

722

**MODIFIED PROCESSOR SCHEDULING ALGORITHM
WITH A SIMULATION EXPERIMENT**

651

**Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the Degree of
MASTER OF TECHNOLOGY**

SANJAY NAGPAL



**SCHOOL OF COMPUTER AND SYSTEM SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110 067
JUNE 1989**



जवाहरलाल नेहरू विश्वविद्यालय
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110067

CERTIFICATE

It is certified that the contents of this dissertation which carries the title MODIFIED PROCESSOR SCHEDULING ALGORITHM WITH SIMULATION EXPERIMENT has been submitted by Sanjay Nagpal, has not been previously submitted for any other degree of this or any other University.

Prof. Karmeshu
(Dean)
SCSS/JNU

SANJAY NAGPAL
(Student)

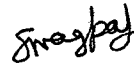
Dr. P.C. Saxena
(Supervisor)
SCSS/JNU

A C K N O W L E D G E M E N T

I take this opportunity to express my sincere thanks to Dr. P.C. Saxena, Associate Professor, School of Computer & Systems Sciences, Jawaharlal Nehru University for his help and valuable guidance during the completion of my dissertation and my M.Tech. studies.

Thanks are also to other teachers and members of staff of the school for providing me help.

Finally I am thankful to the Library Staff of Jawaharlal Nehru University and Indian Institute of Technology, New Delhi, for their cooperation.



SANJAY NAGPAL

C O N T E N T S

CHAPTER	Page No.
	Abstract 1
I	Operating System Concepts 2
1.1	Evolution of Operating Systems 2
1.2	Operating System Functions 3
1.2.1	Memory Management 4
1.2.2	Device Management 5
1.2.3	Information Management 6
1.2.4	Processor Management 6
II	Scheduling Algorithms 9
2.1	Introduction 9
2.2	Scheduling Levels 9
2.3	Scheduling Objectives 10
2.4	Non-Preemptive and Preemptive Scheduling 11
2.5	Performance Evaluation 12
2.5.1	Why Performance Evaluation ? 12
2.5.2	Performance Measures 13
2.5.3	Performance Evaluation Techniques 14
2.6	Common Scheduling Algorithms 16
2.7	A new Modified Multi-level Feed Back Scheduling Algorithm 19
III	Analytic Modelling 22
3.1	Introduction 22
3.2	Queueing Model for New Algorithm 22
3.3	Analysis of waiting Time Behaviour 24
IV	Simulation Experiment 43
4.1	Development of simulation program 43
4.2	Flow Charts 52
4.3	Program Listings 71
4.4	Results 87
V	Inference & Conclusion 95
	Bibliography 98

A B S T R A C T

This dissertation presents a modified multi-level feed back scheduling algorithm in uni-processor time sharing environment. Jobs are categorized based on their run time requirements. Depending upon the knowledge to which category a particular job belongs, it has been shown through a simulation experiment that the algorithm in question gives a better system throughout and in general has reduced average waiting time for the job over available feed-back algorithm in the literature. A queueing model for the new algorithm is also given and analytical expressions for the average waiting time are obtained. Simulation results are validated by analytical results.

Chapter - I

OPERATING SYSTEM CONCEPTS

1.1 Evolution of Operating Systems

The fact that operating systems have become an indispensable part of a computer system hardly needs to be emphasized. In fact, the development of operating systems is a major investment on the part of computer manufacturers.

Historically, operating systems have evolved through a series of 'generations'. The term generation was originally intended to suggest differences in hardware technologies, but it has come to be applied to the entire hardware-software-firmware system rather than the hardware alone. Zeroth generation computer systems of 1940's had no operating systems. Programs had to be written in the internal binary code and any error occurring during its run would have to be decoded and corrected by the user himself holding out all other activities of the computer. This mode of computer operation was known as 'OPEN JOB ACTIVITY'. Then came the first generation (1950's) systems, which had batch processing capabilities. The jobs were gathered in groups or batches. Once a job was running, it had total control of the machine. As each job terminated (either normally or abnormally), control was returned to the operating system that 'cleaned up after the job' and initiated the next job. The second generation (early 1960's) of operating

systems was characterized by the development of shared systems with multi-programming and multi-processing facilities. Device independence, time sharing and real time processing also began to appear. Third generation systems (1960-70) were primarily general purpose systems featuring multi-node operation. Such systems involved the interposition of a software layer between the hardware and the user. The fourth generation is the generation of computer net-working, personal computers, virtual machine operating systems, database systems and distributed data processing systems.

1.2 Operating System Functions

An operating system (also known as the control, the monitor, the supervisor etc) is a collection of those program modules implemented in either software or firmware within a computer system which govern the control of equipment resources such as processors, main storage, secondary storage, I/O devices and files. These modules resolve conflicts, attempt to optimize performance and simplify the effective use of the system. In other words, it acts as an interface between the user's program and physical computer hardware. Operating systems can also be viewed as a resource manager managing four separate resources - memory, processors, devices and information. Hence in order to analyze existing operating systems, to design new operating

systems and to study theoretical work, one can group all the functions of the operating system in four categories :

- a) Memory Management
- b) Device Management
- c) Information Management
- d) Processor Management

The definition of operating systems as given above implies that like any other manager it must do the following :

- a) Keep track of the resources
- b) Enforce policies to determine who gets what, when and how much
- c) Allocate the resources
- d) Reclaim the resources.

1.2.1 *Memory Management*

The memory or storage management deals with the problem of sharing an internal store of limited capacity among concurrent computations with and without the use of a larger, slower backing store. The functions of it are to keep information about what parts of memory are in use and by whom. It also keeps a record of free memory locations. In case of multi-programming it decides 'which process to get memory, when and how much'. Memory management also does the job of allocating and reclaiming the resource as and when the process requests or

releases it. There are a number of techniques which are used to achieve this goal of greater utilization of memory and more flexibility for the user; the costs of greater complexity, sophisticated hardware and increased overhead acting as constraints. Some of the most popular among these techniques are :

- a) Single contiguous memory management
- b) Partitioned memory management
- c) Relocatable partitioned memory management
- d) Paged memory management
- e) Demand-Paged memory management
- f) Segmented memory management
- g) Segmented and Demand paged memory management

1.2.2 *Device Management*

The device management modules are concerned with the assignment of device, I/O units, channels, control units to the jobs and the efficient operation of these devices. Once the job-scheduler selects a job, it may request any device according to the requirements of the job. The module which keeps track of all device resources is typically called I/O traffic controller. Scheduling of shared devices is done by I/O scheduler. Initiation and termination operation of I/O devices is also a part of device management modules.

1.2.3 *Information Management*

Information management is concerned with the storage and retrieval of information entrusted to the system in much the same manner as a library. It keeps a file directory sometimes called the Volume table of contents (VTOC). These tables contain the name, location and accession rights of all information within the system. It also selects the policy for determining where and how information is to be stored and who gets access to the information. Factors influencing this policy are efficient utilization of secondary storage, efficient access, flexibility to user and protection of access rights to the information requested. The modules of information management are also called 'File System'. This particular module is intended to free the programmer from problems of allocation of space for his information, physical storage formats and I/O accession and to allow him to concern himself only with the logical structure and operations performed in processing his information.

1.2.4 *Processor Management*

Processor management modules explain how concurrent processes and synchronizing primitives can be implemented on a system with one or more processors and a single internal store. It also evaluates the influence of these abstractions on the

real time characteristics of the system. In other words, it is concerned with the management of physical processors; specifically, the assignment of processors to processes. There are three major modules of processor management : the Traffic controller, the Job scheduler and the Processor scheduler. Before going into further details, let us define the term 'Process' formally. Unfortunately the term process which was first used by the designers of multics system (1960) doesn't have a unique definition. Some common interpretations are :

- a] A program in execution
- b] An asynchronous activity
- c] The 'animated spirit' of a procedure
- d] The 'Locus of control' of a procedure in action
- e] The entity to which processes are assigned.

Many other definitions can be found in literature but the 'program in execution' is the most frequently used. Generally a process goes through three different discrete states : the running state (if it currently has the CPU), the ready state (if it could use the CPU if one were available) or the blocked state (when it is waiting for some event to happen e.g. I/O completion before it can proceed).

The traffic controller keeps track of the status of the process. There are also modules who do the synchronization

between processes and jobs. On the process levels there are mechanisms to prevent race conditions which occur when the result of a computation varies, depending upon the timing of other processes. It also tries to solve the problems of deadlocks arising out of situations when there are two or more processes each of which is waiting for resources that the other has and will not give up. The job scheduler creates the processes and in a non multiprogramming environment decides which process is to receive a processor. It also maintains a job control block, which keeps information about the job's status and its position in a job queue.

The process scheduler in a multi-programming environment decides about which ready process should get the processor, at what time and for how long. Enforcement of this policy of assigning ready process to processor(s) in order to reduce the average waiting time for the job and in turn to increase the system throughput is done with the help of process scheduling algorithms.

Even though problems of asynchronous concurrent processes and deadlock etc. are important, our studies here will deal with processor scheduling algorithms which use some prior information about jobs.

Chapter - II

SCHEDULING ALGORITHMS

2.1 Introduction

The sharing of a computer installing by a group of users in an economic necessity. It leads to situations in which resources become scarce, there are not enough physical processors and storage for simultaneous execution of all processes requested by users. The available resources can be shared among the processors either by executing them one at time till completion or by executing several of them in rapid succession of short periods of time. In both cases each processor must pause every now and then and decide whether to continue the execution of its present process or switch to some other process instead. The rule according to which this decision is made is called a 'scheduling algorithm'.

2.2 Scheduling Levels

To make the scheduling problem manageable, it is usually considered at several levels of abstraction. The view of scheduling presented here recognized three main levels:

- 1] High level scheduling
- 2] Medium level (intermediate level) scheduling
- 3] Low level scheduling

High level scheduling is an 'admission scheduling' in the sense that it determines which jobs gain admission to the system. Once admitted, jobs becomes processes or groups of processes. The intermediate level scheduling determines which processes shall be allowed to compete for the CPU. This particular scheduler responds to short term fluctuations in the system load by temporarily suspending and activating processes to achieve smooth system operation and thus helps in realizing certain system wide performance goals. The low level scheduling is performed by the 'dispatcher' which determines policies to allocate ready process to CPU and actually assigns them.

2.3 Scheduling Objectives

The objectives that we should have before us in deciding a particular scheduling policy are :

- [a] A scheduling discipline should be fair in the sense that all processes are treated in the same fashion and no process suffers from indefinite postponement.
- [b] A scheduling discipline should maximize the throughput by serving the maximum number of processes per unit time.
- [c] It should be such that a given job runs in about the same time and at about the same cost regardless of the load on the system.

- [d] It should try to minimize the overhead.
- [e] The scheduling mechanisms should keep the resources of the system busy. Processes that will use under utilized resources should be favoured.
- [f] It should also try to achieve a balance between responses and utilizations.

Apart from these, it should also try to enforce priorities, give preferences to processes holding key resources, provide better service to processes exhibiting desirable behaviour. In addition the mechanism should not collapse under the weight of a heavy system load.

One can immediately see that many of these objectives are in conflict with one another thus making scheduling a complex problem.

2.4 Nonpreemptive And Preemptive Scheduling

If the scheduling algorithm is such that once a process has been allocated the processor, it can not be taken away from the process unless and until it voluntarily leaves it or asks for some I/O or it finishes the quantum allocated; then the discipline is called a 'nonpreemptive scheduling' discipline; otherwise a scheduling discipline is said to be 'preemptive' Preemptive scheduling is useful in system in which

high-priority processes require rapid attention. In real time systems and interactive time sharing systems preemptive scheduling is important in guaranteeing good response times. In nonpreemptive systems short jobs are made to wait by longer jobs, but the treatment of all processes is fairer. Response times are more predictable because incoming higher-priority jobs can not displace waiting jobs.

2.5 Performance Evaluation

2.5.1 *Why Performance Evaluation?*

H. LUCAS [8] mentions three common purposes for performance evaluation :

- a) Selection evaluation : Here the performance evaluator decides whether or not selecting a particular system is appropriate for his work.
- b) Performance projection : The goal of the evaluator here is to estimate the performance of a system that does not exist. It may be a complete new software component.
- c) Performance monitoring : The evaluator collects performance data on an existing system or software component to be sure that system is meeting its performance goals. This helps him in estimating the impact of planned changes and enables him to make

strategic decisions such as whether or not to modify an existing job priority system.

2.5.2 *Performance Measures*

Performance means the manner in which or the efficiency with which a computer system meets its goals. Thus performance is relative rather than an absolute quantity, although one can talk of absolute performance measures such as the number of jobs per hour a given computer system can service. But whenever a performance measure is taken, it is normally to be used as a basis of comparison.

In simulation and modelling studies of systems, some performance measures often employed are :

- a) Variance in Response Times : A small variance means that the various response times experienced by users are relatively close to mean. A large variance is undesirable.
- b) Throughput - This is the work/time unit performance measurement.
- c) Work Load - This is the measure of the amount of work that has been submitted to the system, and which the system must process in order to be functioning acceptably.

- d) Capacity - This is a measure of maximum throughput a system may have assuming that whenever the system is ready to accept more jobs another job is immediately available.
- e) Utilization - This is the fraction of a time the resource is in use. But this is misleading since if there are a number of processes all of which are in infinite loops, a higher utilization is obtained.

2.5.3 Performance Evaluation Techniques

The importance performance evaluation techniques are listed below in tabular form. The table shows the techniques, and their applicability for various purposes of software performance evaluation :

TABLE - 2.5.1

Evaluation Techniques	Purpose of Evaluation		
	Selection Evaluation (system exists elsewhere)	Performance Projection (system does not exist)	Performance Monitoring (system in Operation)
Analytical Models	1	1	-
Bench Marks	3	2	2
Synthetic Programs	3	2	2
Simulation	3	3	3

Technique not applicable

- a: Has been used but is inadequate
- b: Provides some assistance but is insufficient, should be used with other techniques
- c: Satisfactory.

(Taken from 'Performance Evaluation and Monitoring'
H. LUCAS ACM Computing Survey, Sept. 1971).

