>402.931</td>
<td>0.24822</td>
<td>56.40</td>
<td>1.52</td>
<td>35.99</td>
<td>0.23195</td>
</tr>
<tr>
<td>6.7083 (8/6)</td>
<td>1.05</td>
<td>400.976</td>
<td>0.16946</td>
<td>55.60</td>
<td>1.28</td>
<td>36.01</td>
<td>0.23966</td>
</tr>
</tbody>
</table>
Table 31. I/O Bound--Loss in $Q_4$--Even Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMSXZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3750 (5/4)</td>
<td>1.04</td>
<td>524.582</td>
<td>0.15121</td>
<td>18.60</td>
<td>1.24</td>
<td>35.71</td>
<td>0.18244</td>
</tr>
<tr>
<td>4.8333 (6/4)</td>
<td>1.04</td>
<td>528.475</td>
<td>0.15107</td>
<td>17.85</td>
<td>1.36</td>
<td>35.62</td>
<td>0.17754</td>
</tr>
<tr>
<td>5.7083 (7/5)</td>
<td>1.04</td>
<td>418.847</td>
<td>0.17591</td>
<td>35.30</td>
<td>1.24</td>
<td>35.55</td>
<td>0.16106</td>
</tr>
<tr>
<td>6.8607 (8/6)</td>
<td>1.05</td>
<td>404.753</td>
<td>0.12143</td>
<td>37.50</td>
<td>1.32</td>
<td>35.87</td>
<td>0.15687</td>
</tr>
</tbody>
</table>

Table 32. I/O Bound--Loss in $Q_4$--Behind Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMSXZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8333 (5/4)</td>
<td>1.03</td>
<td>393.381</td>
<td>0.67381</td>
<td>61.00</td>
<td>1.44</td>
<td>35.30</td>
<td>0.13778</td>
</tr>
<tr>
<td>5.2917 (6/4)</td>
<td>1.05</td>
<td>339.625</td>
<td>0.48323</td>
<td>66.20</td>
<td>1.32</td>
<td>35.94</td>
<td>0.11784</td>
</tr>
<tr>
<td>6.2500 (7/5)</td>
<td>1.02</td>
<td>239.837</td>
<td>0.41189</td>
<td>76.20</td>
<td>1.48</td>
<td>34.88</td>
<td>0.11357</td>
</tr>
<tr>
<td>7.0833 (8/6)</td>
<td>1.05</td>
<td>202.047</td>
<td>0.25575</td>
<td>80.40</td>
<td>1.64</td>
<td>35.82</td>
<td>0.11790</td>
</tr>
</tbody>
</table>

Table 33. I/O Bound--Loss Mixed $Q_3/Q_4$--Even Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMSXZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7500 (5/4)</td>
<td>1.04</td>
<td>575.135</td>
<td>0.10639</td>
<td>15.40</td>
<td>1.44</td>
<td>35.75</td>
<td>0.20393</td>
</tr>
<tr>
<td>4.8333 (6/4)</td>
<td>1.01</td>
<td>91.428</td>
<td>0.592120</td>
<td>86.60</td>
<td>1.32</td>
<td>34.62</td>
<td>0.05287</td>
</tr>
<tr>
<td>5.5000 (7/5)</td>
<td>1.02</td>
<td>52.630</td>
<td>0.42340</td>
<td>92.20</td>
<td>1.16</td>
<td>34.85</td>
<td>0.07771</td>
</tr>
<tr>
<td>6.4583 (8/6)</td>
<td>1.02</td>
<td>43.088</td>
<td>0.29274</td>
<td>92.00</td>
<td>1.32</td>
<td>34.79</td>
<td>0.08289</td>
</tr>
</tbody>
</table>

