NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

SCHEDULING IN MULTIPLE PROCESSOR REAL-TIME SYSTEMS

by
Jean-Pierre Lachance, CD, B.Eng.

a thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Masters of Applied Science

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa
February 1991

ISBN 0-315-68070-9

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this thesis.

I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Jean-Pierre Lachance

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Jean-Pierre Lachance

The University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign below and give address and date.

Abstract

In most general terms, Real-Time Systems (RTS) scheduling may be divided into two main categories: static and dynamic scheduling. In static scheduling, the characteristics of the environment are assumed to be known a priori, and; therefore, the sequence in which these activities become ready to run can be determined off-line. Although static systems have low run-time cost, they are quite inflexible and can not adapt to a changing environment. When new work units are added to a static system, the schedule for the entire system must be recalculated.

In contrast to static scheduling where the schedules are generated off-line (before actual RTS operation); dynamic scheduling assigns the idle processors to the ready to run work units during RTS operation, in an on-line manner, according to some measure of urgency and criticalness.

The objective of this thesis is to investigate the effects of different dynamic scheduling heuristic rules on the design of RTS which allow for the execution of complex responses. Complex responses are characterized by the concurrent nature of activity execution. RTS design will be done through the use of a simulator which allows for the selection of the external event characteristics, definition of the RTS (processes, due dates, activity-to-resource allocations with activity execution times) and selection of different activity scheduling schemes (first-come first-served, earliest due date, shortest activity first, etc....). Statistics, such as response execution times, waiting times of ready-to-execute activities, percentage of responses that are tardy and resource utilization, are gathered and provided to the designer. These statistics allow the designer to refine the design, by changing system parameters, until satisfactory RTS operation is achieved. In this study, the RTS will be implemented on a centrally coordinated multiple processor system topology.

Acknowlegements

The planning and preparation of this thesis was a lengthy process. I would like to thank Dr. Moshe Krieger for his ideas throughout the preparation of this thesis. I would also like to thank Dr. Aziz Chikhani and Dr. Doug Smith, both Professors of Electrical and Computer Engineering at the Royal Military College of Canada in Kingston Ontario, for their help in helping me prepare for my thesis defence. Special thanks are extended to my fellow graduate students who have made my stay at Ottawa University an enjoyable one. Financial support was gratefully received by the Canadian Department of National Defence (DND).

I would also like to take this opportunity to express my gratitude to my wife Sylvie for her encouragement and support.

TABLE OF CONTENTS

TABLE OF CONTENTS [continued]

LIST OF FIGURES

LIST OF FIGURES [continued]

CHAPTER 1

INTRODUCTION

This thesis investigates the effects of dynamically scheduling executable work units in Real–Time Computer Systems so that the various responses may meet their time constraints. Chapter 1 introduces the main problems that led to the investigation presented in this thesis.

The first section describes the general characteristics of Real–Time Systems. This is followed by a discussion of the importance of scheduling in Real–Time Systems. Then, after stating the thesis objectives, a short outline of the main chapters is given.

1.1 Real–Time Systems

Real–Time Systems (RTS) differ from conventional computing systems in two major ways. First, in RTS, time represents a resource that has to be managed, in contrast to conventional computer systems where time is not part of the abstraction in which problems are expressed [GENT88]. Second, RTS require interactions with the outside world (the environment). The behavior of the environment is beyond the programmer's control. There are many examples of RTS, these include flexible assembly systems, medical monitoring systems, military command and control systems, avionic systems and others.

The proper operation of a RTS depends both on the logical result of computations and also on the time at which the results are produced. As an example, once the control software of an avionic system is initiated, it must not take arbitrarily long to execute. Furthermore, the program must keep pace with the controlled process. The controlled process therefore imposes timing constraints on its control program. RTS may be divided into two classes: **hard** and **soft** RTS. Hard RTS are those whose deadlines must absolutely be met. If a hard deadline is missed then the tardiness of the response will cause a critical system error. Soft RTS allow for some deadlines to be missed with only a degradation in system performance [JENS85] [STAN88]. Figure 1–1 (a) shows a hard RTS in which there is value attached to system computations only if they are carried out prior to a fixed deadline. Hard RTS include nuclear power plants, process control systems and

missile guidance systems. Figure 1–1 (b) shows a soft RTS in which a constant value is attached to system computations carried out prior to their deadlines; followed by a degradation after the deadline. An example of a soft RTS is an aircraft's navigational system where late processing of some radar readings may give less accurate but acceptable results. An alternate definition of soft RTS states that failure to respond to a small number of events may be acceptable. In the case of certain avionic systems, missing the processing of a few radar readings may be acceptable.

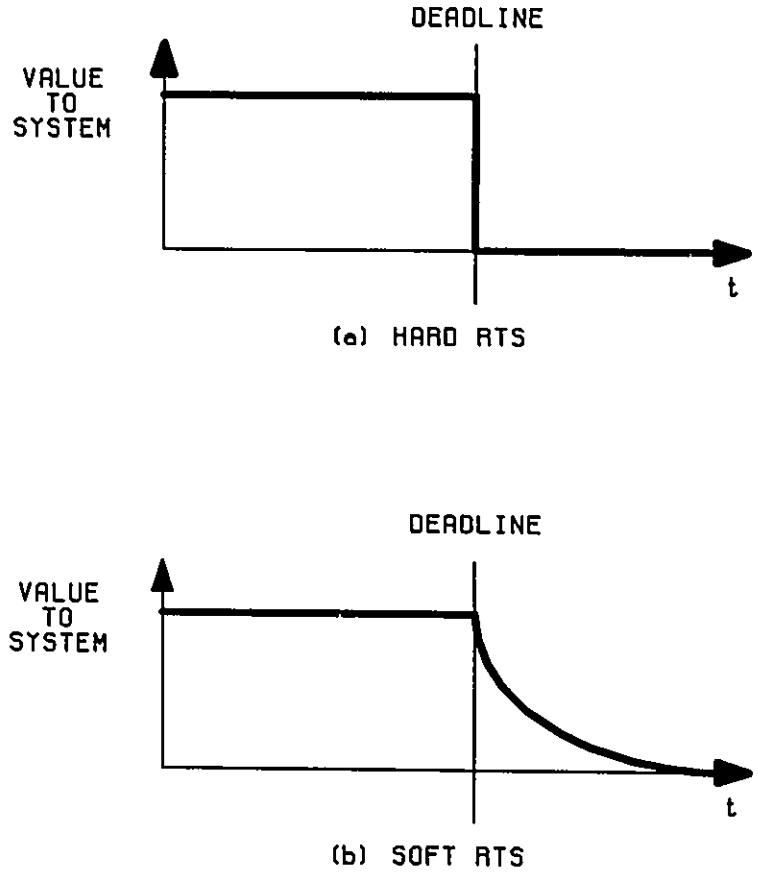


Figure 1–1 Computational Values Attached to Hard and Soft RTS

In RTS, one assumes two distinct entities: first, the **environment,** which is typically a complex dynamic system that includes a multiplicity of devices that have to be controlled; and second, the **controlling entity,** generally a computer–based system consisting of hardware, software, and special purpose interfaces that

coordinate the operation of the elements of the environment in a prescribed way. In RTS, the computer is not the master entity. Rather, it acts as a slave to the environment which prescribes the types of responses that must be generated by the computer. The RTS must react to the occurrence of sensor detected environmental events and must respond to them at appropriate times via actuators. RTS are also known as **embedded** systems. Figure 1–2 shows the interactions between the environment and the controlling entity.

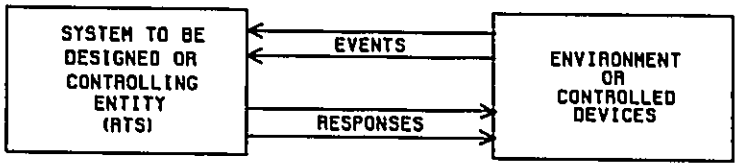


Figure 1–2 Interactions between the Environment and the Controlling Entity

RTS can be abstracted as a set of processes invoked in response to external events. An external event can be anything from the detection of a car's low oil pressure to a computer peripheral interrupt. In this thesis, RTS will be modelled using the **Process Activity Language** (PAL), a high–level general purpose executable specification language. As indicated by figure 1–3, processes (P1, P2, P3, P4, ...) can be partitioned into sub–processes (S1, S2, ...), which in turn can be partitioned, until the lowest level of interest to the system designer is attained. The entities at this level are called activities (A1, A2, ...) and are defined as a meaningful sequence of operations (or actions) that can be executed, once started, from beginning to end without having to wait for data from other activities. The environment may be modified as a result of executing activities.

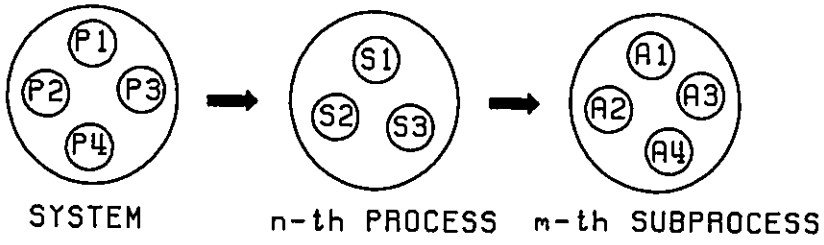


Figure 1–3 System Partitioning

In this study, it is assumed that an **event response** (a process) can be decomposed into a number of activities which must be executed in a given precedence relation. The precedence relation in which the activities have to be executed is determined by the data dependencies that exist between the activities of a process. In any process, the exact dependencies among the constituent activities define the amount of concurrency possible in the process. In addition, the response may have resource requirements which must be met in order to ensure its correct operation. Responses can be categorized as being either **simple** or **complex**. A simple response may either be a single activity or composed of a set of activities that have a purely sequential relationship. Complex responses are characterized by the concurrent nature of activity execution.

Processes can also be classified as being **sporadic** or **periodic** depending on the type of initiating events. Sporadic processes typically arise in control-oriented systems and in operator initiated actions. Sporadic processes are also known as aperiodic processes. In sporadic processes, one or more of the following time constraints may be imposed: **acceptance time** (r_i), **deadline** (d_i), **start response time** (s_i) and **response duration** (t_i). Figure 1-4 shows these sporadic process timing parameters.

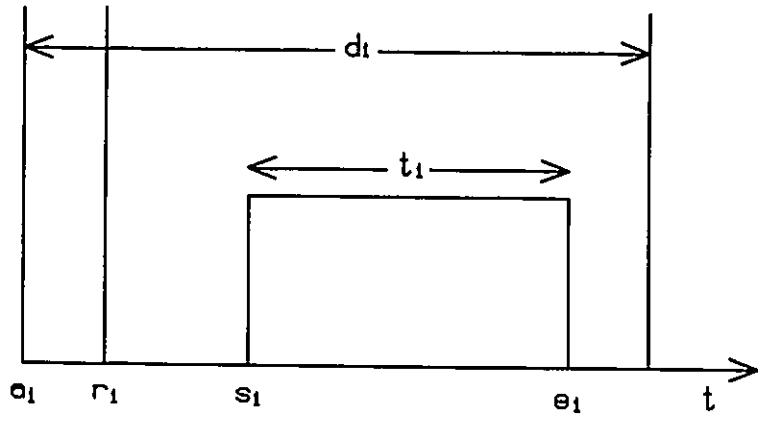


Figure 1-4 Sporadic Process Timing Parameters

The acceptance time represents the point in time by which a sporadic event arrival must be accepted. If the event arrival is accepted then the sporadic process may be executed. If the event arrival has not been accepted by time r_i then the sporadic process will not be executed. The acceptance time is always greater than or

equal to the event arrival time. Deadline d_i is the point in time by which the process execution must be completed. Start response time s_i is defined as the point in time when process execution starts and response duration t_i is the execution time upper bound. Furthermore, e_i is defined as the point in time by which process execution completes. Depending on the system, the time constraints may be hard or soft.

Periodic processes are event responses which must be executed once per period [CHEN88]. Periodic processes are often used in control and signal processing applications to process sensor data and update the current status of the RTS. Periodic processes usually have hard deadlines, but in some applications the deadlines may be soft. Periodic processes are also characterized by one or more of the following time constraints: its **period** (p), its **deadline** (d_i), its **start response time** (s_i) and its **response duration** (t_i). Figure 1–5 shows these periodic process timing parameters.

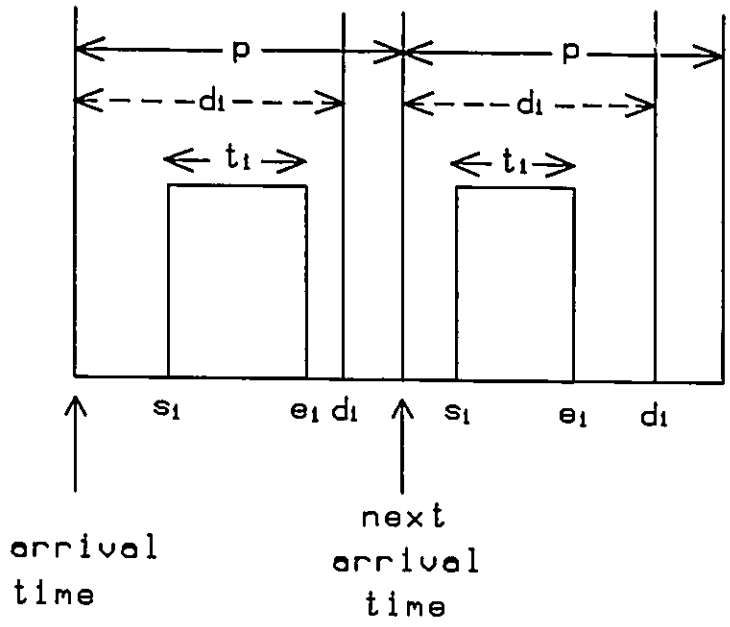


Figure 1–5 Periodic Process Timing Parameters

A periodic process has a regular interarrival time equal to its period. In some cases, the process deadline may coincide with the end of its current period; while in

other situations, the deadline may be set to a point in time before the end of the current period. The remainder of the periodic process timing parameters have the same definition as their counterparts in sporadic processes. Deadline d_i is shown between dotted lines in figure 1-5 to represent the fact that its location in time may vary between process event arrival time and the end of the current periodic process period. Response duration may vary for different instances of the same periodic process as shown by the different lengths of t_is in figure 1-5. When a periodic process completes prior to its deadline, the following relations are assumed to hold: $t_i \leq d_i \leq p$.

In many RTS, the time constraints imposed on the response can be quite complex. For example, consider the armament computer of a small bomber aircraft that has to release two 500-pound bombs. To hit the target, one has to release the bombs at the correct time with as little jitter as possible. To maximize the kill ratio, the bombs have to be released as close in time as possible. On the other hand, if there is not enough time between bomb releases they may collide and explode in or around the bomb bay area of the aircraft.

In order to be able to meet the required process deadlines for periodic and sporadic processes, real concurrency is required in many RTS in order to provide responses in parallel as opposed to apparent concurrency in which responses are provided in an interleaved fashion. This real concurrency can only be achieved through the use of multiple processor systems. When there are different kinds of external events to which responses are required within specified times, it is far easier to dedicate processors to each of the different kinds of external events rather than having to analyse response interactions within a common processor module. Also, the fact that RTS applications are often large and diversified gives rise to another motivation for the use of multiple processors: functional processor specialization. Rather than trying to do the whole problem on one processor, or even on multiple instances of the same processor type, it can be more effective to use specialized processors (i.e. often involving different requirements) and address the question of how to interface them smoothly [GENT88]. Furthermore, RTS require multiple processor implementation because they are part of systems with a very long life cycle which means that these systems have to be modular to allow for easy modular incremental growth as the required computing power of the system grows. Finally, because they control real life applications, RTS must still be extremely reliable. True reliability can only be achieved through the redundancy and reconfigurability provided by multiple processors.

From the above discussion, we can see two major aspects of RTS: first, because of the complexity and the nature of the interactions, at any given time, there may be a number of active processes involving activities that have to be executed on the available system resources; second, to meet all the requirements imposed, RTS must include a multiplicity of assignable resources that can execute the various activities.

1.2 Scheduling in Real–Time Systems

To meet the timing constraints of the RTS, a scheduler must coordinate the use of all RTS resources to meet the following objectives: first, guarantee that all processes with hard timing constraints will always meet their deadlines; and second, provide fast average response times for processes with soft deadlines. When designing a RTS, the application specialist can implement it either as a multitasking or as a multiactivity system. In multitasking systems, related activities are grouped into tasks and the tasks are the schedulable work units; while in multiactivity systems, the activities are the schedulable work units.

In most general terms, scheduling may be divided into two main categories: **static** and **dynamic** scheduling. In static scheduling, the characteristics of the environment are assumed to be known a priori, and; therefore, the sequence in which these activities become ready to run can be determined off–line [STAN88]. These ready to run activities can therefore be scheduled so as to optimize some performance measure such as the maximal utilization of system resources, etc... Although static systems have low run–time cost, they are quite inflexible and can not adapt to a changing environment. When new work units are added to a static system, the schedule for the entire system must be recalculated, which is expensive in terms of money and time. Furthermore, in the case of even moderately complex static systems, the computation time required to produce feasible schedules becomes immense.

Static scheduling is directly applicable to RTS with only periodic processes. A well–understood scheduling algorithm for guaranteeing the hard deadlines of periodic processes is the **rate monotonic** algorithm produced by Liu and Layland [LIUL73]. In this algorithm, fixed priorities are assigned to processes based on the rate of their requests (i.e. a process with a short period is given higher priority than another process with a longer period). The algorithm guarantees that **n** periodic single task processes can always be guaranteed to meet their deadlines if the total resource

utilization is less than 69% for large **n**.

In addition to scheduling periodic processes, one must also be able to schedule sporadic processes so that both types of processes meet their hard time constraints. One approach used for servicing sporadic processes is that of **polling processes**. This polling technique consists of creating a periodic process for servicing sporadic requests. The period of this periodic process is set to the estimated minimum interarrival time between sporadic event arrivals. At regular intervals, the polling process is started and services any pending sporadic requests. If no requests are pending, the polled periodic process suspends itself until the next period and the time is used by other periodic processes [GAGN89] [SPRU89]. As for the execution of soft deadline sporadic processes, these processes are normally executed when the resource is idle (i.e. as their execution is not as critical as for process with hard deadlines, they have lower priority).

The term "static scheduling" is often also used in RTS where one assigns fixed priorities to the various ready to run work units and then schedules them accordingly. There are two major drawbacks with this approach. First, it is not clear how one translates the real-time constraints and the relative importance of a task or activity into priorities so that one may allocate system resources appropriately. Second, work unit (activity or task) execution times are, in general, stochastic. This means that the worst case execution times of a work unit may be much higher than its average execution time [STAN88]. Static RTS must be built so that the time critical processes will meet their timing constraints especially in overloaded situations. These requirements often result in the creation of a highly underutilized RTS.

In contrast to static scheduling where the schedules are generated off-line (before actual RTS operation); dynamic scheduling assigns the idle processors to the ready to run work units during RTS operation (i.e. in an on-line manner) according to some measure of urgency and criticalness. Dynamic scheduling is applied only to RTS with soft constraints. Some of the measures of urgency that are used are heuristics such as the earliest due date or least laxity first dispatching rules.

There are many drawbacks with the current trends associated to the research in RTS scheduling. First, the various scheduling studies reported in the literature assume single work unit responses. No parallelism (concurrent work unit execution) is allowed. This implies that all work unit execution must be serialized and

therefore time constraints are harder to meet. Second, work unit execution is normally restricted to one resource. Most instances of the RTS scheduling problem do not consider the possibility of either allocating a work unit to more than one resource or do not consider the case where multiple resources are required to execute a given work unit. Third, overheads associated with the preemption of a work unit on one resource and resumption of execution of the remainder of the same work unit on another resource are not taken into account in most scheduling models. Fourth, work unit execution time is normally assumed to be deterministic in most scheduling models while it is known that in most applications it is stochastic.

The problem of RTS scheduling is further exacerbated by the fact that the derivation of the physical model of the environment is almost always incomplete and never completely accurate [GENT85]. This is caused by two main factors. First, the controlled entity is not easily described in terms of analytical models or mathematical equations because the environment changes in time as a function of complex interactions of timed discrete events [HO89] [GARZ86]. The second difficulty lies in the fact that the RTS only has access to the environment through a finite number of sensors and actuators. Because of this, the RTS can not be in complete control and therefore can not provide a perfect solution that could account for all environmental states. While it is not possible to do a complete analysis of any but the simplest RTS, a feel for the system behavior may be obtained through the use of simulators and prototypes.

1.3 Thesis Objectives

The objectives of this thesis were threefold:

1. provide background and indicate pertinent results in scheduling;

2. discuss the design of Real-Time Systems using the Process Activitiy Language and its associated toolset; and

3. investigate the effects of different dynamic scheduling heuristic rules on the design of Real-Time Systems which allow for the execution of complex responses.

1.4 Thesis Outline

Chapter 2 presents the scheduling background required for this thesis. First, to allow for the discussion of results from different specialization areas that have considered the scheduling problem, a common terminology is introduced. The next section discusses what was learnt in Operations Research scheduling. This is followed by a discussion of scheduling in general purpose computer systems. The next section discusses the effects of scheduling in RTS. The chapter ends with a few concluding remarks.

Chapter 3 presents the design of RTS using the Process Activity Language (PAL) and its associated toolset. The chapter begins by presenting the elements of PAL. This is followed by a discussion of multiactivity systems. The chapter ends by presenting the different steps that may be used in the design of RTS using PAL and its associated toolset.

Chapter 4 presents the RTS simulation environment and some results for a medium-sized example. The chapter begins with a description of the general organization of the PAL simulator. A medium-sized RTS application and the parameters that were varied during simulation is then presented. Some results are then introduced for this example showing the effects of various scheduling techniques. The chapter ends with a few concluding remarks.

Chapter 5 presents the thesis conclusion and suggests recommendations for further work.

CHAPTER 2

SCHEDULING BACKGROUND

The purpose of this chapter is to provide background and indicate pertinent results in scheduling. Section 2.1 introduces a general scheduling terminology that allows us to present results from both Operations Research (OR) and computer systems. Section 2.2 presents a brief survey of scheduling results in OR, while sections 2.3 and 2.4 consider the scheduling problem from the point of view of general purpose and real-time computer systems respectively. Section 2.5 presents a few concluding remarks about chapter 2.

2.1 Scheduling Terminology

Scheduling is the orderly assignment of resources to units of work. Scheduling is required in all systems where, at any given time, there can be a number of ready to run **work units** and a number of assignable **resources** that can execute them. In most general terms, scheduling involves two elements: **sequencing** and **allocation.** Sequencing represents the order in which work units are presented to resources. Allocation deals with which resource can execute a specific work unit.

In the terminology associated with OR, work units represent the various **operations** that are associated with a given **job** and the resources are the different **machines** capable of executing these operations. In RTS, the work units represent the various components (tasks or activities) of a response and the resources are the different elements of the computer system (processors, memory, I/O, etc....) required to execute these tasks or activities. The nomenclature that will be used in this thesis is summarized in figure 2-1 shown at the top of the next page.

The prime components of any scheduling model are the **system processes**, the **system resources** and the **scheduling criteria.** Each of these components will now briefly be discussed.

GENERIC TERMINOLOGY	OPERATIONS RESEARCH TERMINOLOGY	REAL-TIME SYSTEMS TERMINOLOGY
PROCESS	JOB	RESPONSE/PROCESS
WORK UNIT	OPERATION	ACTIVITY/TASK
RESOURCE	MACHINE	PROCESSORS, ETC...

Figure 2-1 Scheduling Nomenclature

2.1.1 System Processes

As indicated in chapter 1, a system is made up of a number of processes with the following characteristics: first, each process can either be categorized as sporadic or periodic depending upon the type of initiating event; second, each process has a given response duration (t_i) which may be data dependent; third, processes may be partitioned into work units and depending on the process, the constituent work units may be executed independently or in some precedence relation; fourth, each process may have a well defined deadline (d_i); and additionally or alternately, each process may have a maximum start-up time (m_i). Figure 2-2 shows the timing parameters of a process i that is partitioned into three work units (w1, w2 and w3) which are executed in strict sequence. In this figure, the origin (a_i) corresponds to the start of a cycle in the case of periodic processes or to the arrival of an event in the case of sporadic processes. Furthermore, it is assumed that all events are accepted and there is no appreciable delay between event arrival and acceptance by the system.

Quantitative measures for evaluating schedules are usually a function of process completion times and/or due dates. Four such quantities are: process **lateness**, **tardiness**, **laxity** and **flowtime**. Process lateness (l_i) is defined as being the difference between process end time and its due date ($l_i = e_i - d_i$) and is negative if the process completes before its due date. Process tardiness ($tard_i$) is defined as process lateness if a process fails to meet its due date; or zero if a process completes prior to its deadline ($tard_i = \max(0, l_i)$). Process laxity (lax_i) is equal to the difference between process due date and the work remaining on the process. Work

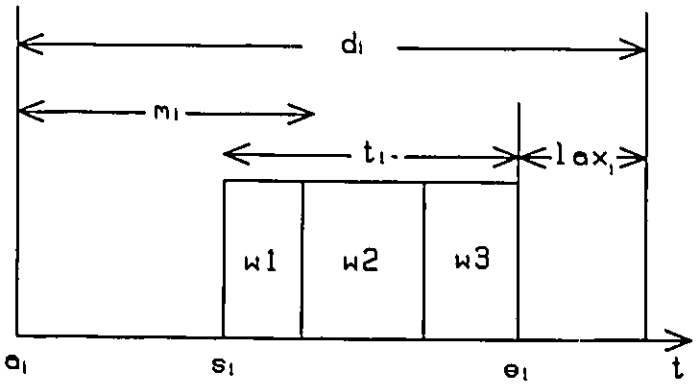


Figure 2–2 Process Timing Parameters

remaining on a process is the sum of the execution times of the work units that have not yet been executed. Process flowtime (f_i) is equal to the amount of time process i spends in the system: $f_i = e_i - a_i$. In some systems, it is not the completion time of a process that is critical but the time by which it is initiated. In this case, the quantitative measure may be process **delay**, the time by which a process initiation is delayed beyond a maximum start-up time.

2.1.2 System Resources

In terms of resources, systems may be divided into two categories: **single** resource or **multiple** resource systems. In single resource systems, the scheduling problem is restricted to the sequencing of process work units on the single resource. In multiple resource systems, concurrent execution of work units is allowed. The multiple resources can either be **homogeneous** or **heterogeneous**. Heterogeneous resources differ in their functionality, speed, reliability, or in other aspects, while homogeneous resources do not. Another aspect of system resources is the allocation of work units to resources. That is, if each resource can execute all of the work units or only a subset of them. In heterogeneous resources, work units are generally allocated to resources according to their specialized need. The execution of the same work unit on two different heterogeneous resources may result in different execution times. As a rule, in multiple homogeneous resource systems, all the work units can be executed on each of the resources and the execution of the same work unit on different resources requires the same processing time.

2.1.3 Scheduling Criteria

The final element of the scheduling model is the choice of scheduling criteria to be optimized. Five commonly used criteria are [GONZ77]: maximize resource utilization; maximize the number of processes that meet their deadlines; minimize average process lateness; minimize maximum process lateness; and minimize overall completion times (i.e. the time by which all processes terminate). To meet one or more of these scheduling criteria, one has to carry out a proper allocation of resources and sequencing of work units. In terms of sequencing, at any given time, one has to determine the urgency of executing a particular work unit. Selection of a particular ready to run work unit for execution may be done using a **fixed priority scheme**, some **optimization technique** or some **heuristic dispatching rule** such as earliest due date or least laxity first.

Another important decision is whether or not to allow a currently active work unit to run until completion. **Preemption** refers to the suspension of one work unit to permit the execution of another work unit. In **non−preemptive** scheduling, once a work unit has begun execution on a resource it can not be preempted by another work unit and is left to run until completion. There may be a significant switchover penalty associated with preemption. In OR, the present operation has to be stopped, the component replaced and a new operation has to be initiated. In the case of computer systems, every preemption requires a context switch which may be expensive in terms of overhead and memory requirements. It is generally agreed that preemptive scheduling is preferable if the scheduling criteria to be optimized is the minimization of the number of processes that are tardy [GONZ77]. A simple example where preemption maximizes the number of processes meeting their deadlines is illustrated in figure 2−3.

In this example, if no preemption is allowed, process 2 can not complete execution by its deadline. When preemption is allowed, both processes can be completed before their deadlines.

2.2 Scheduling in Operations Research

In this section, some of the results obtained in the field of OR will be presented for single resource systems, multiple resource systems and job shop systems. In single and multiple resource systems, the problem is restricted to the sequencing of single work unit processes (single operation jobs) with or without given precedence

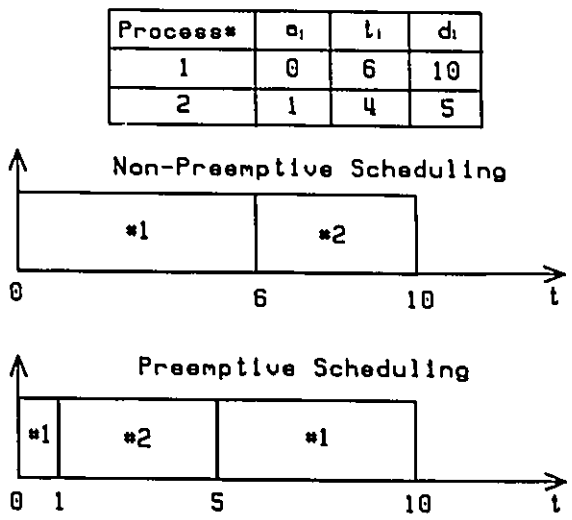


Figure 2-3 Example of Preemptive vs Non-Preemptive Scheduling

relations between processes on one or more resources. In job shop systems, each process (job) consists of m work units (operations) that have to be executed in a given order on m different resources (machines).

2.2.1 Single Resource

The scheduling of single resource systems has been extensively investigated in OR. The simplest single resource scheduling problem is the static non-preemptive scheduling that assumes that there exists a static set of n independent processes available at time t=0; that each process has a well defined response duration (t_i) and deadline (d_i); that once a process has begun execution, it must be executed to completion.

The sequence in which processes are presented to the single resource will greatly affect the various scheduling criteria for a particular system. Figure 2-4 demonstrates that different sequences produce different results for various scheduling criteria. It is easy to show that for these systems, one can obtain minimum mean completion time (i.e. flowtime in OR) and minimum mean lateness by sequencing the processes in nondecreasing order of their response duration; this is called shortest processing time or SPT sequencing (figure 2-4(b)). Sequencing the processes according to their deadlines, which is called earliest due date or EDD

Process i	t_i	d_i
1	2	7
2	4	5
3	5	12
4	1	4
5	3	6
6	6	8

(a) Process Specifications

Process i	s_i	e_i	d_i	l_i	$tard_i$	
4	0	1	4	-3	0	mean completion time=9.33
1	1	3	7	-4	0	maximum lateness =13
5	3	6	6	0	0	maximum tardiness =13
2	6	10	5	5	5	mean lateness=2.33
3	10	15	12	3	3	mean tardiness=3.5
6	15	21	8	13	13	number of tardy processes=3

(b) Shortest Processing Time Sequencing

Process i	s_i	e_i	d_i	l_i	$tard_i$	
4	0	1	4	-3	0	mean completion time =10.16
2	1	5	5	0	0	maximum lateness=9
5	5	8	6	2	2	maximum tardiness=9
1	8	10	7	3	3	mean lateness=3.16
6	10	16	8	8	8	mean tardiness=3.66
3	16	21	12	9	9	number of tardy processes =4

(c) Earliest Due Date Sequencing

Process i	s_i	e_i	d_i	l_i	$tard_i$	
4	0	1	4	-3	0	mean completion time=9.66
5	1	4	6	-2	0	maximum lateness =13
1	4	6	7	-1	0	maximum tardiness =13
3	6	11	12	-1	0	mean lateness=2.66
2	11	15	5	10	10	mean tardiness=3.83
6	15	21	8	13	13	number of tardy processes=2

(d) Moore Sequencing

Process i	s_i	e_i	d_i	l_i	$tard_i$	
6	0	6	8	-2	0	mean completion time=15.66
3	6	11	12	-1	0	maximum lateness =17
2	11	15	5	10	10	maximum tardiness =17
5	15	18	6	12	12	mean lateness=8.16
1	18	20	7	13	13	mean tardiness=8.66
4	20	21	4	17	17	number of tardy processes=4

(e) Longest Processing Time Sequencing

Process i	s_i	e_i	d_i	l_i	$tard_i$	
3	0	5	12	-7	0	mean completion time=14.33
6	5	11	8	3	3	maximum lateness =17
1	11	13	7	6	6	maximum tardiness =17
5	13	16	6	10	10	mean lateness=7.33
2	16	20	5	15	15	mean tardiness=8.5
4	20	21	4	17	17	number of tardy processes=5

(f) Latest Due Date Sequencing

Process i	s_i	e_i	d_i	l_i	$tard_i$	
4	0	1	4	-3	0	mean completion time=9.83
2	1	5	5	0	0	maximum lateness =13
1	5	7	7	0	0	maximum tardiness =13
5	7	10	6	4	4	mean lateness=2.83
3	10	15	12	3	3	mean tardiness=3.33
6	15	21	8	13	13	number of tardy processes=3

(g) Sequence Determined using a Branch and Bound Algorithm

Figure 2–4 Single Resource Sequencing Example

sequencing, minimizes maximal lateness and tardiness (figure 2-4(c)). It was shown by Moore that starting with earliest due date sequencing and reordering the processes in a prescribed way minimizes the number of tardy processes (figure 2-4(d)).

Another important scheduling criteria, in such systems, is mean tardiness. The minimization of mean tardiness requires a complete search of all possible sequences. There is no simple algorithm. For the n process problem, this would involve the investigation of n! sequences which becomes computationally time consuming as n increases. To reduce the work, one may use some of the better known combinatorial optimization techniques approaches such as branch and bound, dynamic optimization, etc... Figure 2-4(g) shows the sequence that produces the minimum mean tardiness for this problem. This sequence was obtained by using a branch and bound technique. Figure 2-4 also includes longest processing time (LPT) sequencing (figure 2-4(e)) and latest due date (LDD) sequencing (figure 2-4 (f)) to indicate how some sequences can produce poor results for the different scheduling criteria discussed here. Figure 2-5 has been included to show that the use of different sequences, for a simple problem like that of figure 2-4, will yield a wide range of values for different scheduling criteria.

	Minimum Value	Maximum Value
Mean Completion Time	9.33	15.06
Maximum Lateness	9	17
Maximum Tardiness	9	17
Mean Lateness	2.33	8.16
Mean Tardiness	3.33	8.66
Number of Tardy Processes	2	5

Figure 2-5 Range of Values for Different Scheduling Schemes

Some of the previous scheduling criteria schemes may be expanded; in particular, in the case of scheduling preemptable processes that arrive dynamically

over time. For example, to minimize maximal tardiness, at each process completion or at the arrival of a new process assign the resource to the available process with the earliest deadline. To minimize the average process computation time, at each process completion time or when a new process arrives assign the resource to the available process with the shortest remaining processing time [HORN74].

For other scheduling criteria or if the processes are dependent, there is no simple extension to these results and, as indicated by Garey and Johnson [GARE76], deriving an optimal sequence becomes NP–complete. * Therefore except for very simple scheduling criteria and a small number of processes, even in the simple case of single resource systems one can not expect to derive optimal sequences dynamically.

2.2.2 Multiple Resources

In OR, multiple resource systems were investigated for two types of jobs: jobs requiring a single machine for their execution and jobs that require multiple machines for their execution. The first class of problems are generally referred to as **parallel machine** problems; the machines can either be identical, each being able to execute all jobs, or they can be different, some being capable to execute only a subset of jobs. The second class of problems has been considered under two headings: **flow shop** problems where the different jobs require the machines in the same order and **job shop** problems where the order in which the machines are required can be different from job to job. Due to the widespread interest in automated manufacturing, the recent work in scheduling on multiple resources is very extensive, therefore we shall only review the basic results for parallel machines and job shop problems.

In the case of the static multiple identical resource problems, it is relatively easy to show that one can extend the minimum mean completion time scheme of the

* An NP–complete problem is defined as a problem for which no known polynomial time algorithm is available to solve it. A polynomial time algorithm is defined as an algorithm whose time complexity is of the order (O(p(n))) for some polynomial function p where n denotes the number of processes to be scheduled [GARE79].

single resource by applying the following two-step approach:

1. construct a shortest processing time (SPT) ordering of all processes; and

2. assign non-preemptively the resource with the least amount of processing already allocated to the next process on the ordered list of processes.

As an example, figure 2-6(a) lists a set of 8 processes to be scheduled. Figure 2-6(b) shows the produced schedule using the SPT rule for three identical resources R1, R2 and R3. The resulting mean completion time becomes (1+5+12+2+7+15+3+9)/8= 6.75.

Another important scheduling criteria for multiple identical resources is minimizing **makespan** (i.e. the minimum time required to complete all jobs). In this case, the following LPT based scheduling algorithm provides an effective heuristic scheduling algorithm:

1. construct an LPT ordering of the jobs; and

2. schedule the jobs in order, each time assigning a job to the machine with the least amount of processing already allocated.

The schedule corresponding to the LPT algorithm is shown in figure 2-6(c). The resulting makespan and mean completion times are 13 and 9.87. The problem with this algorithm is that it does not guarantee minimal makespan. For this problem, there exists a number of schedules with a makespan of 12. An example of such a schedule, obtained by trial and error, is shown in figure 2-6(d).

For the case where preemption is allowed, there exists a simple algorithm developed by McNaughton [MCNA59] which minimizes makespan:

1. select some process to begin on machine 1 at time zero;

2. choose any unscheduled process j and schedule it as early as possible on the same resource. Repeat this step until the resource is occupied beyond M^* (or until all processes are scheduled); and

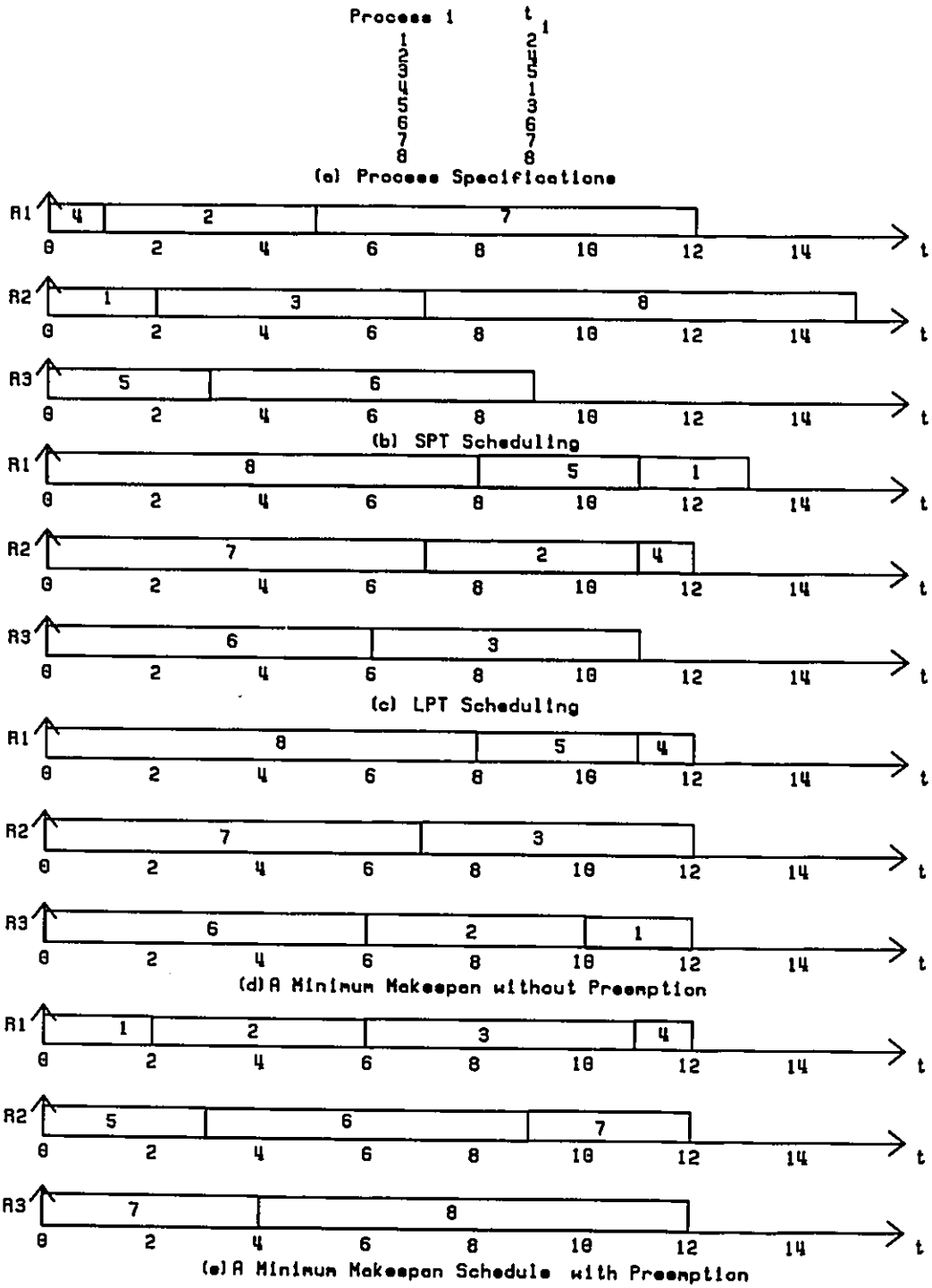


Figure 2–6 A Scheduling Example on Parallel Identical Resources

3. reassign the processing scheduled beyond M* to the next resource instead, starting at time zero. Return to step 2.

where $M^*=1/m\ (\sum_i t_i)$; m=number of resources and

$\sum_i t_i$ = sum of all process completion times.

The schedule corresponding to the McNaughton algorithm is shown in figure 2-6(e). The resulting makespan and mean completion times are 12 and 8.87.

The more general case of scheduling processes with arbitrary precedence constraints and unit computation time on a number of homogeneous resources was shown to be NP-complete for both the preemptive and non-preemptive cases when the scheduling criteria was the minimization of makespan [ULLM75].

Some of the previous scheduling criteria results may be extended; in particular, in the case of scheduling non-preemptable processes that arrive dynamically over time. For example, the algorithms based on the SPT and LPT scheduling rules (i.e. which respectively minimized mean completion time and makespan) may be extended to the dynamic case in order to include random arrivals.

Another important scheduling criteria is determining whether a set of jobs may be scheduled preemptively on a set of parallel machines with different processing speeds such that each job with a given processing requirement may be executed between its start-up time and its due date. Martel developed an algorithm based on the network flow technique to determine whether such a schedule exists [MART82]. Balakrishnan extended Martel's work to consider scheduling of jobs in a manufacturing facility that consisted of special purpose machines, each capable of performing only one particular job, and a group of identical machines that can perform all the required jobs [BALA89].

The last branch of OR scheduling to be examined is the job shop scheduling problem. The job shop scheduling problem is interesting because the processes to be scheduled consist of m work units that have to be executed in a given order on m different resources. The most common form of the job shop problem is characterized by the following assumptions [BAKE74]:

1. there exists a set of n processes available for scheduling (each process consists of m strictly ordered work units);

2. the job shop is composed of m different resources;

3. a given work unit is allocated to a single resource;

4. work unit computation times are known prior to run-time;

5. work unit execution is non-preemptive;

6. a work unit may not begin until its predecessors are completed; and

7. each resource can only execute one work unit at a time.

Figure 2-7 shows a simple job shop example with figures 2-7(a) and 2-7(b) indicating the computation times and the resource requirements of the different work units.

With each work unit one associates a triplet **ijk** to indicate that the j^{th} work unit of process i requires resource k. Figure 2-7(c) presents a **feasible** (one that respects the restrictions of the problem) solution for our simple job shop example. Note that at time t=0, the first work units of each process (112 and 212) can be started simultaneously because they require different resources. However, even though resource 1 is available at t=3, the second work unit of job 1 (121) can not be started until t=5 when the first work unit of job 1 (112) has completed.

There has been a lot of research carried out on the scheduling of static systems. The performance criteria to be optimized were often the minimization of mean completion time and the minimization of the completion times on each resource. Finding an optimal schedule for any realistic problem is computationally excessive as it first requires finding all feasible solutions and next selecting the best solution. Finding a feasible solution in itself is quite complex as it involves satisfying simultaneously three types of constraints:

1. the precedence constraints between work units of a job;

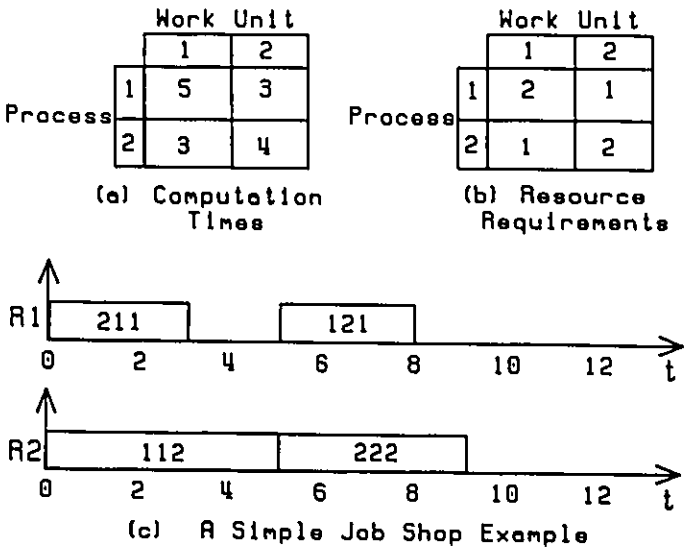


Figure 2-7 A Simple Job Shop Example

2. the resource requirements of the work units; and

3. the fact that each resource can execute only one work unit at a time.