Analytic models are mathematical representation of computer systems. The models of queueing theory and Markov Processes are most useful. A bench-mark is a real program that the evaluator actually submits for execution on the computer system being evaluated. The evaluator knows the performance characteristics of bench-mark on existing equipment, so running on new equipment helps him to estimate the performance of the system quickly and relatively accurately. Synthetic programs are real programs that have been custom-designed to exercise specific features of a computer system. Simulation is a technique in which the evaluator develops a computerized model of the system being evaluated. The model is then run on a computer system over some simulated period of time which reflects the behaviour of the system quickly and accurately.

2.6 Common Scheduling Algorithms

The simplest scheduling discipline is first-in-first out (FIFO). As the name itself suggests, in this, processes are dispatched according to their arrival time on the ready queue. FIFO is a non-preemptive discipline. It is fair in the formal sense but somewhat unfair in that long jobs make short jobs wait and unimportant jobs make important jobs wait. Today's operating systems do not have FIFO as the master scheme but it is often embedded within other schemes. The next relatively simple scheduling discipline is 'Round Robin', in which processes are dispatched FIFO but are given a limited amount of CPU time called a time-slice or a quantum. If a process does not complete before its CPU time expires, the CPU is preempted and given to the next waiting process. The preempted process is then placed at the back of the ready list. It is better than FIFO in time sharing environments in which the system needs to guarantee reasonable response time for interactive users. The choice of the optimal quantum in RR discipline varies from system to system and it varies under different loads.

Shortest job first (SJF) is another non-preemptive scheduling discipline in which the waiting job (or process) with the smallest run time-to-completion is run next. SJF reduces average waiting time over FIFO. SJF favours the short

job at the expense of larger ones. The obvious problem with SJF is that it requires precise knowledge of how long a job or process will run and this information is rarely available. The best SJF can do is to rely as user estimates. 'Shortest remaining time' (SRT) is the preemptive counterpart of SJF and is useful in time-sharing. In SRT the process with the smallest estimated run time-to-completion is run next, including new arrivals. In SRT a running process may be preempted by a new process with a shorter estimated run time. Brinch Hansen ('Short term scheduling in Multi-programming System; 1971 ACM) developed the highest response ratio next (HRN) strategy which is non-preemptive but has dynamic priorities which are calculated according to the formula:

$$\text{Priority} = \frac{\text{time waiting} + \text{service time}}{\text{service time}}$$

and jobs are served according to priority.

Next we focus our attention to the multi level feed back queues. These favour short jobs and I/O device utilization, determines the nature of a job as quickly as possible and schedules the job accordingly. In this structure a new process enters the queueing network at the back of the top queue. It moves through that queue FIFO until it gets the CPU. If the job gets over or relinquishes the CPU to wait for completion of some I/O or other event, the job leaves the network. If the quantum expires before the process voluntarily relinquishes the CPU,

the process is placed at the back of the next lower queue (See Fig. 2.1). A multi-level feed back queueing network is an example of an adaptive mechanism that responds to the changing behaviour of the system it controls. This mechanism achieves good device utilization and responsiveness to interactive users favouring I/O bound processes. The CPU bound processes are also given fair treatment by this discipline.

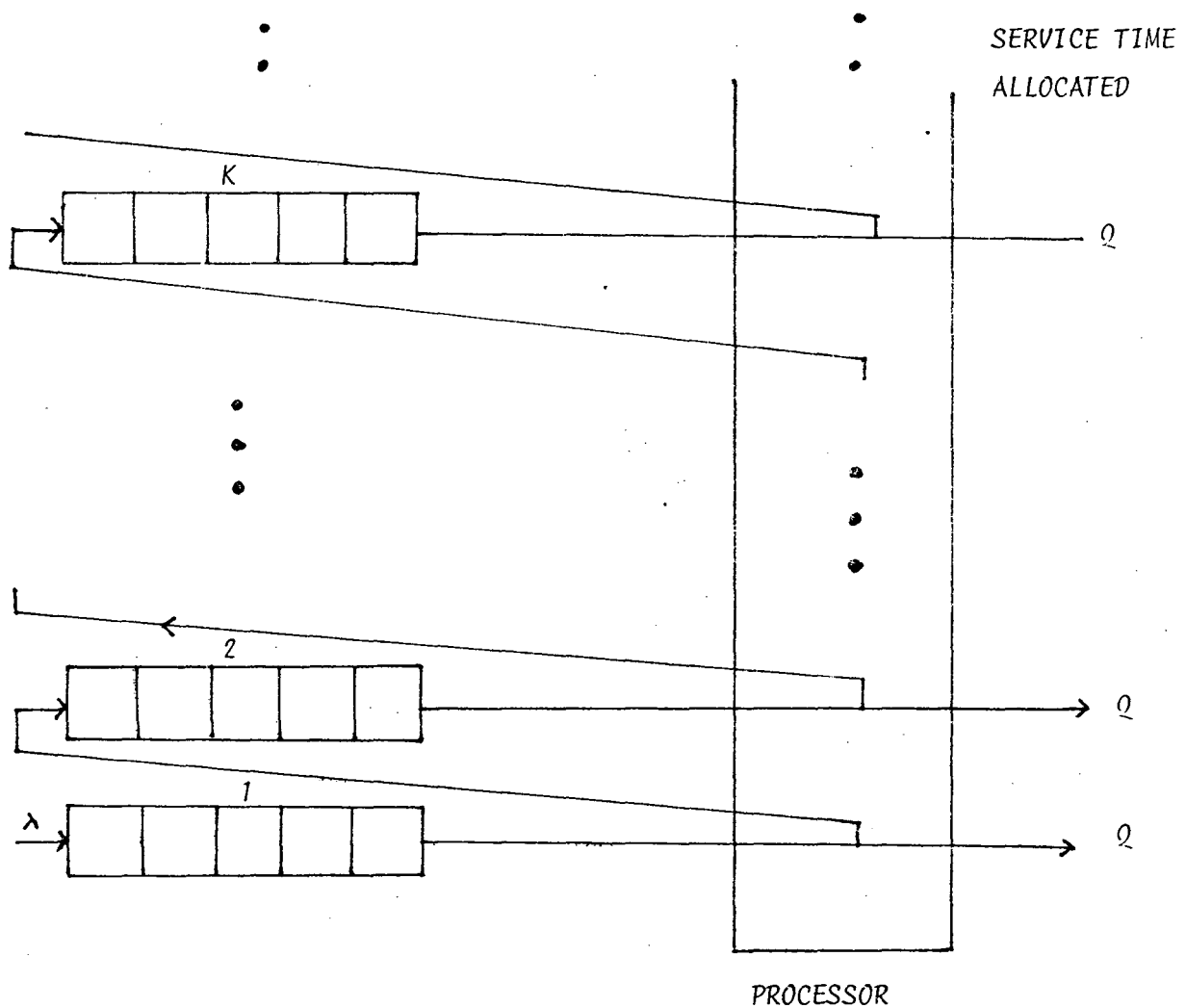


Fig. 2.1

2.7 A Modified Multi-Level Feed Back Algorithm

The problem with the multi-level feed back queue is that being a 'no prior information' discipline, if some prior informations are available it does not have any flexibility as such to make the changes so as to get the improved system throughput and reduce the weighted average waiting time for the jobs (or processes).

In this dissertation we shall study a modified feed back system which makes effective use of certain prior informations. For this purpose we assume that jobs can be classified into a number of categories based on their run time estimates. For simplicity sake, let us have only two categories 'short jobs' and 'long job's'. Let us denote by t_0 the cut off time point in the sense that any job needing more time than this is considered a long job. Now we define the following modified feed back discipline which accommodates this information (See Fig.2.2).

There are two phases in the queueing network. If an arriving job is short ie enters the Phase I whereas if it is a long job it enters Phase II. In Phase I, the system operates exactly like a multilevel feed back queue i.e. a job at the head of a queue gets the quantum when there is no job waiting in the higher level queues. After completion of the quantum if it needs more it joins the next lower level queue. Within a queue FIFO

rule is adopted. The Phase II is executed only when the Phase I is empty. In Phase II also multi-level queue discipline is adopted but with the difference that when a process gets the processor for the first time in the Phase II, it gets the amount of time equal to what it would have got, until this queue, had it been the usual feed back discipline.

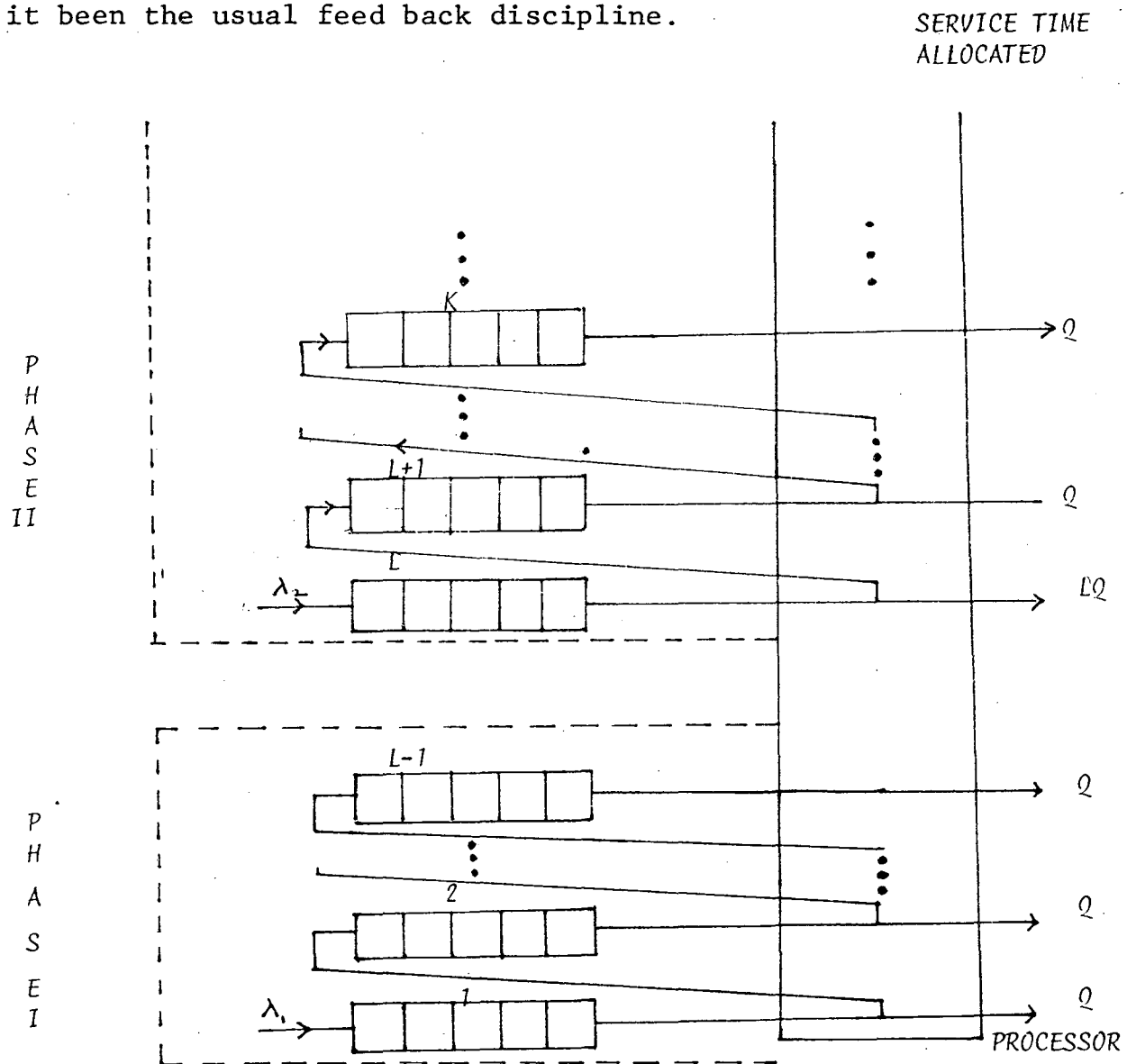


Fig. 2.2

In the following chapters, we will analyse and compare modified feed back queueing algorithm with usual feed back algorithm from the point of view of their average waiting time behaviour and system throughput. For this purpose, we will carry out a simulation experiment which will help us in corroborating certain inferences about the behaviour of these algorithms. An analytical model is also presented which will help us in validating simulation results.



7H-2948

Chapter - III

ANALYTIC MODELLING

3.1 Introduction

Analytic models are mathematical representation of systems that allow the performance evaluator to draw quick and accurate conclusions about a system's behaviour. As a mathematical approach probability models are generally more realistic than deterministic models, because they can represent the irregular and unpredictable demands made by the computer users. When these probability models are formulated to study the properties of dynamic scheduling techniques they take the form of queueing systems.

3.2 Queueing Model for New Algorithm

In this chapter we shall concentrate our attention on the modified multi-level feed back scheduling discipline which we defined in the last chapter. Our specific objective will be to study analytically the waiting time behaviour of the new algorithm in uni-processor time sharing environment. This will help us in providing insight into the properties of the algorithm even though idealizations or simplifying assumptions have to be made in order to keep the models mathematically traceable. This will eventually help us in validating our simulation studies.

A general mathematical statement of the queue discipline is as follows :

There is one processor, Jobs (or processes) arrive in a poisson stream and are assumed to have service time taken from a discrete but otherwise general probability distributions. For the sake of simplicity let us assume there are only two types of job arrivals classified as short job or long job based on their run time requirements. A job requiring less than LQ amount of execution time where Q is the Quantum size and L is any positive finite number, is treated as a short job. All other jobs are long jobs. If the job is a short job it joins the Phase I of the system. Otherwise it joins the Phase II of the system. Phase II is executed only when there is no job left in Phase I of the system. In Phase I, after the processor has completed the quantum allocated to a given job the next job to be executed is the one having received the fewest quanta of all those jobs currently waiting. If there is a tie among several jobs having received the least service, then the job selected is the one with the earliest arrival time. In the Phase II the same rule as in Phase I is adopted but with the following difference. When a job gets the processor for the 1st time in Phase II, it receives the latter for the quantum size which is equal to what it would have got had it been a usual feed back discipline i.e. for LQ amount of time.