Table 34. I/O Bound--Loss Mixed $Q_3/Q_4$--Behind Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMSXZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7917 (5/4)</td>
<td>1.10</td>
<td>971.130</td>
<td>0.15506</td>
<td>13.50</td>
<td>1.44</td>
<td>37.58</td>
<td>0.29359</td>
</tr>
<tr>
<td>5.0417 (6/4)</td>
<td>1.04</td>
<td>111.669</td>
<td>0.81120</td>
<td>89.90</td>
<td>1.36</td>
<td>35.68</td>
<td>0.08369</td>
</tr>
<tr>
<td>5.9583 (7/5)</td>
<td>1.11</td>
<td>80.391</td>
<td>0.54435</td>
<td>92.60</td>
<td>1.48</td>
<td>38.07</td>
<td>0.08030</td>
</tr>
<tr>
<td>6.7083 (8/6)</td>
<td>1.10</td>
<td>75.844</td>
<td>0.36507</td>
<td>93.00</td>
<td>1.44</td>
<td>37.80</td>
<td>0.08045</td>
</tr>
</tbody>
</table>
Table 35. Mixture—Loss in $Q_3$—Even Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1667 (5/4)</td>
<td>1.19</td>
<td>342.211</td>
<td>0.21968</td>
<td>45.60</td>
<td>1.72</td>
<td>39.88</td>
<td>0.18141</td>
</tr>
<tr>
<td>5.7083 (6/4)</td>
<td>1.13</td>
<td>561.414</td>
<td>0.04482</td>
<td>10.65</td>
<td>1.84</td>
<td>37.98</td>
<td>0.24171</td>
</tr>
<tr>
<td>6.0000 (7/5)</td>
<td>1.08</td>
<td>317.034</td>
<td>0.16641</td>
<td>49.50</td>
<td>1.52</td>
<td>36.27</td>
<td>0.15680</td>
</tr>
<tr>
<td>6.7083 (8/6)</td>
<td>1.06</td>
<td>293.186</td>
<td>0.18154</td>
<td>53.30</td>
<td>1.64</td>
<td>35.48</td>
<td>0.13553</td>
</tr>
</tbody>
</table>

Table 36. Mixture—Loss in $Q_3$—Behind Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9167 (5/4)</td>
<td>1.03</td>
<td>686.809</td>
<td>0.18078</td>
<td>25.89</td>
<td>1.56</td>
<td>34.63</td>
<td>0.28604</td>
</tr>
<tr>
<td>5.6667 (6/4)</td>
<td>1.07</td>
<td>539.005</td>
<td>0.29889</td>
<td>40.60</td>
<td>1.72</td>
<td>36.06</td>
<td>0.24298</td>
</tr>
<tr>
<td>6.1667 (7/5)</td>
<td>1.06</td>
<td>444.456</td>
<td>0.18048</td>
<td>51.90</td>
<td>1.44</td>
<td>35.47</td>
<td>0.24182</td>
</tr>
<tr>
<td>7.1250 (8/6)</td>
<td>1.08</td>
<td>386.023</td>
<td>0.21551</td>
<td>79.50</td>
<td>1.84</td>
<td>36.24</td>
<td>0.10297</td>
</tr>
</tbody>
</table>

Table 37. Mixture—Loss in $Q_4$—Even Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9267 (5/4)</td>
<td>1.02</td>
<td>959.631</td>
<td>-0.26352</td>
<td>-31.30</td>
<td>1.44</td>
<td>34.16</td>
<td>0.33836</td>
</tr>
<tr>
<td>5.5000 (6/4)</td>
<td>1.02</td>
<td>914.537</td>
<td>-0.13775</td>
<td>-38.90</td>
<td>1.48</td>
<td>34.18</td>
<td>0.09950</td>
</tr>
<tr>
<td>5.7500 (7/5)</td>
<td>1.02</td>
<td>357.830</td>
<td>0.20084</td>
<td>45.80</td>
<td>1.44</td>
<td>34.16</td>
<td>0.16305</td>
</tr>
<tr>
<td>6.7500 (8/6)</td>
<td>1.05</td>
<td>415.910</td>
<td>0.11664</td>
<td>36.85</td>
<td>1.40</td>
<td>35.17</td>
<td>0.16154</td>
</tr>
</tbody>
</table>

Table 38. Mixture—Loss in $Q_4$—Behind Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9167 (5/4)</td>
<td>1.02</td>
<td>1242.540</td>
<td>-0.19979</td>
<td>-23.00</td>
<td>1.44</td>
<td>34.16</td>
<td>0.39876</td>
</tr>
<tr>
<td>5.5417 (6/4)</td>
<td>1.02</td>
<td>329.573</td>
<td>0.41032</td>
<td>67.20</td>
<td>1.40</td>
<td>34.34</td>
<td>0.13451</td>
</tr>
<tr>
<td>6.1667 (7/5)</td>
<td>1.02</td>
<td>330.356</td>
<td>0.23782</td>
<td>67.20</td>
<td>1.36</td>
<td>34.31</td>
<td>0.14050</td>
</tr>
<tr>
<td>7.0833 (8/6)</td>
<td>1.02</td>
<td>412.044</td>
<td>0.16084</td>
<td>58.40</td>
<td>1.64</td>
<td>34.36</td>
<td>0.15338</td>
</tr>
</tbody>
</table>
### Table 39. Mixture—Loss Mixed $Q_3/Q_4$—Even Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMAXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4167(5/4)</td>
<td>1.05</td>
<td>269.156</td>
<td>0.47340</td>
<td>60.30</td>
<td>1.36</td>
<td>35.14</td>
<td>0.11003</td>
</tr>
<tr>
<td>5.0000(6/4)</td>
<td>1.04</td>
<td>557.789</td>
<td>0.09066</td>
<td>16.10</td>
<td>1.32</td>
<td>35.04</td>
<td>0.20504</td>
</tr>
<tr>
<td>5.6667(7/5)</td>
<td>1.01</td>
<td>72.030</td>
<td>0.43053</td>
<td>89.40</td>
<td>1.28</td>
<td>34.02</td>
<td>0.06606</td>
</tr>
<tr>
<td>6.2917(8/6)</td>
<td>1.01</td>
<td>44.798</td>
<td>0.33418</td>
<td>93.50</td>
<td>1.24</td>
<td>34.05</td>
<td>0.63600</td>
</tr>
</tbody>
</table>