This fact led researchers to the development of algorithms that used heuristics that generated sub-optimal scheduling solutions for large and complex problems. Some examples of the algorithms used include the branch and bound technique and the integer programming approach.

One of the most interesting and widely studied areas of scheduling research is the dynamic job shop model. In the dynamic job shop, jobs arrive at the shop randomly over time so that the shop itself performs like a network of queues. In this context, the scheduling problem is transformed; namely, that at any given time, there may be one or more ready to run work units that can be assigned. For each free resource, there may be one or more ready to run work units that can be assigned to it. Scheduling is generally carried out by means of dispatching decisions: at a time a resource becomes free, a decision must be made as to what it should do next. The following is a list of some of the dispatching rules examined in various job shop scheduling studies:

1. first-come first-served (FCFS): select the work unit that arrived at the queue first. This rule is not oriented towards processes meeting their due dates.

2. shortest processing time (SPT): select the work unit with the minimum computation time. Use of this rule normally results in the best resource utilization;

3. most work remaining (MWKR): select the work unit belonging to the process having the largest amount of processing remaining to be done.

4. least work remaining (LWKR): select the work unit belonging to the process having the smallest amount of processing remaining to be done;

5. most operations remaining (MOPNR): select the work unit that has the most successor work units left to be executed;

6. random: select a work unit at random;

7. earliest due date (EDD): select the work unit that belongs to the process that has the earliest due date.

8. minimum slack time (MST): select the work unit that belongs to the process with the least process laxity.

The scheduling in dynamic job shops has made considerable progress with the use of computer simulation models. Experimentation with computer simulation models has made it possible to compare different dispatching rules and develop greater insight into job shop operation. Although it is impossible to identify any dispatching rule as the best in all circumstances; several rules have been identified as exhibiting good performance in general. In particular, the SPT dispatching rule was found to minimize the number of tardy processes [CONW67] [ELVE73] and mean process completion times [LEGR63] [NANO63] [CONW65] in a number of studies. The only disadvantage in using the SPT rule is that a few long processes will become very late.

Other dispatching rules were found to perform well when the scheduling objective involved meeting process due dates. For example, in a study carried out by Elvers, the LWKR rule performed well [ELVE73]. In another study, the EDD rule provided good performance [ROCH76] while, in another Conway showed that the S/OPN (slack per operation=process laxity/number of operations left in the process) also performed well. The principal advantage of due-date-based rules is often a smaller variance of job lateness and tardiness [BLAC82].

Unfortunately, there are also many drawbacks associated to the studies carried out on job shop scheduling. First, no parallelism or concurrent work unit execution is allowed. This implies that all precedence relations must be serialized and therefore time constraints are harder to meet. Second, work unit execution is normally restricted to one resource. Most instances of the job shop scheduling problem do not consider the possibility of either allocating a work unit to more than one resource or do not consider the case where multiple resources are required to execute a given work unit.

2.3 Scheduling in General Purpose Computer Systems

In general purpose multiprogrammed computer systems, there are many resources that have to be scheduled. Some of these resources include processors, memory and printers. Most of the early work only considered the development of scheduling algorithms for the CPU as it was the most expensive component of the system. These algorithms had two main directions: first, to use multiprocessors to exploit the concurrencies in a single program and second, to increase the performance of the early multiuser systems.

Coffman and Denning [COFF73] present a good survey of the early work in multiprocessor systems. They consider the problem of scheduling processes, that have been partitioned into tasks, with or without precedence constraints, on a number of homogeneous processors. Although most of the early work parallels OR work of the time, Coffman and Denning present results that are more applicable to computer systems. First, they consider the case when all tasks have equal execution times and they show that there is an optimal scheduling algorithm if the tasks have a tree-like precedence relation. This result is based on the work of Hu who presented a labeling mechanism and associated scheduling for dependent tasks whose

precedence relation defines a tree [HU61]. Next, they present a preemptive scheduling algorithm for the general case of tasks with arbitrary precedence relations and the number of processors is restricted to two. The major drawback with these results is that they assume that the task execution times are known prior to run-time. This assumption is only realistic for very simple cases. In general, task execution times are highly data dependent and their value can vary a lot.

More recently, with the development of special multiprocessor systems, interest has shifted to the development of algorithms that fit specific architectures such as hypercube, array processors, systolic arrays, etc. Lately, there has also been some interest in developing scheduling algorithms that fit multiprogrammed parallel environments [MAJU88].

With the development of time-shared systems, interest has shifted to providing better services to the various users. For these systems, very little work has been done to exploit the concurrency within a program and therefore we will present scheduling only in terms of uniprocessor systems. In these systems, the ready to run queue of executable work units is made up of the following: first, one or more processes generated by users interactively at their terminal; second, one or more batch processes; and third, the operating system's own internal programmed activities. In the remainder of this section, the term "process", not "task", will be used because the schedulable entity is the whole process.

In such multiprogrammed systems, we have to consider two types of CPU schedulers: the **long-term scheduler** and the **short-term scheduler**. The long-term scheduler determines which processes are admitted to the system for processing. For example, there are often more batch processes submitted than may be executed immediately by the processor. These processes are typically spooled to a mass storage device where they are kept for later execution. The long-term scheduler selects processes from these batch jobs and loads them into memory for execution. In most time-sharing systems, system processes and interactive processes generated by system users are directly placed into the ready to run queue of processes.

The short-term scheduler, in turn, selects one of the ready to run processes and allocates the processor to it. The main difference between the two schedulers is the frequency of their execution. The short-term scheduler is invoked more often. As an example, a process may execute only for a short period of time before becoming blocked waiting for an I/O request. In this case, the process is placed in the

appropriate I/O waiting queue and the short-term scheduler is once again called upon to make a decision. The short-term scheduler may again be invoked if a process is **timed-out** because its time-slice was used up or **swapped-out** due to an overcommitment of available memory or to improve on the process mix (i.e. a good combination of CPU-bound and I/O-bound processes).

The long-term scheduler executes much less frequently. It may be minutes between the arrival of new processes in the system. In general, most processes may be described as I/O-bound or CPU-bound processes. It is the function of the long-term scheduler to select a good mix of I/O-bound and CPU-bound processes. If all selected processes are I/O-bound, the ready to run queue of processes will almost always be empty. If all processes are CPU-bound, the I/O waiting queues will almost always be empty and again the system will be unbalanced. The system with the best performance will have a good mix of CPU-bound and I/O-bound processes [PETE83]. Figure 2-8 shows a simplified queuing diagram for the single processor system.

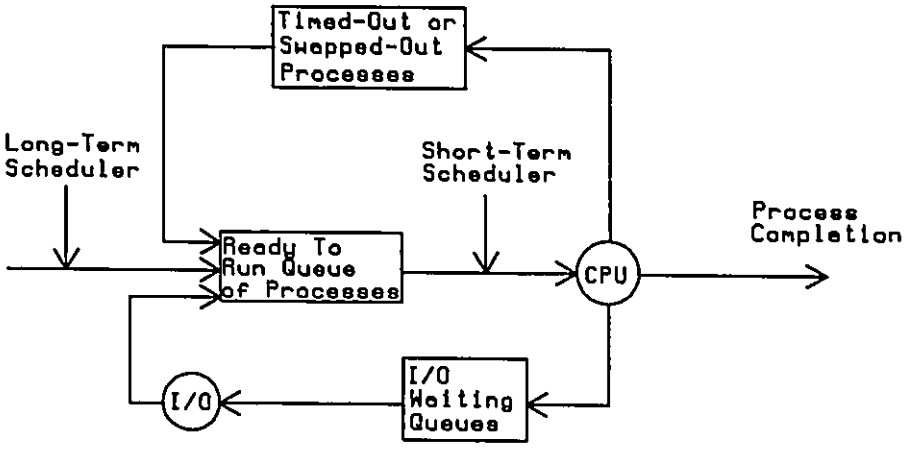


Figure 2-8 Block Diagram for Single Processor System Schedulers

There has been much work carried out on the short-term scheduler based on queuing theory models [HANS73] [KLEI76]. It is generally agreed that the single most important performance measure, for the short-term scheduler, is to minimize the **mean process response time.** Studies of different scheduling policies, such as First Come First Served (FCFS), Shortest Processing Time (SPT), Round Robin (RR) and Shortest Remaining Processing Time (SRPT) have been carried out in the

context of general purpose uniprocessor systems [KLEI76].

In FCFS and SPT scheduling, running processes are allowed to run to completion. In RR scheduling, each ready to run (RTR) process, in turn, is assigned a time interval, called its quantum, during which it is allowed to run on the processor. If the process has not completed execution at the end of the quantum, the running process is suspended and placed at the end of the RTR queue and the processor is allocated to the next RTR process. The SRPT scheduling rule is also based on time-slicing. At the end of its quantum, an uncompleted process is put back into the RTR queue. The SRPT discipline then allocates the processor to the RTR process with the least remaining processing time left. A summary of the relative performance of these four scheduling policies was presented by Majumdar [MAJU88] and is reproduced here as figure 2-9.

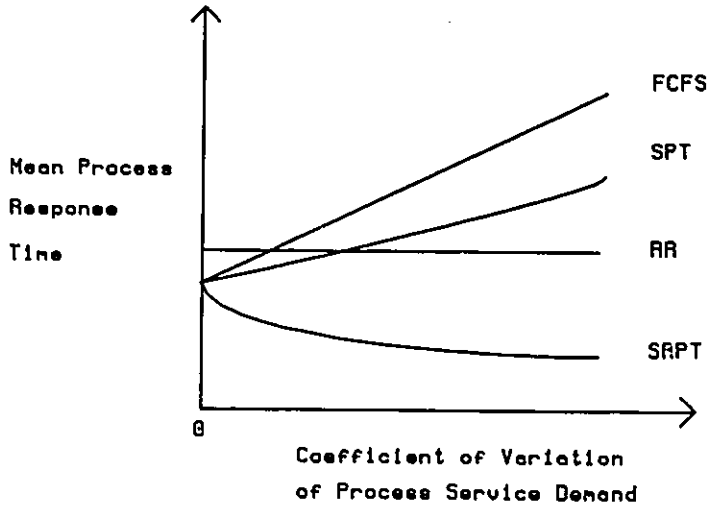


Figure 2-9 Performance of Scheduling Policies in Uniprocessor Systems

By comparing the performance of the different scheduling rules of figure 2-9, two observations can be made. First, "time slicing" or allocating to each process a small quantum of service before switching to another process improves system performance. For example, we see that the RR and SRPT scheduling rules lead, most of the time, to lower mean response times than the FCFS and SPT scheduling rules. Second, knowledge of estimated process execution times enables the use of time based scheduling rules which improve mean process response time. For

example, the SPT rule performs better, most of the time, than the FCFS rule. The SRPT scheduling rule, in turn, always performs better than the RR scheduling rule.

It is worth noting that although detailed a priori knowledge of estimated process execution times would lead to better system performance, such information is often not available in most real general purpose computer systems. It is for this reason that scheduling of the ready to run processes in general purpose computer systems has almost always consisted of one of the following algorithms: **round robin** or **priority scheduling.** Since round robin scheduling has been described before, it will not be discussed any further.

In priority scheduling, each process is assigned a priority and the ready to run process with the highest priority is allowed to gain control of the processor. In order to prevent high priority processes from running continuously, the scheduler may decrease the priority of the currently executing process at each clock interrupt. If this action causes its priority to drop below that of the next ready to run process then a context switch occurs [TANE87]. Priorities may be assigned statically or dynamically. In static priority determination, each process is given a priority at system start-up and maintains it throughout time. For example, in early time-sharing systems, the priority of a process was proportional to the business the user gave the computer center per month. Priorities may also be assigned dynamically. For example, processes that are highly I/O bound should be allocated to the processor as soon as possible to enable them to start its next I/O request which can then proceed in parallel with another process actually computing.

2.4 Scheduling in Real-Time Systems (RTS)

The purpose of this section is to discuss the effects of scheduling in RTS. As discussed in chapter 1, RTS may be divided into hard and soft RTS. In hard RTS, processes with hard timing constraints must always meet their deadlines. If a hard deadline is missed then the tardiness of the response will cause a critical system error. In soft RTS, the objective is to ensure that processes with hard timing constraints meet their deadlines while fast average response times are provided to processes with soft deadlines [STAN89].

A general scheduling taxonomy for RTS was presented by Casavant and Kuhl [CASA88]. Figure 2-10 represents a slightly modified version of this classification involving only the top three levels.

At the top level, scheduling may be divided into single processor and multiple processor scheduling. Single processor and multiple processor scheduling, in turn, may be divided into two categories: static and dynamic. In static scheduling, the characteristics of the environment are assumed to be known a priori, and therefore, the sequence in which the process activities become ready to run can be determined

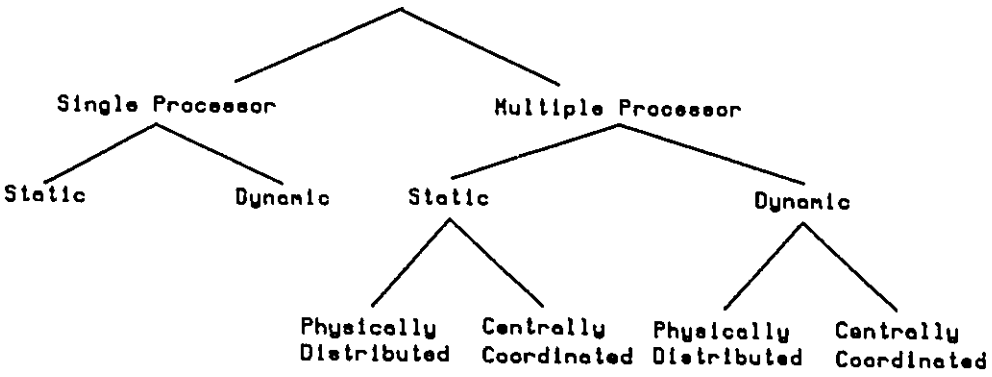


Figure 2-10 A Classification of RTS Scheduling

off-line. In dynamic scheduling, processors are assigned to the ready to run work units in an on-line manner according to some measure of urgency and criticalness.

In this section, we shall briefly review the main results in RTS scheduling. It should be pointed out that much of the RTS scheduling research assumes that the event responses are computational processes made up from one single work unit. Furthermore, processes are normally restricted in three additional ways: first, no parallelism (concurrent work unit execution) is allowed within a process; second, processes are not allowed to synchronize with other processes; and third, no resource contention is permitted except for processors.

2.4.1 Single Processor Scheduling

From the results presented on single resource scheduling in Operations Research (section 2.2.1), one can see that one can use various scheduling schemes such as Shortest Processing Time, Earliest Due Date or Moore sequences to meet soft deadlines. It was also shown that the general problem of scheduling a set of

processes with precedence constraints and deadlines so that all processes meet their deadline is an NP–complete problem.

2.4.1.1 Static Scheduling

Many RTS, such as those found in control and signal processing applications, require only the execution of periodic processes. In these systems, since all interactions are predetermined, static scheduling is possible. Typical functions accomplished by periodic processes in RTS include the periodic checking of various sensors and the update of the current status of the RTS. For example, consider the example of a radar altitude avionic sub–system (RAAS) available today on most commercial and military aircrafts. The RAAS is used to determine an aircraft's "distance above ground". Examples of periodic processes in the RAAS include: first, checking if an input signal is available at the RAAS antenna; second, transforming the input signal into a usable "distance above ground" value; and third, update the pilot's RAAS display with the current "distance above ground" value.

Liu and Layland [LIUL73] considered the problem of creating a schedule for a set of periodic processes on a single processor. The processes were assumed to be preemptable and independent and each process had a deadline (d_i) and fixed upper bound on its response time (t_i). For these conditions, they developed the **rate monotonic** assignment rule in which periodic processes are assigned priorities inversly proportional to their period. They proved that this algorithm can guarantee the deadlines of a set of **n** periodic processes if condition (1) is satisfied:

$$t_1/p_1 + \ldots + t_n/p_n \leq n\,(2^{1/n} - 1) = u(n) \qquad (1)$$

where t_i and p_i are the execution time and period of a process T_i respectively. Essentially, condition (1) ensures that as long as the processor utilization for all processes lies below a certain bound u(n), all periodic processes will meet their deadlines. The values of the scheduling bounds for one to five independent processes are as follows: u(1)=1.0; u(2)=0.828; u(3)=0.779; u(4)=0.756 and u(5)=0.743. The scheduling bound converges to 0.69 or ln 2 as the number of processes approaches infinity.

For example, as shown in figure 2–11(a), suppose that processes 1 and 2 are periodic with periods of 25 and 5 respectively. Both of these processes are initiated at t=0. Assume that process 1 requires 5 units of computation time and its first

deadline is at t=25, while process 2 requires 1 unit of computation time and its first deadline is set at t=5. The total processor utilization of these two processes is:

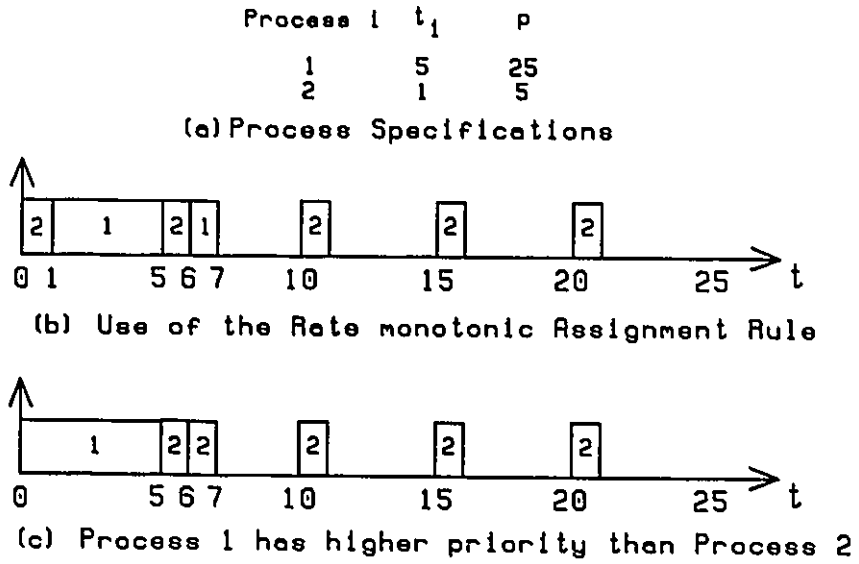


Figure 2–11 Periodic Process Scheduling on a Single Processor

$t_1/p_1 + t_2/p_2 = 5/25 + 1/5 = 0.4$ which is well below condition (1)'s bound of $2(2^{1/2} - 1) = 0.828$ for two periodic processes. This means that these two processes are schedulable; that is, they will meet their deadlines if a higher priority is assigned to process 2 since process 2 has a smaller period than process 1. As shown in figure 2–11(b), this priority assignment ensures that both processes 1 and 2 meet their deadlines. However, as shown in figure 2–11(c), had process 1 been assigned a higher priority than process 2, then the first instance of process 2 would not have met its due date of t=5 and would have completed at t=6. Sha [SHA86] recently described a technique to modify the periods of processes in such a way that while the process deadlines continue to be met, better processor utilization is achieved.

An alternate way to determine whether a set of periodic processes are schedulable is to set the length of the schedule to the **least common multiple** (LCM) of the periods specified in the processes and then to schedule the processes according to the earliest deadline scheme [LAWL84]. For example, if the periods of the periodic processes specified in the scheduling model are set at 40, 50 and 100

time units, then the corresponding schedule length would be set at 200 time units. Then, the earliest deadline scheme is applied to determine whether all the instances of the periodic processes in the interval can be scheduled to meet their deadlines. If they can be scheduled, then all other instances of the periodic process may also be scheduled. If the period of the periodic processes are relatively prime, the length of the schedule may be large and the cost may become very high.

2.4.1.2 Dynamic Scheduling

Many RTS require the execution of both periodic and sporadic processes. Sporadic processes typically arise in control oriented systems and in operator initiated actions. In most RTS, periodic processes normally have hard deadlines; while sporadic processes may have hard or soft deadlines. In these systems, one has to guarantee the hard deadlines of periodic processes and also be able to provide good average response times for the soft deadline sporadic processes. The scheduler must be at least partially dynamic to accomodate sporadic responses.

Two commonly used approaches for handling soft deadline sporadic processes is to use either one of two techniques: **background processing** or **polling processes.** Background servicing of sporadic processes occurs whenever the processor is idle. If the load of the periodic process set is high, then the processing time left for background processing is low; and therefore the average response time to sporadic requests will be high. The "polling processes" technique consists of creating a periodic process in order to service sporadic requests. At regular intervals, the polling process is started and services any outstanding sporadic request. However, if no sporadic requests are pending, the polling process suspends itself until the next period and the time allocated for sporadic sevice is used instead by periodic processes. Should a sporadic request occur just after the polling process has been suspended, then the sporadic request will have to wait until the start of the next polling process period [SPRU89].

Figure 2–12 presents an example of periodic and sporadic process scheduling using background processing and polling processes. In this example, there are two periodic processes (1 and 2) and two sporadic processes (S1 and S2) that have to be scheduled. As shown in figure 2–12(a), processes 1 and 2 have periods of 10 and 20 respectively. Both of these processes are initiated at t=0. Assume that process 1 requires 3 units of computation time and its first deadline is at t=10; while process 2

requires 9 units of computation time and its first deadline is set at t=20. The rate monotonic algorithm is used to assign priorities to the periodic processes. Process 1

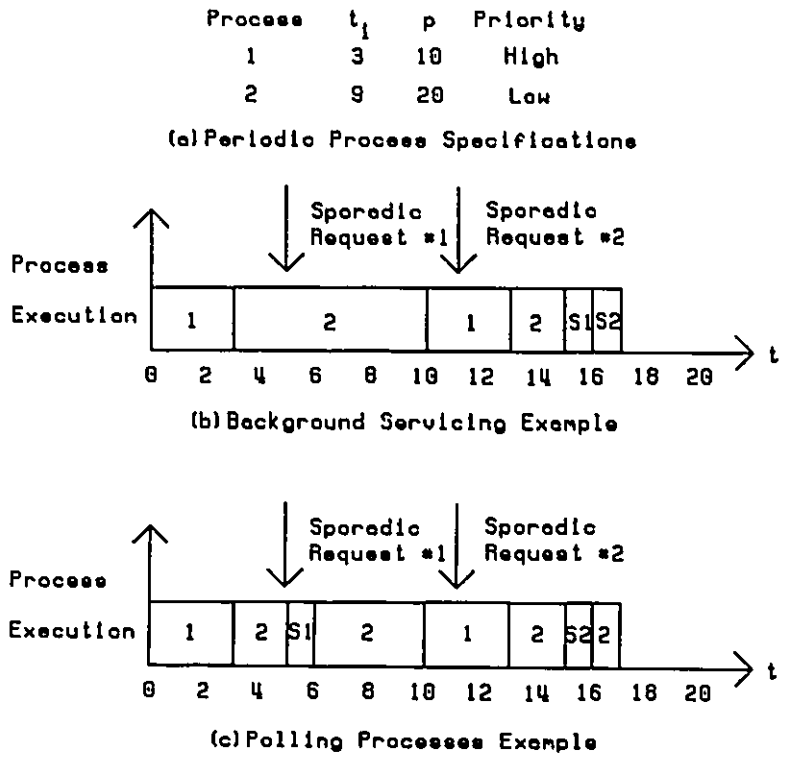


Figure 2–12 Periodic and Sporadic Process Scheduling Using Background and Polling

is assigned a higher priority than process 2 because it has a shorter period. In this example, two sporadic requests occur: one at t=5 and one at t=11. Each sporadic request requires 1 unit of computation time to complete.

Figure 2–12(b) shows the background servicing of sporadic processes for our example. Process execution is shown from t=0 to t=20. Since background service only occurs when the resource is idle, sporadic service cannot begin until t=15. S1 and S2 are then executed successively. The response time to the two sporadic requests are 11 and 6 respectively.

Figure 2-12(c) shows the scheduling of processes using the polling process technique. Process execution is shown from to t=0 to t=20. In this example, a polling server is created with one unit of computation time and a period of 5 time units. Using the rate monotonic algorithm makes the polling server the highest priority process. The polling server's first period begins at t=0. Since there are no sporadic requests pending at t=0, the execution time of the polling server is discarded during its first period and is used instead by the periodic processes. The beginning of the second polling period at t=5 coincides with the occurrence of the first sporadic request, and so, sporadic process S1 is executed immediately. The second sporadic request, however, misses the beginning of the third polling period at t=10, since it is invoked at t=11, and must wait until the fourth polling period at t=15 before being serviced. The response times to the sporadic requests using polling processes are 1 and 5 time units, respectively, which represents better service than for the background service example.

The major drawback to the "polling processes" technique is that by creating a periodic polling server to handle sporadic requests, the remaining portions of the unexecuted periodic processes slip by the execution time of the sporadic processes. In the case where the number of sporadic processes is large or if their computation time is intensive, the amount of induced slippage might cause some hard deadlines for periodic processes to be missed.

To overcome these problems, one method that can be used is to transform all sporadic processes into **pseudo-periodic** processes. This transformation is based on the assumption that consecutive requests for sporadic process execution are generally separated by a minimum interarrival time. The pseudo-periodic process period is set to the minimum interarrival time between consecutive requests. In practice, the minimum interarrival time can be enforced by buffering the sporadic requests and executing the pseudo-periodic process once every period. This approach assumes no request overload for sporadic processes and that spare capacity is available on the single processor to execute these processes. At every time interval, the request buffer is checked to determine if there are any requests pending for a sporadic process. If a request is pending, the process is executed. One drawback of this technique is that whether or not the sporadic process has been invoked, its execution is scheduled once per period. This may lead to processor underutilization in applications with a large number of sporadic processes or where the minimum interarrival times of sporadic processes are relatively prime to the periodic process periods. This transformation technique can also be extended to the scheduling of

periodic processes and sporadic processes with hard deadlines[SPRU89].

The last RTS scheduling problem to be considered, for the single processor, is the ability to guarantee the deadlines for a set of hard deadline sporadic processes. Mok and Dertouzos [MOKD78] showed that, if the set of all possible sporadic processes ever to arrive in the system could be scheduled statically, then this set of processes could also be scheduled at run-time. They also showed that ordering the processes by earliest due date (EDD) or least laxity first (LLF) will always find a feasible schedule.

2.4.2 Multiple Processor Scheduling

From the results presented on multiple resource scheduling in Operations Research (section 2.2.2), one can see that one can use various scheduling schemes such as Shortest Processing Time or Longest Processing Time to meet the soft deadlines of a set of processes with identical arrival times.

Due to the widespread interest in RTS scheduling, the recent work in scheduling on multiple processors is very extensive, therefore we shall only review the basic results involved in meeting process due dates. As shown in figure 2-10, multiple processor scheduling may be divided into static and dynamic scheduling. Static and dynamic multiple processor systems may be divided into centrally coordinated and physically distributed scheduling. In a centrally coordinated scheme, the responsibility for scheduling ready to run work units normally resides within a single logical authority. Furthermore, the processors are normally located at a single point in the system. In a physically distributed scheme, the processors are located at a different point in the system and the interprocessor communication costs are not negligible. Each processor has a certain degree of autonomy in determining how it should be used.

2.4.2.1 Static Scheduling

The first level of complexity in static scheduling involves scheduling a set of independent periodic processes on a centrally coordinated multiple processor system. Scheduling a set of preemptable periodic processes is more complicated than for single processor systems. Guaranteeing real-time constraints for a set of periodic processes is accomplished by partitioning the periodic processes among a number of

processors such that each partition may be scheduled on one processor according to the rate monotonic scheme.

For example, Davari and Dhall [DAVA86] presented a **next-fit** scheduling algorithm that divides a set of periodic processes into different classes. As shown below, each class of periodic process (class-1, class-2, ...,class-M) corresponds to a range of utilization factor (u_i) where u_i is defined to be equal to the ratio of response time (t_i) to period (p). The next-fit algorithm allocates at most K class-K periodic processes to each processor so that the total utilization factors of all periodic processes assigned to each processor does not exceed the bounds specified by Liu and Layland's rate monotonic algorithm (see page 2-21, u(1)=1.0, u(2)=0.828, ...).

Class of periodic process	Range of Utilization Factor($u_i=t_i/p$)
1	$[2^{1/2}-1, 1]$= [0.414, 1]
2	$[2^{1/3}-1, 2^{1/2}-1]$=[0.259, 0.414]
3	$[2^{1/4}-1, 2^{1/3}-1]$=[0.189, 0.259]
.	.
M	$[0, 2^{1/M}-1]$

As an example, figure 2-13(a) show the specifications of six different processes to be scheduled. In this example, there is one class-1 process (process 1) because its utilization rate u_i=0.6 is within the range of [0.414, 1.0], two class-2 processes (processes 2 and 3) because their utilization rates of 0.4 and 0.3 are within the range [0.259, 0.414] and three class-3 processes (processes 4, 5 and 6) because their utilization rates of 0.2 are within the range [0.189, 0.259]. Using the next-fit algorithm: process 1 is allocated to processor 1; processes 2 and 3 are allocated to processor 2; and processes 4, 5 and 6 are allocated to processor 3. All six processes are eligible for execution at t=0. The first deadline is set at t=10 for all 6 processes. The total utilization rates for processors 1, 2 and 3 are 0.6, 0.7 and 0.6, respectively. These utilization rates are well below the scheduling bounds proposed by Liu and Layland of u(1)=1.0 for one process, u(2)=0.828 for two processes and u(3)=0.779 for three processes. This means that all six processes will meet their deadlines when using the Davari and Dhall next-fit algorithm. Figure 2-13(b) show the schedules that would be generated using Davari and Dhall's next-fit algorithm.

Gagne [GAGN89] recently extended Davari and Dhall's work, using a branch and bound enumeration technique, to schedule between their start times and due dates a set of periodic processes with precedence relations on the heterogeneous processors of the Canadian Forces CF-18 Fighter Aircraft Mission Computer System.

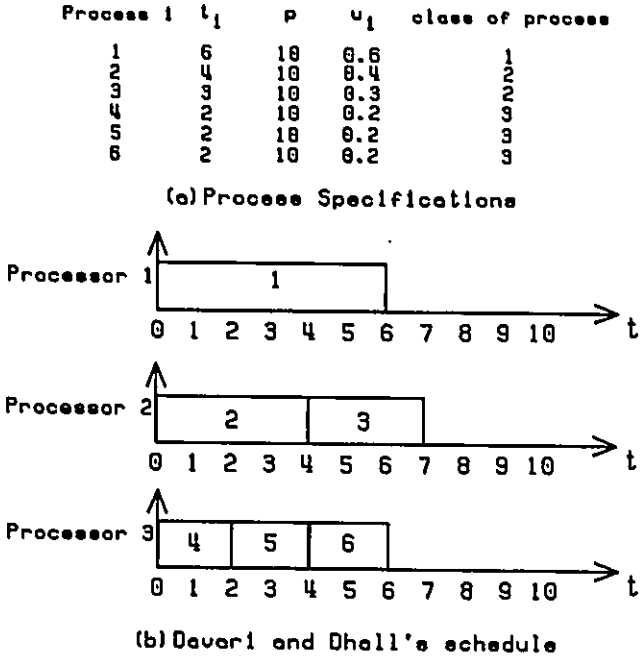


Figure 2-13 Scheduling Periodic Processes on Multiple Processors

The next level of complexity in static scheduling involves scheduling different processes on a physically distributed multiple processor system. The physically distributed category of problems has traditionally been formulated as a **task allocation** or **load balancing** problem. In a physically distributed system, the objective of a load balancing algorithm is to assign work units to processors in the system so that all processors in the system are approximatively evenly loaded. The effect of equally dividing load among processes has often been to optimize throughput.

In static physically distributed scheduling, there are only a few algorithms that apply directly to hard RTS. One of these algorithms was created by Leinbaugh and

Yamini [LEIN82]. Their algorithm computes worst-case completion times for a set of processes which are made up of multiple work units with series-parallel precedence constraints.

The second result of interest is due to Peng and Shin [PENG87]. They developed a generalized stochastic Petri Net (GSPN) to model the behavior of a physically distributed system. Their model considered both periodic processes with precedence constraints and sporadic processes without precedence constraints. The precedence constraints of the periodic processes form an AND-OR graph. They used the Petri-Net to build a sequence of homogeneous continuous-time Markov chains to model concurrent process execution in the system. Given a process selection policy and the local state of each node in the system, the Markov chain model can be used to compute the probability of missing a hard deadline [CHEN88].

2.4.2.2 Dynamic Scheduling

Dynamic scheduling is applicable only to RTS with soft constraints. This fact was proven by Mok and Dertouzos [MOKD78] in one of the most important papers published on multiple processor scheduling. They showed that, for centrally coordinated multiple processor systems, no algorithm can guarantee the deadlines of all processes to be scheduled if the arrival times of a set of independent processes are not known before run-time. For example, many sporadic processes arise unpredictably as a result of operator initiated actions or control oriented systems. In these RTS, Mok and Dertouzos have proved that it is impossible to guarantee all process deadlines.

In a physically distributed system, processes may arrive at any node of the system. The state of each node is dynamic in nature. Because of the communication delay between nodes, the state information received from a remote node is inaccurate by the time it is received. It is for these reasons that scheduling in distributed systems has traditionally consisted of two parts: a **local scheduling** algorithm and a **distributed scheduling** algorithm.

A good example of scheduling in physically distributed RTS is the algorithm developed by Ramamrithan and Stankovic [RAMA84] for scheduling independent processes. The goal of the algorithm is to have as many processes as possible that

meet their deadlines. This performance measure is often referred to as the guarantee ratio. The first step in their algorithm is to use as local scheduling algorithm a heuristic such as earliest due date to guarantee as many process deadlines as possible. The distributed algorithm consists of a **bidding** or a **focussed addressing** algorithm. In focussed addressing, processes that can not be guaranteed locally are sent to a selected remote node that is estimated to have high surplus processing time. In bidding, when a new process arrives, the node initially responsible for its execution broadcasts a request for bids to all other nodes. This message contains details of the resource requirement of the new process. Each node then works out a bid based on its current loading. The bid is returned to the initiating node which selects the lowest one. Simulation results using this algorithm [RAMA84] have shown that a high percentage of processes meet their deadlines under a wide range of system conditions and process parameters.

Other work of interest is that of Craig [CRAI87] which investigated light traffic loss of random hard real-time tasks in a physically distributed network. In that study, task loss rate for a physically distributed network was examined under preemptive local scheduling algorithms and unrestricted routing of transferred tasks.

2.5 Concluding Remarks

The purpose of this chapter was to present a survey of pertinent results in Operations Research (OR), General Purpose and Real-Time Computer Systems. Section 2.1 introduced a general scheduling terminology that allowed us to present results from both OR and computer systems. In section 2.2, results were presented for the single, parallel machine and job shop scheduling in OR. It was shown that one could use various scheduling rules such as Shortest Processing Time or Longest Processing Time to meet the soft deadlines of a set of processes. In OR, event responses are assumed to be simple (i.e. no concurrent work unit execution is allowed within a process). Additionally, two other restrictions are normally placed upon work unit execution in job shop scheduling. First, work unit execution is normally restricted to one resource. Second, most instances of the job shop scheduling problem do not consider the possibility of allocating a work unit to more than one resource.

Section 2.3 dealt with scheduling in General Purpose Computer Systems. In these systems, processes are made up of a single executable work unit. CPU scheduling in General Purpose Computer Systems has almost always consisted of round robin or priority scheduling; although certain attempts have been made to use scheduling rules, such as Shortest remaining Processing Time, which use knowledge of estimated process execution times in order to minimize mean process response times.

In section 2.4, the effects of scheduling in RTS were discussed. RTS scheduling was divided into static and dynamic scheduling. Static scheduling was found to be directly applicable to RTS with periodic processes. The rate monotonic and the next-fit scheduling algorithms presented the conditions under which a set of periodic processes could meet their hard deadlines on a single or multiple processor system, respectively. In addition to scheduling periodic processes, one must also be able to schedule sporadic processes so that both types of processes meet their hard deadlines. It was shown that the transformation of sporadic processes into pseudo-periodic processes could lead to a feasible schedule. The main drawback to this method was that it often led to processor underutilization.

Dynamic scheduling, in turn, is applicable only to RTS with soft constraints. As an example, the polling process technique was presented. It permitted periodic processes to meet their hard deadlines while providing good average response times to sporadic processes with soft deadlines. The major problem with this technique is that in the case where the number of sporadic processes is large or if the computation time is intensive, then some hard deadlines for periodic processes might be missed.

In the general case of scheduling a set of independent processes on a multiple processor system, Mok and Dertouzos showed that the deadlines of all processes to be scheduled could not be guaranteed when the process arrival times are not known before run-time. There are also many other drawbacks with the current trends associated to the research in RTS scheduling. First, much of the RTS scheduling research assumes single work unit responses. Second, event responses are assumed to be simple (i.e. no concurrent work unit execution is allowed). Third, work unit execution is normally restricted to one resource.

To overcome some of these limitations, this thesis will investigate the effects of different dynamic scheduling heuristic rules on the design of RTS which allow for

the execution of complex responses. Complex responses will be modeled using the multiactivity system design paradigm and the Process Activity Language (PAL). In multiactivity systems, the activities are the schedulable work units which can be dynamically scheduled based on the present status of their responses according to some scheduling heuristic rule. In multiactivity systems, event responses allow for concurrent activity execution. In addition to this, activities may be allocated to more than one resource. RTS design will be done through the use of a simulator which allows for the selection of the external event characteristics, definition of the RTS (processes, due dates, activity-to-resource allocations with activity execution times) and selection of different activity scheduling schemes (first-come first-served, earliest due date, shortest activity first, etc....). Statistics, such as response execution times, waiting times of ready-to-execute activities, percentage of responses that are tardy and resource utilization, are gathered and provided to the designer. These statistics allow the designer to refine the design, by changing system parameters, until satisfactory RTS operation is achieved.

CHAPTER 3

PAL BASED DESIGN OF REAL–TIME SYSTEMS

The growing number and diversity of complex real–time system (RTS) applications has highlighted the need for the development of better specification and design tools. The Process Activity Language (PAL) was designed to address this problem by being a design specification language. Furthermore, based on PAL, a number of specification and design tools were built. Section 3.1 presents the elements of PAL. Section 3.2 introduces multiactivity systems in which PAL specifications can be directly implemented. Section 3.3 presents the different design steps that may be used in the design of RTS using PAL and its associated toolset.

3.1 The Elements of PAL

In developing PAL, an attempt was made to strike a balance between the need for formalism, to allow for verification, and the need for understandability. To satisfy the formalism requirement, well defined primitives and structuring or control constructs similar to those found in structured programming languages were chosen. To satisfy the understandability requirement, a limit was placed on the number and complexity of the language elements. These elements were based on intuitively understood concepts.

As mentioned in chapter 1, in RTS, one assumes two distinct entities: first, the environment, which as a rule is a complex dynamic system that includes a multiplicity of devices that have to be controlled; and second, the RTS, a computer based system consisting of hardware, software and special purpose interfaces that coordinate the operation of the environment in a prescribed way. In RTS, the computer is not the master entity. Rather, it acts as a slave to the environment which prescribes the types of responses that must be generated by the computer. The RTS must react to the occurrence of sensor detected environmental events and must respond to them at appropriate instances via actuators.

RTS can be abstracted as a set of processes invoked in response to external events. These processes can be partitioned into subprocesses, which in turn can be partitioned, until the lowest level of interest to the system designer is attained. The entities at this level are called activities. In this context, a process corresponds to a set of activities which are executed in a given precedence relation so that a correct response to an event is achieved. The order in which the activities have to be executed depends on the data dependencies between activities of the same process and between activities of different processes as well.

The processes defined may have multiple threads of control, as many activities may potentially be executed concurrently. This contrasts with the definition of a task which has only one thread of control. In order to meet the real-time requirements of RTS, it is desirable to maximize activity concurrency in the available processors.

A RTS application in PAL is specified as a set of processes $\{P_i, P_j, \ldots, P_n\}$. Each process has a well defined **starting event**, S_i, a **response** X_i, and an **end** E. The general form of a process is as follows:

$$P_i := S_i \cdot X_i \cdot E$$

A given starting event S_i may represent both sporadic and periodic external events. Depending upon the application, if another starting event occurs while the given process is active, the event may be ignored, queued, or another instantiation of the process may be started.

The response X_i, is composed of PAL language primitives whose execution ordering is defined by the PAL control constructs. There may be an abort or shut-down procedure associated with each process. Since these procedures are highly application dependent, they will not be considered any further here.

To represent independent processes requires the use of only three language primitives: **activities, process conditions** and a **delay** primitive. The basic element of a process expression is the **activity** A_i, a non-preemptable schedulable work unit with a bounded execution time. Decision points in PAL expressions are provided by process conditions, pc_i's. The **set process condition** primitive, STP(pc_i, expression), is used to modify the process condition, pc_i, by assigning to it the value resulting from the evaluation of the expression. The scope of a process condition is

limited to the process in which it is declared. The **delay** primitive, D(n), is used for timing purposes as its name implies it provides a delay of n time units. We will use the notation p_i to designate any generic primitive in PAL and X_i to denote any well formed expression of p_i's.

Basic Control Constructs

Precedence relations between primitives or sub-expressions are defined by two control constructs. The first is the **sequence** construct, $(X_1 \; X_2 \; \ldots)$, which specifies that each expression can only be started after the previous expression in the sequence has been executed. The construct is completed when the last expression in the sequence is executed. The other is the **concurrency** construct, $(X_1 || \; X_2 || \; \ldots)$, which specifies that all the expressions in the construct are independent and given the availability of resources, any number of them can be executed in parallel. The construct is completed when all the expressions in the construct have been executed.

Conditional execution of primitives or sub-expressions is provided by two additional control constructs: **case** and **repeat**. The case construct, $(lexp(pc_i)_1 : X_1 | lexp(pc_i)_2 : X_2 | \ldots)$, is used to choose a specific execution path upon the evaluation of the **logic expression** $lexp(pc_i)$ containing the specified process condition. To simplify validation, the set of logic expressions must be defined such that at any time one and only one of them is true. The repeat construct, $<X:lexp(pc_i)>$, specifies that the expression X is repeatedly executed until the logic expression $lexp(pc_i)$ becomes true.

In addition to the algebraic representations introduced above, the PAL control constructs can also be represented in **pseudo code** or **graphical format** as shown in figure 3-1. The graphical representation, referred to as a **process activity net**, provides an easy visual check of the process expression. A simple example of an independent PAL process, represented in algebraic, pseudo code and graphical formats is presented in figure 3-2.