For the sake of simplicity we will assume processing time of the jobs to be exact multiple of quantum size and the discipline to be non-preemptive.

3.3 Analysis of Average Waiting Time Behaviour

We will try to get the expressions for average waiting time for a job (referred to as tagged job) that needs KQ amount of execution time.

Let λ be the poisson arrival rate for the jobs, and g_i represent the unconditional probability that a job requires iQ amount of execution time.

Let

$$G(K) = \sum_{i=1}^k g_i, \quad k = 1, 2, \dots$$

Also, let W_k and V_k represent the mean waiting time for a job of size KQ in the system and in the queue respectively.

Obviously,

$$W_k = V_k + KQ \quad (3.0)$$

We will consider three exclusive and exhaustive cases.

Case I $K < L$

In this case we obtain the average waiting time for the short job which joins the system in Phase I. The probability

that a job requires iQ amount of time given that its a short job is $\frac{g_i}{G(L-1)}$.

Now, let us write mean waiting time in the queue V_k as

$$V_k = V'_k + V''_k \quad (3.1)$$

where V'_k and V''_k are defined as follows :

V'_k = mean time to finish the quantum in progress + mean time to service upto k quanta, all short jobs at the 1st k levels at the time of a-arrival.

V''_k = every short job that arrives while the tagged job (i.e., the job under consideration) is waiting in the queue must be allowed to ascend to the k th queue level if it requires in excess of $(k-1)Q$ units of execution time. The total execution time required by these new arrivals has a mean value

$$V''_k.$$

Let $E_k(s)$ denote the mean amount of execution time used by a job to which kQ time units are allowed given that the job is a short job. Then

$$E_k(s) = \sum_{i=1}^k (iQ) \frac{g_i}{G(L-1)} + KQ \left[\sum_{i=k+1}^{L-1} \frac{g_i}{G(L-1)} \right]$$

for $k = 1, 2, \dots, L-1$

$$= \frac{1}{G(L-1)} \left[\sum_{i=1}^k i g_i + k[G(L-1) - G(k)] \right] \cdot Q$$

for $k = 1, 2, \dots, L-1$ (3.2)

Similarly, the second moment which we will use later is

$$E_k(s^2) = \frac{1}{G(L-1)} \left[\sum_{i=1}^k i^2 g_i + K^2[G(L-1) - G(k)] \right] \cdot Q^2$$

for $k = 1, 2, \dots, L-1$ (3.3)

The rate of arrival to the 1st phase is λ_1 say, where

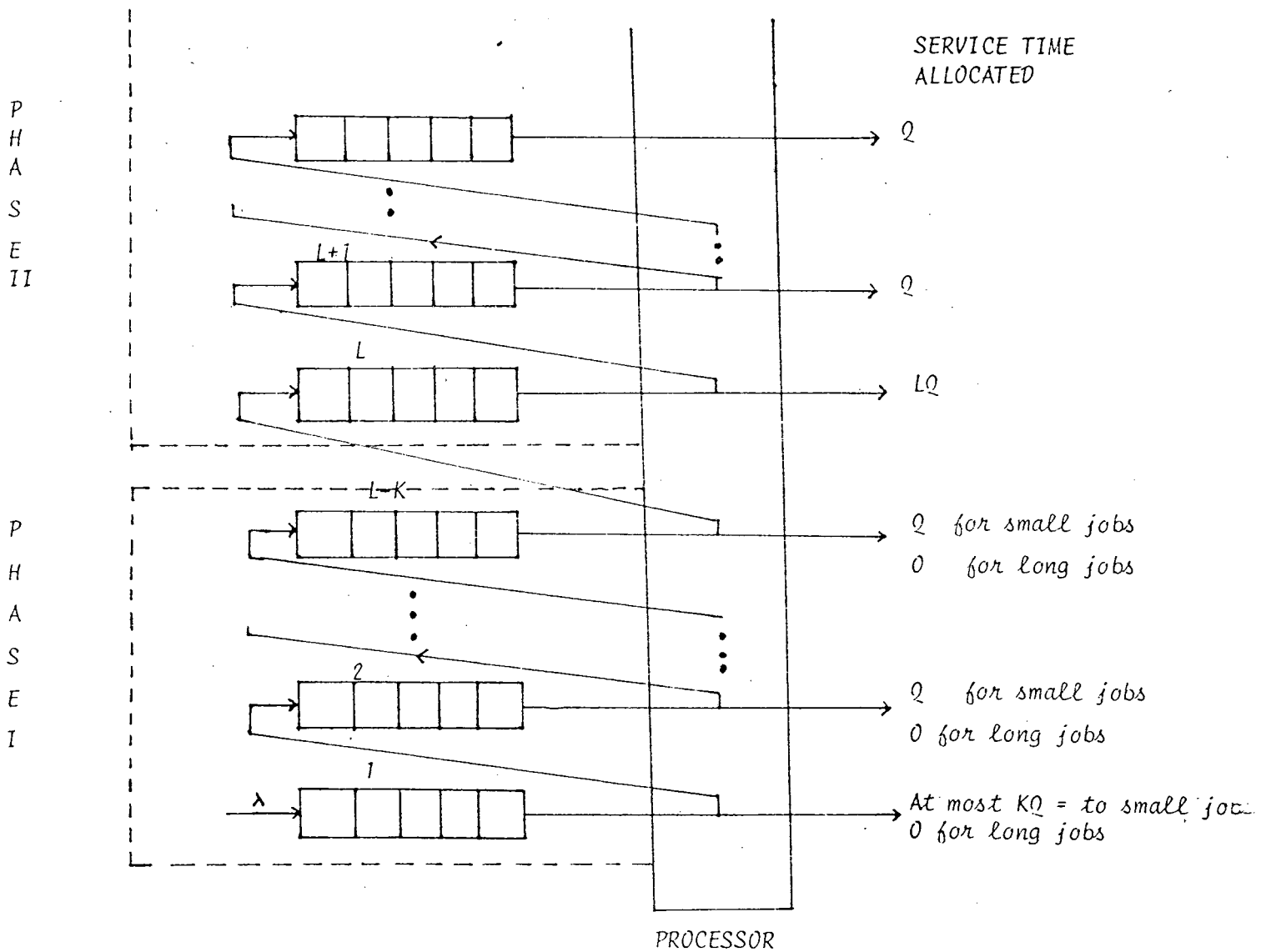
$$\lambda_1 = \lambda \sum_{i=1}^{L-1} g_i = \lambda G(L-1)$$

Hence,

$$V_k'' = \lambda_1 [V_k + (k-1)Q] \cdot E_{k-1}(s) \quad (3.4)$$

This is because, λ_1 being the arrival rate and $V_k + (k-1)Q$ being the time at which the tagged job gets the processor for the last time, the number of arrivals in this time are $\lambda_1 [V_k + (k-1)Q]$. Now, all these arrivals gets the processor for at most $(k-1)$ times and mean execution time used by them is $E_{k-1}(s)$, hence the expression for V_k'' as in (3.4).

To compute V_k' we simplify the matter as follows. Consider a system say S_1 , where in Phase I jobs served at the 1st queue level are allocated at most KQ units of execution time and those served at the higher level are allocated one quantum of service. The Phase II being same as that of modified feed back system. The system S_1 's figurative discription is as given in Fig. 3.1.



One can immediately see that from the point of view of a short job requiring k quanta the mean time spent waiting for these jobs in system S1 is precisely the same as that in our modified feed back system.

Thus, by examining the system S1 we have

$$V'_k = E'(S_r) + N'_1 E_k(S) \quad (3.5)$$

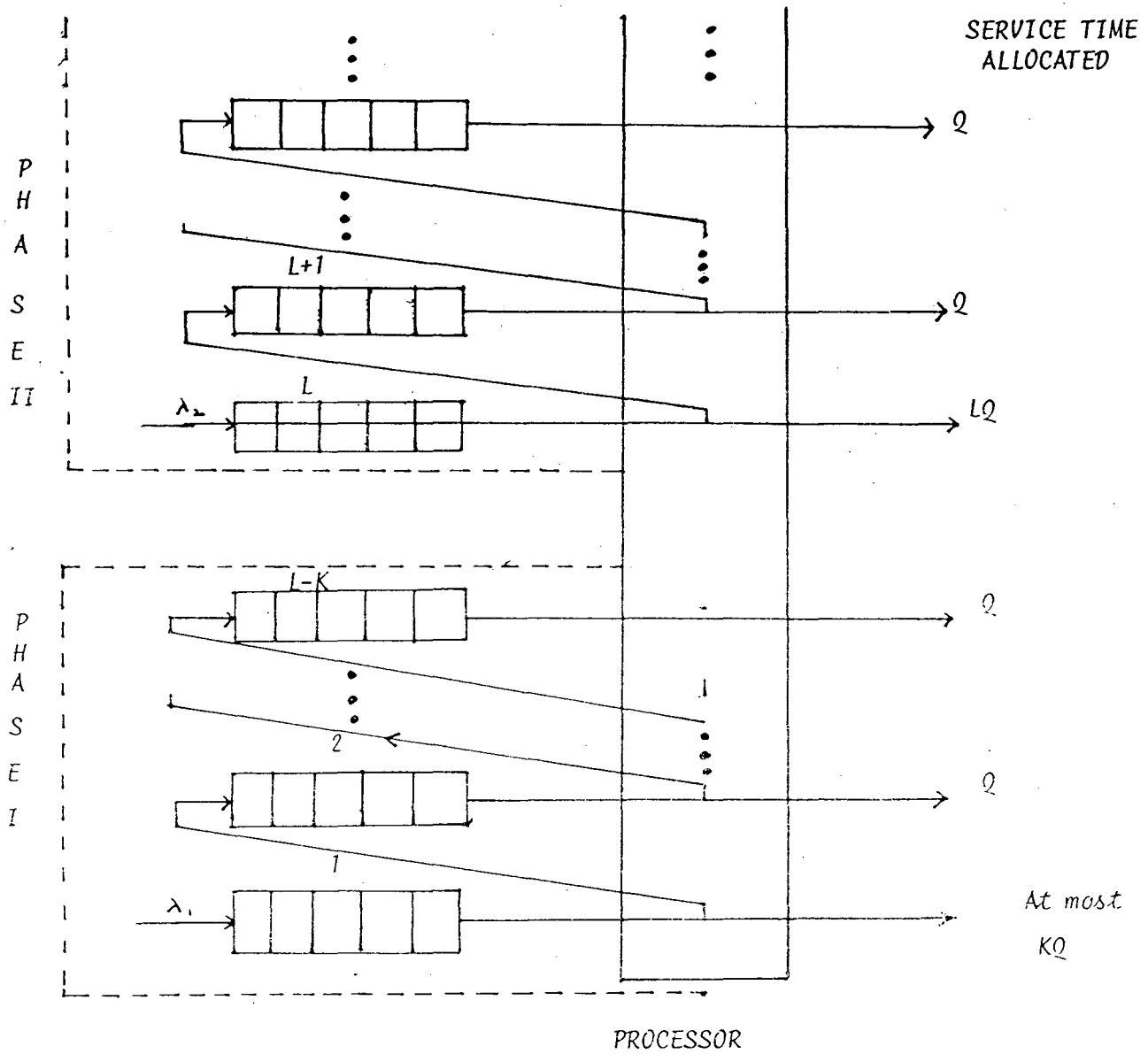
where $E'(S_r)$ is the mean time to complete the quantum in progress and N'_1 is the mean number encountered by the tagged job in the first queue level at the time of a-rival in the system S1.

Using the Little's result which says that number of customers in the queue is equal to the product of their arrival rate and the mean time a customer spends in the queue, we have

$$N'_1 = \lambda_1 \cdot V'_k$$

Hence from (3.5) we have,

$$\begin{aligned} V'_k &= E'(S_r) + \lambda_1 V'_k E_k(s) \\ \implies V'_k &= \frac{E'(S_r)}{1 - \lambda_1 E_k(S)} \end{aligned} \quad (3.6)$$



Fif- 3.2

To calculate $E'(S_r)$, let us consider again a new system S_2 which is equivalent to system S_1 (Hence equivalent to modified feed back system) from the point of view of a short job requiring K quanta of execution time. The figurative discription of which is given above, :

In system S2 all arriving jobs join 1st queue of Phase I. The quantum allocated i.e. the service pattern is as given in the figure. S_2 is equivalent to S_1 for the simple reason that even though long jobs join service queue in Phase I, they do not get any service in Phase I. When any long job is at the head of the queue in Phase I, it immediately joins next lower level queue without any service.

In system S_2 , we have precisely four classes of jobs any one of which may be with the processor when the tagged arrival takes place.

- a) Those receiving an allocation of at most KQ units of execution time having just waited through the 1st queue in Phase I. Their arrival rate is λ and the second moment of the amount of execution time used by them is $E'_k(S^2)$ where

$$E'_k(S^2) = \sum_{i=1}^k (iQ)^2 \cdot g_i + (KQ)^2 \cdot \sum_{i=k+1}^{L-1} g_i + 0 \cdot \sum_{i=L}^{\infty} g_i$$

$$= \left[\sum_{i=1}^k i^2 g_i + K^2 [G(L-1) - G(K)] \right] \cdot Q^2$$

which from (3.3) gives

$$E'_k(S^2) = G(L-1) \cdot E_k(S^2) \quad (3.7)$$

- b) Those receiving an allocation of one quantum upto $(L-1-K)$ th queue in Phase I. The arrival rate for such

ith queue is

$$\lambda_i'' = \sum_{j=k+i-1}^{\infty} g_j = \lambda [1-G(K+i-2)]$$

for $i = 2, 3, \dots, L-K$

The 2nd moment of amount of execution time used by them in the ith queue is, $E_{k,i}''(S^2)$, where

$$\begin{aligned} E_{k,i}''(S^2) &= \frac{1}{[1-G(K+i-2)]} [Q^2 \sum_{j=k+i-1}^{L-1} g_j + 0 \cdot \sum_{j=L}^{\infty} g_j] \\ &= \frac{Q^2 \sum_{j=k+i-1}^{L-1} g_j}{[1-G(k+i-2)]} \end{aligned} \quad (3.8)$$

for $i = 2, 3, \dots, L-K$

- c) Those receiving an allocation of L quanta of executive time in the 1st queue level of Phase II of the system. The arrival rate for such jobs is

$$\lambda_2 = \lambda \sum_{i=L}^{\infty} g_i = \lambda [1-G(L-1)]$$

and the 2nd moment of amount of execution time used by such jobs is $(LQ)^2$.