### Table 40. Mixture—Loss Mixed $Q_3/Q_4$—Behind Arrivals

<table>
<thead>
<tr>
<th>$Q_1/Q_2$ Size</th>
<th>SG</th>
<th>TL</th>
<th>EFF</th>
<th>%IPRV</th>
<th>AVMAXSZ</th>
<th>TP</th>
<th>AVSYT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7917(5/4)</td>
<td>1.07</td>
<td>967.840</td>
<td>0.13432</td>
<td>13.48</td>
<td>1.40</td>
<td>35.81</td>
<td>0.29552</td>
</tr>
<tr>
<td>5.2500(6/4)</td>
<td>1.10</td>
<td>954.637</td>
<td>0.10984</td>
<td>14.40</td>
<td>1.52</td>
<td>37.02</td>
<td>0.29252</td>
</tr>
<tr>
<td>5.8833(7/5)</td>
<td>1.02</td>
<td>67.709</td>
<td>0.54570</td>
<td>93.70</td>
<td>1.36</td>
<td>34.31</td>
<td>0.09231</td>
</tr>
<tr>
<td>6.6250(8/6)</td>
<td>1.01</td>
<td>59.515</td>
<td>0.39825</td>
<td>93.70</td>
<td>1.32</td>
<td>33.80</td>
<td>0.05066</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY

Literature Cited


Other References


Lowe, Thomas C., "Analysis of Boolean Program Models for Time
Shared Paged Environments," Communication of the Association of Com-
puting Machinery, XII (April 1969), 199-205.

McGee, W. C., "On Dynamic Program Relocation," IBM System
Journal, IV (1965), 184-190.


Manacher, G. K., "Production and Stabilization of Real Time Task
Schedules," Journal of the Association of Computing Machinery, XIV

Martin, Francis F., Computer Modeling and Simulation, New York:

Meadow, Charles T., Man-Machine Communication, New York: Wiley
Interscience, 1970.

Meinstein, L. S., "RCA's Time Sharing Operating System," Comparative
Operating Systems--A Symposium, 1-10.


Moore, J. M., "An n Job, One Machine Sequencing Algorithm for
Minimizing the Number of Late Jobs," Management Science, XV (September

Myers, Charles A., The Impact of Computers on Management,
Cambridge, Massachusetts: The Massachusetts Institute of Technology

Nielsen, N. R., "The Simulation of Time-Sharing Systems," Com-
munications of the Association of Computing Machinery, X (July 1967),
379-412.

Nemhauser, G. L., Introduction to Dynamic Programming, New York:

Ohlgren, Earl, "6000 SCOPE Operating Systems," Comparative

Oppenheimer, G., and Weizen, N., "Resource Met for a Medium Scale
Time Sharing Operating System," Communications of the Association of
Computing Machinery, XI (May 1968), 313-322.


VITA

John M. Hoffman was born October 23, 1942, in Atlanta, Georgia. In 1960 he was graduated from Slidell High School, Slidell, Louisiana, as valedictorian. As a result of his scholastic achievements, Mr. Hoffman was able to obtain a scholarship which provided most of his support while attending Louisiana Tech University. Mr. Hoffman's major fields of study included mathematics and physics. He was graduated Magna Cum Laude in 1964 with a Bachelor of Science in Liberal Arts.

While at Louisiana Tech, Mr. Hoffman participated in the AFROTC program and was awarded a commission in the U. S. Air Force in 1964. His first assignment was to the Air Force Institute of Technology (AFIT) to attend the Georgia Institute of Technology to obtain a Master of Science in Information Science. Upon completion of this degree in 1965, Mr. Hoffman was assigned as the Data Automation Officer of the 32nd Fighter Interceptor Squadron, Camp New Amsterdam, The Netherlands. While in this assignment, he was awarded the Air Force Commendation Medal for his work in developing computer software systems to be used Air Force wide.

In the summer of 1968, Mr. Hoffman returned to the United States for another AFIT assignment. Upon completion of this degree he will be assigned to the Electronics System Division of the Air Force Systems Command, Hanscon Field, Massachusetts.