Note that in pseudo code notation, a process is differentiated from the sequence construct by enclosing it in a **begin...end** block. Figure 3-2(d) includes the corresponding **activity timing chart**, a component of PAL that shows the execution times of the various activities of a process. The relative execution times assigned to each activity in the chart are estimates of the average or maximum time needed for

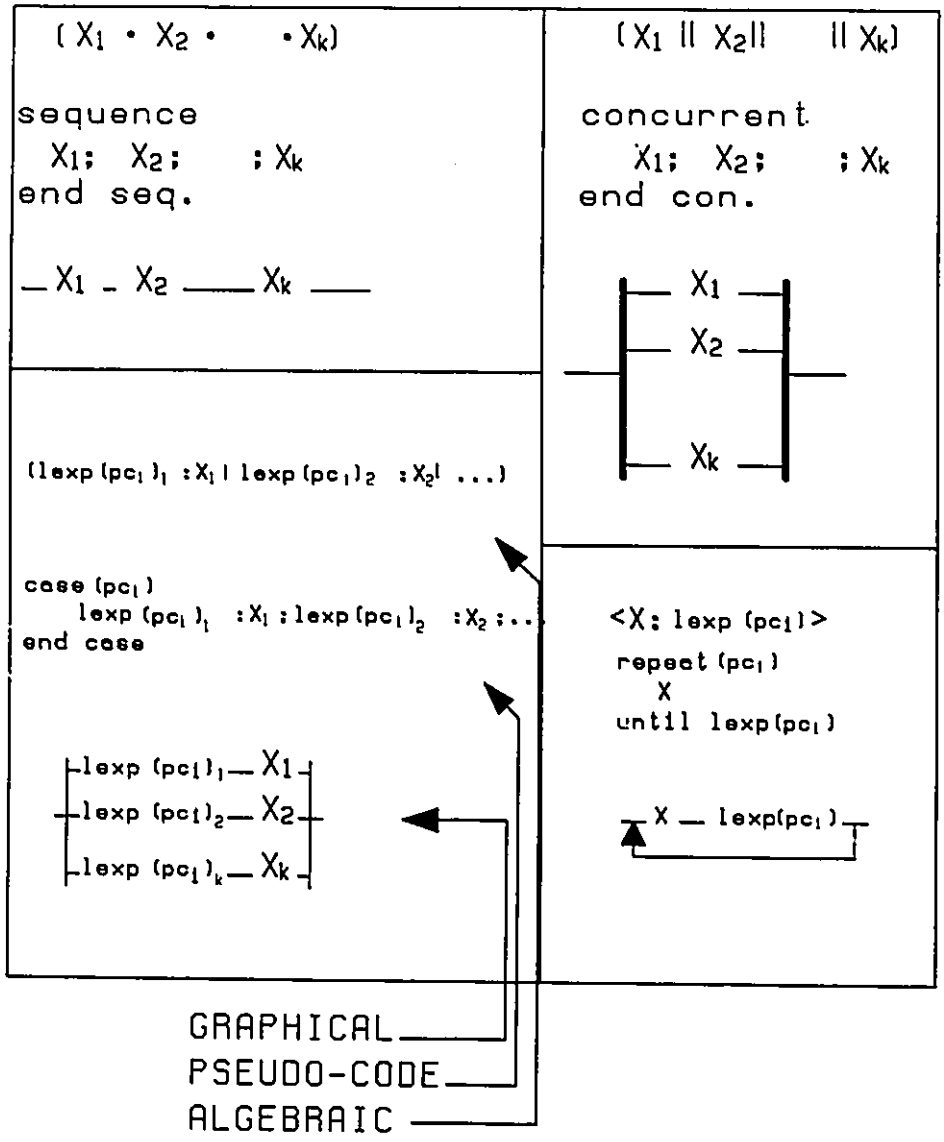


Figure 3–1 Basic PAL Constructs

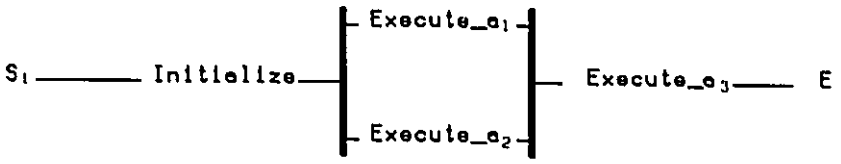
the execution of the activity. The activity timing charts also help to indicate the hierarchical nature of PAL. The shown expression can be viewed, at a higher level of abstraction, as a composite expression with an execution time equal to the combined process timings of the primitive activities.

$P_1 := S_1 \bullet (Initialize \bullet (Execute\_a_1 \parallel Execute\_a_2) \bullet Execute\_a_3) \bullet E$

(a) algebric

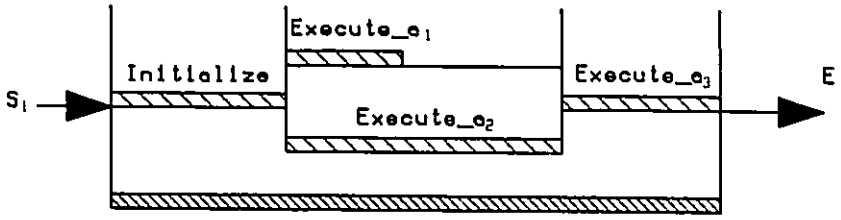
```
P1 := begin
        wait for S1
        sequence
          Initialize;
          concurrence
            Execute_a1 ;
            Execute_a2
          end con.;
          Execute_a3
        end seq.
      end
```

(b) pseudo-code



Process Activity Net

(c) graphical format



Combined process timing

(d) Activity Timing Chart

Figure 3-2 A Simple Example

Interprocess Communication

When it comes to represent the interaction between responses, PAL is not a "pure" specification language. In pure specification languages, the tendency is to represent interprocess dependencies as conditions such as:"certain things can not occur at the same time" or "one has to wait for another", etc..., without stating how they will be resolved. In PAL, interprocess dependencies are explicitly stated in

terms of interprocess communication primitives in the process expressions. While this does require greater expertise on the part of the specifier, we are assuming that the application specialist is computer literate.

In PAL, communication between dependent processes is done via two distinct mechanisms: **message passing** for direct communications and the use of **billboard conditions** for posting conditions of general interest.

Messages are passed between PAL processes via named, one-way, finite capacity channels operated on by two messaging primitives: **transmit message**, TXM(channel, parameters) and **wait for message**, WTM(channel, parameters). If a channel is empty, the process executing the WTM primitive will wait until a message is available. If the channel is full, then the process executing the TXM primitive will wait until after a message has been received. Thus, if the channel is not full, the TXM primitive is non-blocking. For each channel, a queue length of 0, 1 or n, has to be specified in the receiving process.

A billboard condition is basically a global variable defined such that it can be globally read by any process, on a billboard, but can only be modified by one of the processes in the system. The primitives used to set and read system conditions are: **set system condition,** STS(sc_i, expression) and **read system condition**, RDS(sc_i, pc_j). The STS primitive sets the system condition, sc_i, to the value resulting from the evaluation of the expression. To avoid non-determinism, multiple updates of a system condition within a process must be done in sequential order. The RDS primitive assigns to the process condition, pc_j, the current value of the system variable, sc_i.

One can also access an environment condition, ec_i, by using the **read environment condition**, RDE(ec_i, pc_j), primitive that behaves in the same way as the RDS primitive. To use a system or environment condition within a process, the value of that condition must first be assigned to a process variable via the proper read primitive. A process can also wait to receive an event from the environment via the **wait for environment** primitive, WTE(channel, parameters). Messages and events can represent either a pure synchronization event or the transfer of a data item.

Additional Constructs

The above set of primitives and control constructs is enough to specify most systems. In order to simplify the specification of certain systems, PAL can be expanded to include additional primitives and/or control constructs, as long as the "single input/single output" requirement of structured systems is maintained. For example, the control constructs: **select** ($X_1|X_2|$ ...), **simultaneous** {$p_1||p_2||$ ...} and **alternation** [$p_1||p_2$ ...] were found to be useful. The pseudo code and graphical representations of these constructs are shown in figure 3-3.

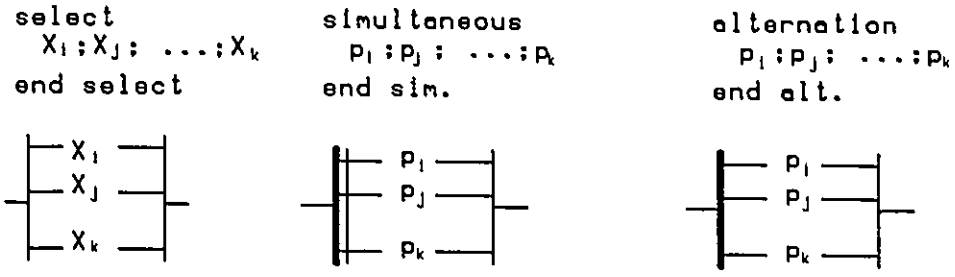


Figure 3-3 Additional Constructs

The select construct is similar to the one used in Ada and works in the following manner. Each path is examined for eligibility to be executed. Eligibility for execution is based on the first primitive in the path and is determined as follows: all read (RDS, RDE) and set (STP, STS) primitives are always eligible, the wait (WTM, WTE) primitives are eligible if there is a message in the specified channel, the transmit (TXM) primitive is eligible if the specified channel is not full and an activity A_i is eligible if there is a processor available to execute it. Of the alternatives that are eligible one path is chosen non-deterministically for execution. If none are eligible, then the first path to become eligible is selected.

The simultaneous and alternation constructs are basically variations of the concurrency construct and are also specified for primitives. In the simultaneous construct, the start of execution of all the primitives within the construct is delayed until all the necessary resources are available to allow them to start simultaneously. The construct is considered completed when all the primitives are executed. In the alternation construct, execution of each of the primitives is started as resources

become available. The construct is considered complete when the first primitive is finished executing, all the others are purged.

3.2 Multiactivity Systems

Traditionally, concurrent operation in multiple processor systems has been achieved through the use of a multitasking operating system. In such an operating system, tasks are the schedulable entities. The fact that there is not a one to one mapping between tasks and the required responses to an event introduces a major difficulty in mapping real-time response requirements into task priorities [SR87]. Furthermore, most general purpose and real-time multitasking operating systems contain additional support facilities for varied applications which are often not necessary in RTS.

An alternate approach is the use of a multiactivity executive in which the activities making up a response are scheduled directly according to the real-time requirements of the parent processes. The multiactivity executive represents a direct implementation of PAL expressions in that it separates activity execution from activity coordination. As indicated in figure 3-4, this executive can be implemented as a simple three-layer structure. The top layer of the executive consists of the **system manager** which interacts directly with the environment and has a general knowledge of the system. Upon detecting an external event, the system manager is responsible for initiating (or continuing) the required response. The system manager also handles delays and interprocess communications (i.e. billboard variables and messages).

The middle layer consists of **process managers**; one for each awakened process in the system. It is the process manager that contains the precedence relations of the activities of a given process and that determines at each instant the ready-to-run activities of that process.

The actual scheduling of activity execution is done by the **scheduler** which forms the bottom layer of the executive. The scheduler assigns activities to processors according to a predetermined scheduling scheme and information about which processors are currently free. The actual executable code for each activity is resident in one or more of the processor's local memories. Thus, activities with special processing requirements can be assigned to specialized processors and

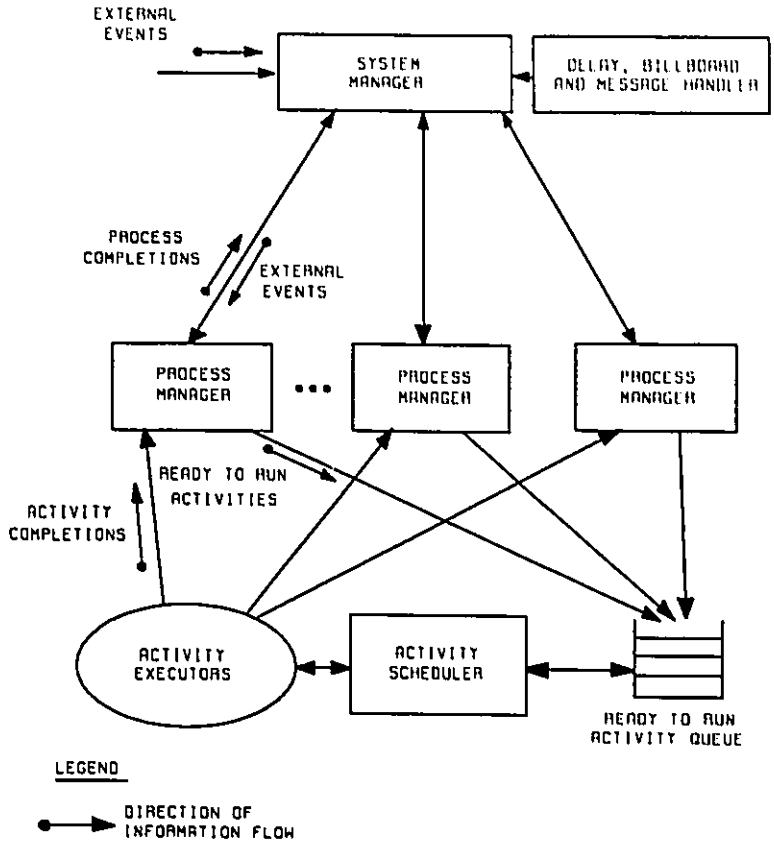


Figure 3–4 The Structure of the Multiactivity Executive

frequently executed activities can be assigned to more than one resource. Information about which processor can execute a particular activity is contained in activity to processor allocation table. The scheduling scheme used by the scheduler can be based on activity priorities such as first–come first–served, shortest activity first, etc..., and/or based on process priorities such as earliest due date, least laxity first, etc...

The main characteristic of a multiactivity executive is that all relevant system information to manage event responses is contained within the executive. The activities do not need to include any information related to the coordination of the response. The executive coordinates and manages the data needed for activity execution. Any data needed for the activity to execute is either transmitted directly

to an activity or a pointer is provided giving the location of the data.

A multiactivity system is well suited for a centrally coordinated multiple processor implementation because of its separation between activity coordination and activity execution. The general organization of a centrally coordinated RTS architecture is shown in figure 3-5. This architecture was originally presented in the paper of Fathi [FATH83]. In such a system, there is a central coordinator which monitors the environment and runs the multiactivity executive, and a set of processing units used to execute the individual activities. The system may also have some global resources such as memory for interprocessor communications, and special purpose I/O.

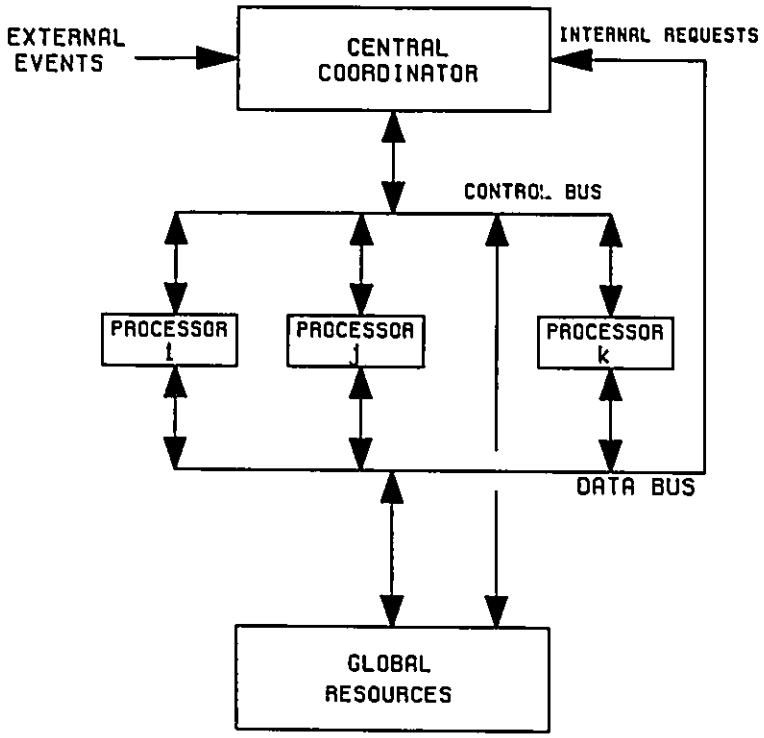


Figure 3-5 The Centrally Coordinated System Architecture

The processing elements can either be homogeneous or heterogeneous. Each processor has its own private program memory, data memory and I/O. Executable code for each of the activities that it can execute (i.e. allocated to it) is permanently stored in the local program memory of the individual processors, thus when an

activity is to be executed, it needs only to be awakened rather than having to be downloaded from a central memory. A typical processor organization and a profile of its program memory is shown in figure 3-6.

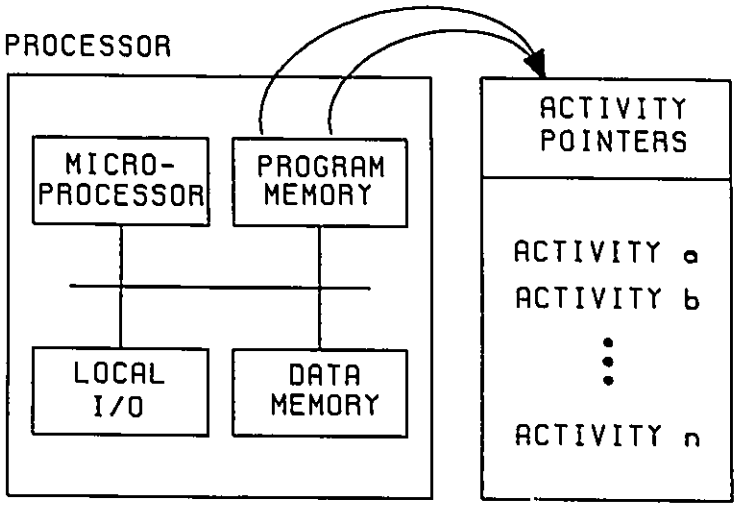


Figure 3-6 A Typical Processor Organization and a Profile of its Program Memory

The centralized architecture provides several benefits in terms of system performance and fault tolerance. Because of the centralized nature of the multiactivity system, there exists the opportunity for more flexible scheduling. The coordinator is aware of the current timing needs of each process, the current list of ready to run activities, and the status of each processor. Proper scheduling can also reduce interprocessor communications by assigning data dependent activities to the same processor.

In terms of fault tolerance, a slave processor's failure will produce little overheads because it will affect a single activity. This problem can be overcome by incorporating a timeout mechanism in the coordinator which when activated declares the processor faulty and restarts the affected activity on a new processor or by executing critical activities redundantly in two or more processors and comparing the results. Note that our Centrally Coordinated system exhibits the same drawbacks generally associated with all centralized systems. These include controller bottleneck, communications bottleneck and questions of controller reliability. Because of the simplicity of the outlined executive, the bottleneck in the coordinator may only appear when there is a very large number of processes, activities and

events. A good partitioning of the system between different processes can reduce interprocess communications and an adequate granularity of the activities should not produce too many communication overheads between the coordinator and the processors. The problem of the reliability of the coordinator can be solved by using a hot stand-by which can assume control in case of a failure.

3.3 Design of Real Time Systems using PAL

The classic waterfall model was defined as early as 1970 by Royce [ROYC70] to help cope with the growing complexity of RTS projects being tackled. The waterfall model is shown here as figure 3-7. This graph was adapted from a book written by Boehm [BOEH81]. This model has become so popular that most commercial corporations and governmental agencies follow some basic variations of the waterfall model in the development of RTS; although different names are used to identify each of the different stages. For example, the system feasibility stage is often called requirement analysis or specifications; the product design stage is often called design specification, preliminary design, high-level design; the detailed design stage is often called program design, module design, etc. For the most part, all these methodologies are equivalent.

The main drawback to the waterfall model has been its emphasis on fully elaborated documents as completion criteria for the requirements and design phases. In practice, this has meant an early freeze in the software plans and requirements stage; which, in turn, led to the design and development of large quantities of unusable code based on incorrect or incomplete requirements [BOEH88].

To overcome this difficulty, a design methodology known as the Rapid-Prototyping Cycle was introduced by [LUQI89]. This methodology addressed the issue of ensuring that the RTS being proposed really met the user's needs by quickly building and evaluating a series of prototypes. A prototype is a concrete model of selected aspects of a proposed system. The potential users utilize these prototypes for a period of time and supply feedback to the developers concerning its strengths and weaknesses. This feedback is used to modify the initial requirements to reflect the real user needs. This thesis proposes that RTS be designed using the **Prototyping-Based Design Methodology** shown in figure 3-8. The Prototyping-Based Design Methodology is a modification of the Rapid-Prototyping Cycle in which two PAL-Based prototypes (i.e. Requirement and Design

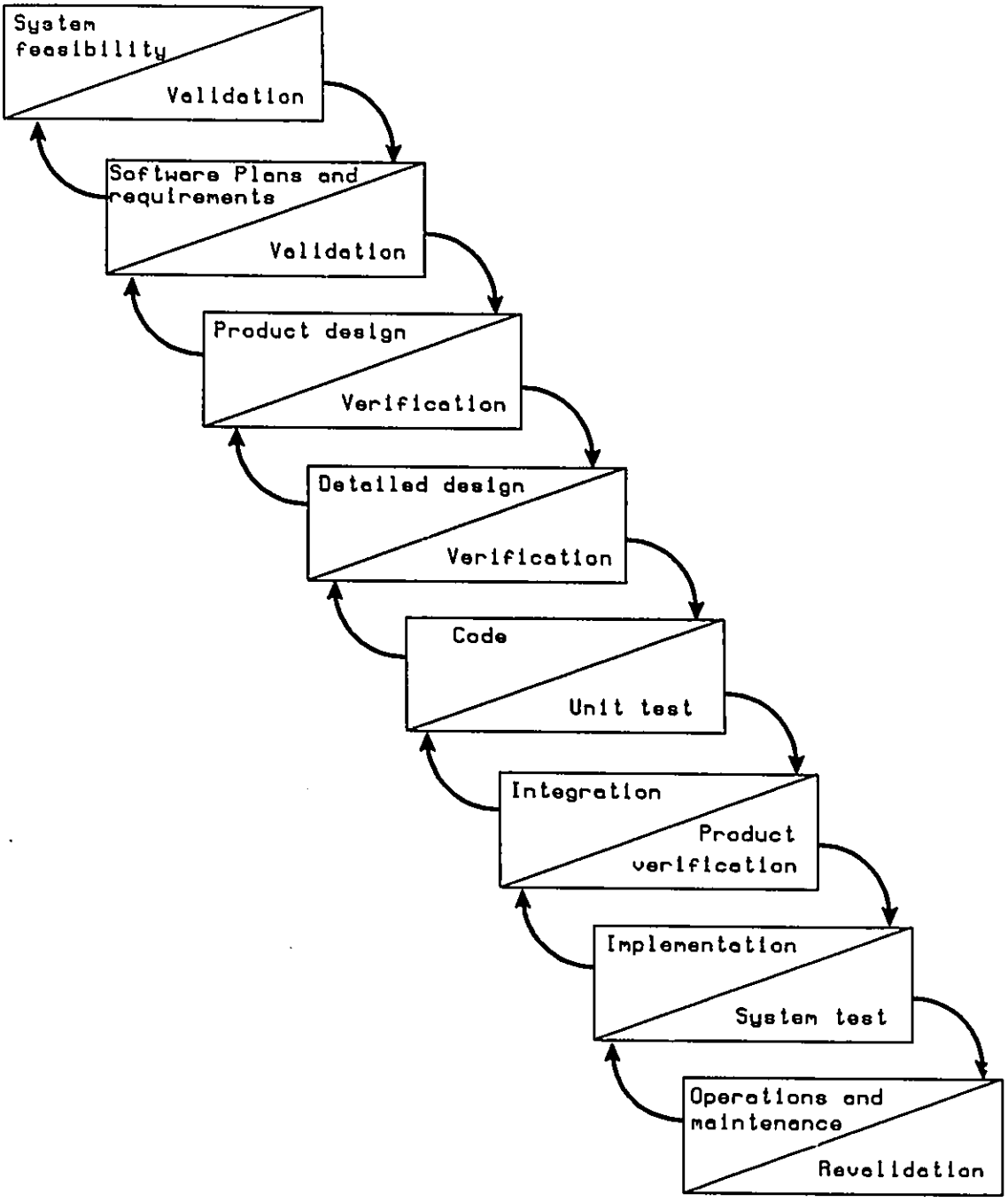


Figure 3–7 The Waterfall Model

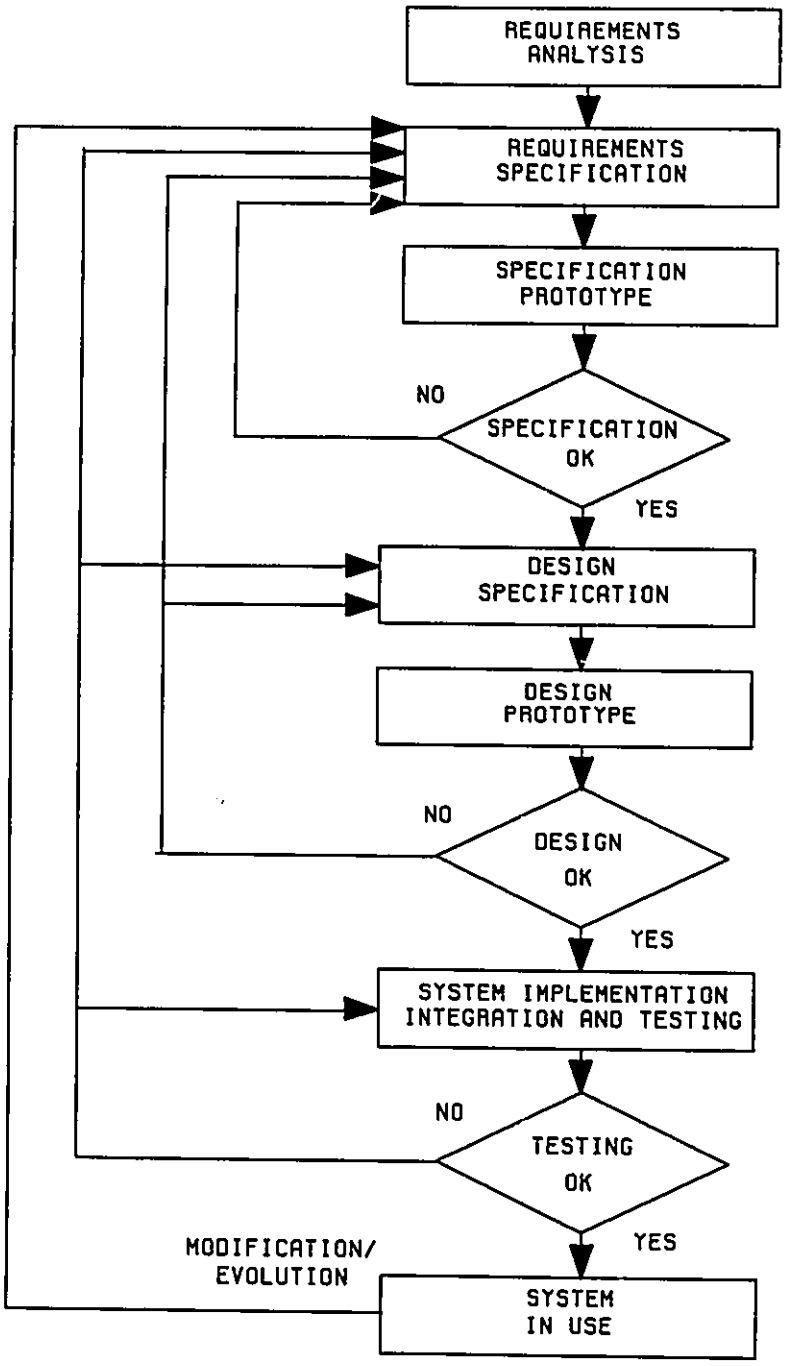


Figure 3-8 The Prototype-Based Design Methodology

Prototypes) have to be built.

The Prototyping–Based Design Methodology is composed of four major phases:

- The **Requirements Analysis**, in which the requirements of the controlled system are studied and understood;

- The **Requirements Specification**, which consists of an unambiguous specification of the functional/behavioral requirements of the RTS, as well as other requirements such as timing constraints, etc.... In this phase, a **Specification Prototype** of the behavioral/functional requirements is obtained as well;

- The **Design Specification**, which translates the Requirements Specification into a hardware and software specification; it includes the generation of a **Design Prototype** that includes the structure and functionality of the system;

- The **System Implementation, Integration and Testing**, which is a transformation of the Design Specification into a real implementation and its testing.

As RTS are often present in the field for extended periods of time (i.e. 30 years is common for avionic systems), an additional **System in Use** phase has been added at the bottom of figure 3–8 to represent the many modifications that the system will undergo throughout its lifetime. The purpose of the remainder of this section is to present how the PAL simulator can be used as the Specification Prototype and Design Prototype in conjunction with other PAL tools.

3.3.1 PAL–Based Specification Prototype

The objective of creating a Specification Prototype is to provide an early feedback in the design cycle and to ensure that the behavior of the system being designed is in accordance with the behavioral and functional requirements and furthermore, that these requirements were correctly specified. This prototype does not take into account any of the system constraints, such as performance desired, timing constraints or costs and does not assume any specific hardware platform. This phase is as much as possible implementation independent.

The generation of this prototype is possible if the specifications are done in an executable language such as PAL because they can be easily simulated using an interpreter of the language. Furthermore, this specification becomes a major part of the design specification and thus, saves time and costs in the development phase.

The Requirements Specification cycle is shown in figure 3-9. It consists of three major phases, each of them using a specific PAL tool:

1. Functional and Behavioral Specification using the PAL editor (PALED): This is the first phase and follows directly the requirement analysis of the problem. The PALED editor allows the user to enter the system specification either as PAL algebraic expressions or in pseudo-code format. It allows step-wise refinement of the specification by allowing abstractions to represent subexpressions which the user can define at a later stage. It performs syntactical and lexical checking of the entered specifications. The PALED editor is also capable of generating Process Activity Nets which are graphical representations of the PAL expressions and show the precedence relations of the various PAL elements. If errors are found, they must be corrected and the process repeated until no more errors are present. A more detailed discussion of the capabilities of the editor is available in the thesis of [CHUB90].

2. Temporal Logic Checking using the Verifier: In this phase the specification is checked using the Temporal Logic Verifier which will test it regarding temporal errors such as deadlocks, critical races and starvation. If errors are encountered, the functional specification will have to be redefined and the process goes back to step 1 above. A complete description of the verifier is available in the thesis of [HARV89].

3. Functionality and Behavior Checking using the PAL Simulator: In this phase, the specification is simulated using the PAL simulator assuming infinite resources. The PAL simulator is able to execute any processes written in PAL. The control flow of processes can be easily observed and checked. Using the PAL simulator, the user is able to define all events of the system and he is able to activate events individually to analyse specific processes. Checking the specification consists on running the system and seeing how it performs, how the processes interact and what are the effects on the environment, without checking specific timing or performance

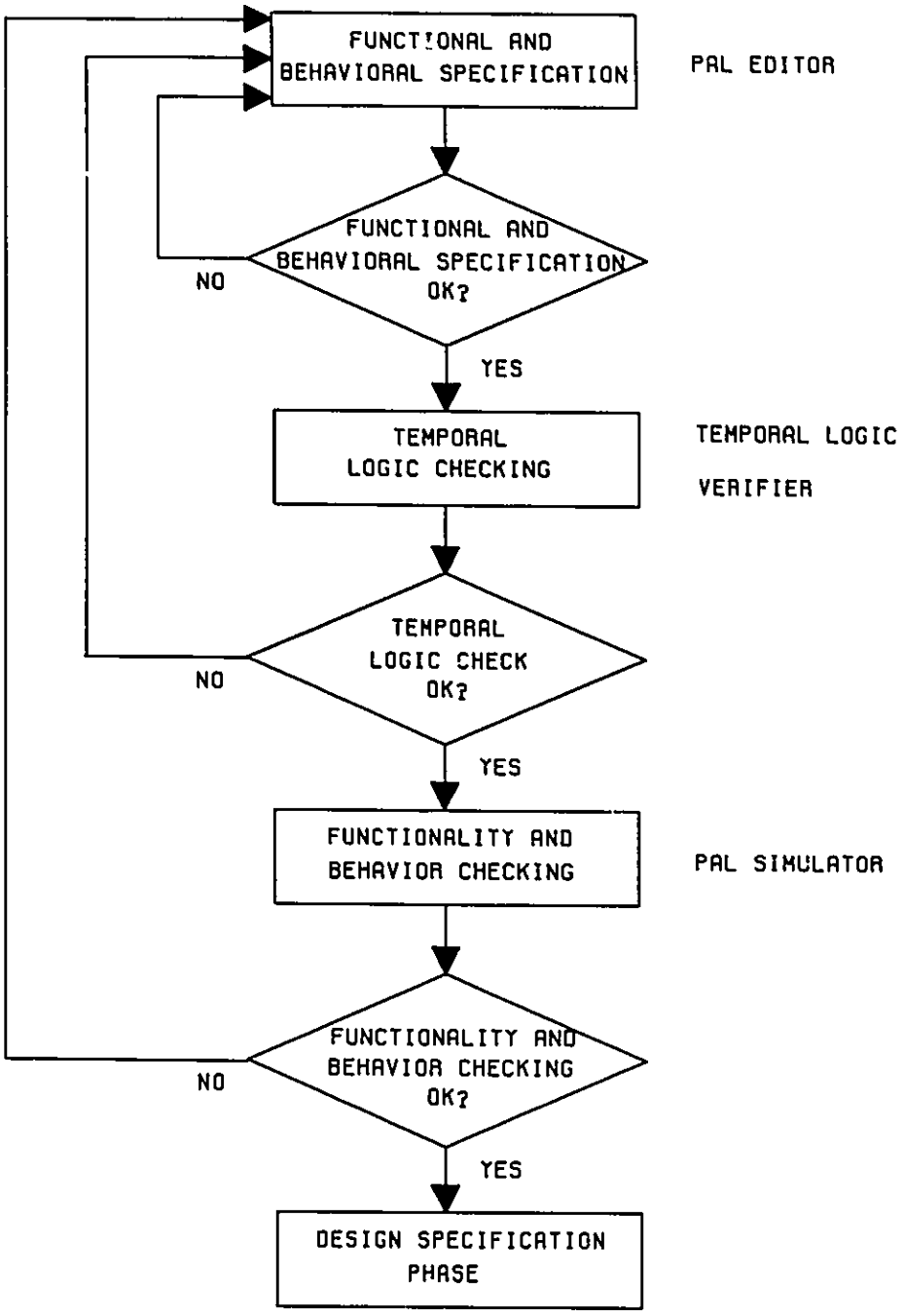


Figure 3–9 The Requirements Specification Cycle

requirements and without assuming any specific hardware platform. The logical and functional behavior can be checked and the user can see if this is really what he was expecting and compare the observed with the desired result. If the observed behavior is not acceptable, the specification has to be redefined and the process goes back to step 1. When the specification successfully passes through all phases of the Requirements Specification cycle, it is considered ready and can then be used and refined in the Design Specification phase, where all constraints will be checked. The PAL simulator will be explained in more detail in Chapter 4.

3.3.2 PAL as a Design Prototype

In this section, a methodology for building the Design Prototype using the PAL simulator is presented. This section describes the steps that can be taken to obtain such a prototype and which can be used to test the prototype with the PAL simulator. The tests will determine efficient operating parameters and other aspects of the final implementation. The Design Specification Cycle is shown in figure 3-10. It consists of two major phases:

1. Definition of the RTS: In this first stage, the following RTS characteristics have to be defined:

   a. the PAL processes: In this step, for each PAL process, a number of characteristics have to be defined: first, each process is defined using the PAL executable language and each process may be partitioned into activities and depending on the process, the constituent activities may be executed independently or in some precedence relation; second, each process can be categorized as sporadic or periodic depending on the type of initiating event; and third, each process may have a well defined deadline;

   b. the number and types of resources available: this step consists of an initial estimation of the number of resources that will be necessary to implement a specified solution. A detailed discussion of how to determine the number of resources may be found in the thesis of [SANM90].

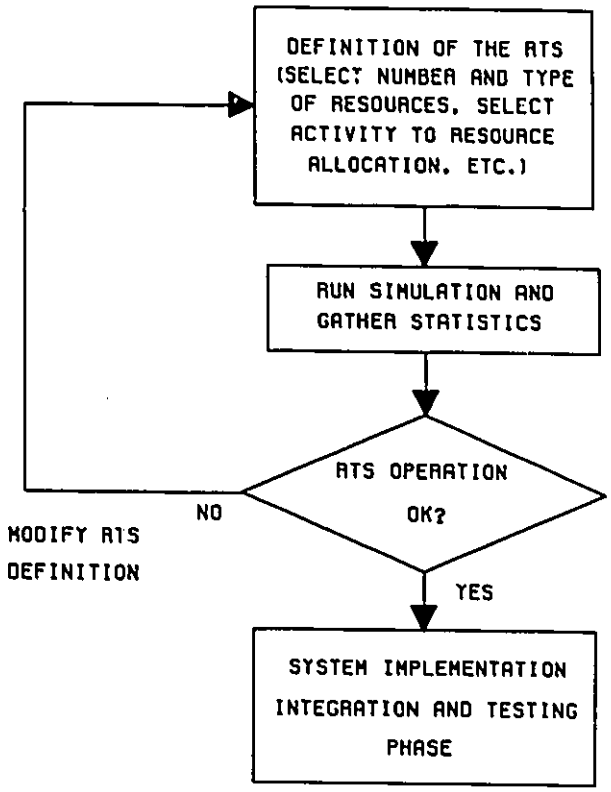


Figure 3–10 The Design Specification Cycle

c. the activity–to–resource allocation: information about which resource can execute a particular activity is contained in the activity–to–resource allocation table. There are two aspects to consider when assigning a resource to an activity. First, the designer has to check whether the activity requires a special purpose resource or not; this is the case where the activities have special requirements which can only be satisfied by specific resources (DSPs, for instance). In this case, a free resource with the required capabilities must be assigned right away. Second, if the activity does not require a special purpose resource, then a resource may be allocated in any number of different ways. Particularly, there are two interesting approaches for this choice: load balancing and sequential

allocation. In the case of load balancing, the assignment of a general purpose resource to a ready to run activity tries to achieve a good distribution of work among all resources. In the case of sequential allocation, it is assumed that all resources can execute all activities.

d. the activity scheduling scheme: the PAL simulator implements seven different activity scheduling rules: first-come first-served (FCFS), shortest activity first (SAF), longest activity first (LAF), least estimated laxity first (LELF), earliest due date (EDD), least estimated time to completion (LETC) and most estimated time to completion (METC). These activity scheduling rules will be discussed in more detail in chapter 4. The RTS designer must select one of these scheduling rules to run a simulation.

e. the external event characteristics: in general, the event arrival rates may be determined through knowledge of the environment.

2. Simulation and Statistics: Since the PAL Specification has been validated in the Requirements Specification cycle and the RTS has been defined in step one of the Design Specification cycle, the PAL simulator may now be used to run simulations and gather statistics about the RTS under study. Various statistics, such as the percentage of tardy processes, percentage of missed events and resource utilization levels, are gathered and provided to the RTS designer. In the event that the RTS operation is satisfactory to the system designer then the Prototyping-Based Design Methodology moves to the system implementation, integration and testing phase. In the case where the RTS is not satisfactory to the system designer, the process reverts back to stage one; and one or more of the RTS characteristics have to be modified. It is important to note that the evaluation and testing of a RTS for all possible operating parameters sometimes require too many different situations to be analysed and too many simulations to be performed. Because of the complexity of the Design Specification phase, this thesis only considers the problems related to scheduling. In particular, since RTS emphasize the study of time related characteristics, this thesis will investigate the effects of different dynamic scheduling rules on such scheduling criteria as the percentage of tardy processes. The rationale for using a simulation method in RTS design is not different from the rationale for simulation in dealing with any other complex system: short of testing alternative scheduling

policies in the actual RTS, there is no way to anticipate fully how different scheduling rules will affect RTS behavior.

It is important to note that although certain scheduling rules are expected to perform well for a specific RTS, these results may not be easily extended to a different application. Experimentation with the PAL simulator will make it possible to select a best scheduling rule for a particular RTS application in order to optimize a specific scheduling criteria.

Chapter 4

SIMULATION RESULTS

In section 4.1, a general description of the design of the PAL simulator will be presented. The principal reason for creating the PAL simulator was to investigate the effects of different dynamic scheduling rules on the design of multiple processor RTS which allow for the execution of complex responses. Section 4.2 presents a medium-sized RTS example. It was decided that, in terms of identifying trends, it is more meaningful to use a single medium-sized example and run a large number of simulations rather than use a large number of small examples. Section 4.3 presents selected results for this example. Section 4.4 ends with a few concluding remarks.

4.1 The Simulator

The simulator developed for this thesis had two main functions. First, the simulator serves as a tool for testing the effect of different scheduling schemes on the behavior of RTS with complex event responses. Second, the simulator serves as a model for the design of a general purpose multiactivity executive and it also serves as an essential component of the prototyping-based design cycle methodology outlined in chapter 3. Because at the time of building this simulator, we did not have a PAL editor and we wanted to test it with various PAL expressions, it was decided to implement the simulator in Franz Lisp* which allows for easy symbol manipulation. The underlying hardware used was the Digital Equipment Vax Station II** Unix*** based Ultrix 32. The PAL simulator software is available in a separate user guide [LACH88].

Because of the time constraints, it was decided to implement only the basic composition constructs, namely sequence, concurrency, repeat until and case. Another reason for not including additional composition constructs is that the effects of various scheduling rules (i.e. the main subject of this thesis) can be checked using only sequence and concurrency constructs. All of the PAL primitives are

---

*Franz Lisp is copywrited by "The Regents of the University of California".

**Vax Station II is a trademark of the Digital Equipment Corporation

***Unix is a trademark of AT&T Bell Laboratories.

implemented in the simulator.

The PAL simulator consists of five separate functional modules: a simulation definition and initialization module, an event generator, a PAL expression control sequencer, an activity scheduler and a systems statistics module. Figure 4-1 shows the five functional modules of the PAL simulator.

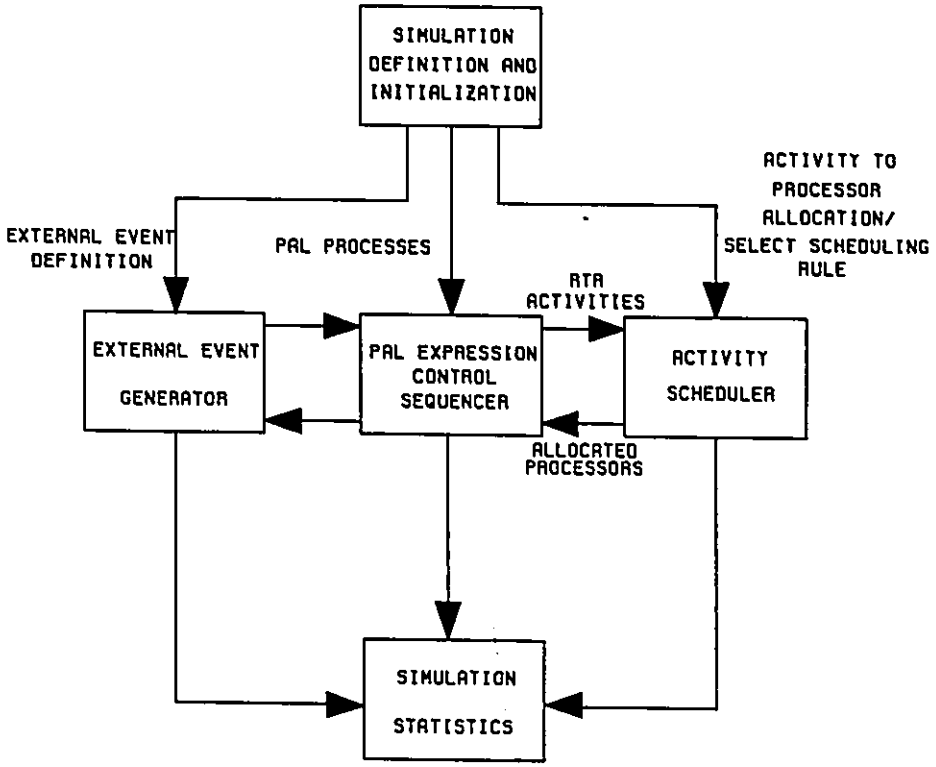


Figure 4-1 The Modules of the PAL Simulator

Beyond the obvious need for modularity, the simulator was logically partitioned into five parts for the following reasons:

- o the simulator serves as a general design tool that can be adapted to many different types of systems. While PAL expression sequencing and execution remains similar throughout different applications, the event arrival rates and the scheduling requirements of various systems can be quite different; and

- in a centrally coordinated system, the PAL expression interpreter, which is part of the PAL expression control sequencer, and the activity scheduling software may be used as the basic modules of the system executive.

The five functional modules of the PAL simulator and the interactions between them are described below:

1. The Simulation Definition and Initialization Module

This module is responsible for the following functions: to initialize all variables associated with the simulation environment; to obtain from the user the amount of simulation time desired; to prompt the user for the PAL processes, their due dates and estimated time to completion (ETC). This module also prompts the user for the filename of two input files: the PAL Expression Information file and the Activity to Processor Allocation file. The first file includes information about the PAL expression primitives and composition constructs; while the second file contains information about the activity to processor allocation and the activity execution times.

The user is also asked for information about the external events of the system under study: the number of events, their arrival rates, and for each event, a maximum queue length and the maximum allowable time an event may be queued before being rejected. The module is also responsible for gathering some information about the messages included in the system: the number of messages and a maximum queue length for each message. The user is then asked to select an activity scheduling rule.

Once all these simulation parameters have been fixed, the user is then prompted for the desired mode of operation, either single-step or continuous. In single step simulation, the simulation advances one clock tick at a time. This enables the user to carefully monitor certain simulation aspects to see if the statistics produced correspond to the expected results. In the continuous simulation mode, the simulator runs from beginning to the end of the simulation without interruption.

2. The External Event Generator

The external event generator developed for this simulator handles only sporadic events with Markovian arrivals. In the PAL simulator, events are queued as they occur. At every clock tick of simulation, for each event included in the RTS under study, the event generator simulates Poisson arrivals by generating a random number between 0 and 1 and then compares that random number against the probability of arrival of each event as set by the user. If the random number is smaller than the event probability for a specific event then the event queue is incremented by one. If the number of events in an event queue exceeds the maximum number of events to be queued as specified by the user, then the number of rejections is incremented by one.

3. The PAL Expression Control Sequencer

The PAL expression control sequencer module receives from the event generator the event queues associated with each event. The control sequencer also has access to a list of PAL expressions and their associated due dates and it provides, as a single output to the activity scheduler module, a global ready to run activity queue. Since a PAL expression editor and a syntax checker have been developed as part of another individual's thesis [CHUB90], this simulator assumes that the PAL expressions, entered at system start-up, are well formed and free of errors.