- d) Those receiving an allocation of Q units of execution time in the 2nd phase of the system at subsequent queue levels. The arrival rate for such jobs is,

$$\lambda'_{L+i} = \lambda \cdot \sum_{j=L+i}^{\infty} g_j \quad \text{for } i = 1, 2, \dots, \infty$$

The second moment of amount of execution on time used by them is Q^2 .

The arrival process in the first queue level is poisson, but the arrivals to the higher level queues do not constitute poisson process. Note that the arrivals to the lower level queues occur only within processor busy periods but since the tagged job is assumed to be random (i.e. poisson) arrival, we can make use of the residual waiting times result [3] pp162 which says that in case the arriving jobs can be grouped into classes each having a distinct service distribution and let λ_p and $B_p(x)$ denote the arrival rate and service time distribution respectively for jobs of class p , then the expected mean time to finish the job in progress is given by

$$E'(S_r) = \frac{1}{2} \sum_{p \geq 1} \lambda_p E(S_p^2) \quad (3.9)$$

where $E(S_p^2)$ is the second moment of service time required by a job. They have also showed that result is also true even if higher level queues do not constitute strictly a poisson process. Hence, using (3.9) we have

$$\begin{aligned}
E'(S_r) &= \frac{\lambda}{2} \cdot G(L-1)E_k(S^2) + \sum_{i=2}^{L-K} \frac{\lambda}{2} [1-G(k+i-2)] \cdot \\
&\quad \frac{1}{[1-G(k+i-2)]} \cdot Q^2 \sum_{j=k+i-1}^{L-1} g_j + \\
&\quad \frac{\lambda_2}{2} \cdot (LQ)^2 + \frac{\lambda}{2} \cdot \sum_{j=1}^{\infty} \sum_{i=L+j}^{\infty} g_i \cdot Q^2 \\
E'(S_r) &= \frac{\lambda_1}{2} E_k(S^2) + \frac{\lambda}{2} \sum_{i=2}^{L-K} [G(L-1)-G(k+i-2)] \cdot Q^2 \\
&\quad + \frac{\lambda_2}{2} (LQ)^2 + \frac{\lambda}{2} \sum_{j=1}^{\infty} [1-G(L+j-1)] \cdot Q^2 \quad (3.10)
\end{aligned}$$

Then substituting (3.4) and (3.6) in (3.1) we have,

$$\begin{aligned}
V_k &= \frac{E'(S_r)}{1-\lambda_1 E_k(S)} + \lambda_1 [V_k + (k-1)Q] \cdot E_{k-1}(S) \\
V_k &= \frac{\frac{E'(S_r)}{1-\lambda_1 E_k(S)} + \lambda_1 (k-1)Q E_{k-1}(S)}{1-\lambda_1 E_{k-1}(S)}
\end{aligned}$$

Thus from (3.0) we

$$\begin{aligned}
\text{get -} \\
W_k &= \frac{\frac{E'(S_r)}{1-\lambda_1 E_k(S)} + \lambda_1 (k-1)Q E_{k-1}(S)}{1-\lambda_1 E_{k-1}(S)} + KQ \quad (3.11)
\end{aligned}$$

where, the expressions for $E'(S_r)$, $E_k(S)$ (and $E_{k-1}(S)$) are given by (3.10), (3.2) respectively.

Case II $K = L$

In this case tagged job is a long job of size L .

Arguments here are similar to that in case I. Waiting time in the system for such a job is given by

$$W_L = V_L + LQ \quad (3.12)$$

Let

$$V_L = V'_L + V''_L \quad (3.13)$$

where,

V'_L = mean time to complete the quantum in progress + mean time to complete all jobs in Phase I of the system at the time of arrival + mean time to complete jobs in the first queue level of Phase II of the system.

and

V''_L = mean time to service all new jobs arrivals in Phase I of the system during the waiting time in the queue for the tagged arrival.

To compute V'_L and V''_L , consider on equivalent new system (say S_3) in which first phase of our modified feed back system is replaced by a single queue in which every arrival is allocated a

maximum of $(L-1)$ quanta of execution time while the second phase is exactly the same as the second phase of the original system. It is immediately apparent that from the point of view of the tagged arrival the mean time spent waiting for these jobs is same in both the systems.

From (3.2) we have,

$$E_{L-1}(S) = \frac{\sum_{i=1}^{L-1} (iQ) \cdot g_i}{G(L-1)}$$

and

$$E_{L-1}(S^2) = \frac{1}{G(L-1)} \left[\sum_{i=1}^{L-1} i^2 g_i \right] \cdot Q^2 \quad (3.14)$$

where, $E_{L-1}(S)$ and $E_{L-1}(S^2)$ are, as before, the mean and second moment of amount of execution time used by the arrivals in Phase I.

Since λ_1 is the arrival rate for Phase I and V_L is the waiting time in the queue for the tagged arrival the number of arrivals in this period in Phase I is $\lambda_1 V_L$. All these arrivals use $E_{L-1}(S)$ amount of time on the average and hence V_L'' is given by

$$V_L'' = \lambda_1 V_L E_{L-1}(S) \quad (3.15)$$

on the other hand V_L' can be written as

$$V_L' = E'(S_r) + N_1' E_{L-1}(S) + N_2' .LQ \quad (3.16)$$

where $E'(S_r)$ is the mean time to complete the quantum in progress.

N_1' is the mean queue size in Phase I.

N_2' is the mean queue length in the first queue level of the Phase II encountered by the tagged arrival.

Note that, $N_1' E_{L-1}(S)$ is the mean time to complete all jobs in Phase I of the system and $N_2' .LQ$ is the average time to complete jobs in the first queue level of Phase II of the system. To find N_1' , N_2' and $E'(S_r)$ we proceed as follows :

Using Little's result we have

$$N_1' = \lambda_1 V_I$$

where, V_I is the mean waiting time in Phase I queue of the new system S_3 .

Note that

V_I = mean number waiting in the queue \times mean amount of execution time used by a job.

Since, Phase I queue system is nothing but a $M|G|1$. queueing system, by making use of the result of $M|G|1$ queues [3] pp 161 we have mean number waiting in the queue

$$= \frac{\rho^2 [1 + C_{L-1}^2(S)]}{2(1-\rho)}$$

where

$$\rho = \lambda E_{L-1}(S)$$

and

$$C_{L-1}^2(S) = \frac{E_{L-1}(S^2)}{E_{L-1}(S)}$$

ρ and $C_{L-1}^2(S)$ are called traffic intensity and coefficient of variation respectively.

Hence,

$$\begin{aligned} N_1' &= \lambda_1 V_1 \\ &= \lambda_1 \cdot \frac{\rho^2 [1 + C_{L-1}^2(S)]}{2(1-\rho)} \cdot E_{L-1}(S) \end{aligned} \quad (3.17)$$

Using Little's result on N_2' , we have

$$N_2' = \lambda_2 V_L' \quad (3.18)$$

That is because, λ_2 is the arrival rate for Phase II and V_L' is the average waiting time needed for all the jobs before the tagged arrival, in Phase I and Phase II get serviced.

To calculate $E'(S_r)$ consider as before a new system S_4 which is equivalent to S_3 from the point of view of the tagged arrival. In system S_4 as shown Fig. 3.3 all jobs join Phase I, but long jobs are not serviced in Phase I. In Phase II system is exactly same as that in S_3 .

We note that there are three types of jobs to be considered now :

- a) All the jobs with the arrival rate λ in Phase I of the new system S_4 . The second moment of the amount of execution time used by them is given by

$$E'_{L-1}(S) = \sum_{i=1}^{L-1} (iQ)^2, g_i + 0^2 \cdot \sum_{i=L}^{\infty} g_i$$

$$= G(L-1) \cdot E_{L-1}(S^2) \quad (3.19)$$

- b) The jobs with arrival rate λ_2 in the first queue level of the Phase II in the new system S_4 . The second moment of amount of execution time used by such job is $(LQ)^2$.

- c) The jobs with arrival rate $\lambda[1-G(L+i-1)]$ for the i th queue in the subsequent queue of Phase II of the new system for $i = 1, 2, \dots, \infty$

Thus by using the result (3.9) we have

$$E'(S_r) = \frac{\lambda}{2} G(L-1) E_{L-1}(S^2) + \frac{\lambda_2}{2} (LQ)^2 + \frac{\lambda}{2} \cdot \sum_{i=1}^{\infty} [1-G(L+i-1)] \cdot Q^2 \quad (3.20)$$

Substituting from (3.17), (3.18) in (3.16) we have

$$V'_L = E'(S_r) + \lambda_1 \frac{\rho^2 [1+C_{L-1}^2(S)]}{2(1-\rho)} [E_{L-1}(S)]^2 + \lambda_2 V'_L LQ$$

$$V'_L = \frac{[E'(S_r) + \frac{\lambda_1}{2(1-\rho)} \rho^2 \cdot [1+C_{L-1}^2(S)] E_{L-1}^2(S)]}{[1 - \lambda_2 LQ]} \quad (3.21)$$

Summing (3.21) and (3.15) we get

$$V_L = \frac{[E'(S_r) + \frac{\lambda_1}{2(1-\rho)} \rho^2 [1+C_{L-1}^2(S)] E_{L-1}^2(S)]}{[1 - \lambda_2 LQ]} + \lambda_1 V_L E_{L-1}(S) \quad (3.22)$$

and from (3.21) we finally get,

$$W_L = \frac{[E'(S_r) + \frac{\lambda_1}{2(1-\rho)} \rho^2 [1+C_{L-1}^2(S)] E_{L-1}^2(S)]}{[1-\lambda_2 LQ] [1-\lambda_1 E_{L-1}(S)]} + LQ \quad (3.23)$$

with the expressions for ρ , $C_{L-1}^2(S)$, $E_{L-1}(S)$ and $E'(S_r)$ as given above.

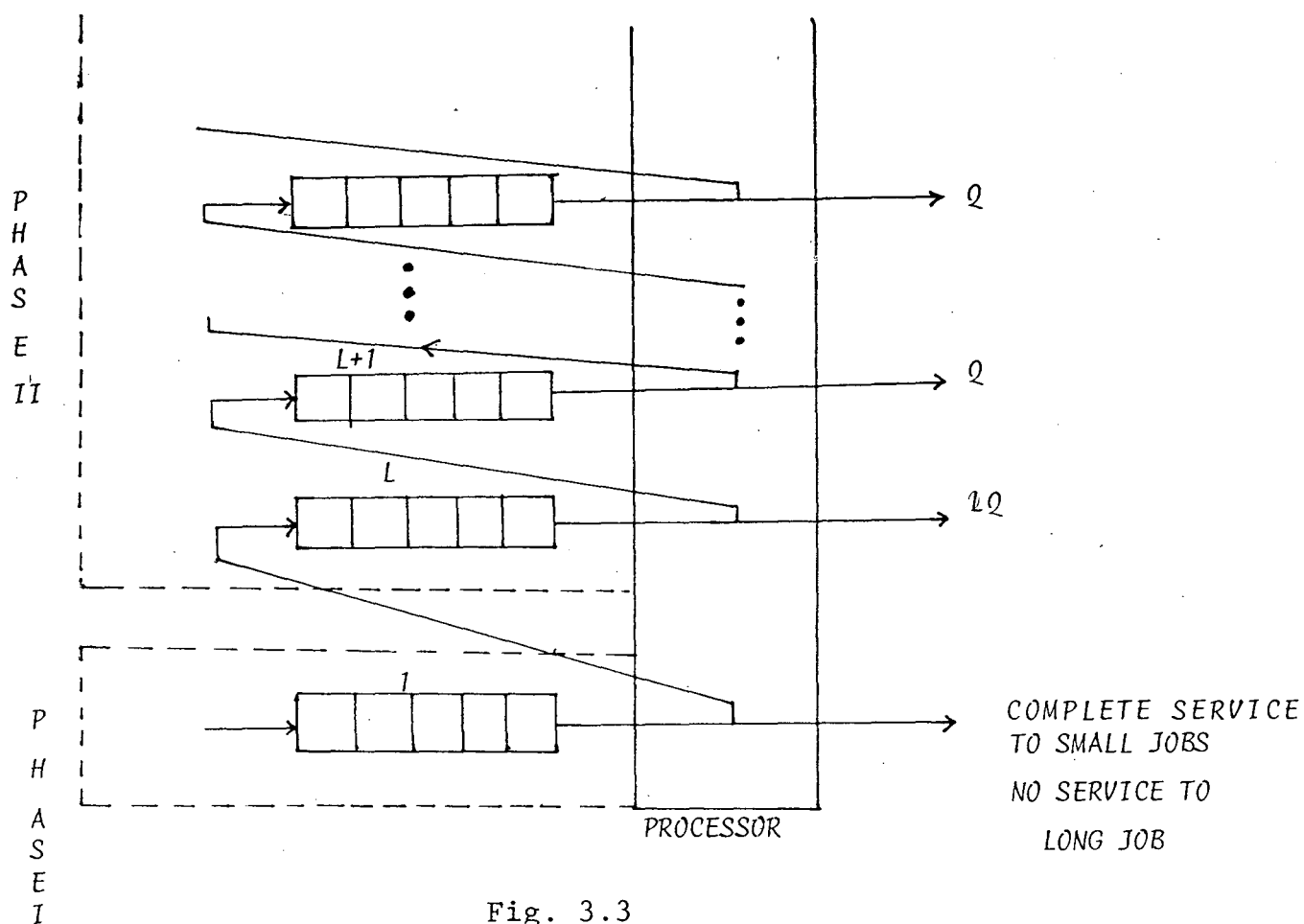


Fig. 3.3

Case III $K > L$

In this case the tagged job is a long job of size greater than L . A close look at both usual multi-level feed back discipline and modified feed back discipline, which we presented earlier will reveal that, from the point of view of a job requiring more than LQ amount of run time, both the systems are equivalent. In fact, usual and modified feed back algorithms vary only with respect to its treatment to jobs of size less than

or equal to LQ. That is because, the tagged job gets its (L+1) th and onwards quanta precisely at the same time in both the algorithms. From the point of view of the tagged job, it is immaterial how we permute small jobs and whether we give them one quantum at a time or in bulk service, because tagged job gets its (L+1)th quantum in both the systems only when all the small jobs before it has already finished service. Instead of getting one quantum at a time for initial L queues in usual feed back discipline it gets LQ amount at one go in modified feed back discipline which helps small jobs to finish service earlier, but makes no difference for tagged job. Hence, waiting time in the system W_k for the tagged job in modified version is same as that in usual multi-level feed back algorithm, which is given as [3].

$$W_k = \frac{\lambda [E_k(S^2) + Q^2 \cdot \sum_{i=k}^{\infty} [1-G(i)]]}{2[1-\lambda E_{k-1}(S)][1-\lambda E_k(S)]} + \frac{(K-1)Q}{1-\lambda E_{k-1}(S)} + Q \quad (3.24)$$

where

$$E_k(S) = \sum_{i=1}^k (iQ) g_i + kQ[1-G(k)]$$

and

$$E_k(S^2) = \sum_{i=1}^k (iQ)^2 \cdot g_i + (kQ)^2 [1-G(k)]$$

Note : Results for the continuous service distribution can be obtained from the expressions of W_k derived above by taking the limit of W_k as $Q \rightarrow 0$, $K \rightarrow \infty$ such that $t = QK$ remains constant

$$\text{i.e. } W(t) = \lim_{k \rightarrow \infty} W_k.$$

$$Q \rightarrow 0$$

$$K \rightarrow \infty$$

$$KQ = t$$

Chapter - IV

SIMULATION EXPERIMENT

4.1 Development of Simulation Program

A simulation program was run to compare usual multi-level feed back discipline and modified feed back discipline, with respect to weighted average waiting time for the job and system throughput. Since, average waiting time is a multiple of quantum size, we define weighted average waiting time as the average waiting time divided by job size, to make the expression independent of quantum size Q . System throughput was defined as

$$1 / \sum \frac{W_k}{KQ} \text{ Prob. [Job Size= } KQ]$$

The above is a valid measure for performance evaluation as increase in throughput according to the above criteria means reduced weighted average waiting time for the jobs and thus number of jobs served/time unit will increase. To obtain system throughput through simulation, an unbiased estimate of

$$\text{Prob[Job Size=KQ]} = \frac{\text{No. of Job Size KQ Served}}{\text{Total No. of Jobs Served}}$$

(which is nothing but the relative frequency) is used.

There are essentially two programs one for the usual

feed back simulation and the other one for the modified feed back simulation. Programming Language used is Pascal. Programs make use of Dynamic data storage, in the sense that no memory requirements are declared beforehand.

Two different types of Record Data structure are used called queue-node and Job-node respectively. Fig. 4.1 shows the node-structure explicitly.

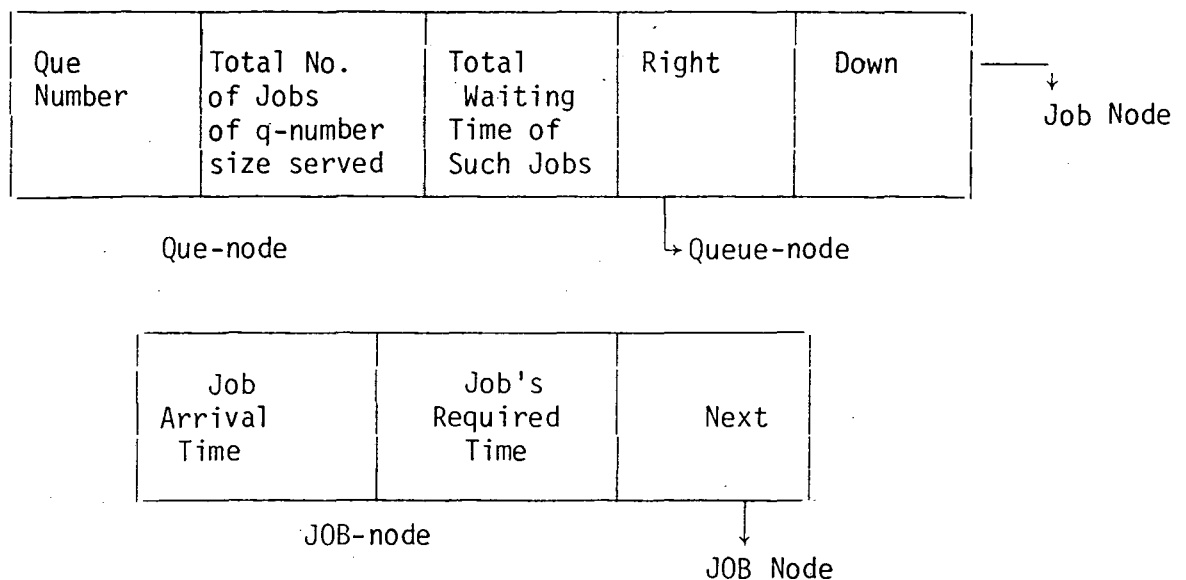


Fig. 4.1

Programs are modular in nature and has following modules :

- i] Procedure EXPO
- ii] Procedure UNIF
- iii] Procedure INITIALIZE

- iv] Procedure ADDQ
- v] Procedure CHANGEQ
- vi] Procedure HEADREPORT
- vii] Procedure REPORT
- viii] MAIN PROGRAM

[i] Procedure EXPO generates inter-arrival time which is exponential since we have assumed a Poisson arrival. The method makes use of Inverse Cumulative Distribution Function. If X is a continuous random variable with cdf $F(X)$ then random variable $Y = F(X)$ has uniform $[0,1]$ distribution. Hence Y is generated from $U[0,1]$ and $X = F^{-1}(Y)$ gives observation from X . To generate pseudo random numbers Tausworthe's feed back shift register method [6] was used. The basic method uses a sequence of polynomials $\{U_k\}$ with coefficients in $GF(2)$, generated by recurrence relation.

$$\begin{aligned}
 U_0 &= 1 \\
 U_k &= x U_{k-1} \pmod{x^p + x^q + 1}
 \end{aligned}$$

$\{U_k\}$ formed in this manner are given a circular shift of $(p-q)$ places to the left to form the sequence of p -tuples $\{W_k\}$ on W_k the following operations are performed. We also make the following assumptions:

Operations ($q < \frac{p}{2}$ and n is integer multiple of p).

- 1) W_n is set in Register A in bit positions 1 through p .
(exclusive of sign bit).
- 2) Register A is copied into Register B and Register B is left shifted to q places bringing 0's into the q right most places.
- 3) Register A is exclusive or-ed into Register B and result is stored back into Register A.
- 4) Register B is right shifted to $p-q$ places bringing 0's in from the left.
- 5) Register B is exclusive or-ed into A.

This method results in Register A containing new Pseudo-random integer.

For assessing the goodness of generated sequences of pseudo-random numbers, following empirical tests (the tests which are applied to samples of generated output) were applied

- a) Kolmogrov Smirnov Test
- b) Marsaglia's Lattice Test

- a) Kolmogrov Smirnov [5] (One sample, non parametric) test for goodness of fit tests null hypothesis H_0 : $F(x) = F_0(x)$ for all x , against the alternative H_1 : $F(x) \neq F_0(x)$, for some x . F_0 is a completely

specified continuous distribution function. The K-S statistic used for this purpose is

$$D_n = \sup_x | S_n(x) - F_0(x) |$$

where n is the sample size and $S_n(x)$ is the empirical distribution function. In our case $F_0(x) = x$ as X follows $U[0,1)$ D_n is a distribution free statistic and critical values of D_n in case of uniform distribution are available. This test gives good result even for small values of n .

b) The lattice test [6] for testing n -space uniformity is given below :

Step 1 : $(n+1)$ set of n -tuples of random numbers were generated successively and were denoted by p_1, p_2, \dots, p_{N+1} .

Step 2 : Let D = the absolute value of the determinant of

$$p_2 - p_1$$

$$p_3 - p_2$$

$$\vdots$$

$$p_{n+1} - p_n$$

If value of D is zero, then Step 1 is repeated.

Step 3 : Step 1 and Step 2 are repeated to form a sequence of D's, D_1, D_2, \dots

Step 4 : Let

$$g_2 = \gcd(D_1, D_2)$$

$$g_3 = \gcd(D_1, D_2, D_3)$$

$$g_4 = \gcd(D_1, D_2, D_3, D_4)$$

Step 5 : If some $g_i = 1$ generator has successfully passed the test.

If the sequence of g_i 's become constant ($\neq 1$) for a number of successive iterations, the generation have failed the test.

The lattice test was used as recommended by Marsaglia for $n = 1, 2, 3, \dots$. The program listings of these tests and Tausworthe generator are given in the Section 4.

(ii) Procedure UNIF generates observation from the distribution which has the pdf given by $f(X) = \frac{1}{N}$ for $X=1, 2, \dots, N$. Though the analytical result hold for any discrete service distribution. We have used only the above distribution to generate service time for the jobs. This procedure also makes use of inverse cumulative distribution function method.

- (iii) Procedure INITIALIZE when called initializes the system. It forms queue node for the first queue and attaches the first job node to the queue-node. In case of modified system, it initializes both Phase I and Phase II of the system and attaches the job node to Phase I queue or Phase II queue depending upon its size.
- (iv) Procedure ADDQ (r) attaches a new queue-node to the already existing queue-node pointed by pointer r. The q-number of the new node is assigned as the q-number of r-node+1.
- (v) Procedure CHANGEQ (p,time) when called takes a job-node at top of the queue-node pointed by p and checks the required run time of the job-node. If the job-node equals q-number it means the job has finished its service and then by making entries about its waiting time, it disposes the job node. Otherwise, it attaches the job-node to next lower-level queue to get further service. If the next lower queue does not exist, it calls procedure ADDQ to construct one such queue.
- (vi) Procedure HEADREPORT prints the headings of the output.

- (vii) Procedure REPORT prints the values obtained of the variables.
- (viii) Main program has a clock. This clock advances in multiples of quantum size Q (which in our model is taken as 1, without any loss of generality). The main program first calls Procedure INITIALIZE to initialize the system. After it becomes free, the 'simulated processor' searches for the highest-level non-empty queue, and calls the Procedure CHANGEQ. After the operation it advances the clock by quantum or by L Quanta if it is the L th queue the modified feed back discipline. As soon as new arrival comes it joins the tail of the highest level queue in usual feed back discipline. In case of modified feed back system, it joins Phase I or Phase II depending upon its JOB-size. The process is continued till the clock exceeds 'Simulation time' limit. After this main program calls Procedure HEADREPORT and Procedure REPORT to print the outputs.

4.2 Flow Charts

Flow charts of usual and modified multi-level feed back queue are listed. Procedure EXPO, Procedure UNIF and Procedure HEADREPORT being very straightforward are not given. Procedure ADDq and Procedure REPORT remain exactly same for both the simulations. Procedure CHANGEq and Procedure MAIN PROGRAM being very similar in both the cases, for the sake of brevity, only the changes required at appropriate places are mentioned, in

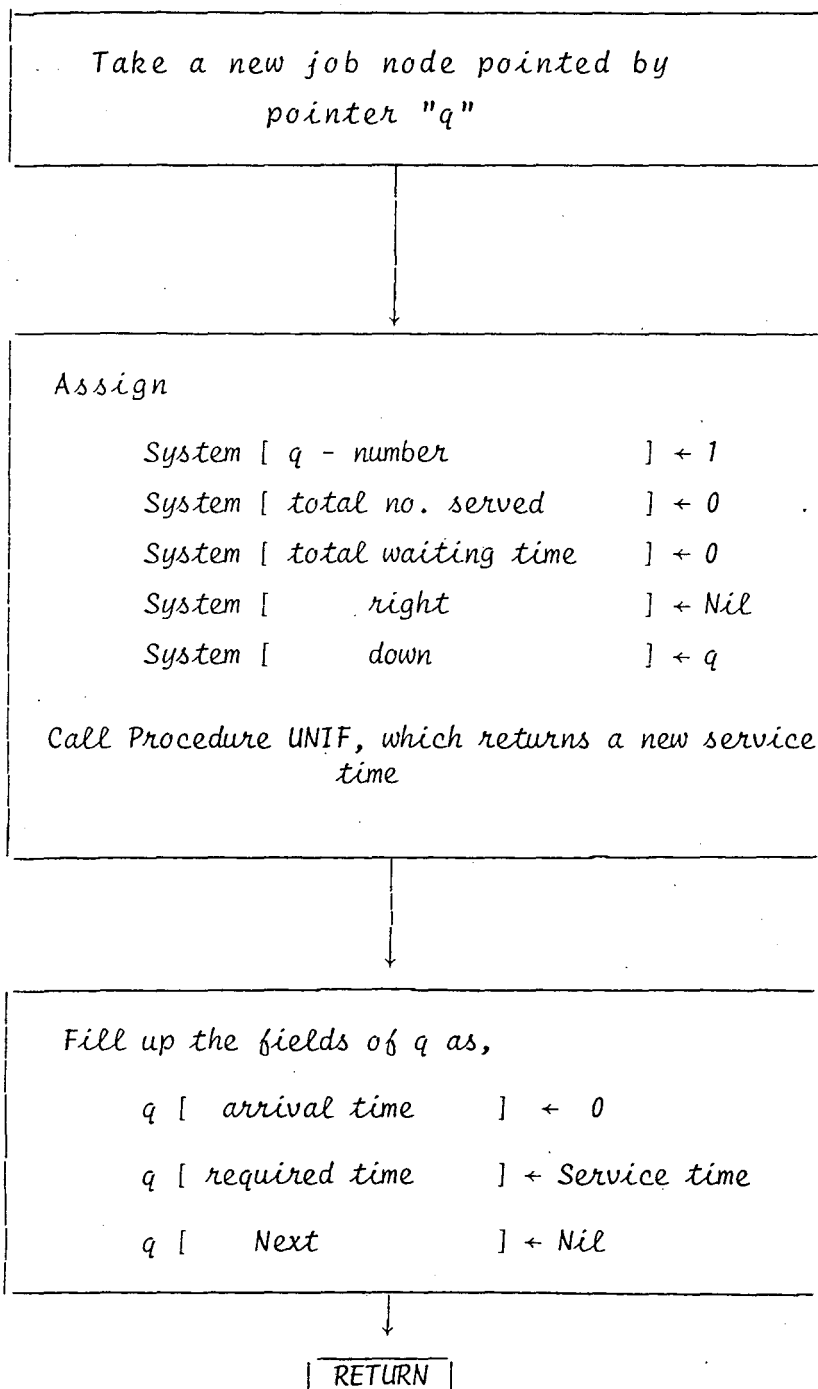
modified feed back case.