In order to simplify the simulator, the system manager and the PAL expression managers were combined into a single PAL Expression Control Sequencer. At every clock tick and for every PAL expression, in turn, the PAL expression control sequencer carries out the following sequence of actions:

1. it initializes the arrays, variables and flags that will enable the PAL expression interpreter to parse the different symbols in a PAL expression;

2. it calls on a PAL expression interpreter to parse and execute the various PAL expression lists and symbols;

3. it places the ready to run activities into a single ready to run activity queue; and

4. it updates the appropriate system statistics.

4. The Activity Scheduler

The activity scheduler is the software module that enables ready-to-run activities to be executed by idle processors. In order to accomplish this, it must know at all times the status of all processors (e.g. if the processor is idle or busy and if busy which activity it is executing and from which PAL expression in order to update network/activity/processor utilization statistics).

Instead of implementing some of the well known dispatching rules from OR such as LLF (least laxity first), most work remaining (MWKR) and least work remaining (LWKR), we opted for similar rules, respectively named, least estimated laxity first (LELF), most estimated time to completion (METC) and least estimated time to completion (LETC). This is due to the fact that **work remaining** is easily calculated in OR theory; because of single operation to resource allocation (i.e. one only has to add up the execution times of all the operations left to execute within a job). All three dispatching rules (LLF, MWKR and LWKR) use this concept of work remaining. In the case of multiactivity systems, activities may be allocated to more than one processor (i.e. with possible different execution times); therefore making the calculation of work remaining a much more difficult task. This is why the LELF, METC and LETC dispatching rules were implemented using the **estimated time to completion** (ETC) concept. ETC is a number entered by the user at system start-up, for each process, that represents his estimate of how long it takes to execute a particular process. **Estimated Laxity** is therefore initialized, at event arrival, to the difference between process due date and ETC.

The activity scheduler can implement a number of different scheduling algorithms. The following is a description of the seven dispatching rules implemented in the PAL simulator:

1. FCFS, First-Come First-Served: the activities are allocated to the idle processors according to the order in which they arrived in the ready to run queue.

2. SAF, Shortest Activity First: when a processor becomes free, the ready to run activity that can be executed in the shortest amount of time on a particular processor is allocated first.

3. LAF, Longest Activity First: the ready to run activity that can be executed in the longest amount of time on a particular idle processor is allocated first.

4. EDD, Earliest Due Date: the activities of the process with the closest deadline are allocated first.

5. LELF, Least Estimated Laxity First: the activities with the least estimated laxity are allocated first.

6. LETC, Least Estimated Time to Completion: the activities of the process with the least estimated time to completion are allocated first.

7. METC, Most Estimated Time to Completion: the activities of the process with the most estimated time to completion are allocated first.

5. System Statistics

System statistics are provided to the user. The information is provided under the following headings:

1. run duration: the duration of the simulation is stated.

2. events: the event handling data shows a set of statistics for each of the events in the system. These include the number of occurrences of an event, the percentage of time an event was rejected and the maximum and average waiting time in an event queue.

3. processor: the processor data shows a set of statistics for each processor. These include the percentage of time that the processors are busy and the number of times each activity was executed on a given processor;

4. network: the network data show a set of statistics for each PAL process. These include the number of times each PAL expression was awakened, maximum/average completion times and the percentage of times each PAL expression was late in meeting its due date.

5. ready to run activity queue: These statistics refer to the maximum/average queue length before and after the scheduling was performed on the ready to run activity queue.

4.2 The Example

The selected RTS example models an application with 3 PAL expressions (PAL 1, PAL 2 and PAL 3) and 25 activities total. The PAL expressions chosen for this RTS example are complex responses. These complex responses are decomposed into 25 activities which are characterized by sequential and concurrency precedence relations. No other PAL primitive or composition construct were allowed within the RTS example. Multiple simulations of the example were to be run and data gathered to investigate the effects of different dynamic scheduling rules on the design of the RTS which allows for the execution of complex responses. Figure 4-2 presents the graphical and pseudo-code representation of the three PAL processes.

Symbols S_1, S_2 and S_3 represent three different sporadic events. Symbols A_1 through A_{25} represent 25 different activities; while symbol E represents the end primitive. In this example, there are six processors available to execute these activities. An explanation of why six processors were selected for this RTS example will be given later on in the section.

In order to gather data, it was decided to run multiple simulations of the example while varying different operating parameters. The parameters that are varied during simulation can be grouped under four headings:

1. activity scheduling rules;

2. activity to processor allocation;

3. event arrival rates; and

4. event rejection policy.

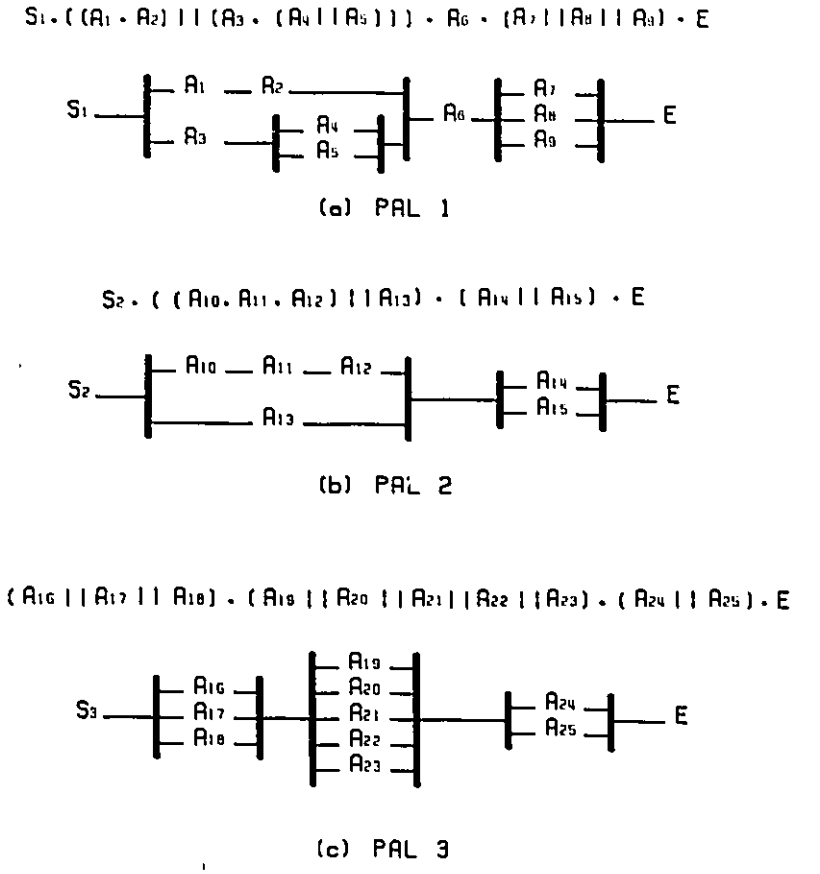


Figure 4–2 Graphical and Pseudo–Code Representation of the RTS Example

1. Activity Scheduling Rules

The simulator implements seven different scheduling rules as described previously in section 4.1. For sake of completeness, these scheduling rules are: First–Come First–Served (FCFS), Shortest Activity First (SAF), Longest Activity First (LAF), Least Estimated Laxity First (LELF), Earliest Due Date (EDD), Least Estimated Time to Completion (LETC) and Most Estimated Time to Completion (METC).

2. Activity to Processor Allocation

In this RTS example, each activity was allocated to two processors. The activity execution times were set up in such a manner that the total load assigned to each processor was about the same. Furthermore, two activity to processor allocation schemes are implemented; namely, a **good** and a **bad** one. Figure 4–3 presents the good activity to processor allocation in graphical form; while figure 4–4 shows the same allocation as a table. In figure 4–3, each four tuple (PX,B,PY,C) associated with an activity A_z represents the fact that activity A_z has been allocated to processors PX and PY and may be executed by processor PX in B time units or by processor PY in C time units.

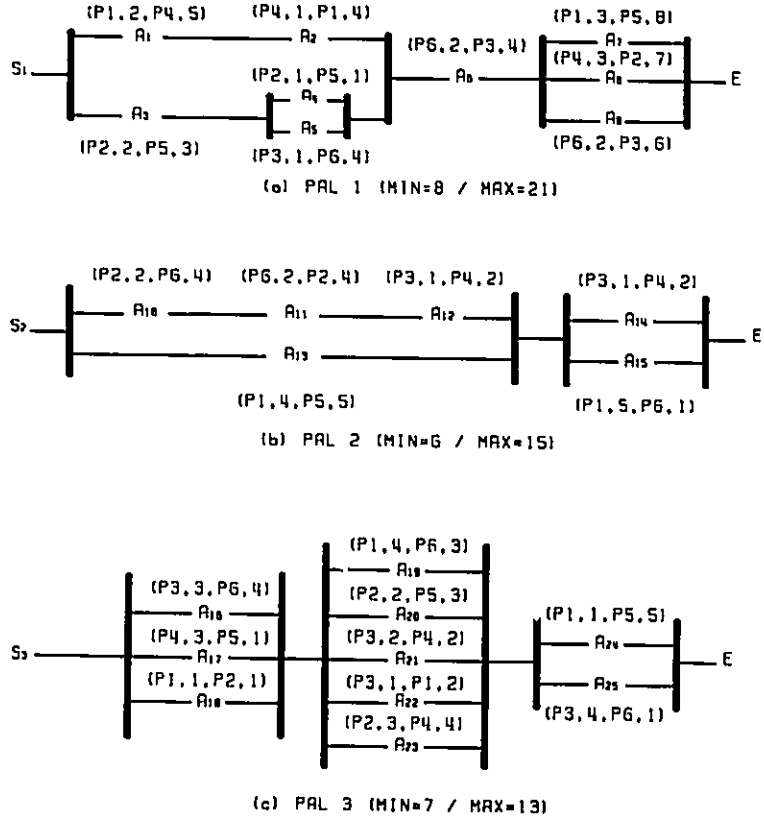


Figure 4–3 Graphical Representation of the Good Activity to Processor Allocation for our RTS Example

Activity	Processor	Execution Time	Activity	Processor	Execution Time
A1	1	2	A14	3	1
A1	4	5	A14	4	2
A2	1	4	A15	1	5
A2	4	1	A15	6	1
A3	2	2	A16	3	3
A3	5	3	A16	6	4
A4	2	1	A17	4	3
A4	5	1	A17	5	1
A5	3	1	A18	1	1
A5	6	4	A18	2	1
A6	3	4	A19	1	4
A6	6	2	A19	6	3
A7	1	3	A20	2	2
A7	5	8	A20	5	3
A8	2	7	A21	3	2
A8	4	3	A21	4	2
A9	3	6	A22	1	2
A9	6	2	A22	3	1
A10	2	2	A23	2	3
A10	6	4	A23	4	4
A11	2	4	A24	1	1

Figure 4–4 Good Activity to Processor Allocation

Activity	Processor	Execution Time	Activity	Processor	Execution Time
A11	6	2	A24	5	5
A12	3	1	A25	3	4
A12	4	2	A25	6	1
A13	1	4			
A13	5	5			

Figure 4-4 Good Activity to Processor Allocation [continued]

As can be expected, the good activity to processor allocation (GA) makes efficient use of the data independence between process activities by allocating potentially concurrent activities to different processors. For example, an examination of PAL 1 shows that data independent activities A_2, A_4 and A_5 are allocated, respectively, to different processor pairs each, namely processors 1 and 4, processors 2 and 5, and processors 3 and 6. In the same fashion, potentially concurrent activities A_7, A_8 and A_9 have all been allocated to different processor pairs. With no event queuing and assuming no contention for processor execution time, close examination of PAL 1 reveals a minimum and maximum process computation time of 8 and 21 time units, respectively.

The data independent activities of PAL 2 have also been allocated to different processor pairs. For example, activity A_{12} has been allocated to processors 3 and 4; while potentially concurrent activity A_{13} has been allocated to processors 1 and 5. Activities A_{14} and A_{15} have also been allocated to different processor pairs. By adding up the execution times of PAL 2, assuming no event queuing and no contention for processor execution time, it can be seen that the minimum and maximum process computation time is set to 6 and 15 time units, respectively.

In PAL 3, data independent activities A_{16}, A_{17} and A_{18} have also been allocated to different processor pairs. The same applies to activities A_{24} and A_{25}. In the case of activities A_{19}, A_{20}, A_{21}, A_{22} and A_{23} which all become ready to run at the same time, it would have required 10 processors to allocate the five potentially

concurrent activities to different processor pairs. In this RTS example, it was decided to use only six processors to show the effects of contention for processor execution time. It is estimated that the effects of activity contention for processor execution time will become increasingly important as the event arrival rates increase. A close look at PAL 3, assuming no event queuing and no contention for processor execution time reveals a minimum and maximum process computation time of 7 and 13 time units, respectively.

On the other hand, figure 4-5 shows the bad activity to processor allocation (BA) in graphical form; while figure 4-6 shows the same allocation in table form. Notice that the activity execution times are identical for both the GA and BA. The bad activity to processor allocation is bad because it does not make use of the data independence between activities by allocating potentially concurrent activities to the same processors.

For example, in PAL 1, data independent activities A_1 and A_3 have both been allocated to the same processors 1 and 4. It is expected that this allocation will often have the effect of increasing process completion times and the percentage of tardy processes. In the same fashion, activities A_7 and A_8 have also been allocated to the same processor pair. The same observations can be extended to other potentially concurrent activities of PAL 2 and PAL 3 which have also been allocated to the same processor pair. It is expected that the effects of the bad activity to processor allocation will be felt at higher event arrival rates when the contention for processor utilization will increase significantly.

3. Event Arrival Rates

Three different sets of random arrival rates were chosen for sporadic events S_1, S_2 and S_3. First, an initial data gathering would be accomplished with all event arrival rates set at 0.02. Second, the arrival rates were to be set at 0.04; and, finally, the event arrival rates were to be set at 0.06. The event arrival rates of 0.02, 0.04 and 0.06 were chosen such that they would create low, medium and high levels of contention for processor utilization.

4. Event Rejection Policy

As mentioned in section 4.1, the PAL simulator allows for event queuing. Simulation data was gathered with two distinct event rejection policies. First, a No

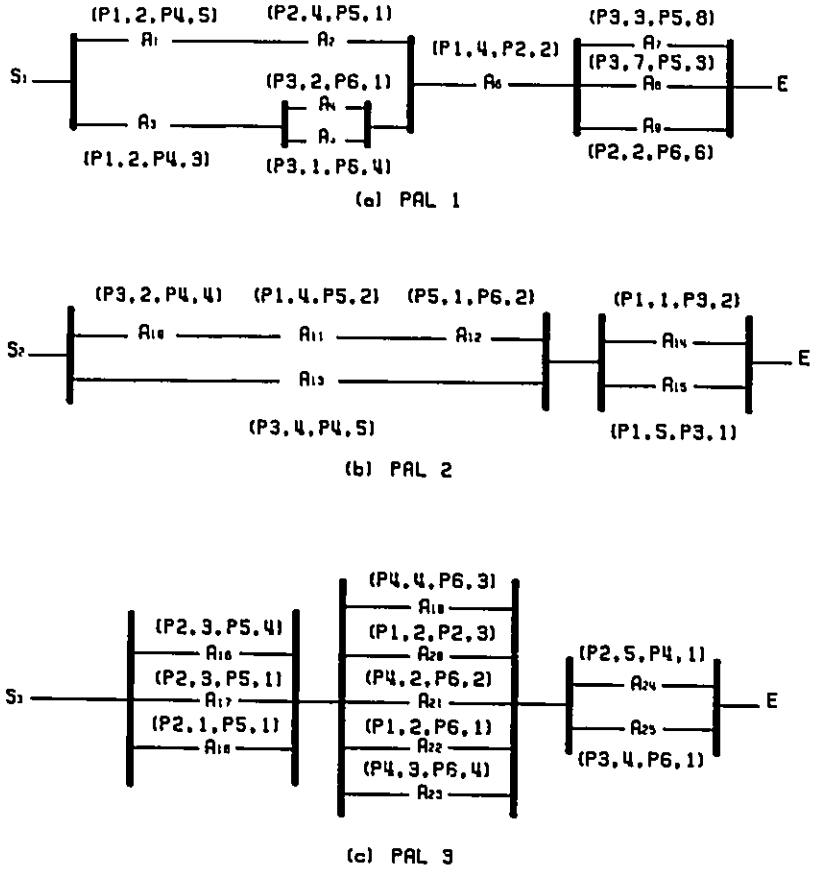


Figure 4–5 Graphical Representation of the Bad Activity to Processor Allocation

Activity	Processor	Execution Time	Activity	Processor	Execution Time
A1	1	2	A14	1	1
A1	4	5	A14	3	2
A2	2	4	A15	1	5

Figure 4–6 Bad Activity to Processor Allocation

Activity	Processor	Execution Time	Activity	Processor	Execution Time
A2	5	1	A15	3	1
A3	1	2	A16	2	3
A3	4	3	A16	5	4
A4	3	2	A17	2	3
A4	6	1	A17	5	1
A5	3	1	A18	2	1
A5	6	4	A18	5	1
A6	1	4	A19	4	4
A6	2	2	A19	6	3
A7	3	3	A20	1	2
A7	5	8	A20	2	3
A8	3	7	A21	4	2
A8	5	3	A21	6	2
A9	2	2	A22	1	2
A9	6	6	A22	6	1
A10	3	2	A23	4	3
A10	4	4	A23	6	4
A11	1	4	A24	2	5
A11	5	2	A24	4	1
A12	5	1	A25	3	4
A12	6	2	A25	6	1

Figure 4–6 Bad Activity to Processor Allocation [continued]

Activity	Processor	Execution Time	Activity	Processor	Execution Time
A13	3	4			
A13	4	5			

Figure 4-6 Bad Activity to Processor Allocation [continued]

Rejection (NR) policy where all events are queued and no event rejection is carried out. Second, a **Rejection=20** (R=20) policy where events are rejected if they have been queued for more than 20 simulation time units.

4.3 Simulation Results

The purpose of this section is to present some key simulation results for the RTS example presented in section 4.2 under different operating parameters. For a more elaborate look at all simulation results obtained in this thesis, the reader is referred to annexes A and B. The data shown in annexes A and B was obtained by averaging out the results of five simulation runs. Each simulation run ran for 2000 time units. In annex A, figures A-1 through A-36 show the percentage of processes tardy versus increasing due dates for the seven different activity scheduling rules, a specific PAL expression (PAL 1, PAL 2 or PAL 3), a particular activity to processor allocation (i.e. Good (GA) or Bad (BA)), an event arrival rate (0.02, 0.04 or 0.06) and an event rejection policy (i.e. No Rejection (NR) or Rejection=20 (R=20)). For example, with an event arrival rate set at 0.02, a good activity to processor allocation (GA) and a no event rejection policy (NR), the average results of these simulation runs can be found at Annex A figure A-1 for PAL 1, at figure A-13 for PAL 2 and at figure A-25 for PAL 3.

Annex B, pages B-1 to B-46, show the tardiness data corresponding to the figures of Annex A. In addition to this, annex B provides additional simulation statistics regarding event queuing, processor utilization, PAL expression completion times and ready to run activity queue related information. The effect of varying various parameters (event arrival rates, activity scheduling rules, activity to processor allocation and event rejection policy) are now presented.

Event Arrival Rates

Figures 4-7 and 4-8 show the effects of increasing event arrival rates on process tardiness. In figure 4-7, it can be easily seen that as the event arrival rate increases so does the percentage of tardy processes for a particular due date. This set of data was obtained using the FCFS activity scheduling rule, the good activity to processor allocation (GA) and the no event rejection policy (NR) for PAL expression 1.

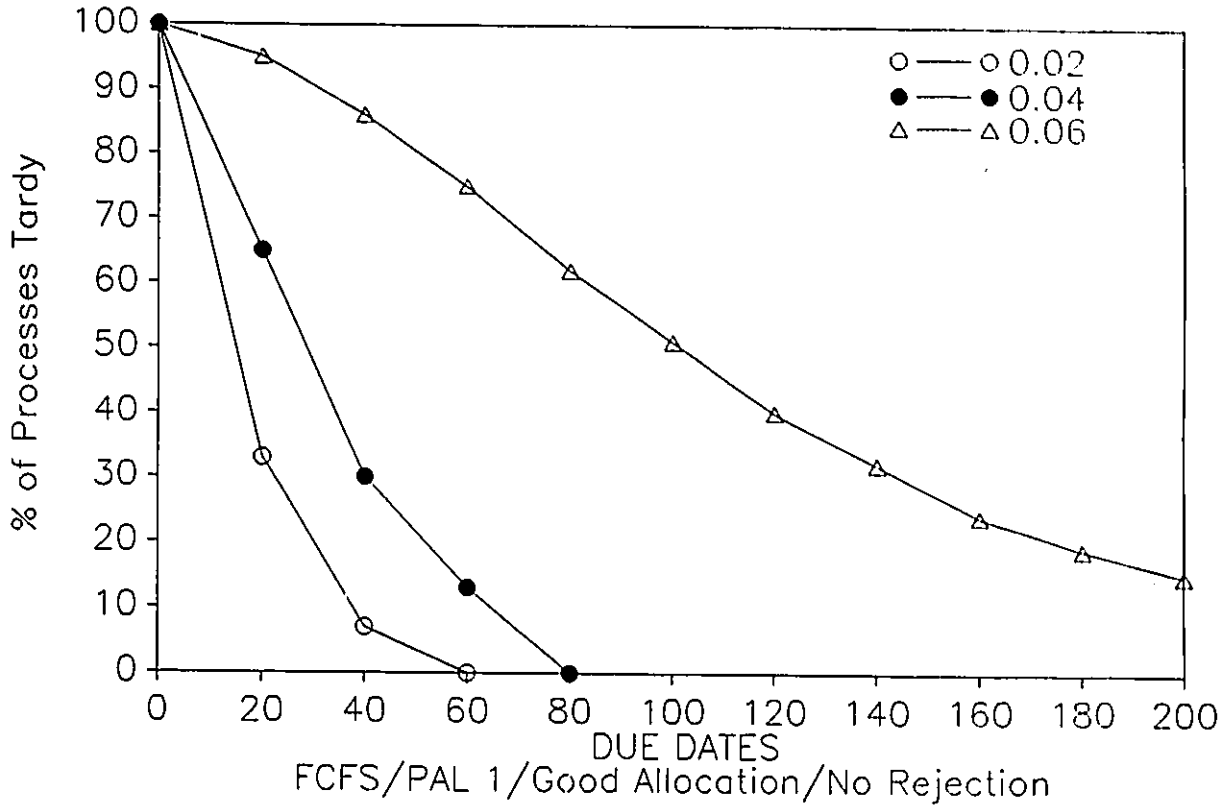


Figure 4-7 A Graph of the Effects of Increasing Event Arrival Rates on Process Tardiness

Event Arrival Rates	0.02	0.04	0.06
Due Dates	Percentage of Processes Tardy		
0	100	100	100
20	33	65	95
40	7	30	86
60	0	13	75
80	0	0	62
100	0	0	51
120	0	0	40
140	0	0	32
160	0	0	24
180	0	0	19
200	0	0	15
Average Completion Time	22	35	118
Average Event Queuing Time	2	9	51
Average Processor Utilization (%)	26	49	72
Average Ready to Run Activity Queue Length	0.6	1.0	2.2

Figure 4–8 The Effects of Increasing Event Arrival Rates on Different Scheduling Criteria

For example, with a due date of t=60, with the event arrival rate set at 0.02, there are no tardy processes; while, with the event arrival rates set at 0.04 and 0.06, respectively, the percentage of tardy processes increases to 13% and 75%. There are

many reasons for this increase in process tardiness. First, as shown by the data of figure 4-6, as the event arrival rates increase, an increasing amount of time is spent in the event queue (i.e. the average event queuing time increases from 2 to 9 then 51 for event arrival rates of 0.02, 0.04 and 0.06 respectively). Second, the average processor utilization also increases from 26% to 49% to 72% when the event arrival rates increase from 0.02 to 0.04 to 0.06. It is for this reason that the ready to run activities spend more time in the ready to run queue as the average ready to run queue length increases from 0.6 to 1.0 to 2.2 when the event arrival rates increase from 0.02 to 0.04 to 0.06.

Activity Scheduling Rules

As discussed in chapter 2, one of the most commonly used scheduling rules is the FCFS rule. This rule will be our baseline. All comments regarding performance improvements are made relative to the FCFS rule.

At the low event arrival rates of 0.02 and 0.04, for the three PAL expressions, with a good activity to processor allocation (GA) and a no event rejection (NR) policy, there is no significant difference in the tardiness data gathered for the different activity scheduling rules (see annex A, figures A-1, A-5, A-13, A-17, A-25 and A-29; and annex B, pages B-1, B-2, B-9, B-10, B-23, B-24, B-27, B-35).

However, with the higher event arrival rate of 0.06, figures 4-9 and 4-10 show that the choice of a particular scheduling rule can provide a significant improvement over the baseline FCFS rule. For example, with a due date of t=40, the use of the SAF heuristic rule produces 30% less tardy processes than the FCFS rule. Figure 4-9 also shows that other rules, such as LELF, EDD and LETC, also lead to much improved tardiness data. Figure 4-10 shows that these scheduling rules lead to performance enhancements over the FCFS rule in other scheduling criteria such as the minimization of the average process completion times and the average event queuing times. These results were obtained for PAL 3 with a good activity to processor allocation (GA) and a no event rejection (NR) policy. Tardiness data for the LAF and METC scheduling rules were not shown in figures 4-9 and 4-10 as they did not lead to performance improvements over the FCFS rule. For a more detailed look at the results obtained for PAL 1, PAL 2 and PAL 3 with GA and NR, see annex A, figures A-9, A-21 and A-33; and annex B, pages B-17, B-18.

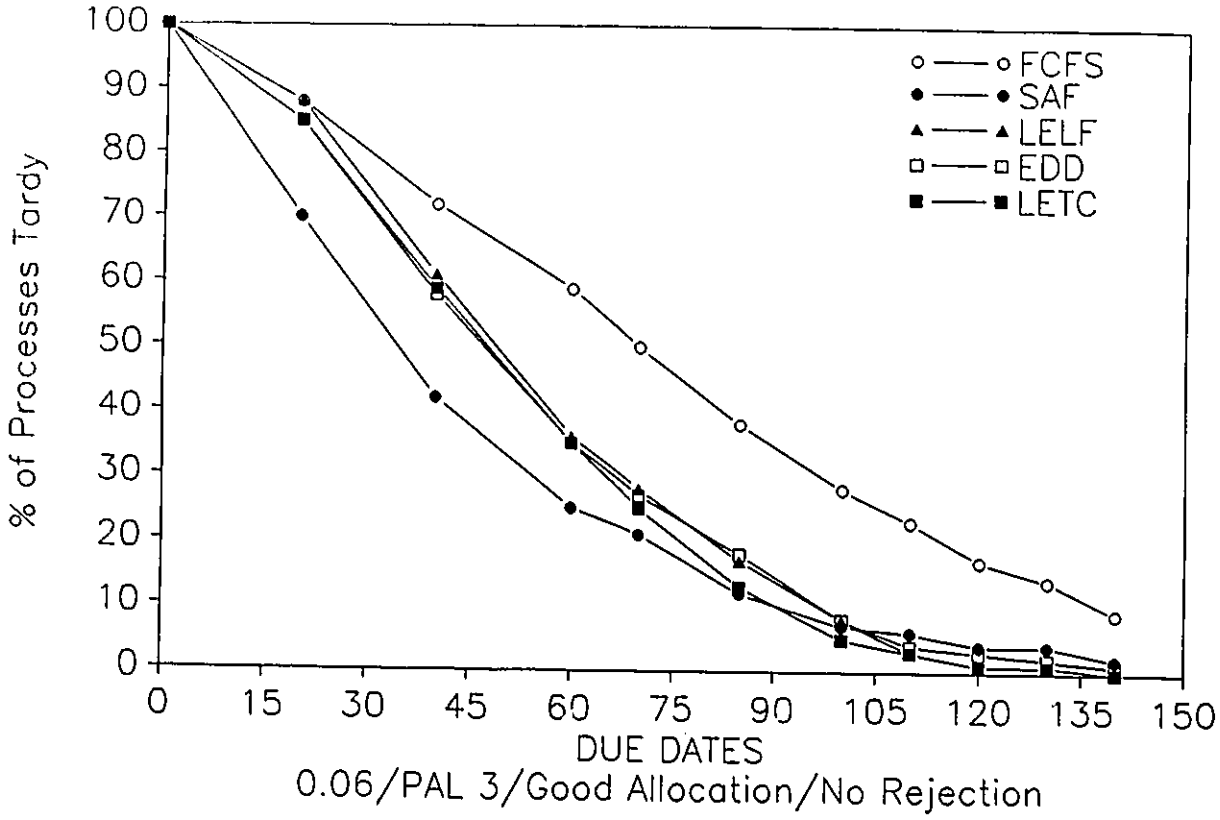


Figure 4–9 The Effects of Different Scheduling Rules on Process Tardiness

	% of Processes Tardy				
Due Dates	FCFS	SAF	LELF	EDD	LETC
0	100	100	100	100	100
20	88	70	88	85	85
40	72	42	61	58	59
60	59	25	36	35	35
70	50	21	28	27	25

Figure 4–10 The Effects of Varying Scheduling Rules on Different Scheduling Criteria

	% of Processes Tardy				
85	38	12	17	18	13
100	28	7	8	8	5
110	23	6	3	4	3
120	17	4	1	3	1
130	14	4	1	2	1
140	9	2	0	1	0
Average Time of Completion	79	45	53	53	58
Average Event Queue Time	33	16	19	19	18
Average Processor Utilization (%)	72	66	72	72	72

Figure 4–10 The Effects of Varying Scheduling Rules on Different Scheduling Criteria [continued]

B–31 and B–43.

The fact that the SAF, LELF, EDD and LETC scheduling rules seem to perform better than other scheduling rules at higher event arrival rates and processor utilizations parallels the results obtained in job shop scheduling by Conway and others (see page 2–14). In their case, the scheduling rules used had been the SPT (shortest processing time), the S/OPN (slack per operation), the Earliest Due Date (EDD) and the Least Work Remaining (LWKR) rules.

It is important to note that although certain scheduling rules (i.e. SAF, EDD, LELF, LETC, etc.) performed better for our medium-sized RTS example; these results can not be easily extended to a different application. Experimentation with the PAL simulator will make it possible to select a "best" scheduling rule for a particular RTS application in order to optimize some scheduling criteria.

Activity to Processor Allocation

As shown in figure 4-11, with the event arrival rates set at 0.02 and 0.04, the bad activity to processor allocation (BA) yields only a very small increase in the number of processes that are tardy. The results presented in figure 4-11 were obtained with the FCFS scheduling rule and are valid for PAL 1 and the no event rejection (NR) policy. The effect of changing from GA to BA, with event arrival rates of 0.02 and 0.04, was also negligible for all other scheduling rules except for the LAF rule which performed significantly worse. For a more detailed look at the effects of changing to a bad activity to processor allocation (BA) with arrival rates of 0.02 and 0.04, and a no event rejection policy (NR), see annex A, figures A-3, A-7, A-15, A-19, A-27 and A-31; and annex B, pages B-5, B-6, B-13, B-14, B-25, B-29, B-37 and B-41.

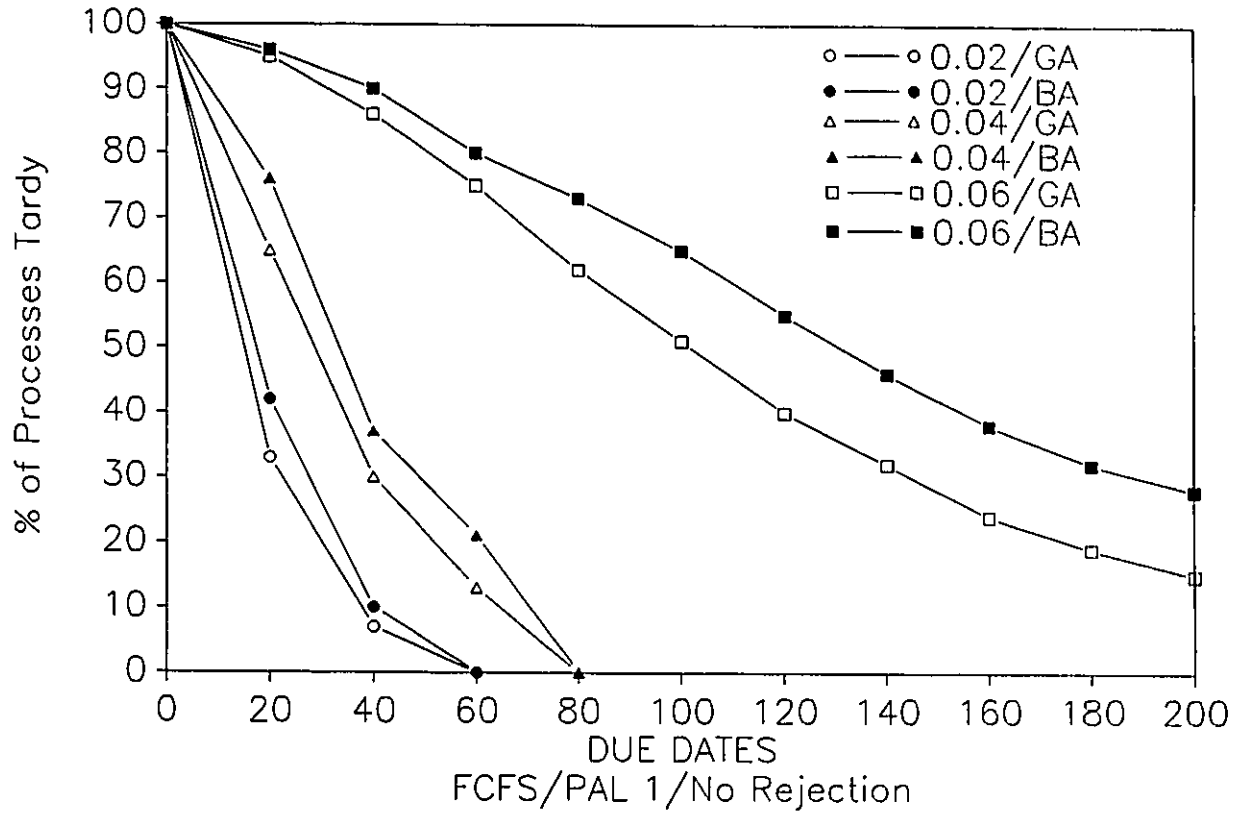


Figure 4-11 The Effects of Different Activity to Processor Allocations on Process Tardiness

With the event arrival rate set at 0.06, figure 4–11 shows that a BA yields a significant increase in the percentage of processes that are tardy when compared to a GA. Similarly, all other scheduling rules performed significantly worse when using a BA as opposed to using a GA. When comparing the other scheduling rules to our baseline FCFS rule, it is interesting to note that, except for the LAF scheduling rule which performed significantly worse than the FCFS rule, the other rules did not consistently perform better or worse for all three PAL expressions. For example, although the SAF rule produced a much smaller number of tardy processes than the FCFS rule for PAL 1 and PAL 2, it performed much worse than FCFS for PAL 3. It would seem that the use of a BA far outweighs the potential benefits of using alternative scheduling rules such as SAF, LELF, etc.... For a more detailed look at the effects of a BA, with an event arrival rate of 0.06, for all three PAL expressions, see annex A, figures A–11, A–23 and A–35; and annex B, pages B–20, B–21, B–33 and B–45.

Event Rejection Policy

The effects of an event rejection policy are not visible for the low event arrival rate of 0.02. However, as soon as the arrival rate reaches 0.04 or 0.06, there is a significant difference due to the event rejection policy. Figure 4–12 shows the effects of the GA or BA event rejection policy on process tardiness. In addition to the process tardiness data, figure 4–13 shows the effects of event rejection on different scheduling criteria. FCFS is the activity scheduling rule and PAL 1 is the process used to obtain these results.

For example, looking at figure 4–10, with a due date of t=40, for 0.04/GA/NR, 30% of processes do not meet their due dates; while with 0.04/GA/R=20 and an event rejection rate of 15%, there are no tardy processes for PAL 1. At the higher event arrival rate of 0.06, the effects of event rejection are even more significant. For example, looking at figures 4–10 and 4–11, with a due date of t=40, one can see that for 0.06/GA/NR; 86% of processes are tardy; while for 0.06/GA/R=20 and an event rejection rate of 24%, there are no tardy processes. For this due date of t=40, the difference in % of tardy processes is 86%.

In the same fashion, it can be observed that one can overcome the negative effects of a BA by using an event rejection policy of R=20. In figure 4–10, it can be observed that for 0.06/BA/R=20, with a due date of t=40 and an event rejection rate of 36%, there are 4% of tardy processes; while with 0.06/GA/NR, there are

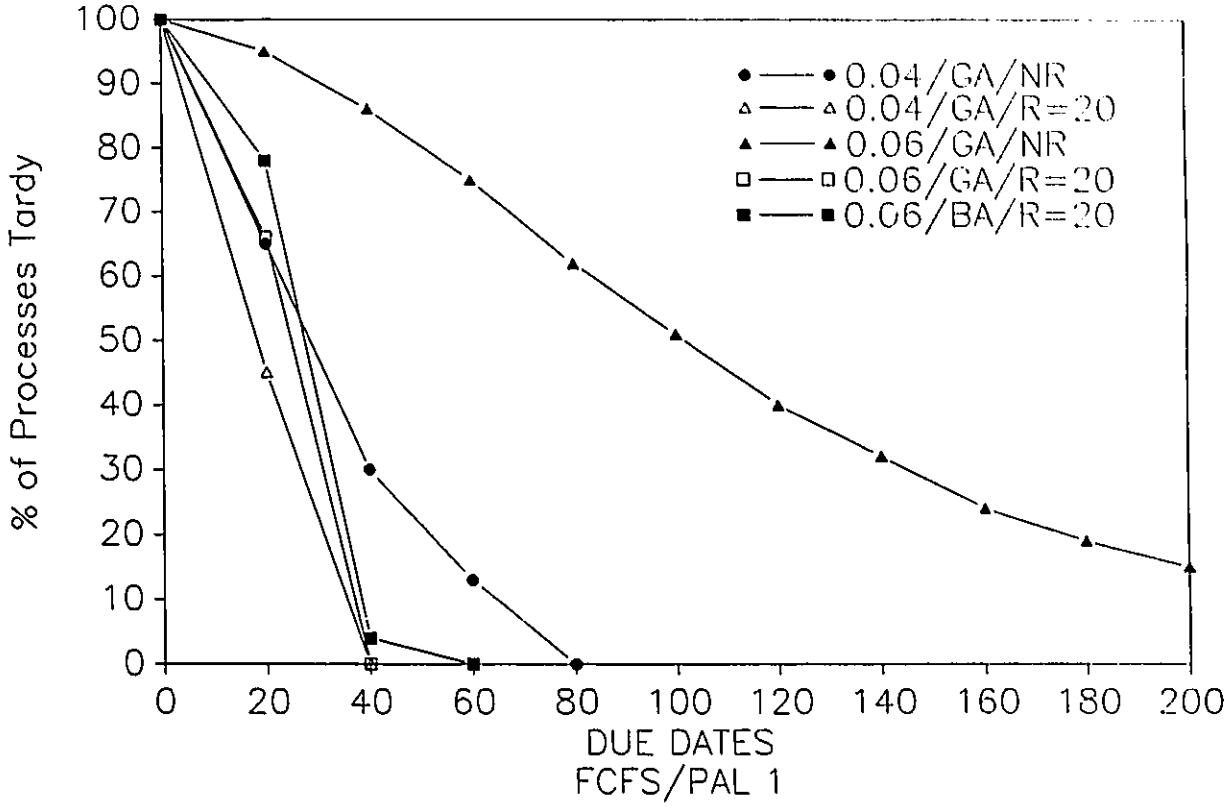


Figure 4–12 The Effects of Different Event Rejection Policies on Process Tardiness

Event Arrival Rate/ Activity to Processor Allocation /Event Rejection Policy	0.04/ GA/ NR	0.04/ GA/ R=20	0.06/ GA/ NR	0.06/ GA/ R=20	0.06/ BA/ R=20
These Results are Valid for a FCFS Scheduling Rule and for the PAL 1 Process					

Figure 4–13 The Effects of Different Event Rejection Policies on Different Scheduling Criteria

Event Arrival Rate/ Activity to Processor Allocation /Event Rejection Policy	0.04/ GA/ NR	0.04/ GA/ R=20	0.06/ GA/ NR	0.06/ GA/ R=20	0.06/ BA/ R=20
Due Dates	% of Processes Tardy				
0	100	100	100	100	100
20	65	45	95	66	78
40	30	0	86	0	4
60	13	0	75	0	0
80	0	0	62	0	0
100	0	0	51	0	0
120	0	0	40	0	0
140	0	0	32	0	0
160	0	0	24	0	0
180	0	0	19	0	0
200	0	0	15	0	0
Average Comple- tion Time	35	22	118	25	28
% of Events Rejected	0	15	0	24	36

Figure 4-13 The Effects of Different Event Rejection Policies on Different Scheduling Criteria [continued]

Event Arrival Rate/ Activity to Processor Allocation /Event Rejection Policy	0.04/ GA/ NR	0.04/ GA/ R=20	0.06/ GA/ NR	0.06/ GA/ R=20	0.06/ BA/ R=20
Average Event Queuing Time	9	2	51	4	4
Average Processor Utilization (%)	49	44	72	60	51
Average Ready to Run Queue Length	1.0	1.2	2.2	1.7	2.5

Figure 4-13 The Effects of Different Event Rejection Policies on Different Scheduling Criteria [continued]

86% of tardy processes. The use of the R=20 event rejection policy, even when using a BA, results in 82% less tardy processes for PAL 1.

4.4 Concluding Remarks

The purpose of this chapter was to present the PAL-Based simulation environment and some results for a medium-sized example. Section 4.1 presented a description of the PAL simulator. Section 4.2 presented a medium-sized RTS exmaple. Multiple simulations of this example were run while varying different operating parameters. The parameters that were varied during simulation can be grouped under four headings: activity scheduling rules, activity to processor allocation, event arrival rates and event rejection policy.

Section 4.3 presented some simulation results for this RTS example. In particular, it was shown that an increase in the event arrival rates caused a significant performance decrease in the performance of various scheduling criteria (process tardiness, average process completion times, ...).

Changing over from a good activity to processor allocation to a bad activity to processor allocation (i.e. one that does not make efficient use of the potential concurrencies within processes) yielded significant increases over the baseline FCFS rule in the percentage of tardy processes and other scheduling criteria, at the higher event arrival rates of 0.06. No significant differences were observed at the low and medium event arrival rates of 0.02 and 0.04.

Data was also presented to prove that the use of an event rejection policy, such as rejecting events if they have to wait in an event queue for more than 20 simulation units, could significantly improve the performance of various scheduling criteria for medium and high event arrival rates.

It was also shown that while varying different scheduling rules, under standard operating parameters (good activity to processor allocation and no event rejection allowed), for low and medium event arrival rates, did not result in any increase in the percentage of tardy processes. However, under the same operating parameters, with the high event arrival rate of 0.06, it was shown that the use of the SAF, LELF, EDD and LETC scheduling rules produced as much as 30% less tardy processes than the baseline FCFS scheduling rule. It is important to note that although these scheduling rules performed significantly better for this RTS example; these results can not be easily extended to different applications. Experimentation with the PAL simulator makes it possible to select a best scheduling rule for a particular RTS application in order to optimize some scheduling criteria.

It is also important to note that the data of figures 4–7, 4–9, 4–11 and 4–12 is reliable as standard deviation and variance values for all the data included in these graphs were very small.

CHAPTER 5

CONCLUSION

This concluding chapter is divided into two sections. The first section presents a brief summary of the thesis, emphasizing the achievement of this research, while the second section discusses directions for future research.

5.1 Thesis Summary

The research done in this thesis has addressed the problem of investigating the effects of dynamically scheduling executable work units in Real-Time Systems (RTS) so that various responses may meet their time constraints.

In chapter 1, the general characteristics of RTS were presented. This was followed by a discussion of the importance of scheduling in RTS. Then a statement of the objectives to be achieved in this thesis was presented. These objectives were:

1. provide background and indicate pertinent results in scheduling;

2. discuss the design of RTS using the Process Activity Language (PAL) and its associated toolset; and

3. investigate the effects of different dynamic heuristic rules on the design of RTS which allow for the execution of complex responses.

In chapter 2, a comprehensive litterature survey of pertinent results in scheduling was presented. A general scheduling terminology was first outlined to allow for the presentation of results from both Operations Research and computer systems. RTS scheduling was divided into static and dynamic scheduling. Static scheduling was found to be directly applicable only to RTS with periodic processes. Dynamic scheduling, in turn, was shown to be applicable only to RTS with soft constraints.

In chapter 3, the elements of the design specification PAL were introduced. The different steps that may be used in the design of RTS, using PAL and its associated toolset, were outlined in the Prototyping-Based Design Methodology.

This methodology is based on the multiactivity design paradigm, which separates the coordination of activities from the execution. The methodology generates two system prototypes, the Specification Prototype and the Design Prototype. The Specification Prototype is a model of the behavioral and functional requirements of the system that can be used to check whether the requirements were correctly specified. The Design Prototype is a model of the design specifications of the system, which is used to check if it satisfies the performance and timing requirements and to determine the parameters needed to satisfy them. These parameters include the PAL processes, the number and types of resources available, the activity-to-resource allocation, the activity scheduling scheme and the external event characteristics. The ways in which the PAL simulator can be used to check the above parameters were discussed.