Note In the flow charts, $p[S]$ denotes the field S of the pointer node p . An assignment of the form $p[S] \leftarrow \text{NIL}$ implies that no node of the 'field S ' type is attached to p .

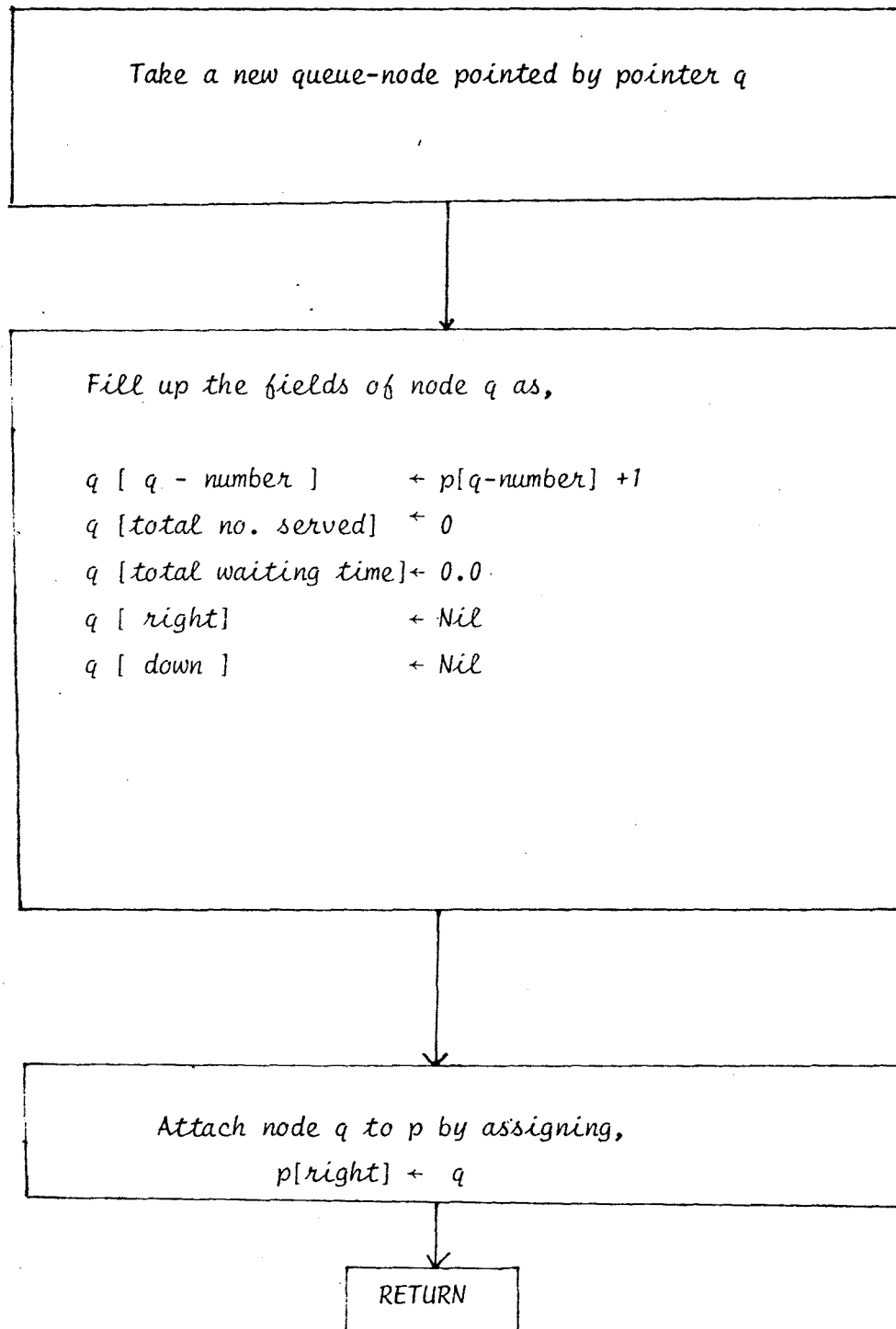
USUAL MULTILEVEL FEEDBACK QUEUE

Flow Chart - 4.2.1.

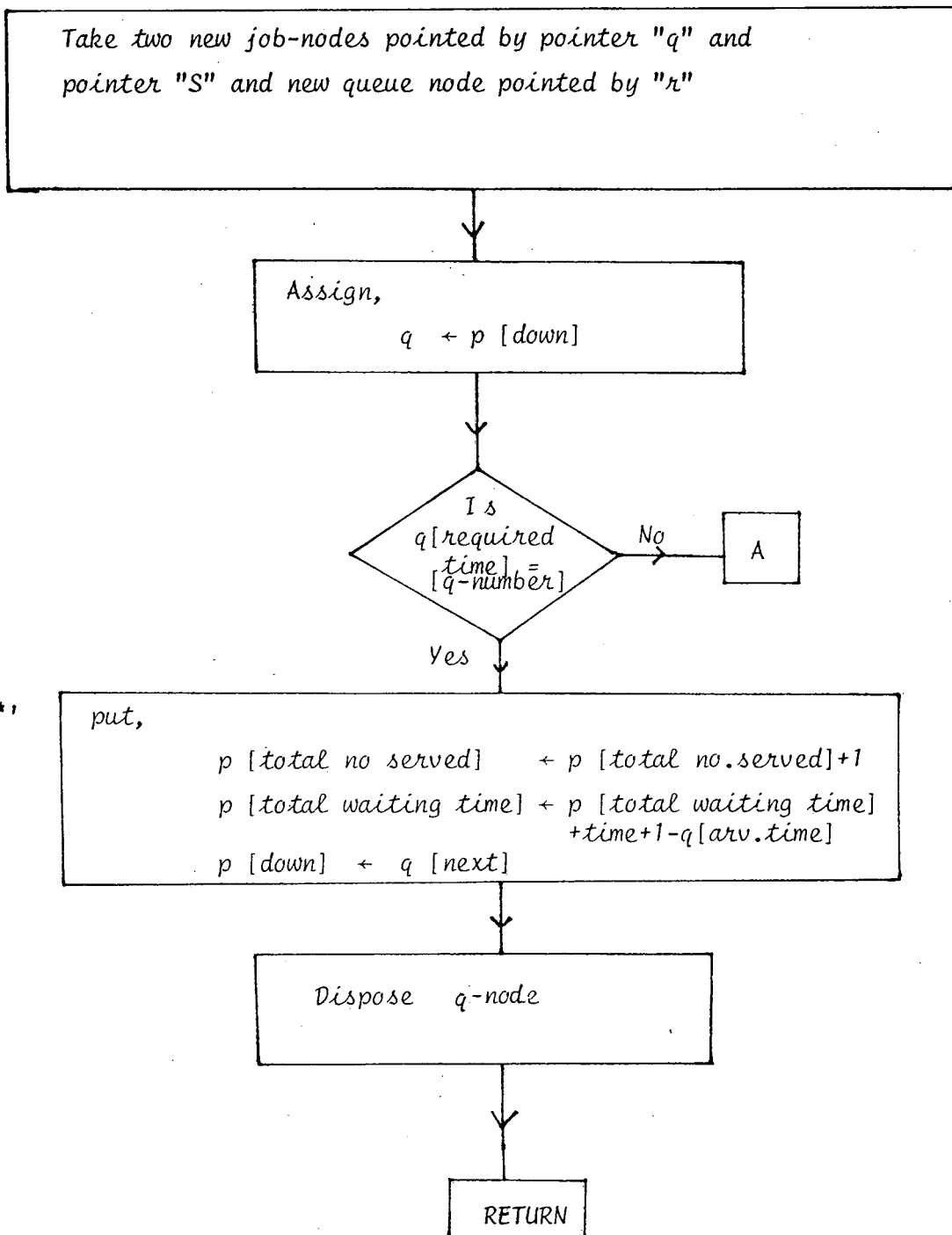
Procedure Initialize (System)