In chapter 4, the RTS simulation environment and some results for a medium-sized example were presented. The PAL simulator was developed for this thesis to accomplish two main functions. First, the simulator can serve as a tool for testing the effects of different scheduling schemes on the behavior of RTS with complex responses. It can also serve as a model for the design of a general purpose multiactivity executive. Different simulations were run and data was gathered for a medium-sized RTS example. It is important to note that although certain scheduling rules (i.e. SAF, LELF, EDD, etc.) were shown to have a significant impact on the minimization of the percentage of tardy processes for this RTS example; these results could not be easily extended to a different application. It was shown that experimenting with the Design Prototype of a particular RTS application on the PAL simulator makes it possible to select a best scheduling rule for a particular application in order to optimize some scheduling criteria.

5.2 Future Research

The following topics are considered important areas for further investigation:

1. The Prototyping-Based Design Methodology presented in this thesis relies on the RTS designer being able to analyse the statistics gathered by the simulator. In the case of complex RTS, this can become quite an arduous task and mistakes can be made. There is a need to somehow automate the analysis process so that the statistics can be presented to the user in a concise preprocessed manner. Depending on the degree of processing, the results should clearly point to the direction that the design should take;

2. Regarding specifically the design of the PAL simulator presented in chapter 4, a number of additional enhancements could be added: first, a better user interface could be provided to enter the activity-to-resource allocation; second, an interface program could be created to make better use of the PAL editor; and third, a number of other scheduling rules could be added to the PAL simulator;

3. What is required now is a final integration of the various PAL tools and an implementation on a hardware platform that is applied to real problems. It would indeed be very interesting to see a real application in which multiactivity systems can be used very efficiently; and

4. Other important aspects that could be studied in multiactivity systems include: resource allocation in multiactivity systems and which types of communication (blocking, non-blocking, etc.) work best in multiactivity systems.

References and Bibliography

BAKE 74 K.R.Baker, "Introduction to Sequencing and Scheduling", J.Wiley & Sons, New-York, 1974.

BALA 89 A.Balakrishnan, "Preemptive Scheduling of Hybrid Parallel Machines", Operations Research, Vol.37, No.2, pp.310-313, March-April 1989.

BLAC 82 J.H.Blackstone, D.T.Phillips and G.L.Hogg, "A State-of-the-art Survey of Dispatching Rules for Manufacturing Job Shop Operations", International Journal of Production Research, Vol.20, No.1, pp.27-45, 1982.

BOEH 81 B.W.Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

BOEH 88 B.W.Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, May 1988, pp.61-72.

CASA 88 T.L.Casavant and J.G.Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems", IEEE Transactions on Software Engineering, Vol.14, No.2, February 1988.

CHEN 88 S.C.Cheng and J.A.Stankovic, "Scheduling Algorithms for Hard Real-Time Systems-A Brief Survey", Hard Real-Time Systems-Tutorial, pp.150-154, IEEE Computer Society Press, Washington, D.C., 1988.

CHUB 90 A.Chubukjian, "The PAL Editor", M.Eng. Dissertation, University of Ottawa, 1990.

COFF 73 E.G.Coffman and P.J.Denning, Operating Systems Theory, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

CONW 65 R.W.Conway, "Priority Dispatching and Work in Process Inventory in a Job Shop", Journal of Industrial Engineering, Vol.16, No.4, July 1965.

CONW 67 R.W.Conway, W.L.Maxwell and L.W.Miller, Theory of Scheduling, Addison-Wesley, Reading, Massachussets, 1967.

CRAI 87 D.W.Craig, "Light Traffic Loss of Random Hard Real-Time Tasks in a Network", Ph.D. Dissertation, Carleton University, 1987.

DAVA 86 S.Davari and S.K.Dhall, "An On Line Algorithm for Real-Time Task Allocation", Proceedings of the IEEE Real-Time Systems Symposium, pp.194-200, New-Orleans, Louisiana, 1986.

DAVI 88 A.M.Davis, E.H. Bersoff and E.R.Comer, "A Strategy for Comparing Alternative Software Development Life Cycle Models", IEEE Transactions on Software Engineering, Vol.14, No.10, October 1988, pp.1453-1461.

ELVE 73 D.A.Elvers, "Job Shop Dispatching Rules using Various Delivery Date Setting Criteria", Production Inventory Management, Vol.14, 1962.

GAGN 89 J.A.M.Gagne, "A Pre-Run-Time Algorithm for the CF-188 Fighter Aircraft", M.Eng. Dissertation, Royal Military College of Canada, Kingston, Ontario, 1989.

GARE 76 M.R.Garey and D.S.Johnson, "Scheduling Tasks with Nonuniform Deadlines on Two Processors", Journal of the Association of Computing Machinery, Vol.23, No.3, pp.461-467, July 1976.

GARE 79 M.R.Garey and D.S.Johnson, "Computers and Intractability- A Guide to the Theory of NP-Completeness", W.H.Freeman, New-York, 1979.

GARZ 86 R.F.Garzia and M.R.Garzia, "Discrete-Event Simulation", IEEE Spectrum, pp.32-36, December 1986.

GENT 88 W.M.Gentleman, "Multiprocessor Realtime Applications", International Specialist Symposium in the Design and Application of Parallel Digital Processing, April 1988.

GONZ 77 M.J.Gonzales, "Deterministic Process Scheduling", Computing Surveys, October 1977.

HANS 73 P.B.Hansen, Operating System Principles, Prentice-Hall, 1973.

HARV 89 R.A.Harvey, "Verification of Concurrent System Specifications Using Temporal Logic", M.A.Sc. Dissertation, University of Ottawa, April 1989.

HO 89 Y.C.Ho, "Dynamics of Discrete Event Systems", Proceedings of the IEEE, pp.3-6, January 1989.

HORN 74 W.A.Horn, "Some Simple Scheduling Algorithms", National Bureau of Standards, 1974.

HU 61 T.C.Hu, "Parallel Sequencing and Assembly Line Problems", Operations Research, Vol.9, No.6, November 1961.

JENS 85 E.D.Jensen, C.D.Locke and H.Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems", Proceedings of the IEEE Real-Time Systems Symposium, pp.112-121, 1985.

KLEI 76 L.Kleinrock, Queuing Systems, Vol.2:Computer Applications, Interscience 1976.

KRIE 88 M.Krieger, R.A.Harvey and J.P.Lachance, "Process Activity Language (PAL) for Planning and Coordination of FMS", Proceedings of the Symposium on Manufacturing Application Languages", Winnipeg, Manitoba, 1988.

LACH 88 PAL Simulator User Guide, University of Ottawa, August 1988.

LAWL 84 E.L.Lawler and C.Martel, "Scheduling Periodically Occuring Tasks on a Multiple Processor", Information Processing Letters, 12(1), February 1984.

LEGR 63 E.LeGrange, "The Development of a Factory Simulation System using Actual Operating Data", Managament Technology, Vol.3, No.1, May 1963.

LEIN 82 D.W.Leinbaugh and M.R.Yamini, "'Guaranteed Response Times in a Distributed Hard Real-Time Environment", Proceedings of the IEEE Real-Time Systems Symposium, pp.157-169, December 1982.

LIUL 73 C.Liu and J.Layland, "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment", Journal of the Association of Computing Machinery, Vol.20, pp.46-61, 1973

LUQI 89 Luqi, "Software Evolution Through Rapid Prototyping", IEEE Computer, May 1989, pp.13-25.

MAJU 88 S.Majumdar, "Processor Scheduling in Multiprogrammed Parallel Systems", Ph.D. Dissertation, University of Saskatchewan, 1988.

MART 82 C.Martel, "Preemptive Scheduling with Release Times, Deadlines and Due Dates", Journal of of the Association of Computing Machinery, vol.29, pp.812-829, July 1982.

MCNA 59 R.McNaughton, "Scheduling with Deadlines and Loss Functions", Management Science, Vol.6, No.1, October 1959.

MOKD 78 A.K.Mok and M.L.Dertouzos, "Multiprocessor Scheduling in a Hard Real-Time Environment", Proceedings of the 7th Texas Conference on Computing Systems, November 1978.

NANO 63 Y.R.Nanot, "An Experimental Investigation and Comparative Evaluation of Priority Disciplines in Job Shop-Like Queuing Networks", Ph.D. Dissertation, UCLA, 1963.

PENG 87 D.Peng and K.G.Shin, "Modelling of Concurrent Task Execution in a Distributed System for Real-Time Control", IEEE Transactions on Computers, Vol. C-36(4), 1987.

PETE 83 J.L.Peterson and A.Silbershatz, Operating System Concepts, Addison-Wesley, pp.103-137, 1983.

RAMA 84 K.Ramamrithan and J.A.Stankovic, "Dynamic Task Scheduling in Hard Real-Time Distributed Systems", IEEE Software, pp.65-75, 1984.

ROCH 76 R.Rochette and R.P.Sadowski, "A Statistical Comparaison of the Performance of Simple Dispatching Rules for a Particular Set of Job Shops", International Journal of Production Research, Vol.14, 1976.

SANM 90 R.San Martin, "The Multiactivity Paradigm in Rapid Prototyping of Embedded Systems", M.A.Sc. Dissertation, University of Ottawa, 1990.

SHA 86 L.Sha, J.Lehoczky and R.Rajkumar, "Solutions for Some Practical Problems in Prioritized Preemptive Scheduling", Proceedings of the IEEE Real-Time Systems Symposium, December 1986.

SPRU 89 B.Sprunt, L.Sha and J.Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems", Real-Time Systems-The International Journal of Time-Critical Computing Systems, Kluwer Academic Publishers, 1989.

STAN 89 J.A.Stankovic and K.Ramamrithan, Hard Real-Time Systems-Tutorial, IEEE Computer Society Press, pp.1-11, Washington, D.C., 1989.

TANE 87 A.S.Tanenbaum, Operating Systems-Design and Implementation, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

ULLM 75 J.D.Ullman, "NP-Complete Scheduling Problems", Journal of Computer and System Sciences, 10, pp.384-393, 1975.