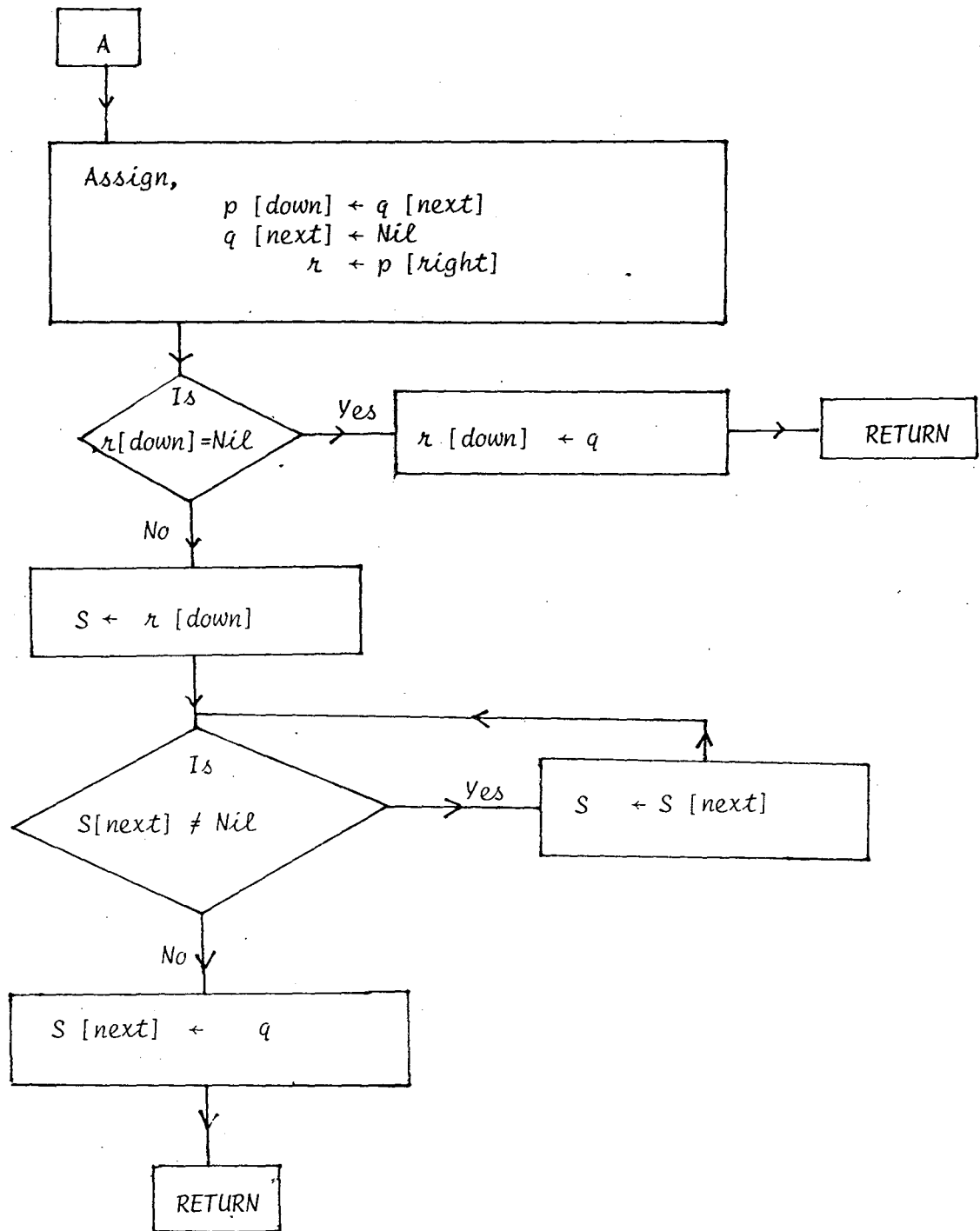


FLOW CHART 4.2.2

PROCEDURE $ADD_q(p)$ 

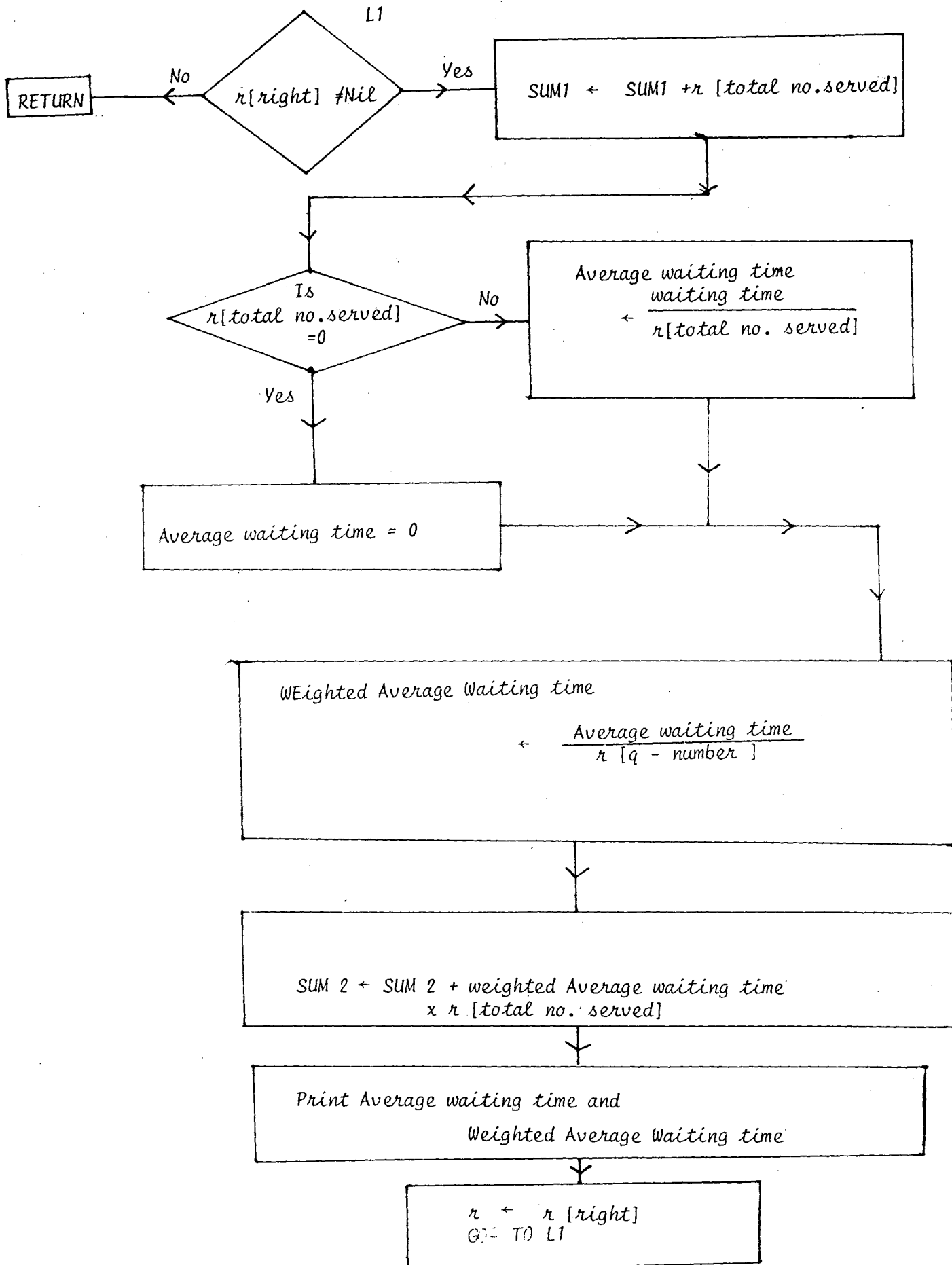
PROCEDURE CHANGE q (p ; $time$)



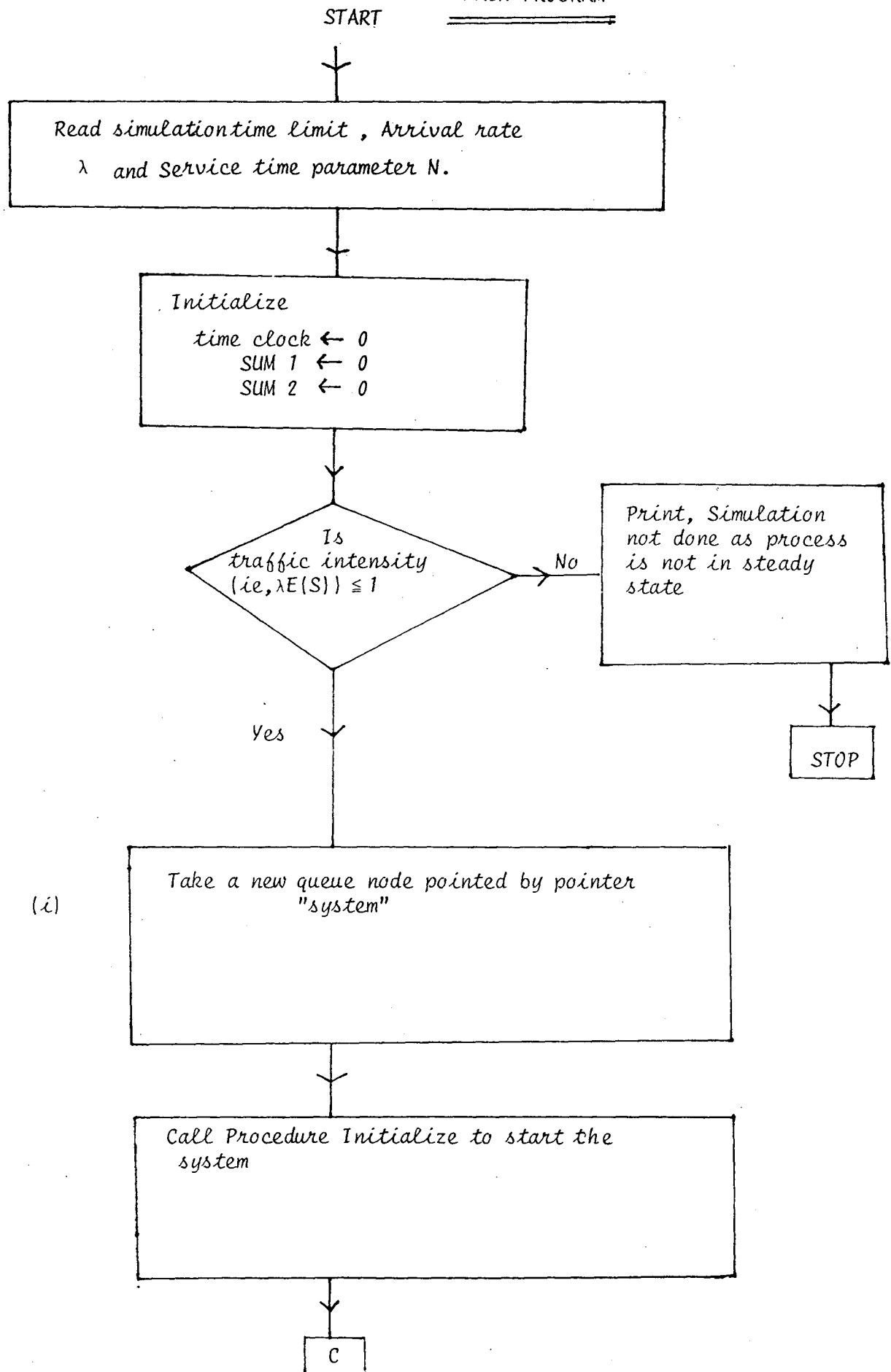


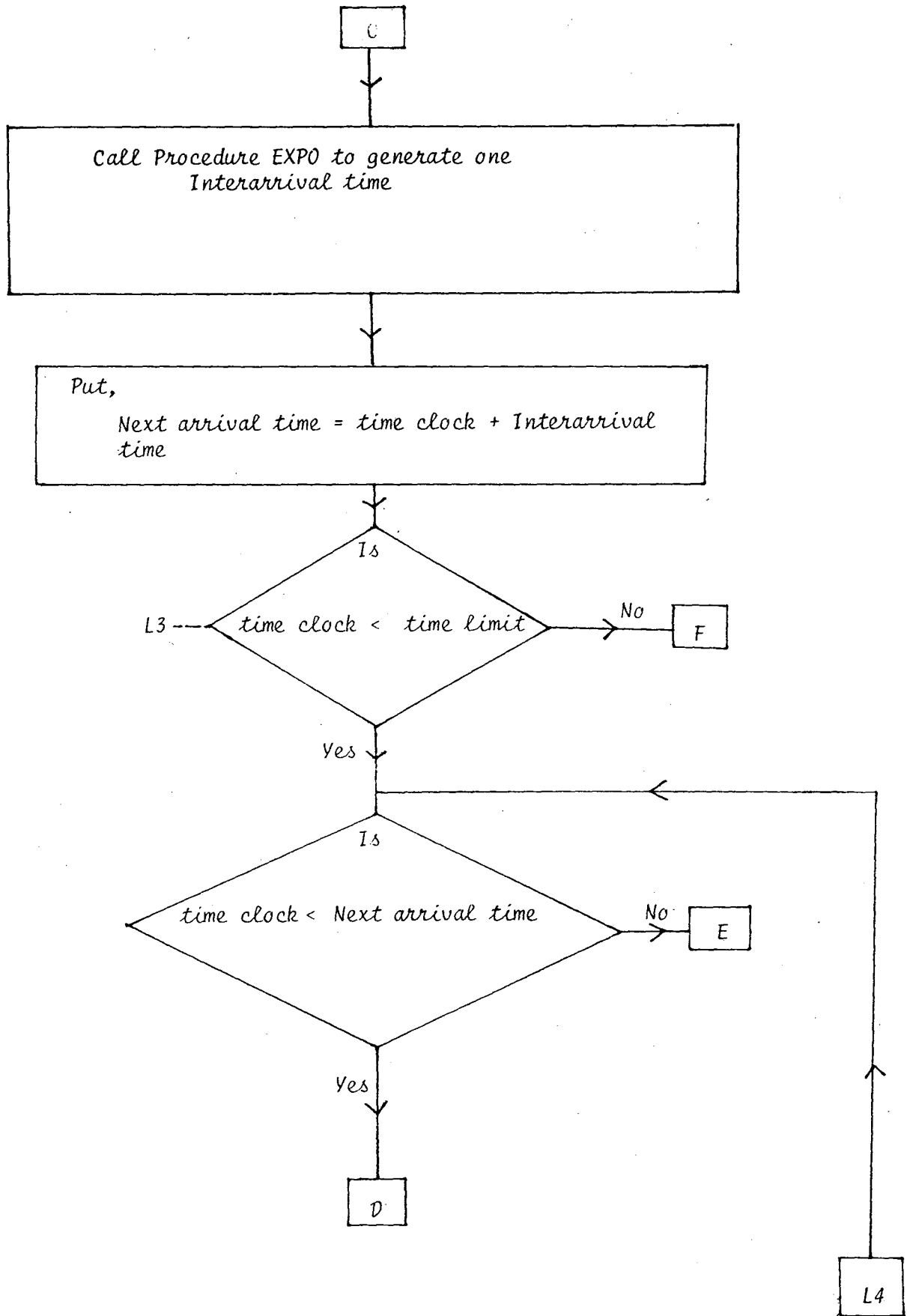
FLOW CHART 4.2.4

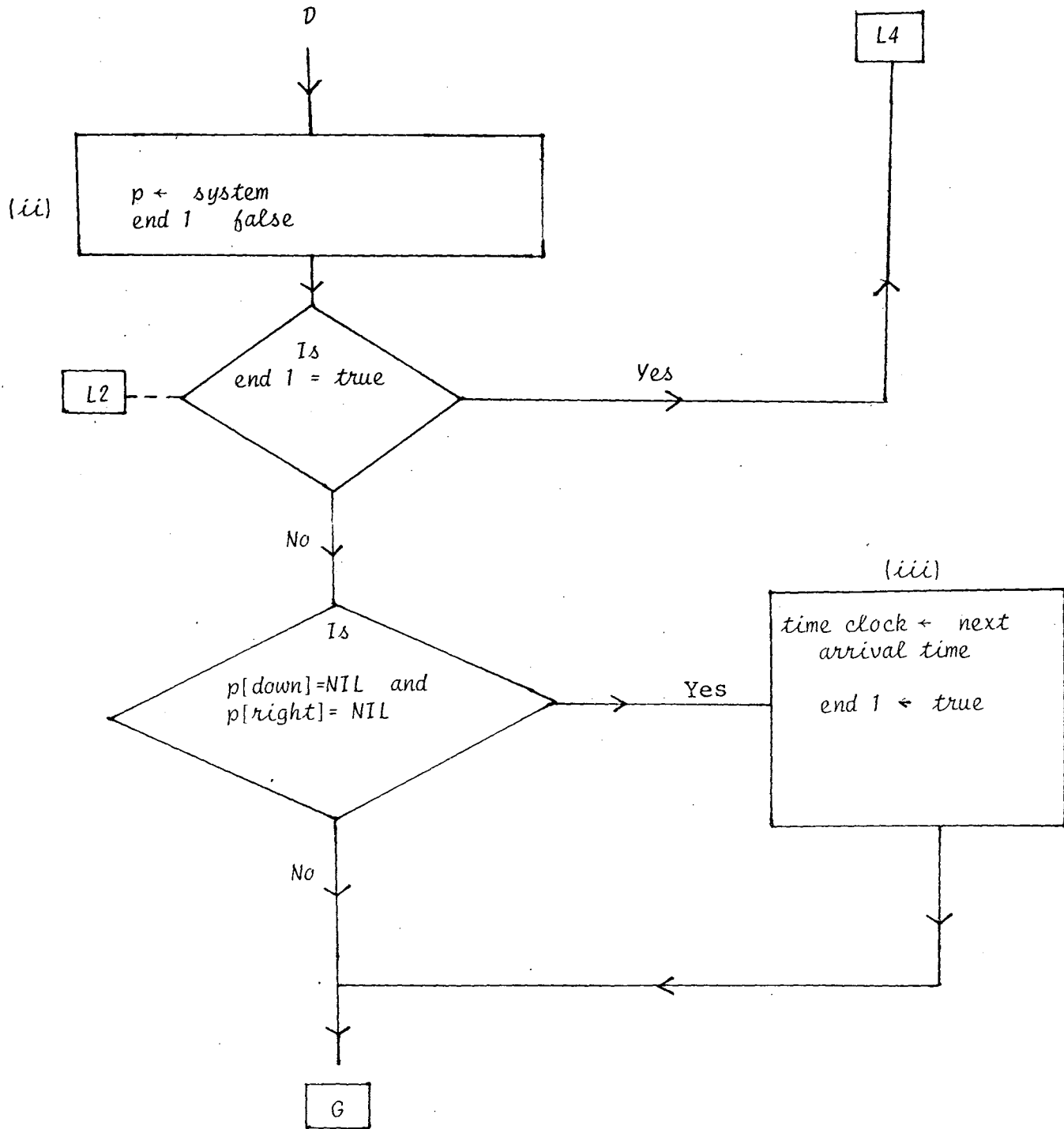
PROCEDURE REPORT (r, SUM1, SUM2)