ANNEX A

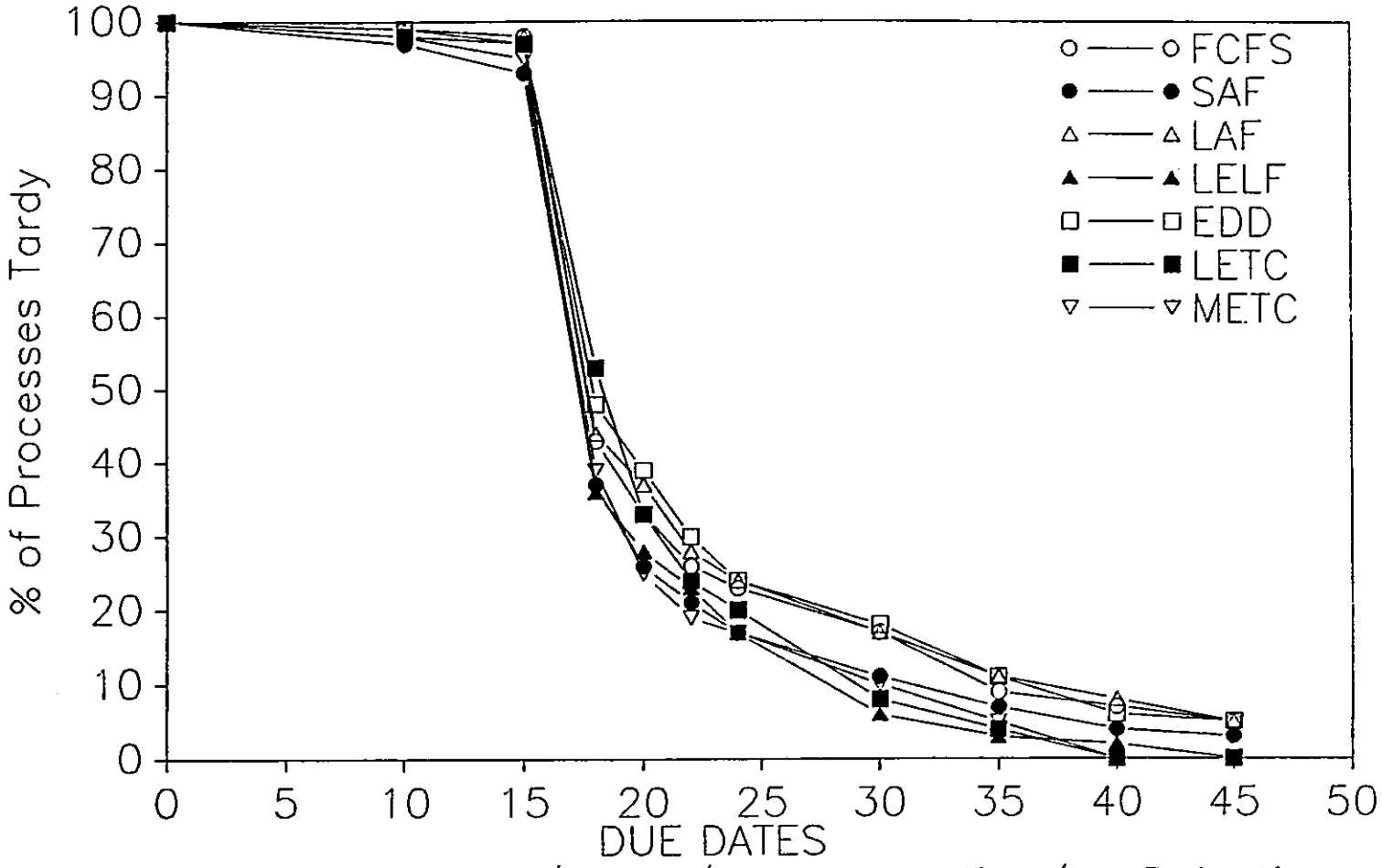


Figure A–1 0.02/PAL 1/Good Allocation/No Rejection

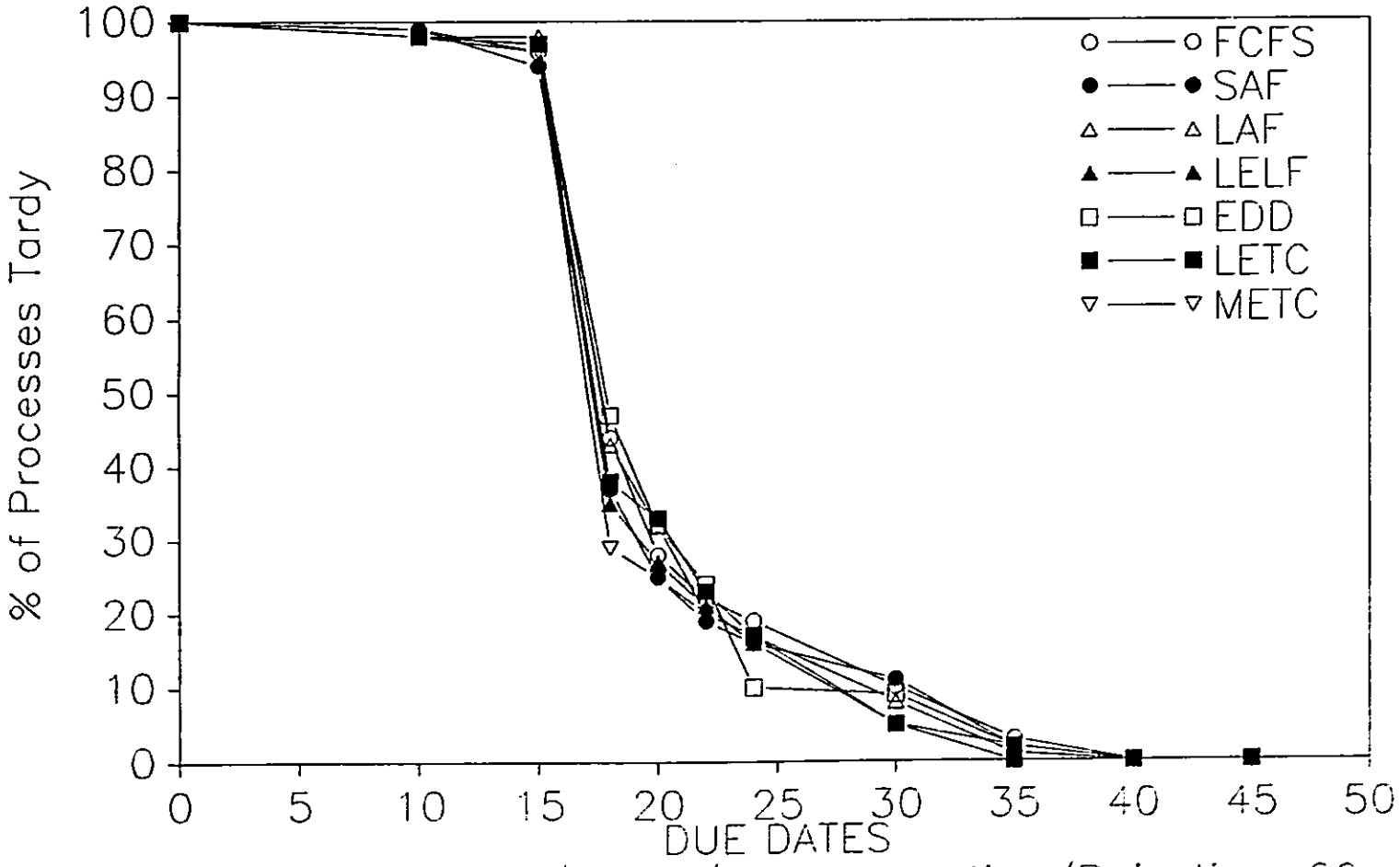


Figure A–2 0.02/PAL 1/Good Allocation/Rejection=20

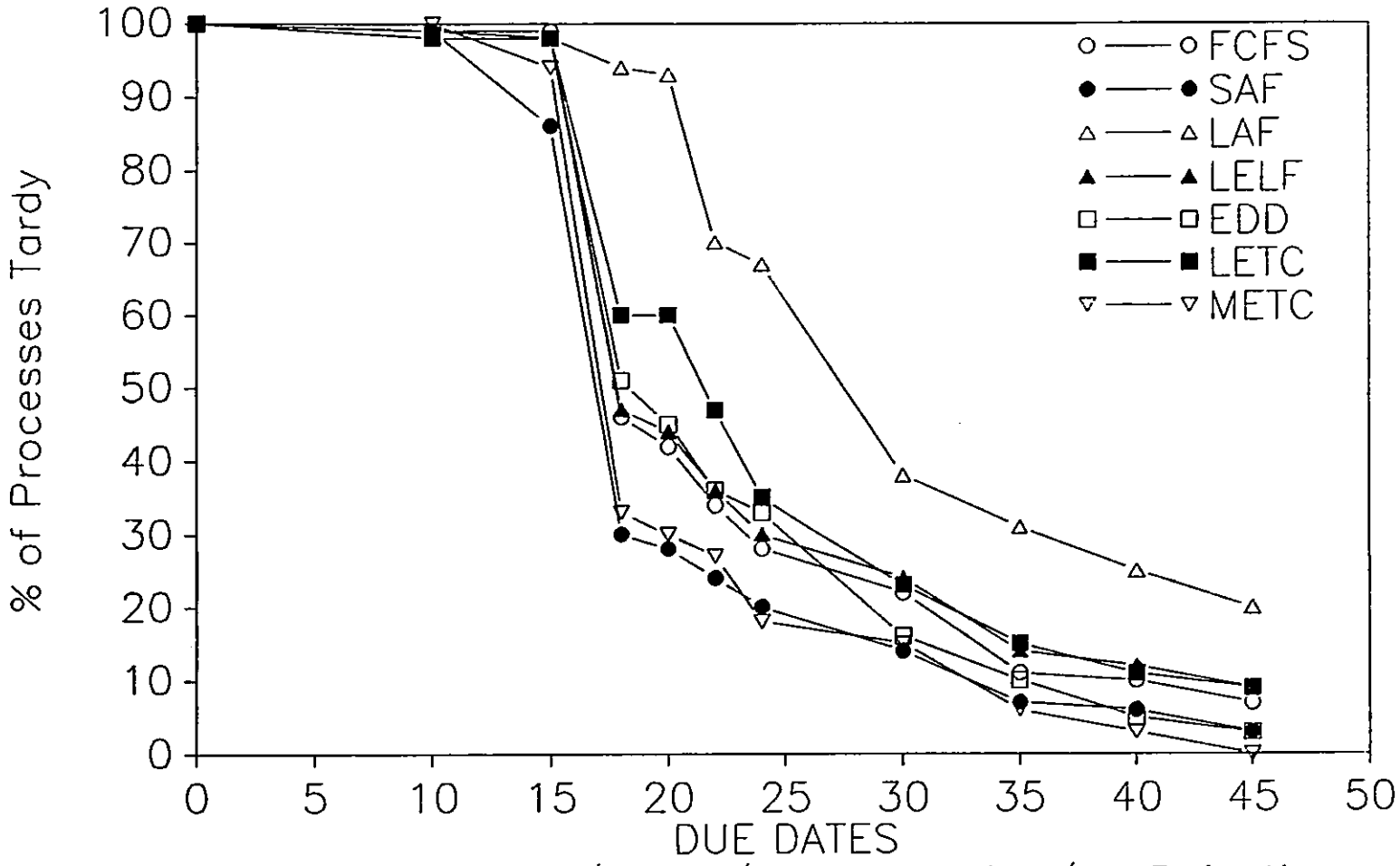


Figure A–3 0.02/PAL 1/Bad Allocation/No Rejection

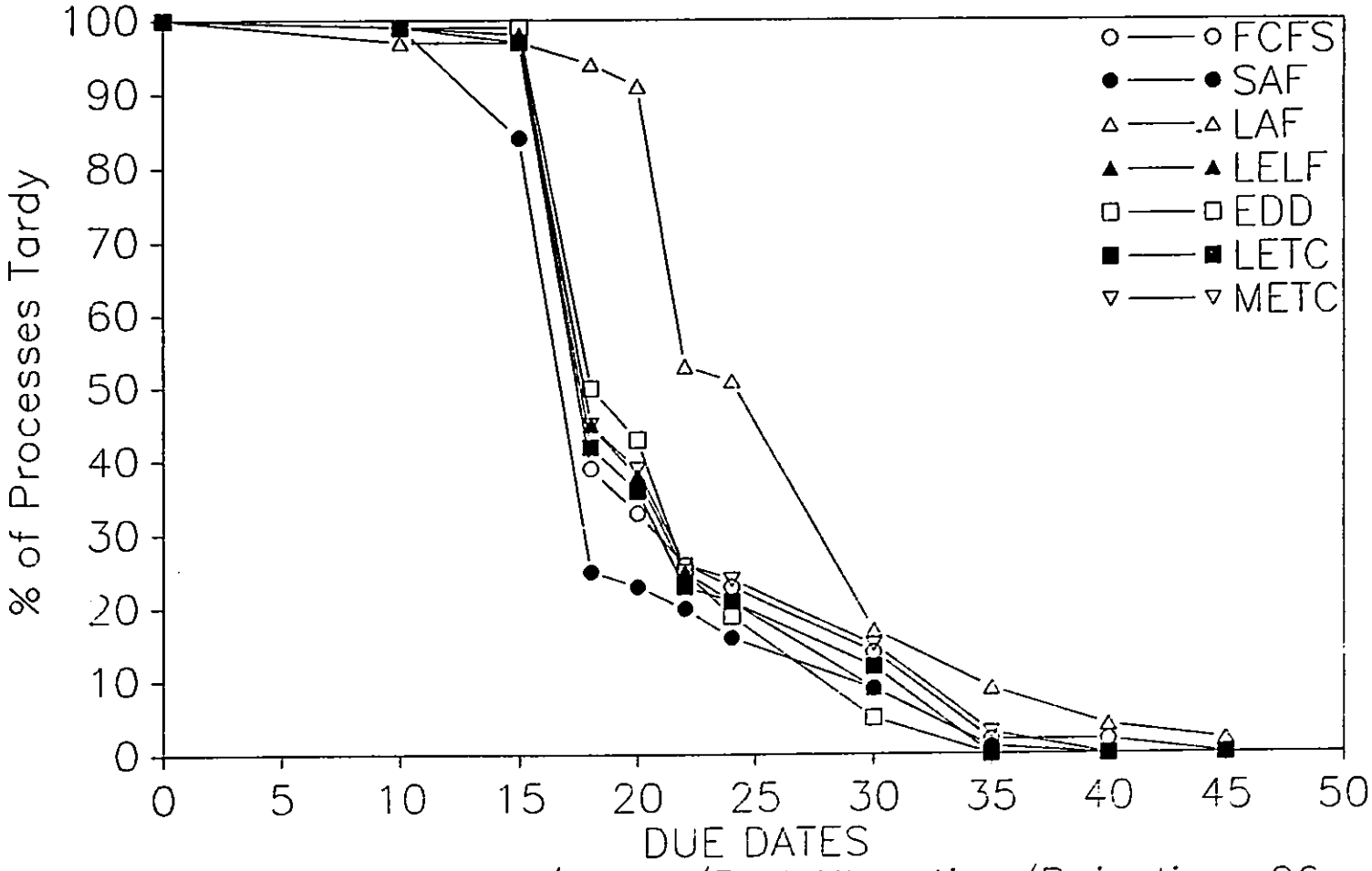


Figure A–4 0.02/PAL 1/Bad Allocation/Rejection=20

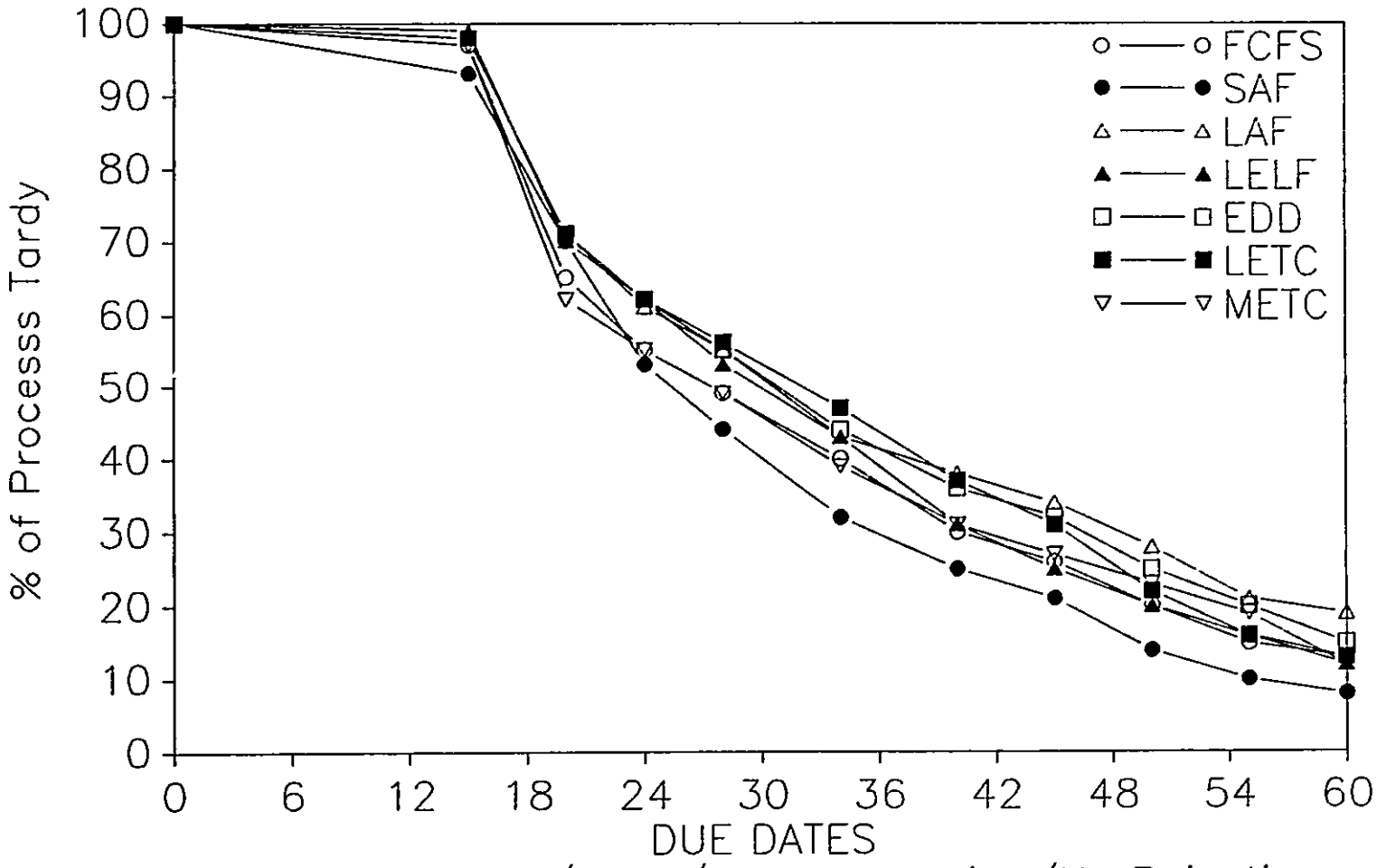


Figure A–5 0.04/PAL1/Good Allocation/No Rejection

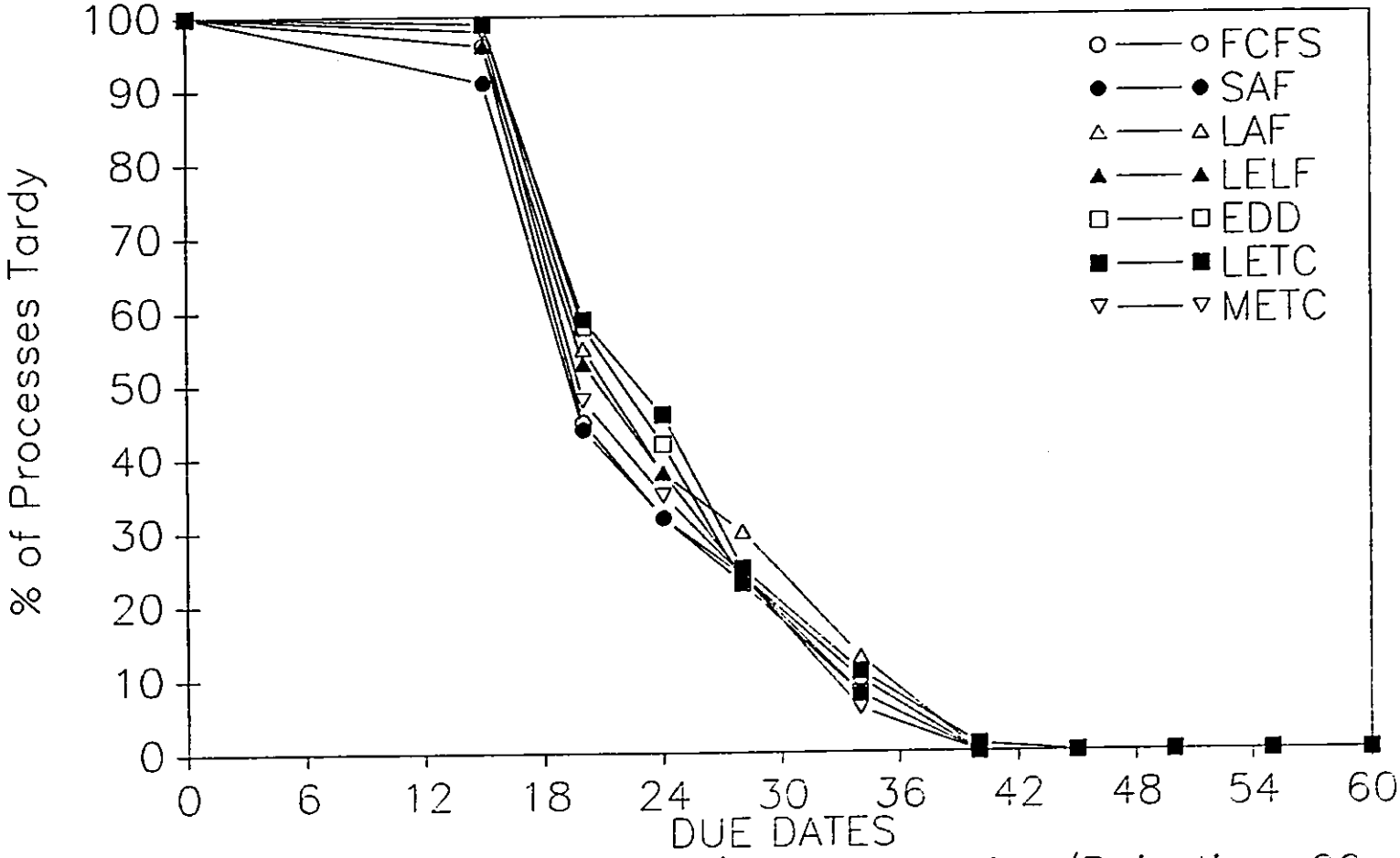


Figure A–6 0.04/PAL 1/Good Allocation/Rejection=20

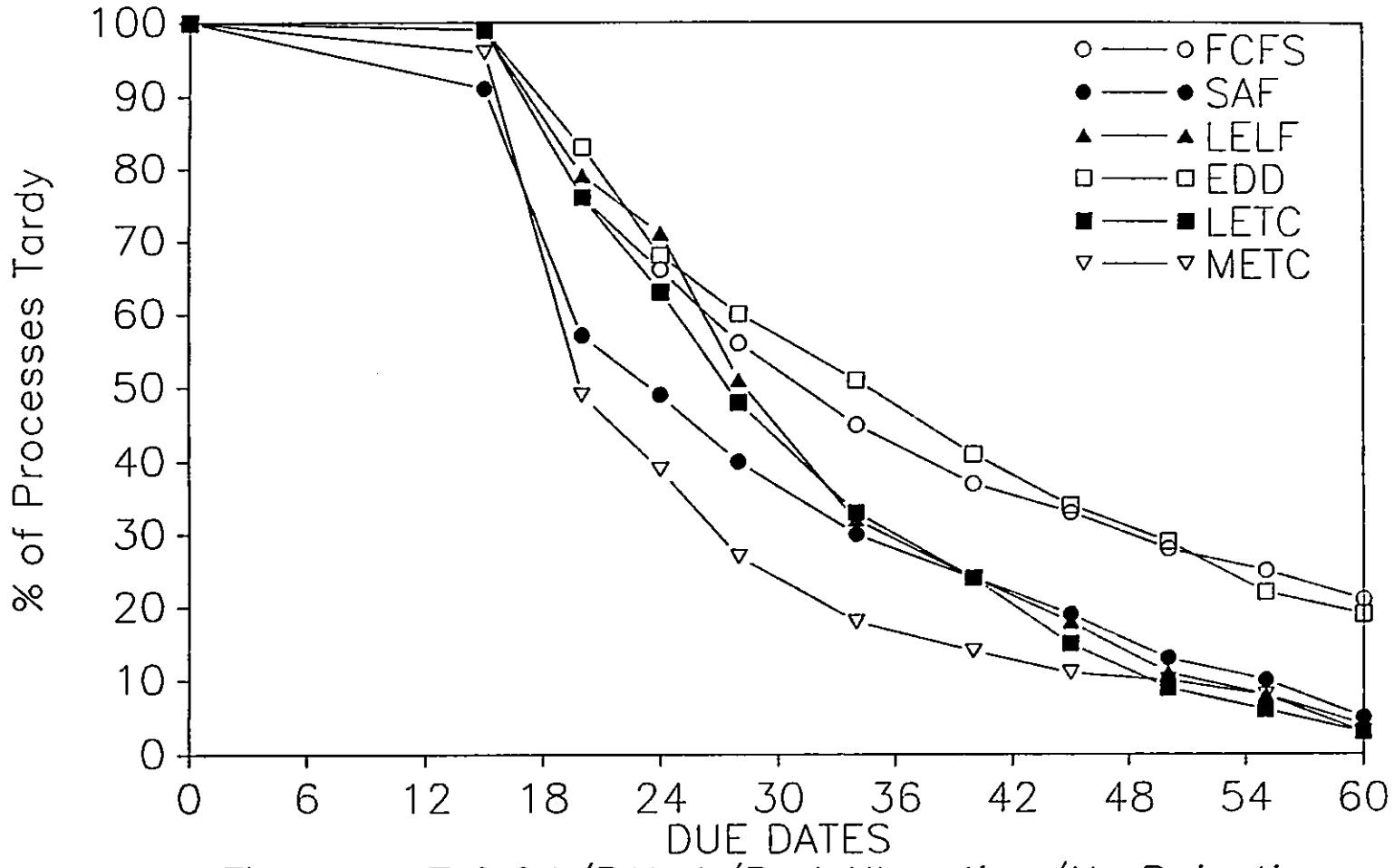


Figure A–7 0.04/PAL 1/Bad Allocation/No Rejection

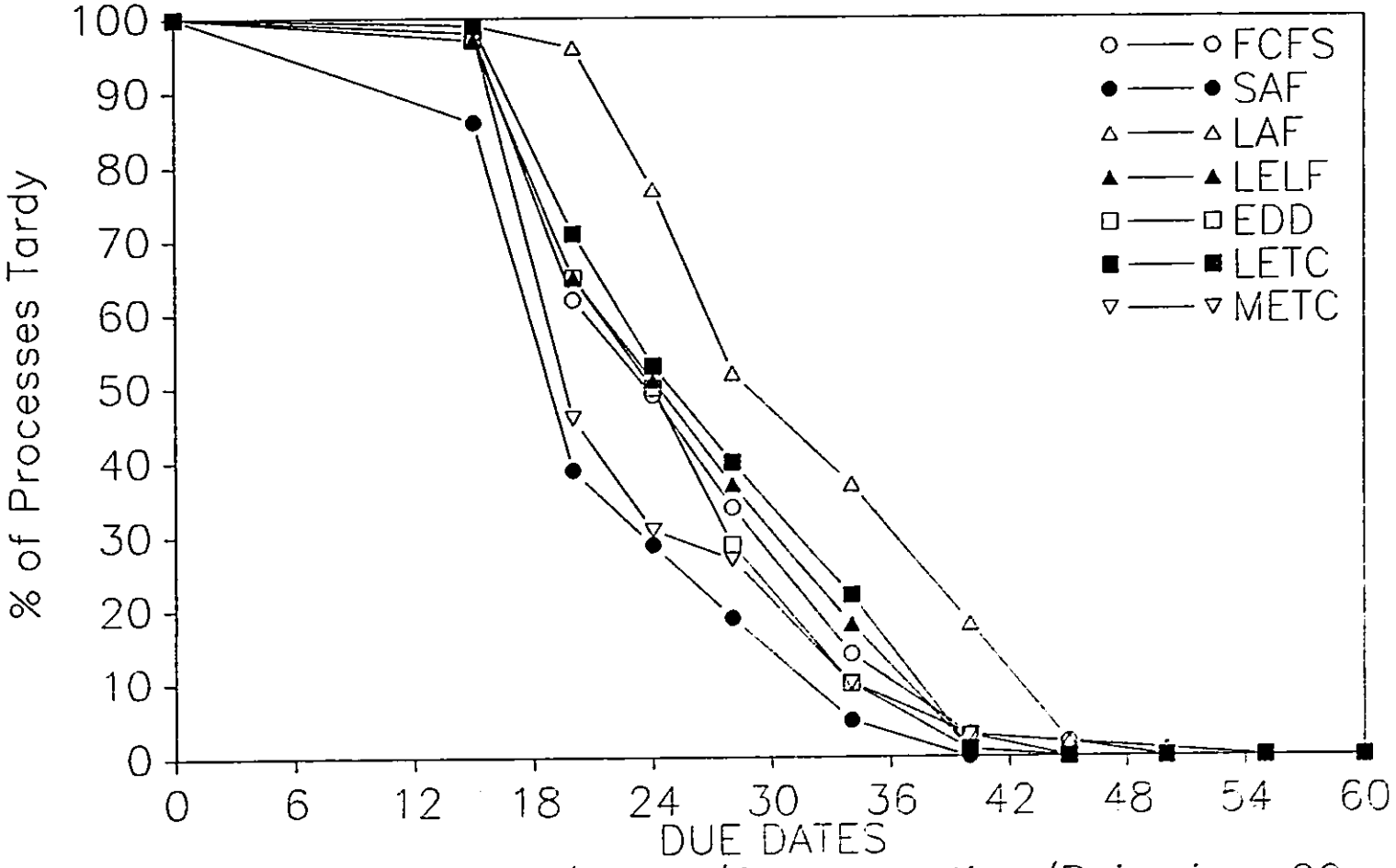


Figure A–8 0.04/PAL 1/Bad Allocation/Rejection=20

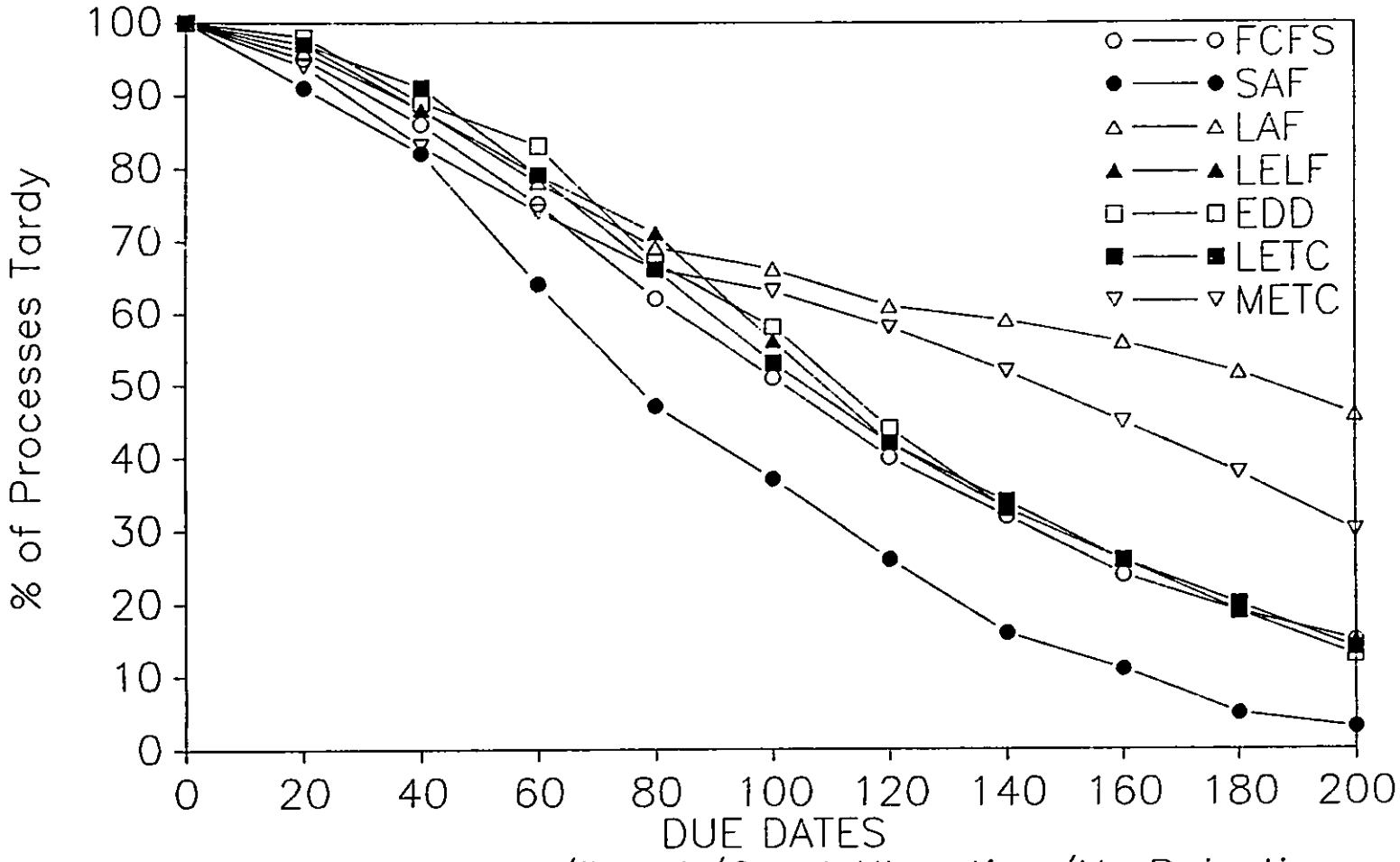


Figure A–9 0.06/PAL 1/Good Allocation/No Rejection

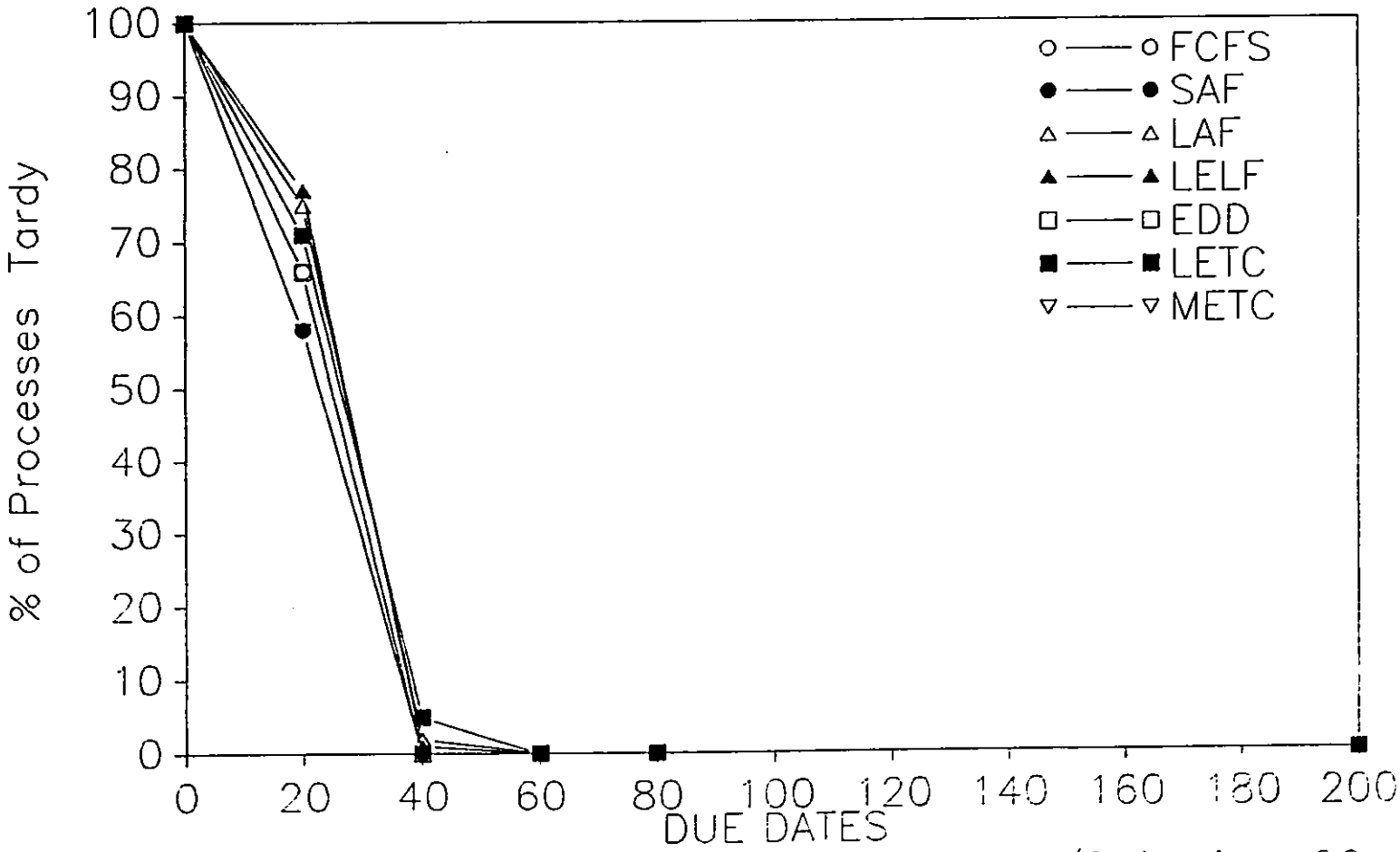


Figure A-10 0.06/PAL 1/Good Allocation/Rejection=20

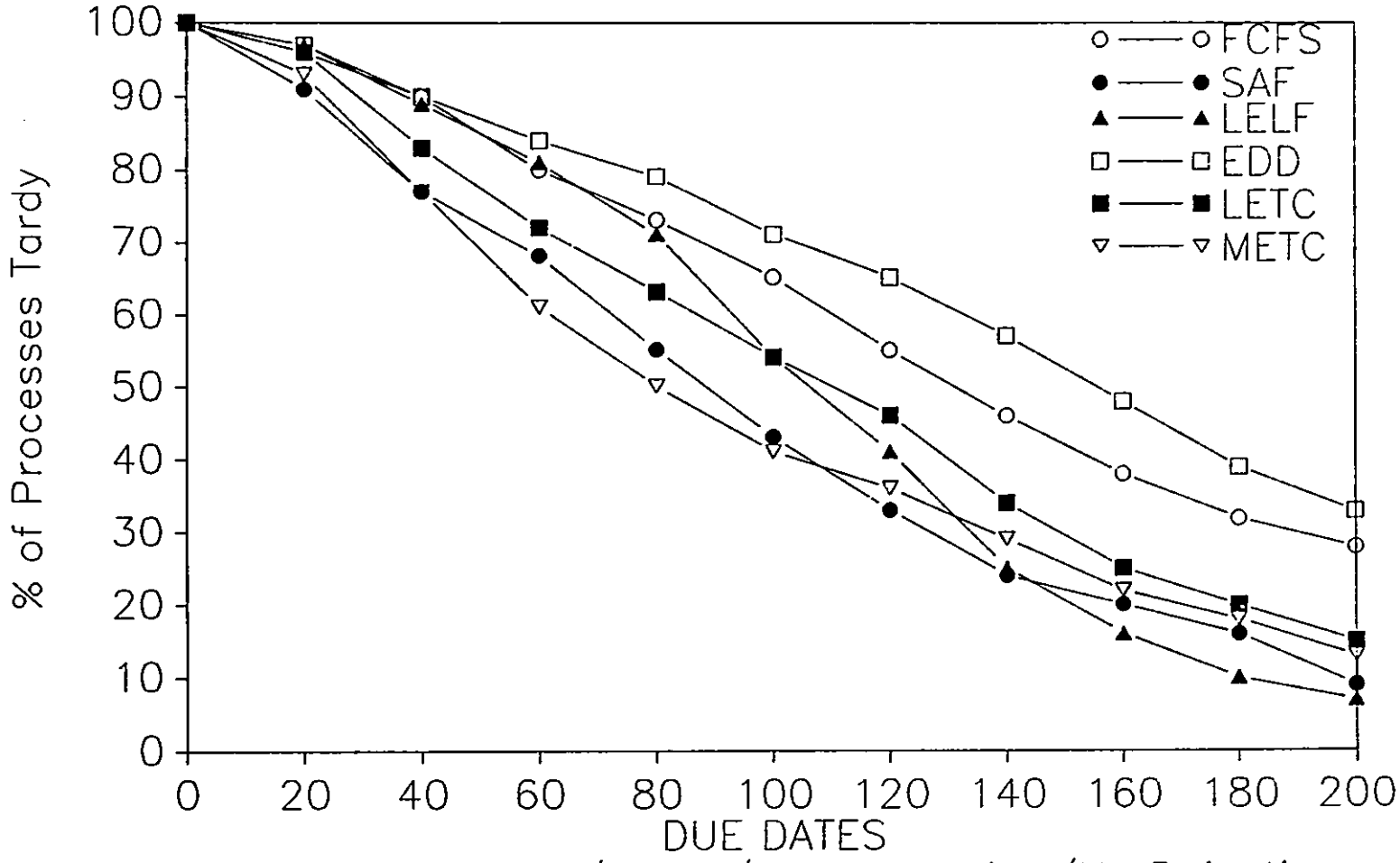


Figure A–11 0.06/PAL 1/Bad Allocation/No Rejection

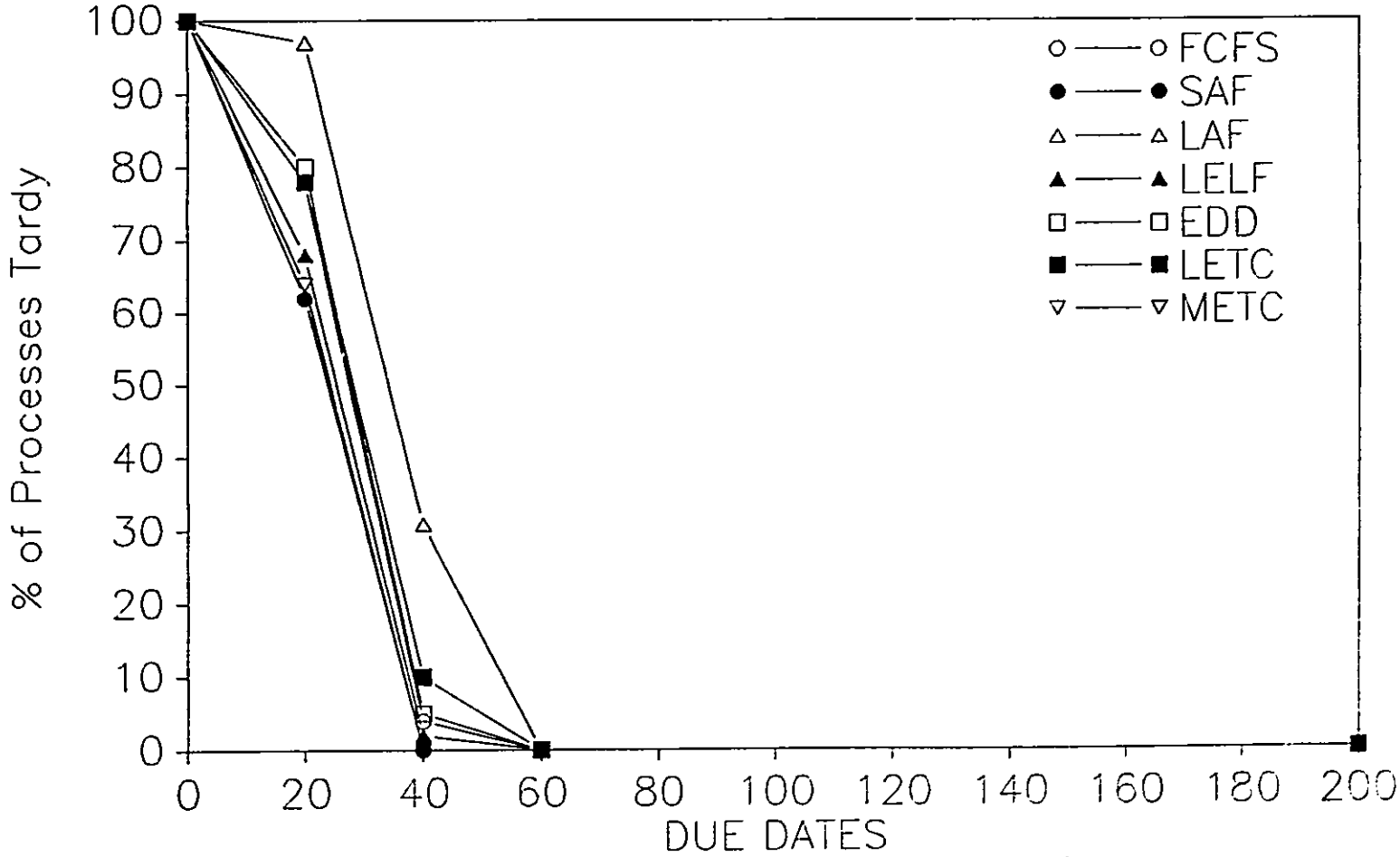


Figure A-12 0.06/PAL 1/Bad Allocation/Rejection=20

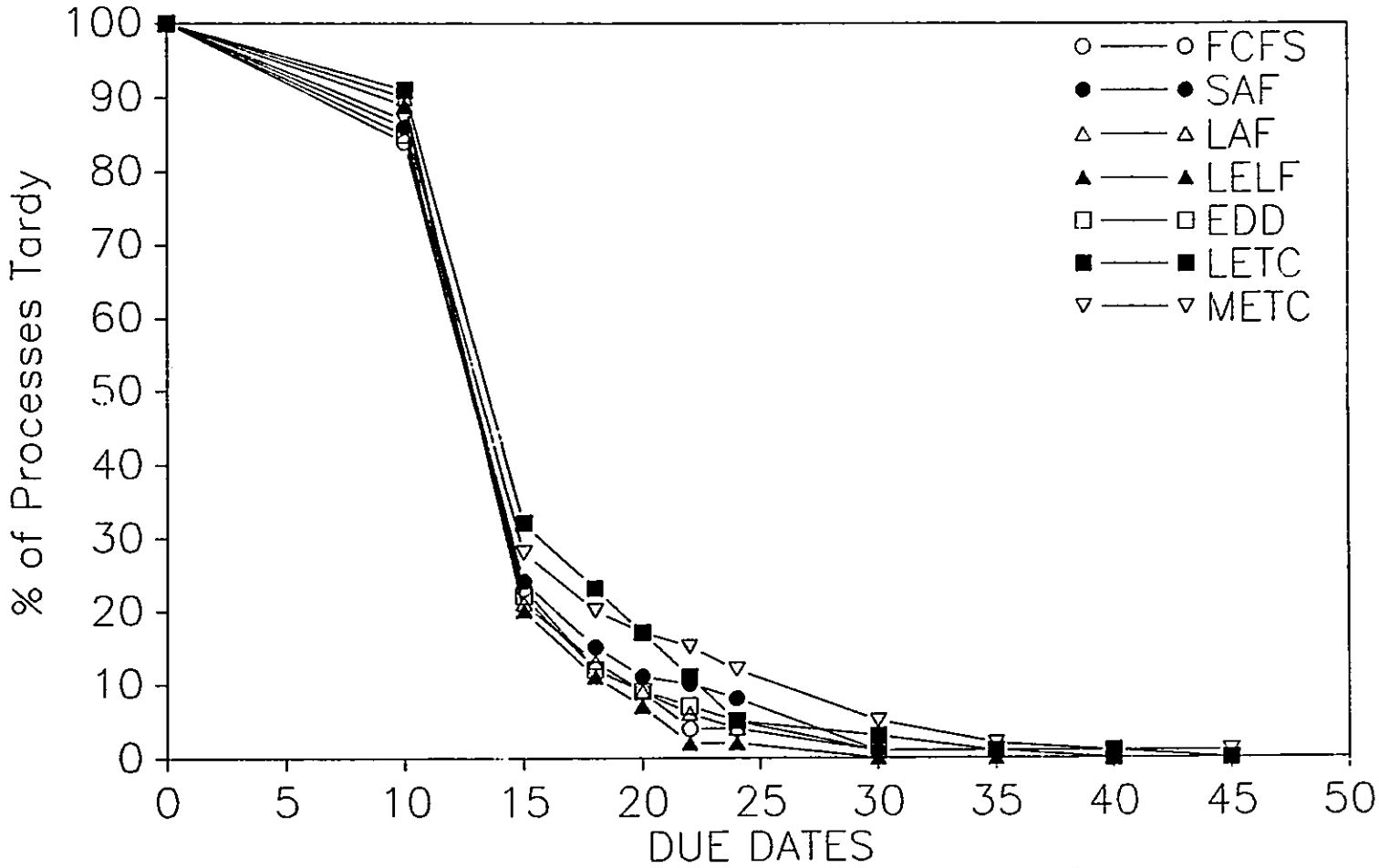


Figure A–13 0.02/PAL 2/Good Allocation/No Rejection

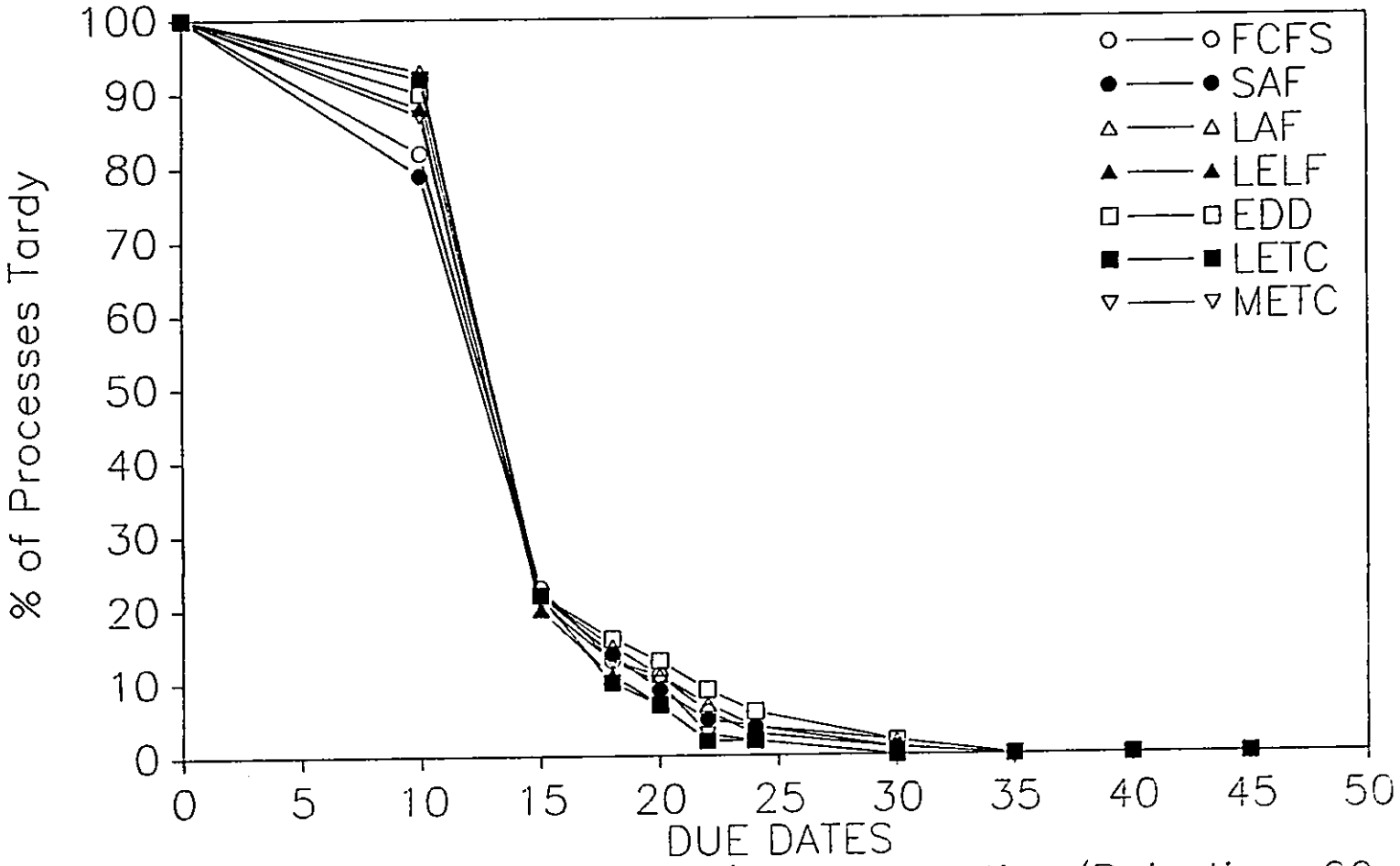


Figure A-14 0.02/PAL 2/Good Allocation/Rejection=20

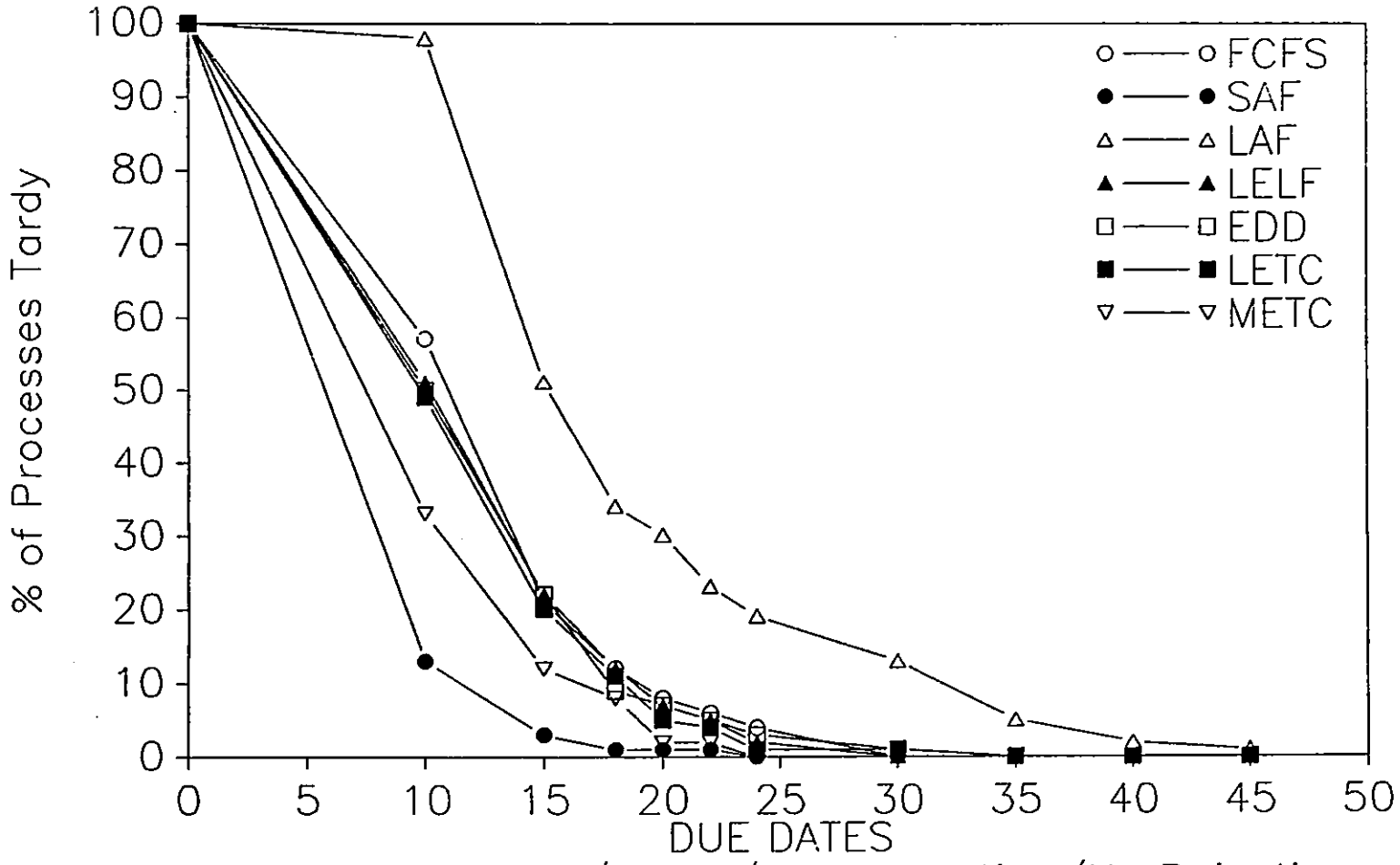


Figure A–15 0.02/PAL 2/Bad Allocation/No Rejection

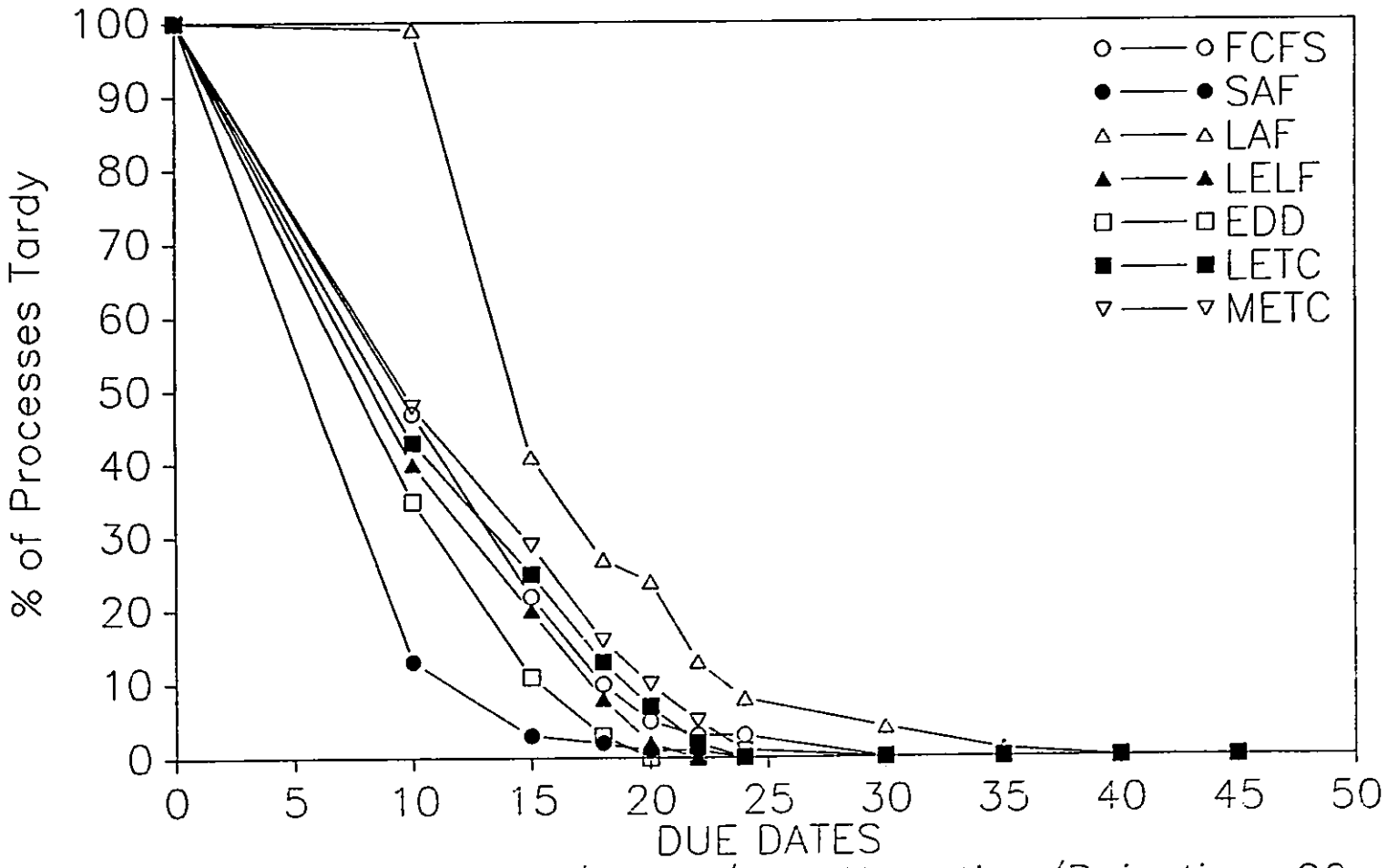


Figure A–16 0.02/PAL 2/Bad Allocation/Rejection=20

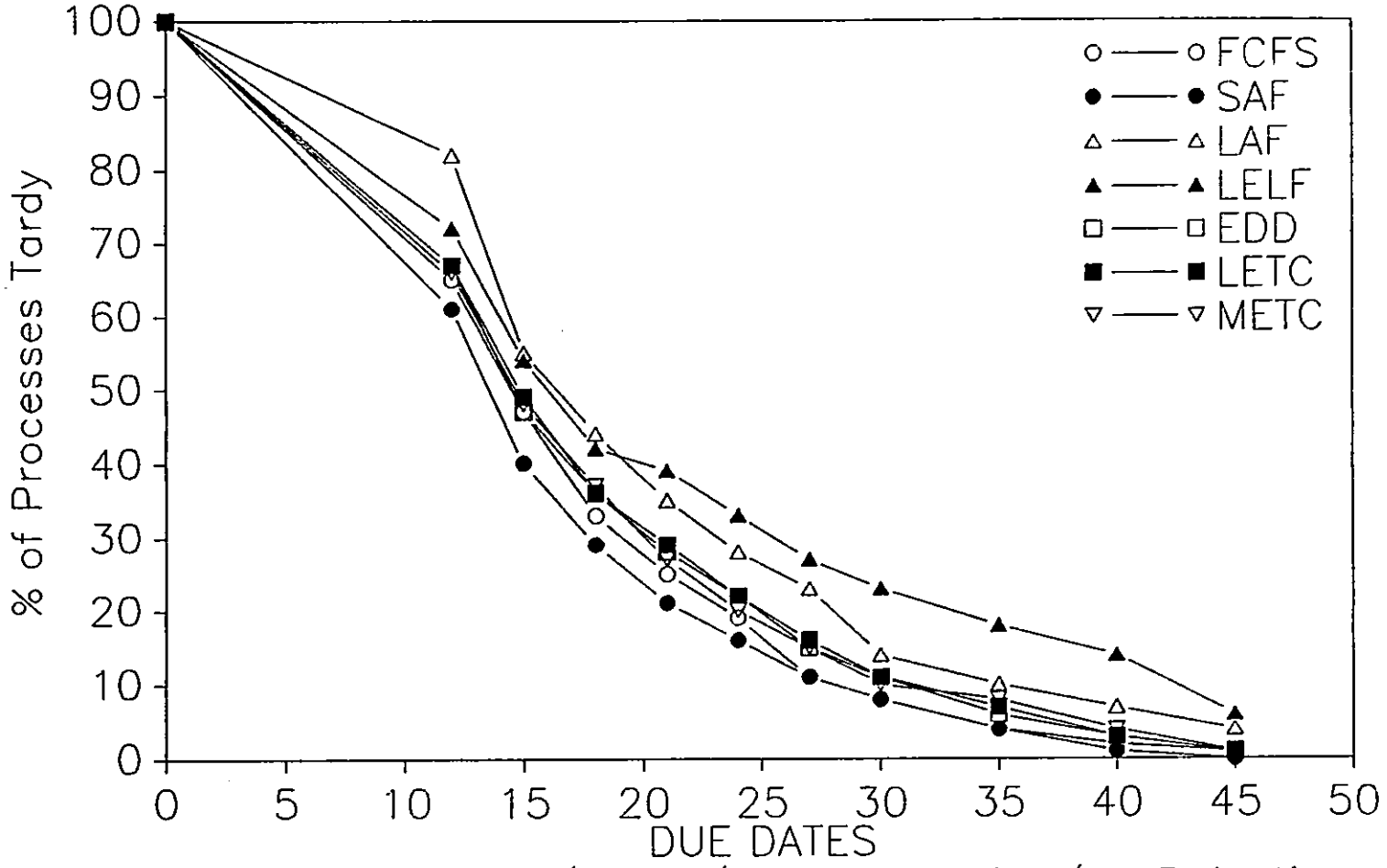


Figure A–17 0.04/PAL 2/Good Allocation/No Rejection

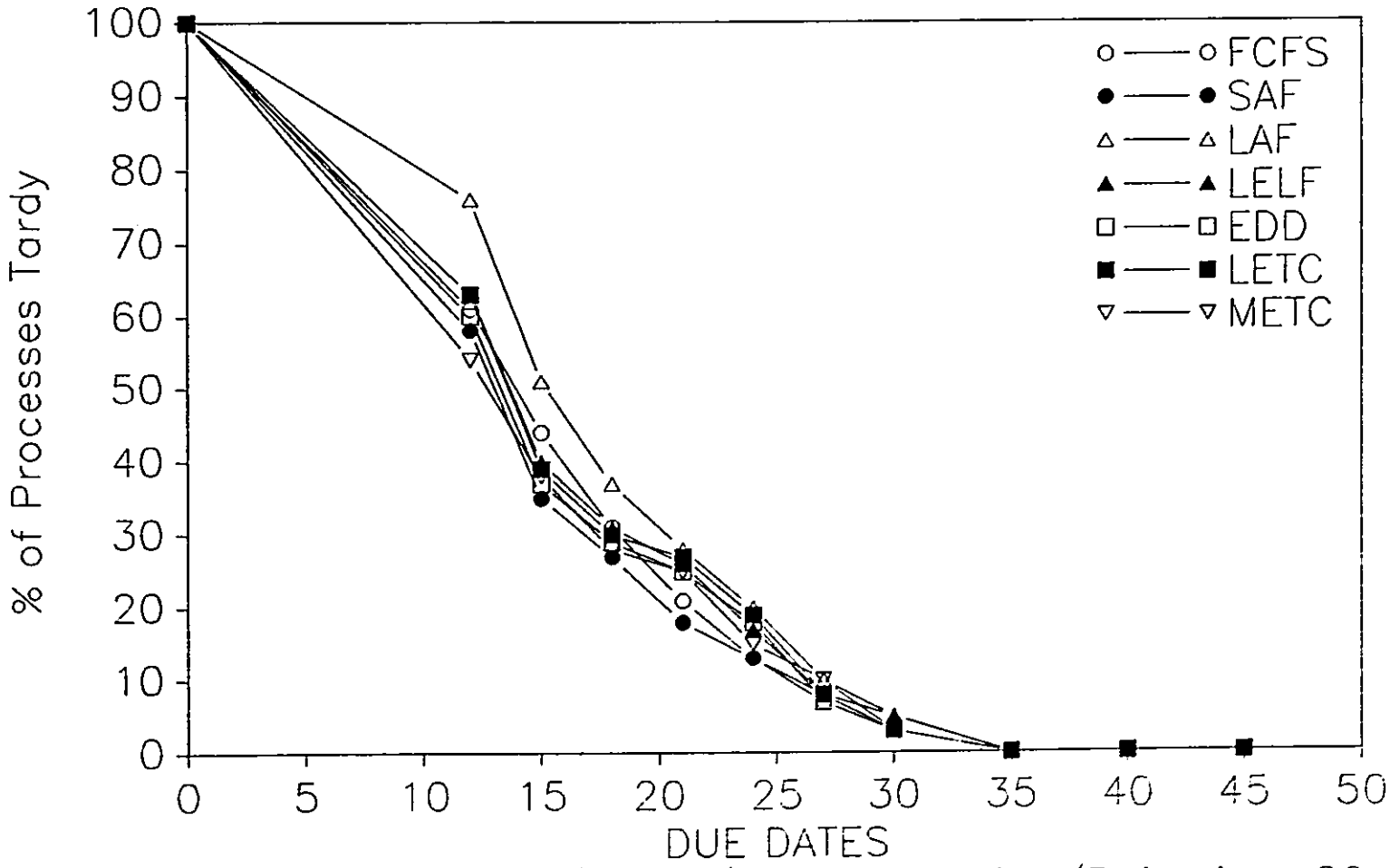


Figure A–18 0.04/PAL 2/Good Allocation/Rejection=20

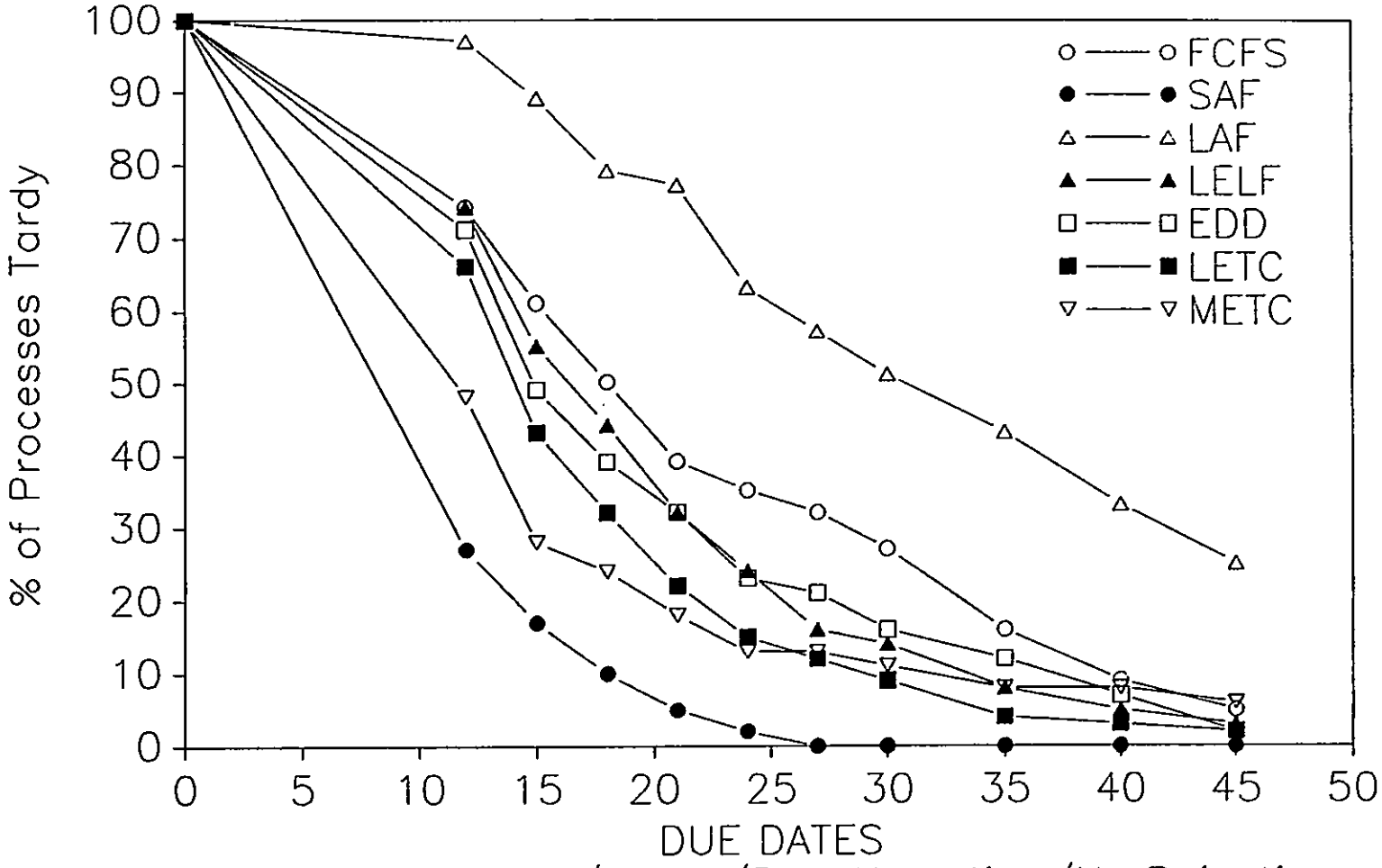


Figure A–19 0.04/PAL 2/Bad Allocation/No Rejection

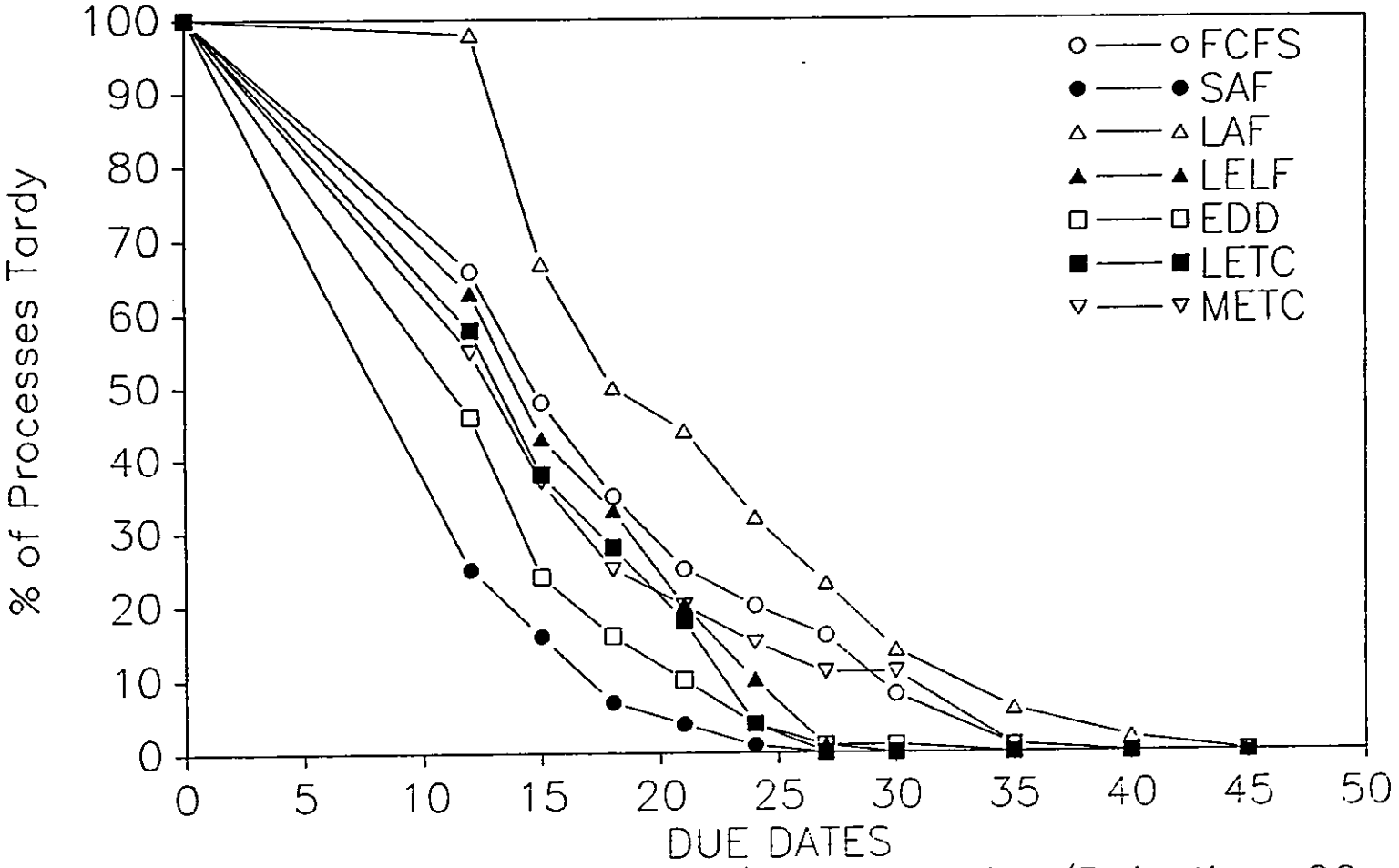


Figure A–20 0.04/PAL 2/Bad Allocation/Rejection=20

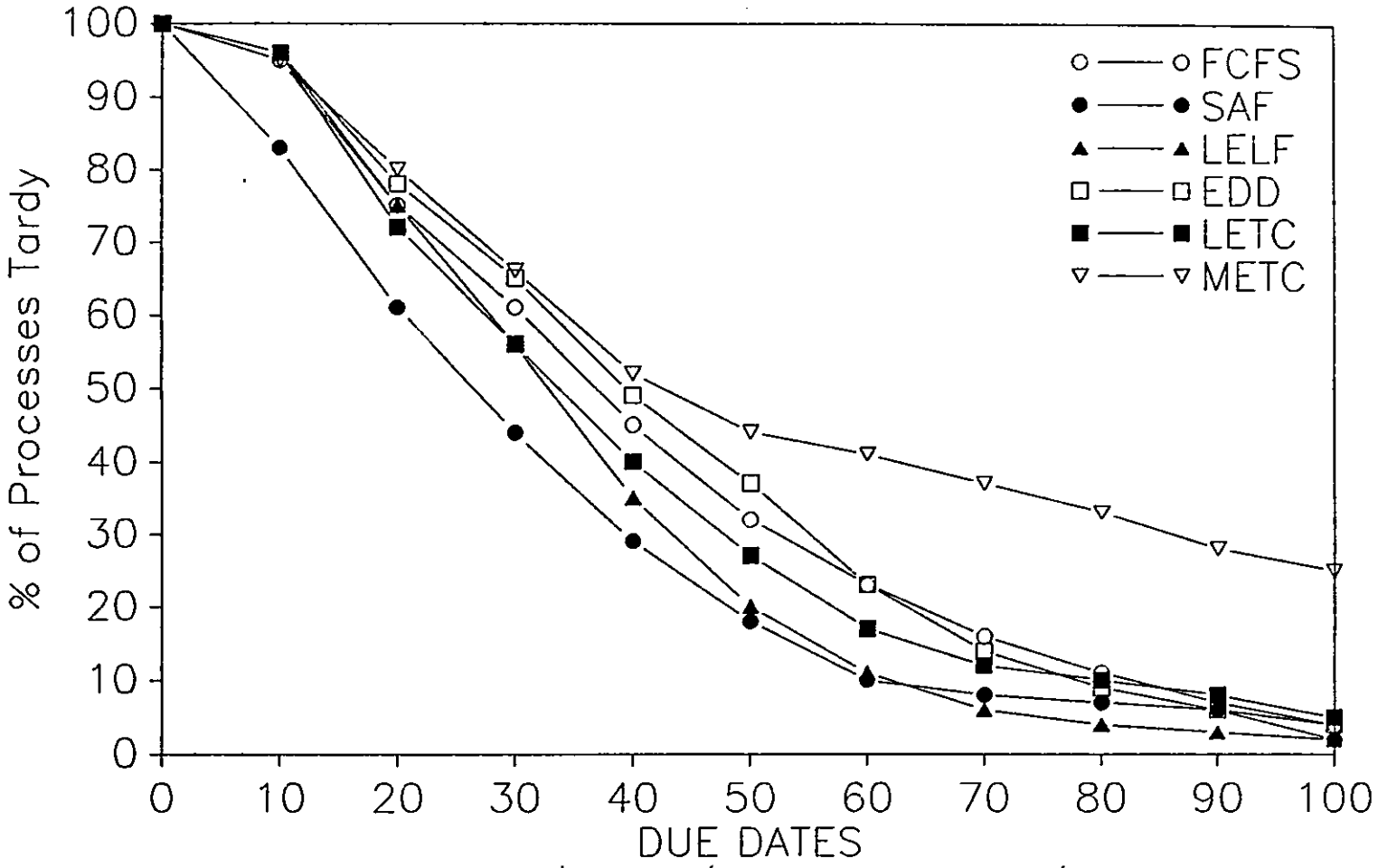


Figure A–21 0.06/PAL 2/Good Allocation/No Rejection

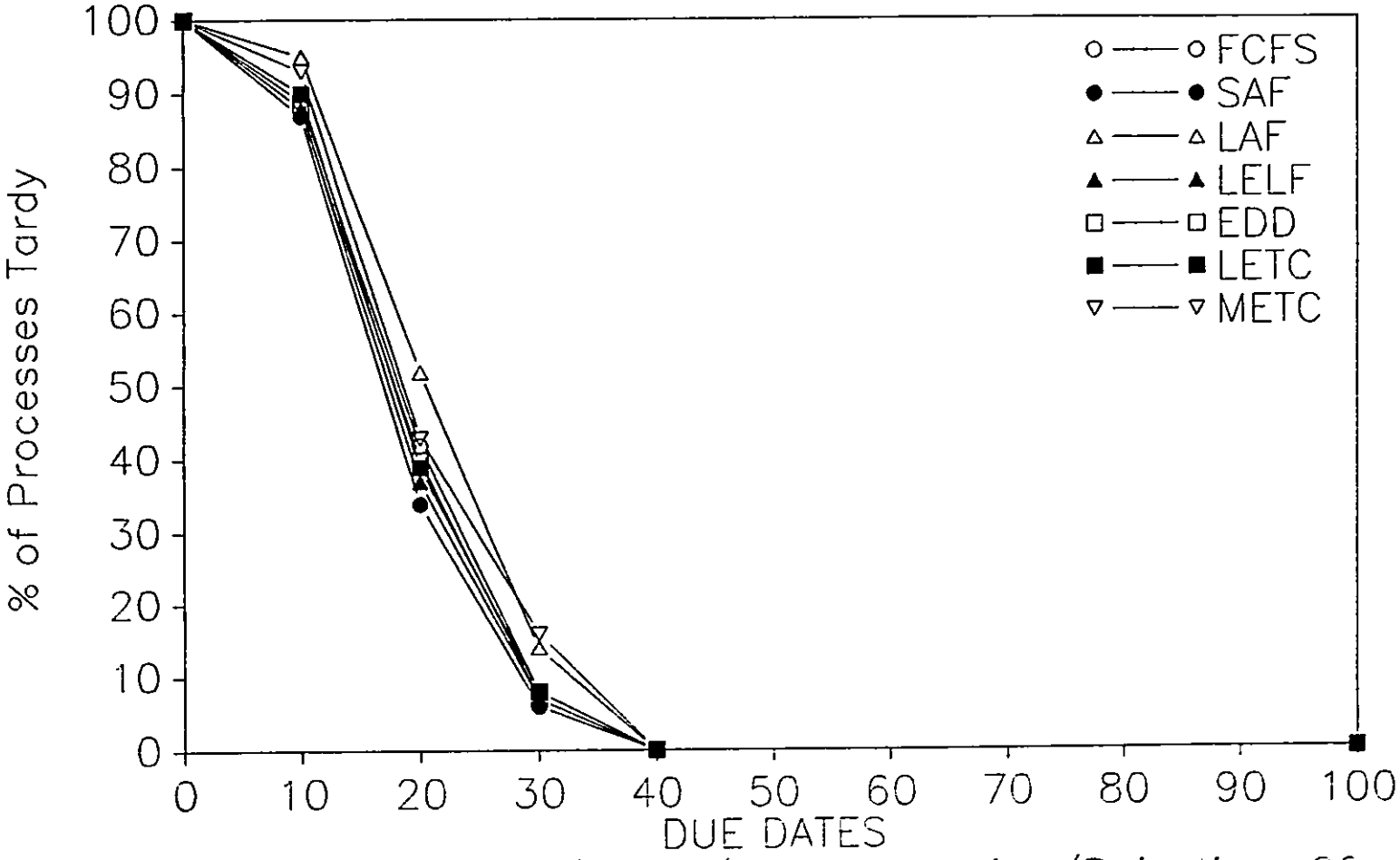


Figure A-22 0.06/PAL 2/Good Allocation/Rejection=20

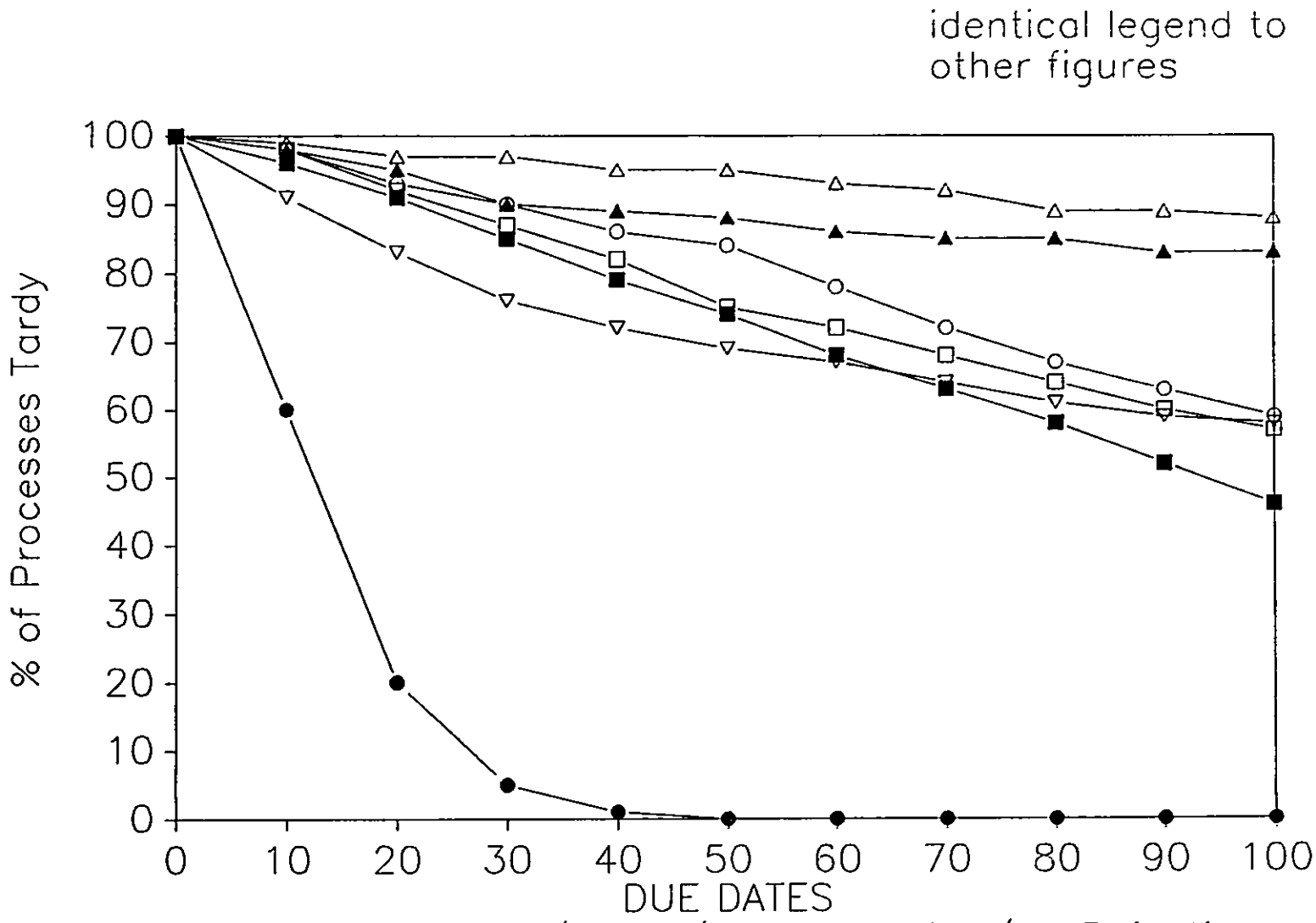


Figure A–23 0.06/PAL 2/Bad Allocation/No Rejection

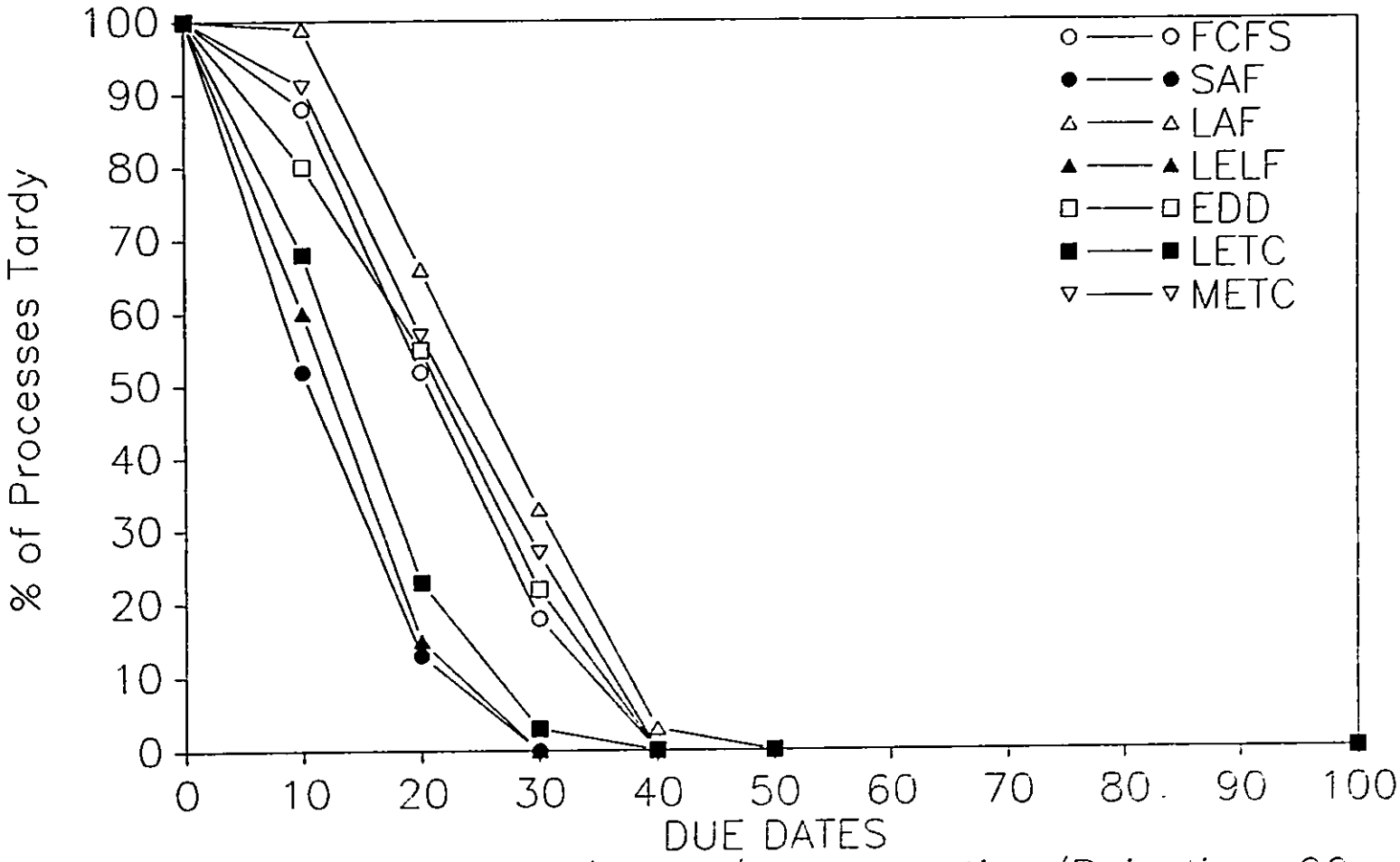


Figure A-24 0.06/PAL 2/Bad Allocation/Rejection=20

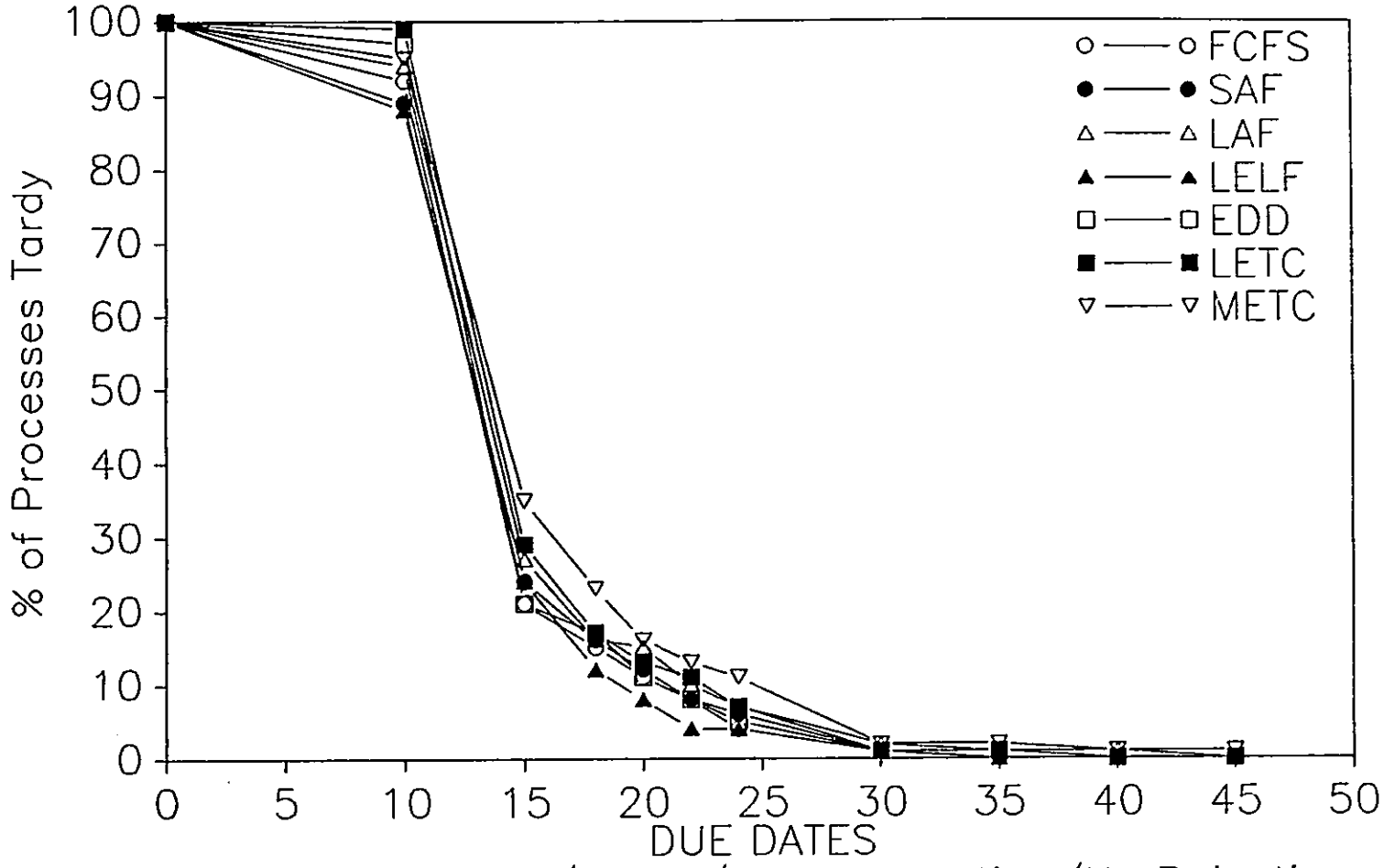


Figure A–25 0.02/PAL 3/Good Allocation/No Rejection

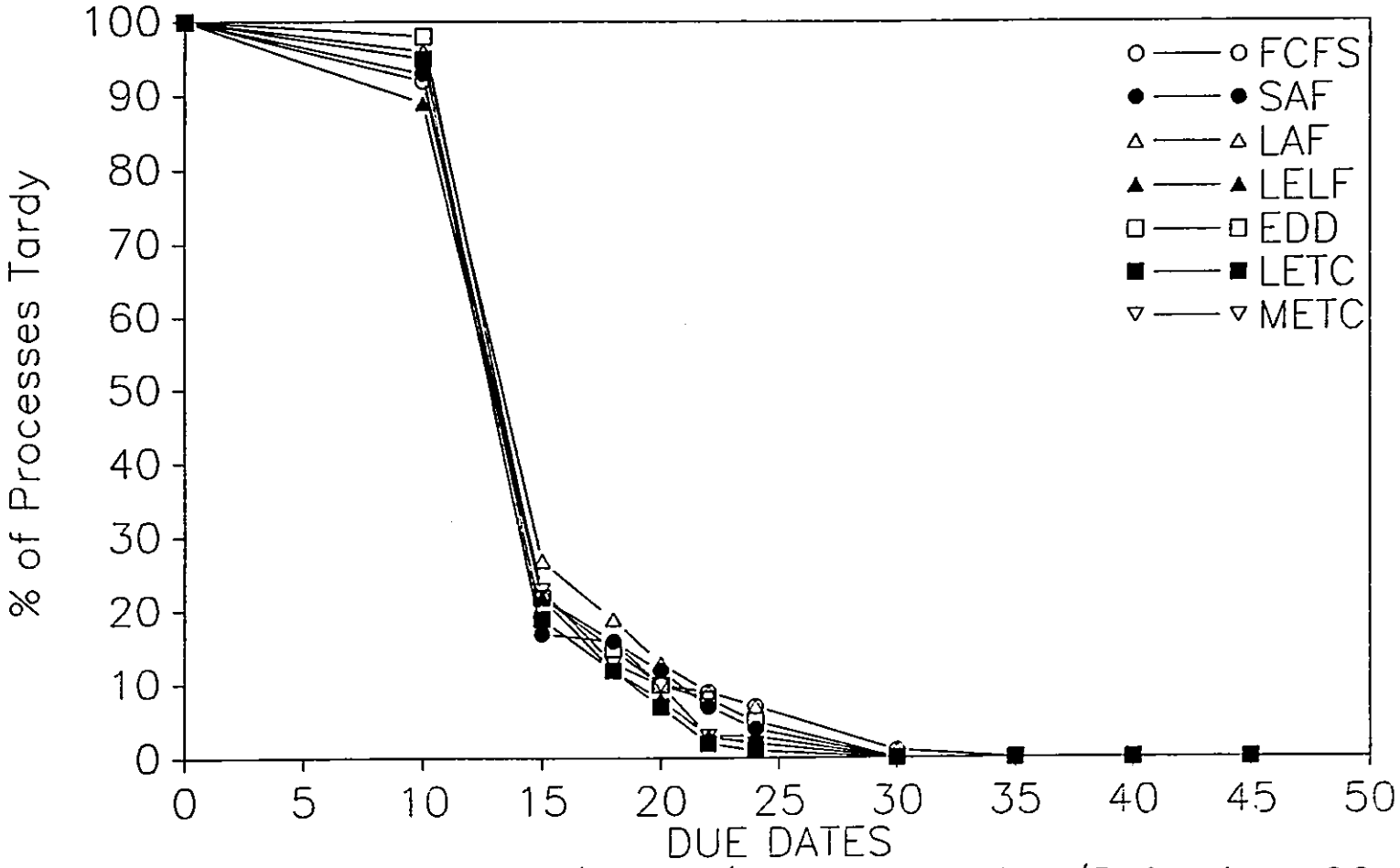


Figure A–26 0.02/PAL 3/Good Allocation/Rejection=20

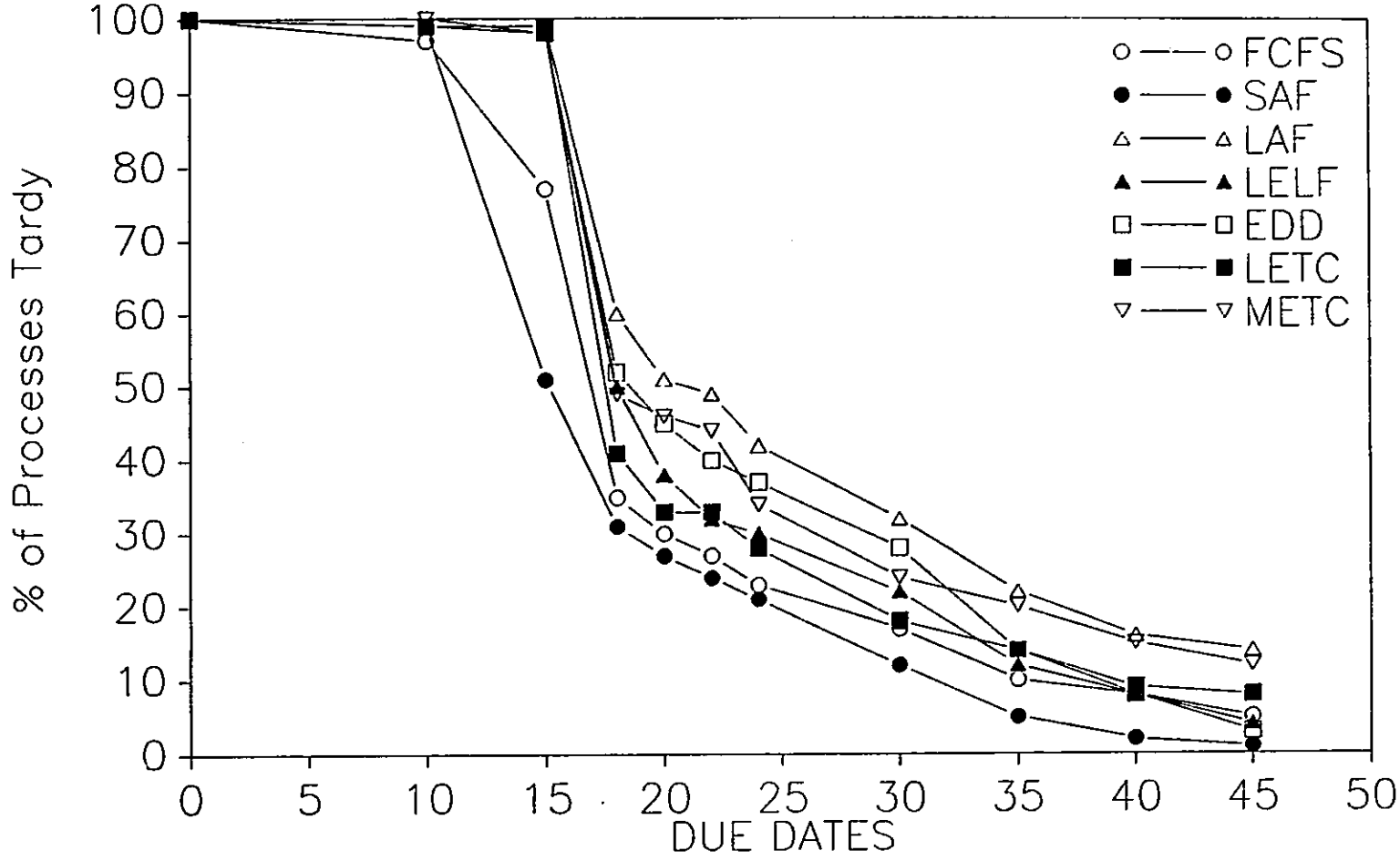


Figure A–27 0.02/PAL 3/Bad Allocation/No Rejection

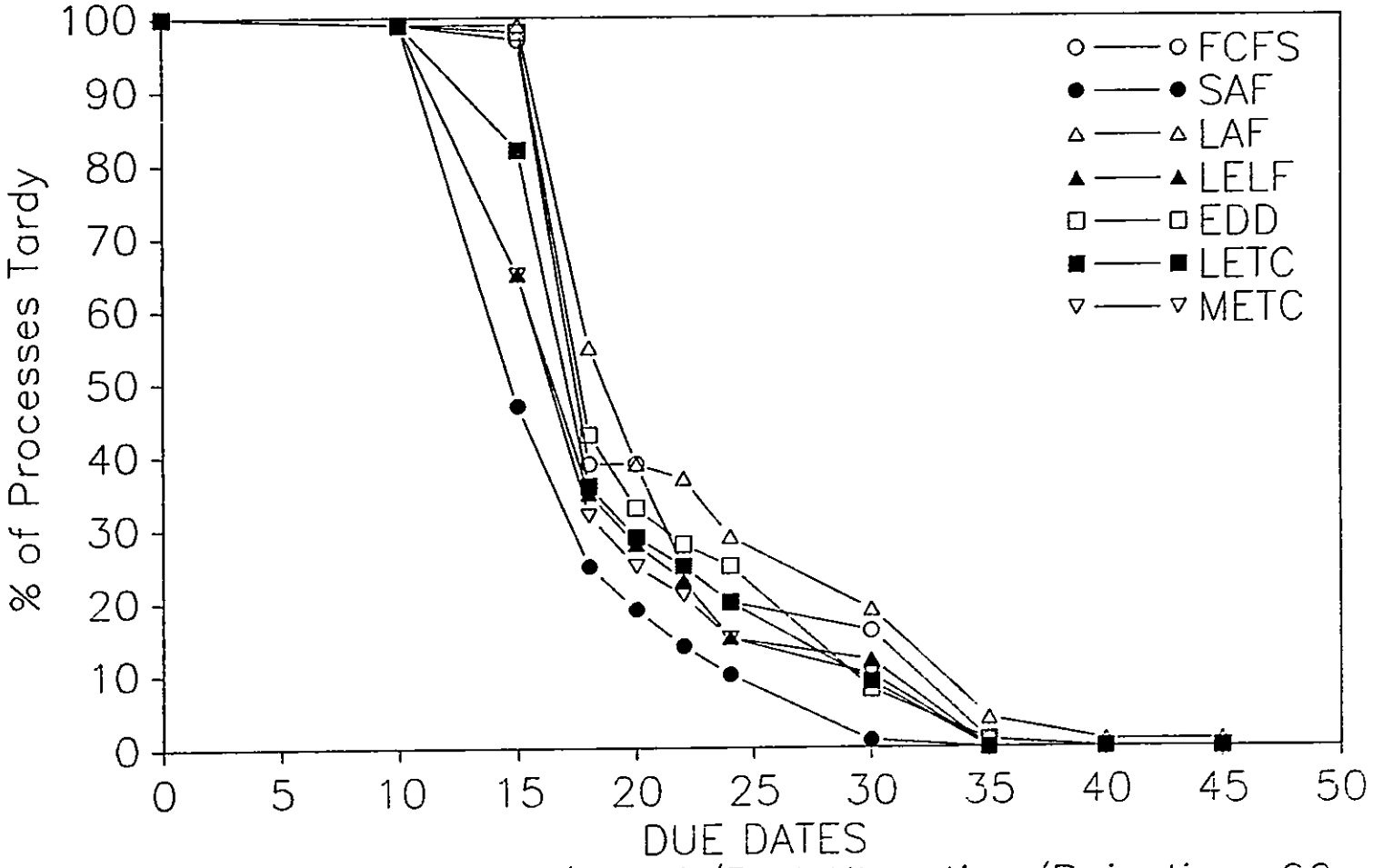


Figure A–28 0.02/PAL 3/Bad Allocation/Rejection=20

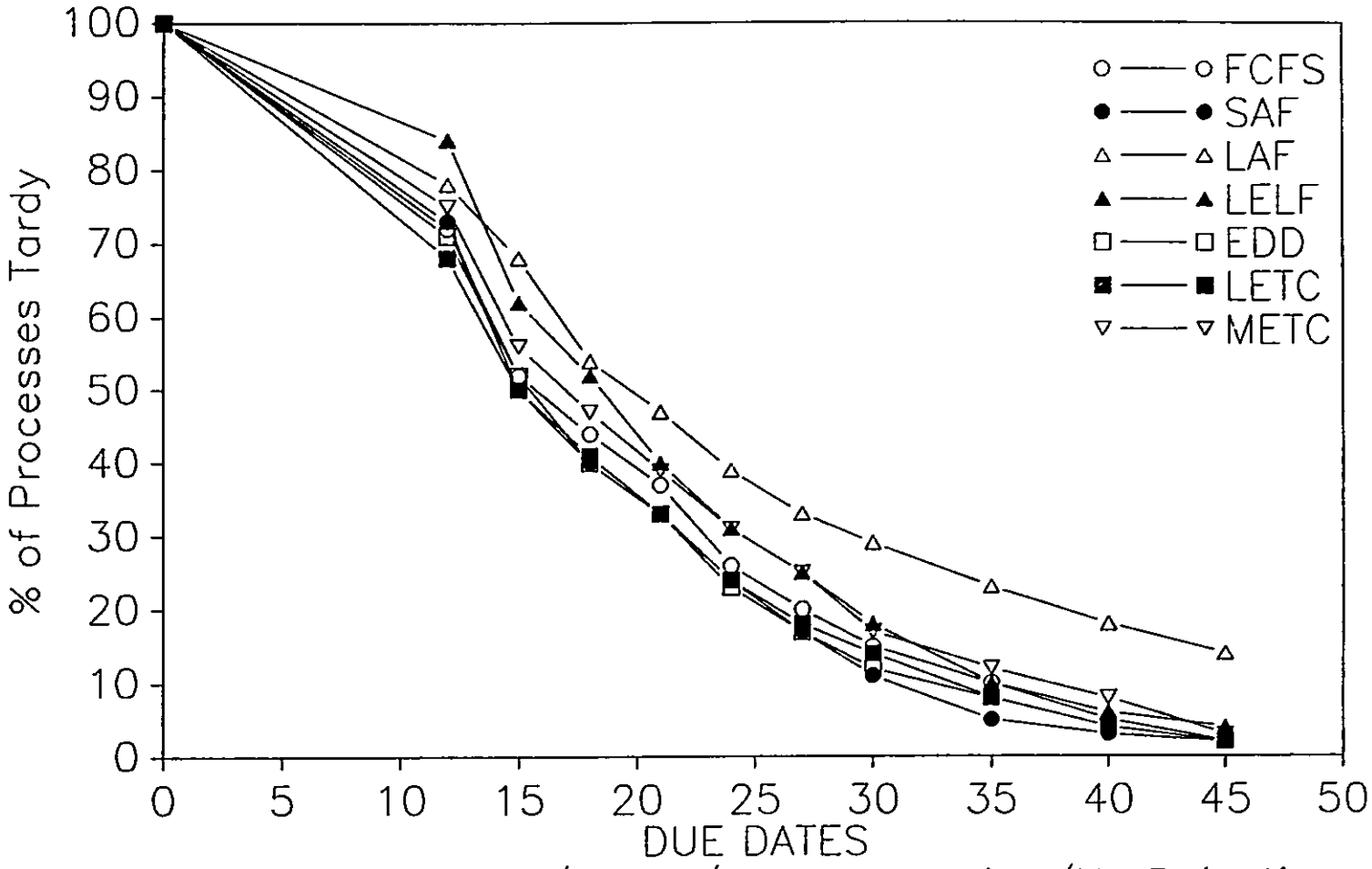


Figure A–29 0.04/PAL 3/Good Allocation/No Rejection

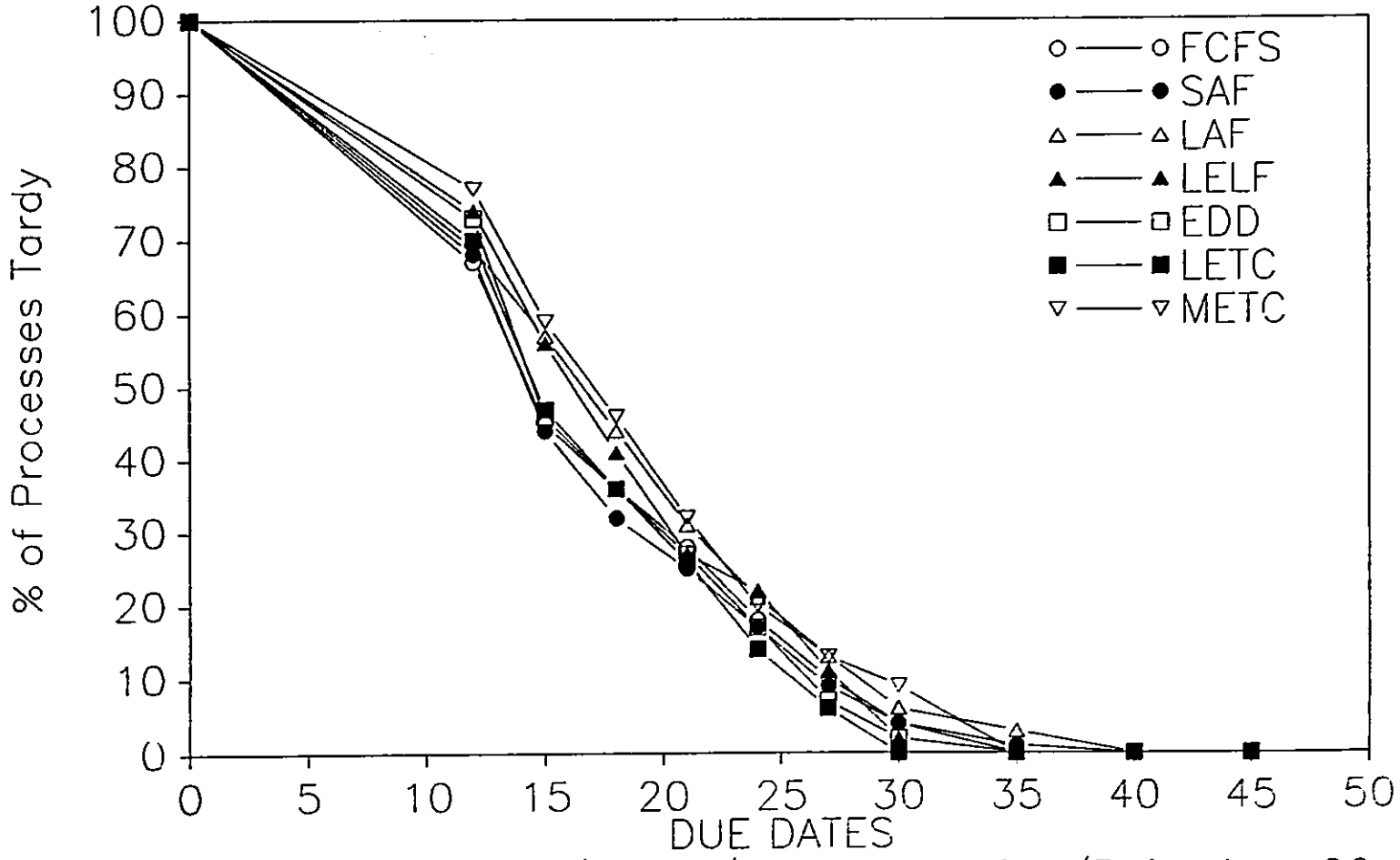


Figure A-30 0.04/PAL 3/Good Allocation/Rejection=20

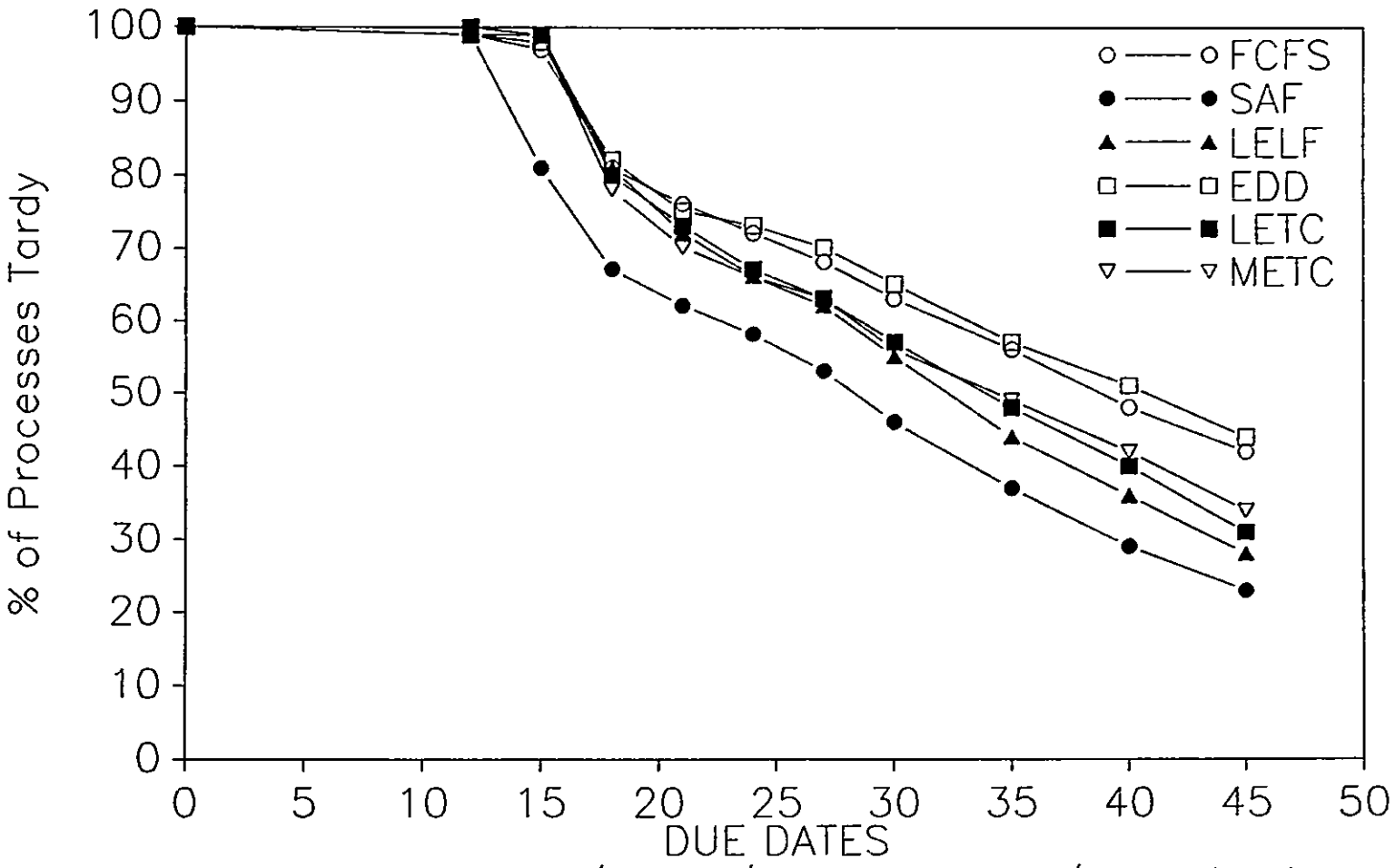


Figure A–31 0.04/PAL 3/Bad Allocation/No Rejection

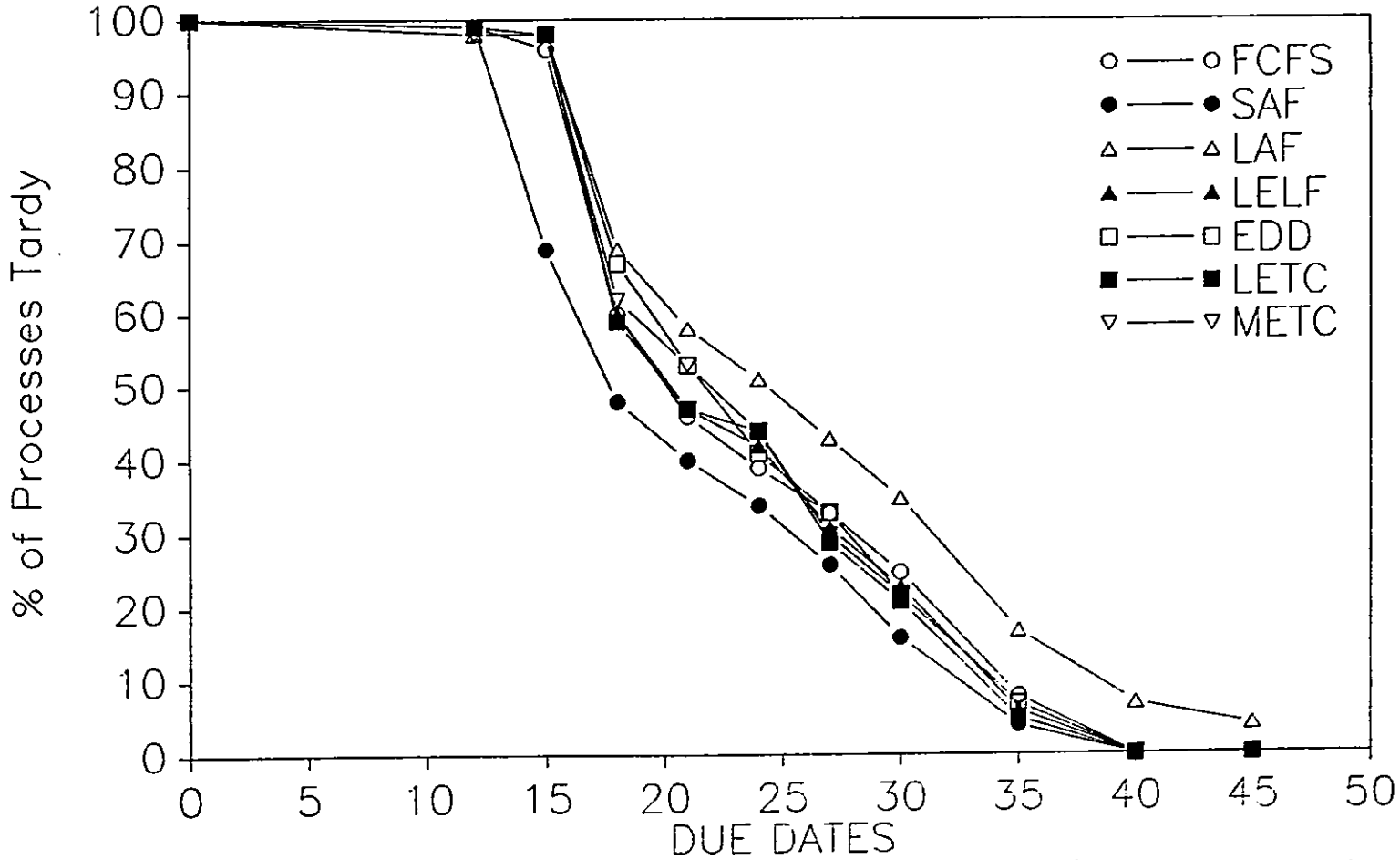


Figure A-32 0.04/PAL 3/Bad Allocation/Rejection=20

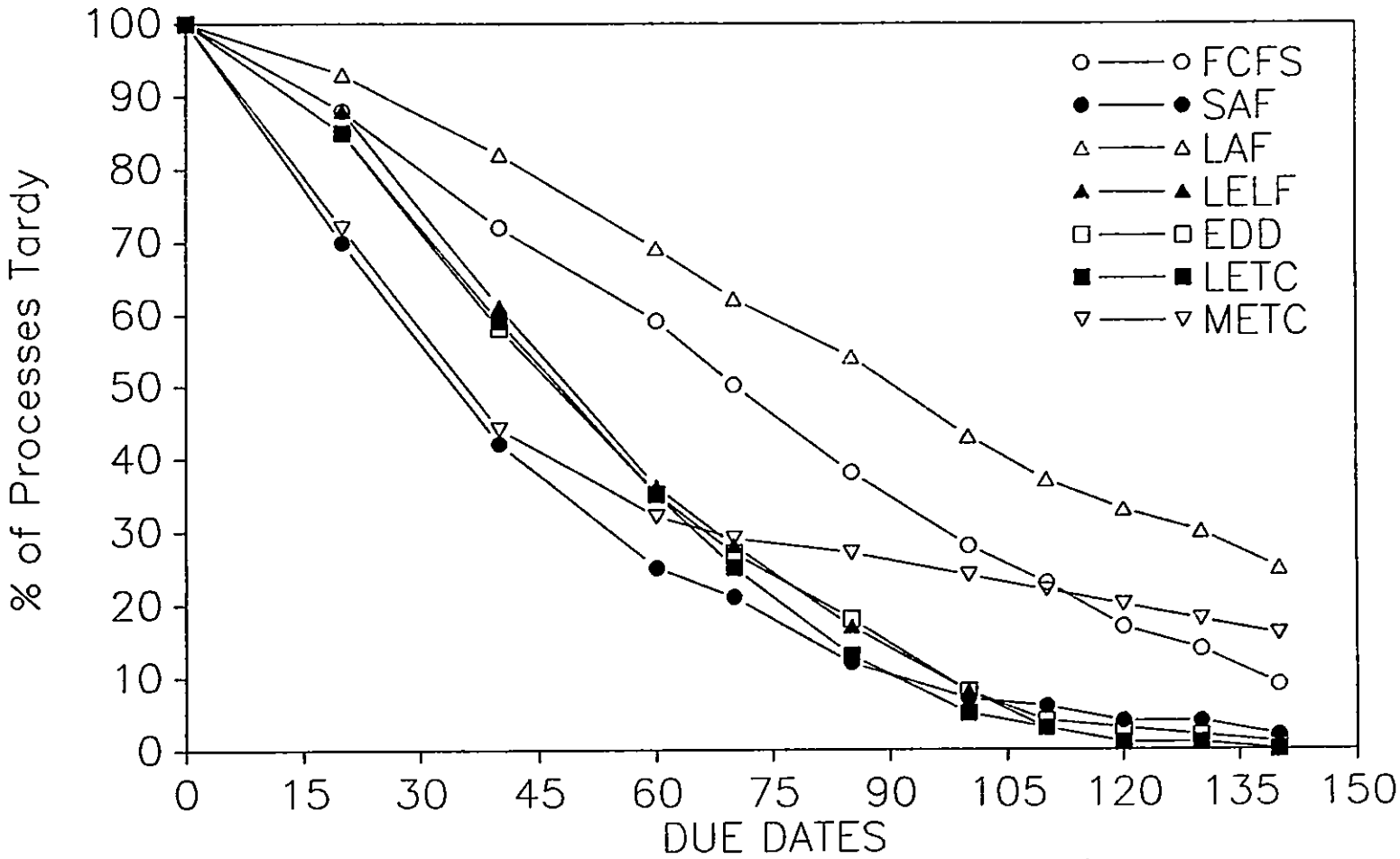


Figure A–33 0.06/PAL 3/Good Allocation/No Rejection

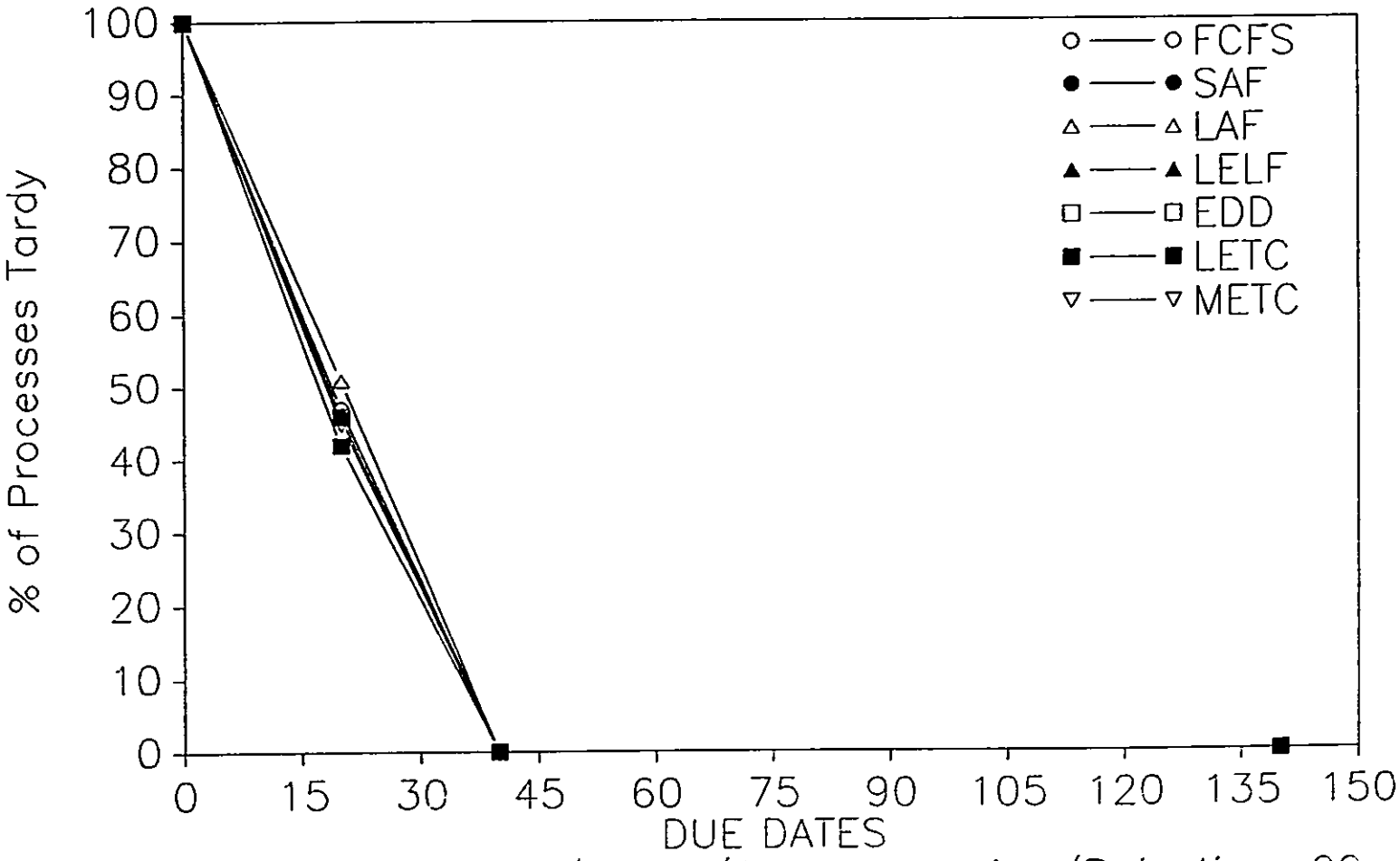


Figure A-34 0.06/PAL 3/Good Allocation/Rejection=20

same legend as
figure A–1

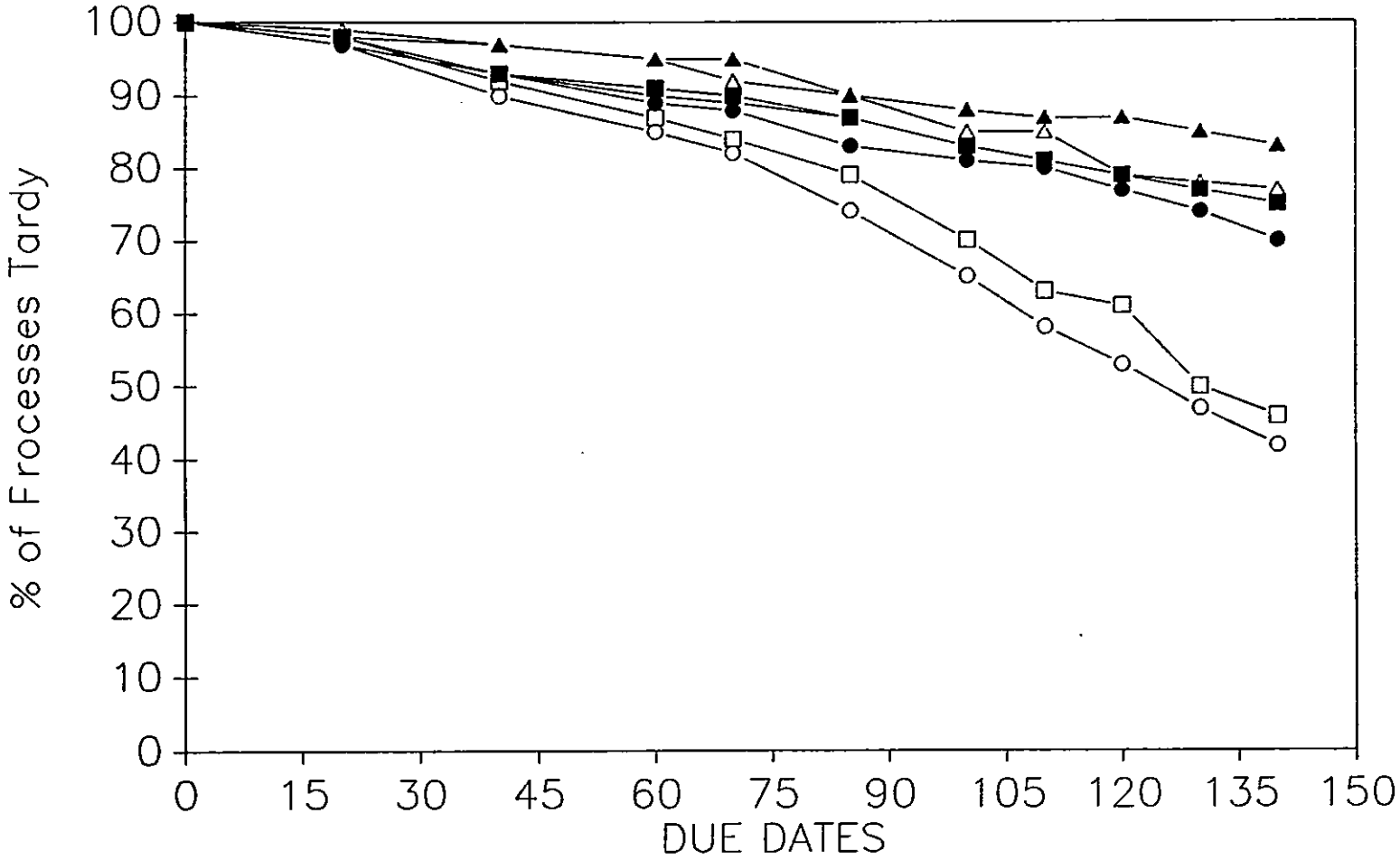


Figure A–35 0.06/PAL 3/Bad Allocation/No Rejection

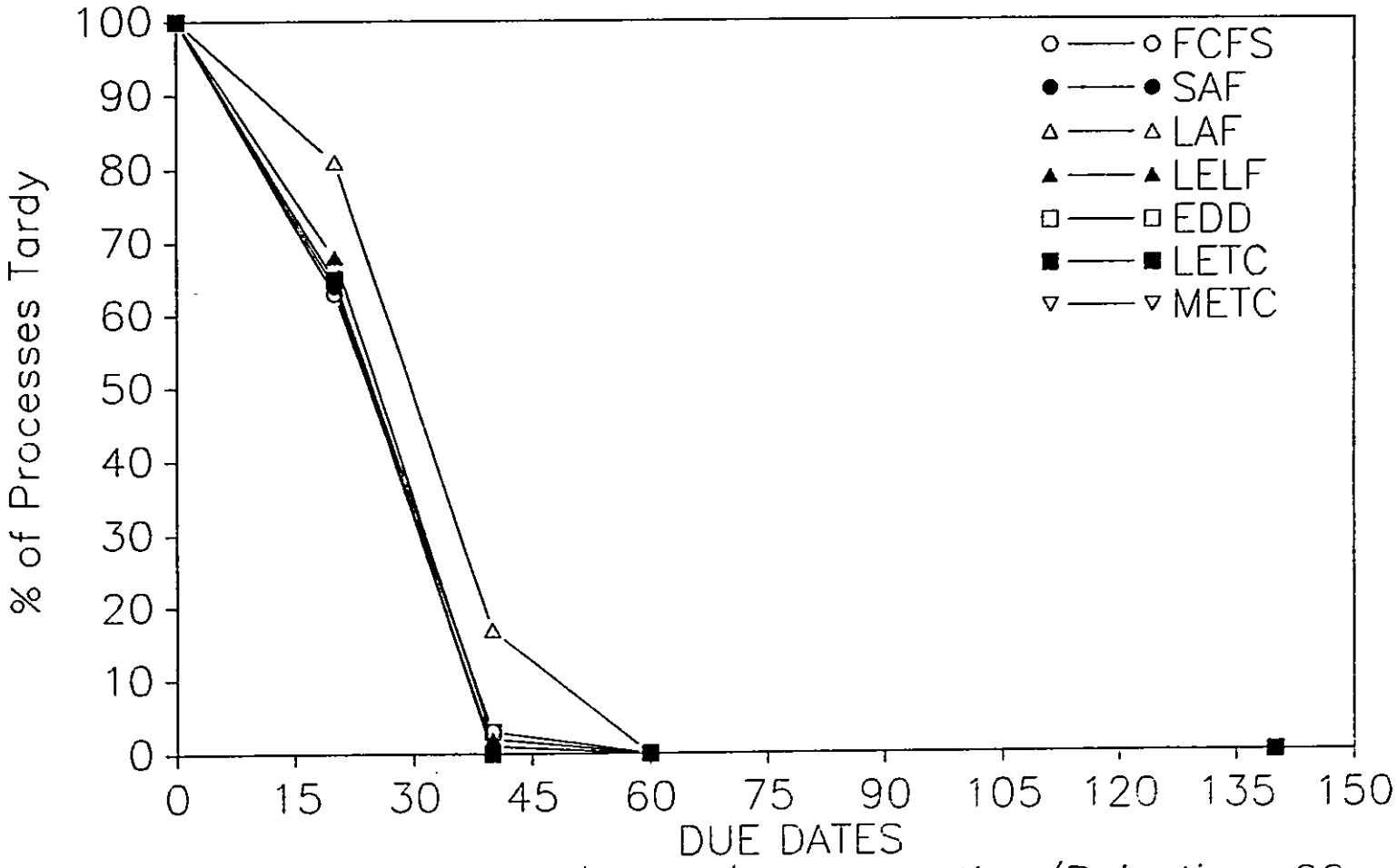


Figure A-36 0.06/PAL 3/Bad Allocation/Rejection=20

ANNEX B

0.02/PAL 1/Good Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	99	97	99	98	99	98	98
15	98	93	98	97	97	97	95
18	43	37	44	36	48	53	39
20	33	26	37	28	39	33	25
22	26	21	28	23	30	24	19
24	23	17	24	17	24	20	17
30	17	11	17	6	18	8	10
35	9	7	11	3	11	4	5
40	7	4	8	2	6	0	0
45	5	3	5	0	5	0	0

Network Statistics (PAL 1)

Average Completion Time	22	20	22	20	22	20	19

Event Statistics (S_1)

Average Event Queuing	2	2	2	2	2	1	1
% of events rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %)

Processor 1:	60	65	59	61	60	60	60
Processor 2:	67	68	66	69	67	62	66
Processor 3:	60	63	59	60	59	60	60
Processor 4:	80	80	85	80	78	77	78
Processor 5:	91	91	85	92	92	91	90
Processor 6:	90	90	91	92	89	90	92

0.02/PAL 1/Good Allocation/No Rejection (continued)

Ready to Run Activity Queue

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	08	08	08	08	08	08	08
MQLA	03	04	04	04	04	04	04
AQLB	0.6	0.6	0.6	0.6	0.6	0.6	0.6
AQLA	0.1	0.1	0.1	0.1	0.1	0.1	0.1

MQLB: Maximum Queue Length Before the dispatching rule is applied;
MQLA: Maximum Queue Length After the dispatching rule is applied;
AQLB: Average Queue Length Before the dispatching rule is applied; and
AQLA: Average Queue Length After the dispatching rule is applied.

0.02/PAL 1/Good Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	99	99	98	98	98	98	98
15	96	94	98	97	97	97	96
18	44	37	43	35	47	38	29
20	28	25	32	27	32	33	25
22	22	19	20	21	24	23	20
24	19	16	17	16	10	17	17
30	10	11	08	05	09	05	05
35	03	02	01	02	02	00	02
40	00	00	00	00	00	00	00
45	00	00	00	00	00	00	00

Network Statistics (PAL 1)

Average time of Completion	20	20	20	20	20	20	20

Event Statistics (S_1)

Average Event Queuing Time	1	1	1	1	1	1	1
% of events rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %)

Processor 1:	61	64	61	62	63	62	61
Processor 2:	68	68	68	69	68	69	69
Processor 3:	61	63	62	63	63	61	60
Processor 4:	80	81	86	81	83	80	80
Processor 5:	91	92	87	92	90	92	93
Processor 6:	90	87	93	93	92	90	93

0.02/PAL 1/Good Allocation/Rejection =20 (continued)

Ready to Run Activity Queue

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	8	8	8	8	8	8	8
MQLA	4	4	4	4	4	4	4
AQLB	0.6	0.6	0.6	0.6	0.6	0.6	0.6
AQLA	0.1	0.1	0.1	0.1	0.1	0.1	0.1

MQLB: Maximum Queue Length Before the dispatching rule is applied;
MQLA: Maximum Queue Length After the dispatching rule is applied;
AQLB: Average Queue Length Before the dispatching rule is applied;
AQLA: Average Queue Length After the dispatching rule is applied.

0.02/PAL 1/Bad Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	99	99	98	98	98	98	100
15	99	86	98	98	98	98	94
18	46	30	94	47	51	60	33
20	42	28	93	44	45	60	30
22	34	24	70	36	36	47	27
24	28	20	67	30	33	35	18
30	22	14	38	24	16	23	15
35	11	07	31	14	10	15	06
40	10	06	25	12	05	11	03
45	07	03	20	09	03	09	00

Network Statistics (PAL 1)

Average Time of Completion	24	21	34	25	23	26	21

Event Statistics (S_1)

Average Event Queuing time	3	2	5	3	2	4	1
% of events rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in percentage)

Processor 1:	77	81	79	80	82	83	82
Processor 2:	76	82	81	78	80	79	78
Processor 3:	68	74	66	73	70	72	72
Processor 4:	75	80	76	74	70	75	75
Processor 5:	76	75	59	72	69	71	73
Processor 6:	87	90	74	85	86	86	88

0.02/PAL 1/Bad Allocation/No Rejection (continued)

Ready to Run Activity Queue Statistics

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	08	08	08	08	08	09	08
MQLA	06	05	06	06	06	06	05
AQLB	1	1	1	1	1	1	1
AQLA	0	0.3	0.8	0.5	0.5	0.5	0.5

MQLB: Maximum Queue Length Before the dispatching rule is applied;
MQLA: Maximum Queue Length After the dispatching rule is applied;
AQLB: Average Queue Length Before the dispatching rule is applied;
AQLA: Average Queue Length After the dispatching rule is applied

0.02/PAL 1/Bad Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	99	99	97	99	99	99	99
15	97	84	97	98	99	97	97
18	39	25	94	45	50	42	45
20	33	23	91	38	43	36	39
22	26	20	53	25	25	23	26
24	23	16	51	21	19	21	24
30	14	09	17	09	05	12	15
35	02	01	09	01	00	00	03
40	02	00	04	00	00	00	00

Network Statistics (PAL 1)

Average Time of Completion	21	19	25	22	20	21	21

Event Statistics (S_1)

Average Event Queuing Time	1	1	1	1	1	1	1
% of Events Rejected	8	5	12	8	3	8	8

Processor Statistics

Processor 1:	82	81	80	82	83	82	82
Processor 2:	80	81	82	80	81	81	81
Processor 3:	70	75	68	70	71	70	70
Processor 4:	71	82	77	75	71	71	71
Processor 5:	71	77	62	71	73	77	72
Processor 6:	86	89	77	80	88	85	85

0.02/PAL 1/Bad Allocation/Rejection=20 (continued)

Ready to Run Activity Queue Statistics

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	8	7	8	8	9	8	8
MQLA	6	5	6	6	6	6	6
AQLB	1	0.7	1.2	1	1	1.2	1.1
AQLA	0.4	0.2	0.7	0.6	0.5	0.4	0.6

MQLB: Maximum Queue Length Before the dispatching rule is applied;
MQLA: Maximum Queue Length After the dispatching rule is applied;
AQLB: Average Queue Length Before the dispatching rule is applied;
AQLA: Average Queue Length After the dispatching rule is applied.

0.04/PAL 1/Good Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
15	97	93	98	99	98	98	97
20	65	70	71	70	71	71	62
24	55	53	61	62	62	62	55
28	49	44	55	53	55	56	49
34	40	32	43	43	44	47	39
40	30	25	38	31	36	37	31
45	26	21	34	25	32	31	27
50	20	14	28	20	25	22	23
55	15	10	21	16	20	16	19
60	13	08	19	12	15	13	12

Network Statistics (PAL 1)

Average Time of Completion	35	31	39	36	38	37	36

Event Statistics (S_1)

Average queuing time	9	8	10	9	10	9	10
% of events rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %)

Processor 1:	36	38	32	33	36	37	34
Processor 2:	41	42	37	40	42	41	41
Processor 3:	34	38	29	33	35	36	33
Processor 4:	58	57	58	59	55	55	58
Processor 5:	69	73	61	70	72	73	70
Processor 6:	70	66	71	72	70	68	71

0.04/PAL 1/Good Allocation/No Rejection (continued)

Ready to Run Activity Queue Statistics

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	9	9	9	9	9	9	9
MQLA	4	4	4	4	4	4	4
AQLB	1.0	1.2	1.4	1.3	1.3	1.3	1.4
AQLA	0.3	0.2	0.4	0.3	0.3	0.3	0.4

MQLB: Maximum Queue Length Before the dispatching rule has been applied;
MQLA: Maximum Queue Length After the dispatching rule has been applied;
AQLB: Average Queue Length before the dispatching rule has been applied;
AQLA: Average Queue Length after the dispatching rule has been applied.

0.04/PAL_1/Good Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
15	96	91	99	96	99	99	98
20	45	44	55	53	58	59	48
24	32	32	38	38	42	46	35
28	24	23	30	24	23	25	24
34	10	08	13	08	08	11	06
40	00	00	00	00	00	01	00
45	00	00	00	00	00	00	00

Network Statistics (PAL_1)

Average Time of Completion	22	22	24	24	24	24	23

Event Statistics (S_1)

Average queuing time	2	2	3	3	3	2	2
% of events rejected	15	17	16	16	16	15	15

Processor Statistics

Processor 1:	40	43	37	38	39	40	40
Processor 2:	46	47	43	46	47	46	45
Processor 3:	38	43	38	40	40	40	40
Processor 4:	63	63	66	65	64	64	65
Processor 5:	75	74	68	76	77	77	77
Processor 6:	74	70	76	77	76	76	77

0.04/PAL 1/Good Allocation/Rejection=20 (continued)

Ready to Run Activity Queue Statistics

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	09	10	09	09	09	10	10
MQLA	04	04	04	04	04	04	04
AQLB	1.2	1.0	1.2	1.2	1.1	1.1	1.1
AQLA	0.2	0.2	0.3	0.3	0.3	0.2	0.2

MQLB: Maximum Queue Length Before the dispatching rule is applied;
MQLA: Maximum Queue Length After the dispatching rule is applied;
AQLB: Average Queue Length Before the dispatching rule is applied;
AQLA: Average Queue Length After the dispatching rule is applied.

0.04/PAL 1/Bad Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
15	99	91	99	99	99	99	96
20	76	57	99	79	83	76	49
24	66	49	97	71	68	63	39
28	56	40	96	51	60	48	27
34	45	30	95	32	51	33	18
40	37	24	94	21	41	24	14
45	33	19	93	18	34	15	11
50	28	13	93	11	29	09	10
55	25	10	91	08	22	06	08
60	21	05	89	04	19	03	03

Network Statistics (PAL 1)

Average Time of Completion	40	30	165	38	40	31	28

Event Statistics (S_1)

Average Queuing Time	10	07	70	10	10	06	05
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %)

Processor 1:	66	63	62	63	65	66	62
Processor 2:	60	63	65	64	60	60	62
Processor 3:	64	52	41	45	46	47	47
Processor 4:	50	62	55	52	48	49	52
Processor 5:	40	54	28	43	43	46	48
Processor 6:	70	77	42	45	42	71	75

0.04/PAL 1/Bad Allocation/No Rejection (continued)

Ready to Run Activity Queue Statistics:

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	10	10	09	10	10	10	09
MQLA	06	06	06	06	06	06	06
AQLB	2.2	1.7	3.0	2.0	2.0	2.0	2.0
AQLA	1.3	0.8	2.0	1.0	1.2	1.0	1.0

MQLB: Maximum Queue Length Before the dispatching rule is applied;
MQLA: Maximum Queue Length After the dispatching rule is applied;
AQLB: Average Queue Length Before the dispatching rule is applied;
AQLA: Average Queue Length After the dispatching rule is applied.

0.04/PAL 1/Bad Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
15	98	86	99	97	97	99	98
20	62	39	96	65	65	71	46
24	49	29	77	51	50	53	31
28	34	19	52	37	29	40	27
34	14	05	37	18	10	22	10
40	03	00	18	01	01	01	03
45	02	00	02	00	00	00	00
50	00	00	00	00	00	00	00

Network Statistics (PAL 1)

Average Time of Completion	25	21	31	25	25	26	23

Event Statistics (S_1)

Average Queuing Time	2	2	2	2	2	3	2
% of Events Rejected	18	13	30	8	11	21	17

Processor Statistics (idle time in %)

Processor 1:	71	66	69	40	69	69	69
Processor 2:	66	66	72	47	64	64	63
Processor 3:	53	57	52	39	52	53	53
Processor 4:	58	66	65	65	57	58	59
Processor 5:	48	59	42	78	52	54	55
Processor 6:	72	80	60	76	74	74	79

0.04/PAL 1/Bad Allocation/Rejection=20

Ready to Run Activity Queue Statistics

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	09	09	09	09	10	09	09
MQLA	06	06	06	04	06	06	06
AQLB	1.9	1.5	2.0	1.2	1.9	1.7	1.7
AQLA	1.0	0.7	1.3	0.3	1.0	1.0	1.0

MQLB: Maximum Queue Length Before the dispatching rule is applied;
MQLA: Maximum Queue Length After the dispatching rule is applied;
AQLB: Average Queue Length Before the dispatching rule is applied;
AQLA: Average Queue Length After the dispatching rule is applied.

0.06/PAL 1/Good Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
20	95	91	96	97	98	97	94
40	86	82	88	88	89	91	83
60	75	64	78	79	83	79	74
80	62	47	69	71	67	66	66
100	51	37	66	56	58	53	63
120	40	26	61	42	44	42	58
140	32	16	59	33	33	34	52
160	24	11	56	26	26	26	45
180	19	05	52	19	19	20	38
200	15	03	46	15	13	14	30

Network Statistics (PAL 1)

Average Time of Completion	118	88	171	120	121	120	147

Event Statistics (S_1)

Average Queuing Time	51	36	78	55	52	51	65
% of events Rejected	0	0	0	0	0	0	0

Processor Statistics

Processor 1:	19	22	14	16	16	16	18
Processor 2:	22	24	16	22	22	22	22
Processor 3:	19	27	14	17	18	17	17
Processor 4:	37	40	28	33	32	31	36
Processor 5:	36	53	32	44	46	43	40
Processor 6:	38	43	52	40	40	39	44

0.06/PAL 1/Good Allocation/No Rejection (continued)

Ready to Run Activity Queue Statistics

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	9	10	10	9	10	9	9
MQLA	4	4	5	4	4	4	4
AQLB	2.2	2.0	2.3	2.1	2.2	2.0	2.0
AQLA	0.7	0.5	1.0	0.8	0.7	0.7	0.6

MQLB: Maximum Queue Length Before the dispatching rule is applied;
MQLA: Maximum Queue Length After the dispatching rule is applied;
AQLB: Average Queue Length Before the dispatching rule is applied;
AQLA: Average Queue Length After the dispatching rule is applied.

0.06/PAL 1/Good Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
20	66	58	75	77	66	71	58
40	00	00	02	01	00	05	00
60	00	00	00	00	00	00	00

Network Statistics (PAL 1)

Average Time of Completion	25	24	28	27	26	26	24

Event Statistics (S_1)

Average Queuing Time	4	3	4	4	4	4	3
% of Events Rejected	24	21	32	28	23	26	22

Processor Statistics (idle time in %):

Processor 1:	26	29	24	24	25	25	25
Processor 2:	32	32	32	34	29	30	27
Processor 3:	26	32	24	29	24	26	23
Processor 4:	46	50	50	49	44	43	43
Processor 5:	56	61	53	57	55	56	54
Processor 6:	57	53	64	61	55	54	54

Ready to Run Activity Queue Statistics

MQLB	9	9	10	9	9	9	9
MQLA	4	4	5	4	4	4	4
AQLB	1.7	1.6	1.7	1.7	1.7	1.7	1.7
AQLA	0.5	0.3	0.5	0.5	0.5	0.5	0.5

MQLB, MQLA, AQLB and AQLA have been defined previously.

0.06/PAL 1/Bad Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
20	96	91	99	97	97	96	93
40	90	77	97	89	90	83	77
60	80	68	96	81	84	72	61
80	73	55	95	71	79	63	50
100	65	43	93	54	71	54	41
120	55	33	92	41	65	46	36
140	46	24	88	25	57	34	29
160	38	20	85	16	48	25	22
180	32	16	84	10	39	20	18
200	28	09	82	07	33	15	13

Network Statistics (PAL 1)

Average Time of Completion	156	102	548	108	178	119	108

Event Statistics (S_1)

Average Queuing Time	70	57	268	46	81	51	45
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %)

Processor 1:	56	47	58	58	46	45	53
Processor 2:	46	48	60	55	50	48	45
Processor 3:	29	35	41	32	31	33	30
Processor 4:	33	50	54	32	32	28	30
Processor 5:	14	42	22	27	29	25	15
Processor 6:	64	67	33	60	63	60	64

0.06/PAL 1/Bad Allocation/No Rejection (continued)

Ready to Run Activity Queue Statistics

	FCFS	SAF	LAF	LELF	EDD	LETC	METC
MQLB	10	9	8	9	9	10	10
MQLA	6	5	5	5	6	6	6
AQLB	3.1	2.7	3.4	3.0	3.0	3.0	3.0
AQLA	1.8	1.3	2.4	1.5	1.8	1.6	1.6

MQLB, MQLA, AQLB and AQLA have been defined previously.

0.06/PAL 1/Bad Allocation/Rejection =20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
20	78	62	97	68	80	78	64
40	04	00	31	02	05	10	00
60	00	00	00	00	00	00	00

Network Statistics (PAL 1)

Average Time of Completion	28	25	35	28	28	28	25

Event Statistics (S_1)

Average Queuing Time	4	4	3	4	4	3	3
% of Events Rejected	36	26	45	35	40	40	32

Processor Statistics (idle time in %)

Processor 1:	61	55	61	60	59	57	59
Processor 2:	54	51	64	62	55	50	54
Processor 3:	41	44	42	40	40	40	38
Processor 4:	47	60	59	61	55	46	43
Processor 5:	31	49	31	33	34	36	40
Processor 6:	63	74	46	49	57	62	69

Ready to Run Activity Queue Statistics

MQLB:	9	9	10	10	10	10	9
MQLA:	6	5	6	6	6	6	6
AQLB:	2.5	2.0	2.8	2.0	2.0	2.2	2.4
AQLA:	1.4	0.9	1.8	1.4	1.3	1.2	1.4

MQLB, MQLA, AQLB and AQLA have been defined previously.

0.02/PAL 2/Good Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	84	86	90	89	85	91	87
15	23	24	21	20	22	32	28
18	12	15	13	11	12	23	20
20	09	11	09	07	09	17	17
22	04	10	06	02	07	11	15
24	04	08	04	02	05	05	12
30	01	01	01	00	01	03	05
35	01	01	01	00	01	01	02
40	00	00	00	00	00	01	01
45	00	00	00	00	00	00	01

Network Statistics (PAL 2)

Average Time of Completion	13	14	14	13	13	15	15

Event Statistics (S_1)

Average Queuing Time	1	1	1	1	1	1	1
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-1

Ready to Run Activity Queue Statistics: see page B-2

0.02/PAL 2/Good Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	82	79	93	88	90	92	87
15	23	22	22	20	22	22	21
18	13	14	15	11	16	10	13
20	11	09	11	07	13	07	10
22	06	05	07	02	09	02	03
24	03	04	04	02	06	02	02
30	01	01	02	00	02	00	00
35	00	00	00	00	00	00	00

Network Statistics (PAL 2)

Average Time of Completion	12	13	14	14	14	14	14

Event Statistics (S_2)

Average Queuing Time	1	1	1	1	1	1	1
% of Events Rejected	5	5	5	5	5	0	0

Processor Statistics (idle time in %): see page B-3

Ready to Run Activity Queue Statistics: see page B-4

0.02/PAL 2/Bad Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	57	13	98	51	50	49	33
15	21	03	51	22	22	20	12
18	12	01	34	12	09	11	08
20	08	01	30	07	07	05	02
22	06	01	23	05	05	04	02
24	04	00	19	02	03	01	00
30	00	00	13	00	01	01	00
35	00	00	05	00	00	00	00
40	00	00	02	00	00	00	00
45	00	00	01	00	00	00	00

Network Statistics (PAL 2)

Average Time of Completion	12	9	19	12	12	12	11

Event Statistics (S_2)

Average Queuing Time	1	1	2	1	1	1	1
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-5

Ready to Run Activity Queue Statistics: see page B-6

0.02/PAL 2/Bad Allocation/ Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	47	13	99	40	35	43	48
15	22	03	41	20	11	25	29
18	10	02	27	08	03	13	16
20	05	01	24	02	00	07	10
22	03	01	13	00	00	02	05
24	03	00	08	00	00	00	01
30	00	00	04	00	00	00	00
40	00	00	00	00	00	00	00

Network Statistics (PAL 2)

Average Time of Completion	12	9	17	12	11	12	12

Event Statistics (S_2)

Average Queuing Time	1	1	1	1	1	1	1
% of Events Rejected	0	0	3	0	0	0	0

Processor Statistics (idle time in %) : see page B-7

Ready to Run Activity Queue Statistics: see page B-8

0.04/PAL 2/Good Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
12	65	61	82	72	67	67	66
15	47	40	55	54	47	49	48
18	33	29	44	42	36	36	37
21	25	21	35	39	28	29	27
24	19	16	28	33	22	22	20
27	11	11	23	27	15	16	15
30	08	08	14	23	11	11	10
35	04	04	10	18	06	07	08
40	02	01	07	14	03	03	04
45	01	00	04	06	01	01	01

Network Statistics (PAL 2):

Average Time of Completion	17	16	20	22	18	18	18
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-9

Ready to Run Activity Queue Statistics: see page B-10

0.04/PAL 2/Good Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
12	61	58	76	63	60	63	54
15	44	35	51	40	37	39	38
18	31	27	37	31	29	30	28
21	21	18	28	26	25	27	25
24	13	13	20	17	18	19	15
27	07	08	10	08	07	08	10
30	03	03	05	05	03	03	03
35	00	00	00	00	00	00	00

Network Statistics (PAL 2)

Average Time of Completion	16	15	18	17	16	17	16

Event Statistics (S_2)

Average Queuing Time	2	2	2	2	2	2	2
% of Events Rejected	4	5	6	5	8	7	6

Processor Statistics (idle time in %): see page B-11

Ready to Run Activity Queue Statistics: see page B-12

0.04/PAL 2/Bad Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
12	74	27	97	74	71	66	48
15	61	17	89	55	49	43	28
18	50	10	79	44	39	32	24
21	39	05	77	32	32	22	18
24	35	02	63	24	23	15	13
27	32	00	57	16	21	12	13
30	27	00	51	14	16	09	11
35	16	00	43	08	12	04	08
40	09	00	33	05	07	03	08
45	05	00	25	03	02	02	06

Network Statistics (PAL 2)

Average Time of Completion	22	11	33	21	20	17	16

Event Statistics (S_2)

Average Queuing Time	4	1	8	3	3	3	3
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-13

Ready to Run Activity Queue Statistics: see page B-14

0.04/PAL 2/Bad Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
12	66	25	98	63	46	58	55
15	48	16	67	43	24	38	37
18	35	07	50	33	16	28	25
21	25	04	44	20	10	18	20
24	20	01	32	10	04	04	15
27	16	00	23	01	01	00	11
30	08	00	14	00	01	00	11
35	01	00	06	00	00	00	01
40	00	00	02	00	00	00	00
45	00	00	00	00	00	00	00

Network Statistics (PAL 2)

Average Time of Completion	17	11	21	16	13	14	15

Event Statistics (S_2)

Average Queuing Time	2	1	2	1	1	3	3
% of Events Rejected	8	0	13	0	4	0	2

Processor Statistics (idle time in %): see page B-15

Ready to Run Activity Queue Statistics: see page B-16

0.06/PAL 2/Good Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	95	83	97	96	96	96	95
20	75	61	91	75	78	72	80
30	61	44	84	56	65	56	66
40	45	29	78	35	49	40	52
50	32	18	72	20	37	27	44
60	23	10	68	11	23	17	41
70	16	08	64	06	14	12	37
80	11	07	60	04	09	10	33
90	07	06	56	03	06	08	28
100	4	02	48	02	04	05	25

Network Statistics (PAL 2)

Average Time of Completion	43	33	109	36	45	41	71

Event Statistics (S_2)

Average Queuing Time	16	10	47	11	15	13	29
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-17

Ready to Run Activity Queue Statistics: see page B-18

0.06/PAL 2/Good Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	89	87	95	88	88	90	93
20	42	34	52	37	40	39	43
30	08	06	14	07	07	08	16
40	00	00	00	00	00	00	00

Network Statistics (PAL 2)

Average Time of Completion	19	18	21	18	19	19	20

Event Statistics (S_2)

Average Queuing Time	3	3	4	3	3	3	3
% of Events Rejected	14	10	17	13	17	16	20

Processor Statistics (idle time in %): see page B-19

Ready to Run Activity Queue Statistics: see page B-19

0.06/PAL 2/Bad Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	98	60	99	98	98	96	91
20	93	20	97	95	92	91	83
30	90	05	97	90	87	85	76
40	86	01	95	89	82	79	72
50	84	00	95	88	75	74	69
60	78	00	93	86	72	68	67
70	72	00	92	85	68	63	64
80	67	00	89	85	64	58	61
90	63	00	89	83	60	52	59
100	59	0	88	83	57	46	58

Network Statistics (PAL 2)

Average Time of Completion	154	15	289	142	125	99	164

Event Statistics (S_2)

Average Queuing Time	70	1	136	64	55	42	75
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-20

Ready to Run Activity Queue Statistics: see page B-21

0.06/PAL 2/Bad Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	88	52	99	60	80	68	91
20	52	13	66	15	55	23	57
30	18	00	33	00	22	03	27
40	00	00	03	00	00	00	00
50	00	00	00	00	00	00	00

Network Statistics (PAL 2)

Average Time of Completion	21	13	26	22	22	15	23

Event Statistics (S_2)

Average Queuing Time	3	2	3	3	3	2	4
% of Events Rejected	20	3	31	25	27	9	24

Processor Statistics (idle time in %): see page B--22

Ready to Run Activity Queue Statistics: see page B–22

0.02/PAL 3/Good Allocation/No rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	92	89	94	88	97	99	95
15	21	24	27	24	21	29	35
18	15	16	16	12	17	17	23
20	11	12	15	08	11	13	16
22	08	08	10	04	08	11	13
24	04	06	07	04	05	07	11
30	01	01	02	01	01	01	02
35	01	00	01	00	01	01	02
40	00	00	01	00	00	00	01
45	00	00	00	00	00	00	01

Network Statistics (PAL 3)

Average Time of Completion	13	14	14	13	14	15	15

Event Statistics (S_3)

Average Queuing Time	1	1	1	1	1	1	1
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-1

Ready to Run Activity Queue Statistics: see page B-2

0.02/PAL 3/Good Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	92	93	96	89	98	95	93
15	22	17	27	22	22	19	23
18	16	16	19	12	15	12	13
20	10	12	13	08	10	07	10
22	09	07	09	03	08	02	03
24	07	04	07	03	05	01	02
30	01	00	01	00	00	00	00
35	00	00	00	00	00	00	00

Network Statistics (PAL 3)

Average Time of Completion	13	13	14	13	13	13	13

Event Statistics (S_3)

Average Queuing Time	1	1	1	1	1	1	1
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-3

Ready to Run Activity Queue Statistics: see page B-4

0.02/PAL 3/Bad Allocation/No rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	97	99	99	99	99	99	100
15	77	51	99	98	98	99	98
18	35	31	60	50	52	41	49
20	30	27	51	38	45	33	46
22	27	24	49	32	40	33	44
24	23	21	42	30	37	28	34
30	17	12	32	22	28	18	24
35	10	05	22	12	14	14	20
40	08	02	16	08	08	09	15
45	05	01	14	04	03	08	12

Network Statistics (PAL 3)

Average Time of Completion	20	18	29	24	23	23	27

Event Statistics (S_3)

Average Queuing Time	2	2	5	4	3	3	4
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-5

Ready to Run Activity Queue Statistics: see page B-6

0.02/PAL 3/Bad Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
10	99	99	99	99	99	99	99
15	97	47	99	65	98	82	65
18	39	25	55	35	43	36	32
20	39	19	39	28	33	29	25
22	25	14	37	23	28	25	21
24	20	10	29	15	25	20	15
30	16	01	19	12	08	09	10
35	01	00	04	00	01	00	00
40	00	00	01	00	00	00	00

Network Statistics (PAL 3)

Average Time of Completion	20	17	22	20	21	20	21

Event Statistics (S_3)

Average Queuing Time	2	1	2	2	2	2	2
% of Events Rejected	8	5	12	6	8	6	7

Processor Statistics (idle time in %): see page B-7

Ready to Run Activity Queue Statistics: see page B-8

0.04/PAL 3/Good Allocation/No rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
12	72	73	78	84	71	68	75
15	52	50	68	62	52	50	56
18	44	40	54	52	40	41	47
21	37	33	47	40	33	33	39
24	26	24	39	31	23	24	31
27	20	17	33	25	17	18	25
30	15	11	29	18	12	14	17
35	10	05	23	10	08	08	12
40	05	03	18	06	04	04	08
45	02	02	14	04	02	02	03

Network Statistics (PAL 3)

Average Time of Completion	20	19	25	21	19	19	21

Event Statistics (S_3)

Average Queuing Time	4	3	6	4	4	3	4
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-9

Ready to Run Activity Queue Statistics: see page B-10

0.04/PAL 3/Good Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
12	67	68	69	74	73	70	77
15	45	44	57	56	46	47	59
18	36	32	44	41	36	36	46
21	28	25	31	27	27	26	32
24	18	17	21	22	17	14	20
27	10	09	13	11	07	06	13
30	04	04	06	02	02	00	09
35	00	01	03	00	00	00	00
40	00	00	00	00	00	00	00

Network Statistics (PAL 3)

Average Time of Completion	17	16	18	18	17	17	19

Event Statistics (S_3)

Average Queuing Time	2	2	2	3	2	2	3
% of Events Rejected	5	5	9	6	6	7	8

Processor Statistics (idle time in %): see page B-11

Ready to Run Activity Queue Statistics: see page B-12

0.04/PAL 3/Bad Allocation/No rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
12	99	99	99	99	99	99	99
15	97	81	99	99	98	99	99
18	81	67	93	81	82	80	78
21	76	62	88	72	75	73	70
24	72	58	84	66	73	67	66
27	68	53	81	62	70	63	63
30	63	46	80	55	65	57	56
35	56	37	75	44	57	48	49
40	48	29	71	36	51	40	42
45	42	23	68	28	44	31	34

Network Statistics (PAL 3)

Average Time of Completion	47	33	104	48	48	38	41

Event Statistics (S_3)

Average Queuing Time	15	9	47	15	15	10	12
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-13

Ready to Run Activity Queue Statistics: see page B-14

0.04/PAL 3/Bad Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
12	99	99	98	99	99	99	98
15	96	69	98	98	98	98	98
18	60	48	69	60	67	59	62
21	46	40	58	47	53	47	53
24	39	34	51	42	41	44	44
27	33	26	43	31	33	29	30
30	25	16	35	23	22	21	22
35	08	04	17	06	07	05	07
40	00	00	07	00	00	00	00
45	00	00	04	00	00	00	00

Network Statistics (PAL 3)

Average Time of Completion	23	21	26	18	24	23	23

Event Statistics (S_3)

Average Queuing Time	3	3	3	3	3	2	3
% of Events Rejected	19	16	28	18	17	16	16

Processor Statistics (idle time in %): see page B-15

Ready to Run Activity Queue Statistics: see page B-16

0.06/PAL 3/Good Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
20	88	70	93	88	85	85	72
40	72	42	82	61	58	59	44
60	59	25	69	36	35	35	32
70	50	21	62	28	27	25	29
85	38	12	54	17	18	13	27
100	28	7	43	08	08	05	24
110	23	6	37	03	04	03	22
120	17	4	33	01	03	01	20
130	14	4	30	01	02	01	18
140	09	2	25	00	01	00	16

Network Statistics (PAL 3)

Average Time of Completion	79	45	120	53	53	58	71

Event Statistics (S_3)

Average Queuing Time	33	16	53	19	19	18	29
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-17

Ready to Run Activity Queue Statistics: see page B-18

0.06/PAL 3/Good Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
20	47	46	51	46	46	42	45
40	00	00	00	00	00	00	00

Network Statistics (PAL 3)

Average Time of Completion	20	20	22	21	21	19	20

Event Statistics (S_3)

Average Queuing Time	3	4	3	3	3	3	3
% of Events Rejected	16	15	19	16	16	12	12

Processor Statistics (idle time in %): see page B-19

Ready to Run Activity Queue Statistics: see page B-19

0.06/PAL 3/Bad Allocation/No Rejection

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
20	97	97	99	98	98	98	97
40	90	93	97	97	92	93	93
60	85	89	95	95	87	91	90
70	82	88	92	95	84	90	89
85	74	83	90	90	79	87	87
100	65	081	85	88	70	83	83
110	58	080	85	87	63	81	81
120	53	077	79	87	61	79	79
130	47	074	78	85	50	77	77
140	42	070	77	83	46	75	75

Network Statistics (PAL 3)

Average Time of Completion	144	192	348	201	149	204	225

Event Statistics (S_3)

Average Queuing Time	64	39	166	93	67	95	105
% of Events Rejected	0	0	0	0	0	0	0

Processor Statistics (idle time in %): see page B-20

Ready to Run Activity Queue Statistics: see page B-21

0.06/PAL 3/Bad Allocation/Rejection=20

% of Processes Tardy

Due Dates	FCFS	SAF	LAF	LELF	EDD	LETC	METC
0	100	100	100	100	100	100	100
20	63	64	81	68	65	65	65
40	03	01	17	02	03	00	00
60	00	00	00	00	00	00	00

Network Statistics (PAL 3)

Average Time of Completion	26	25	30	28	26	28	24

Event Statistics (S_3)

Average Queuing Time	3	4	3	3	3	3	4
% of Events Rejected	31	26	42	30	25	26	25

Processor Statistics (idle time in %): see page B-22

Ready to Run Activity Queue Statistics: see page B-22