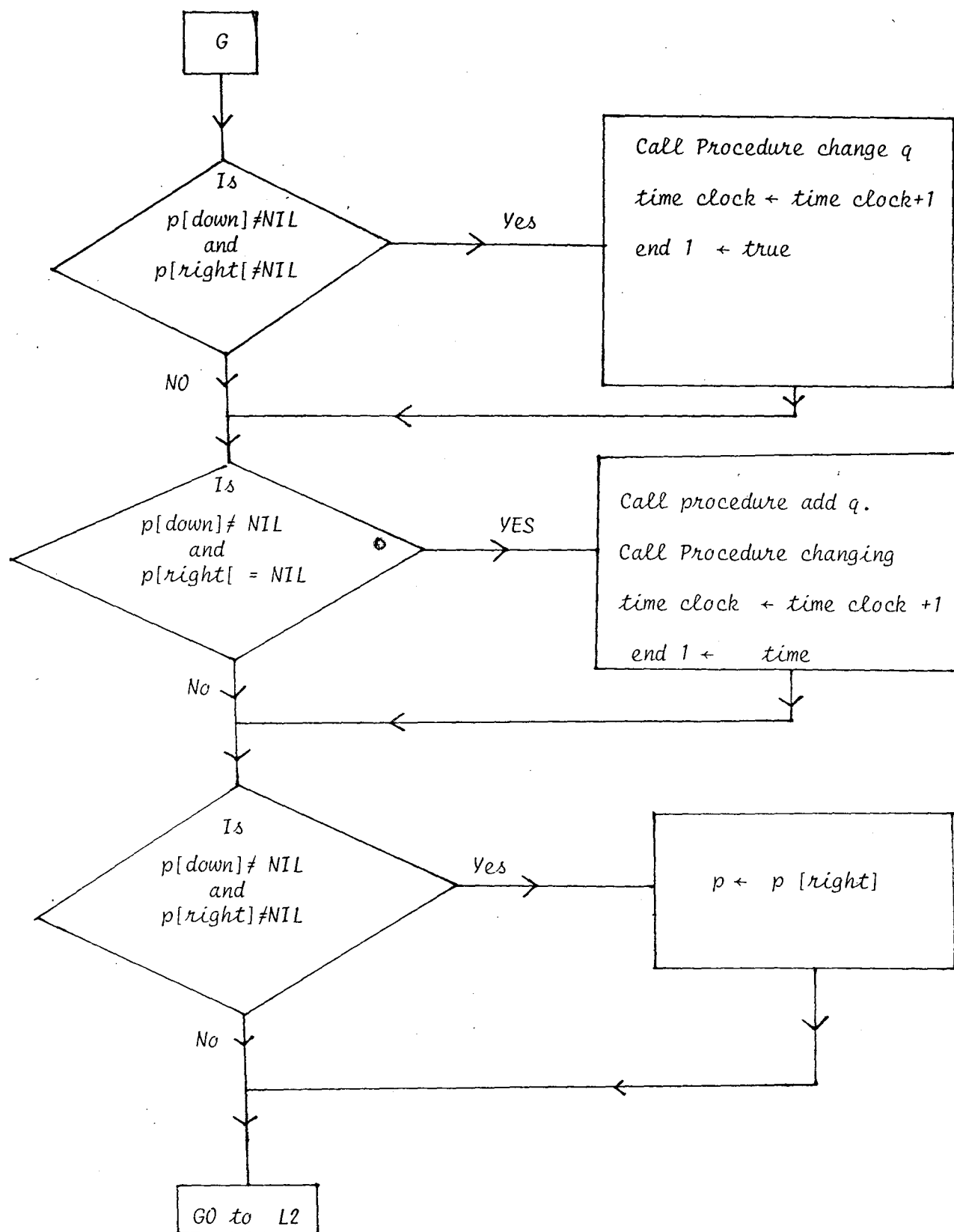


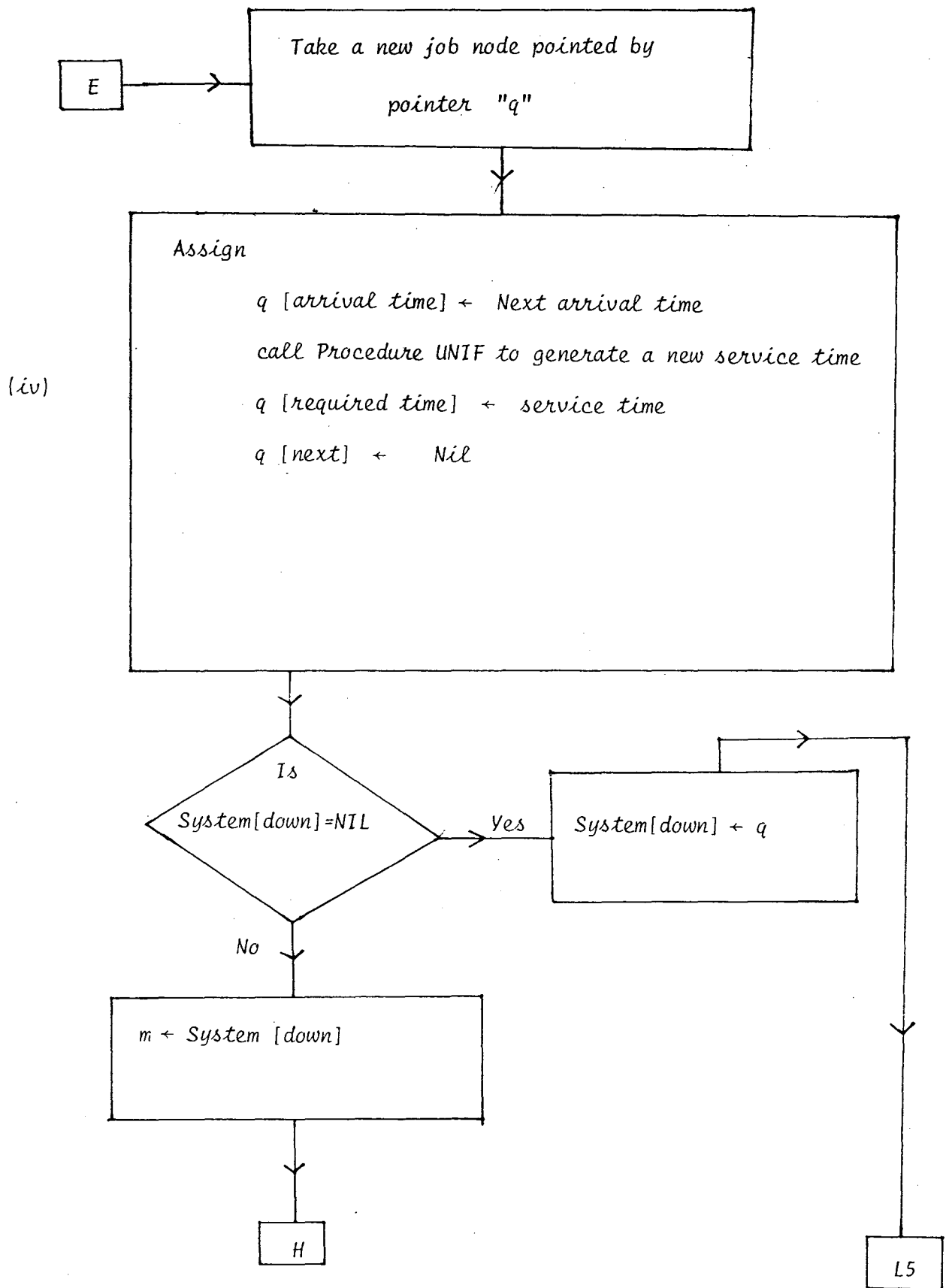
MAIN PROGRAM

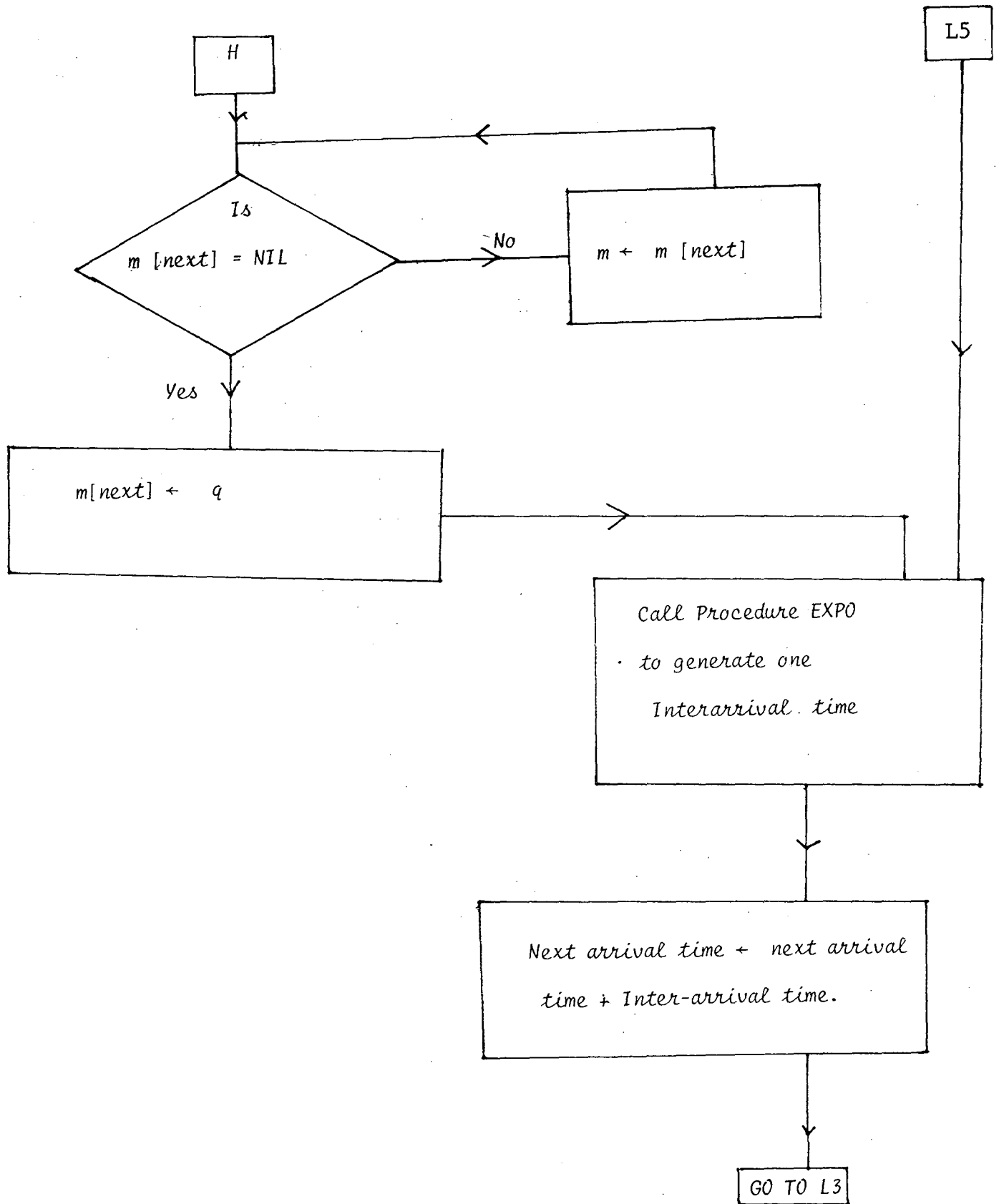


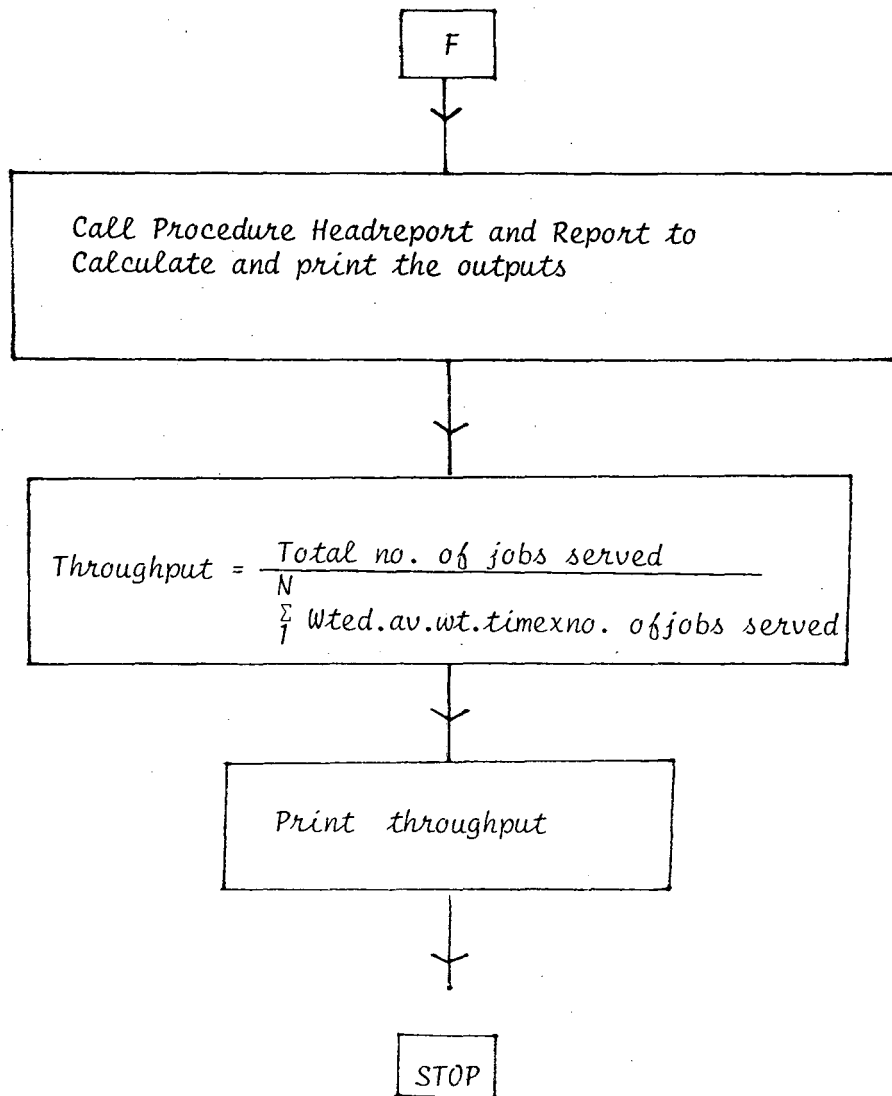








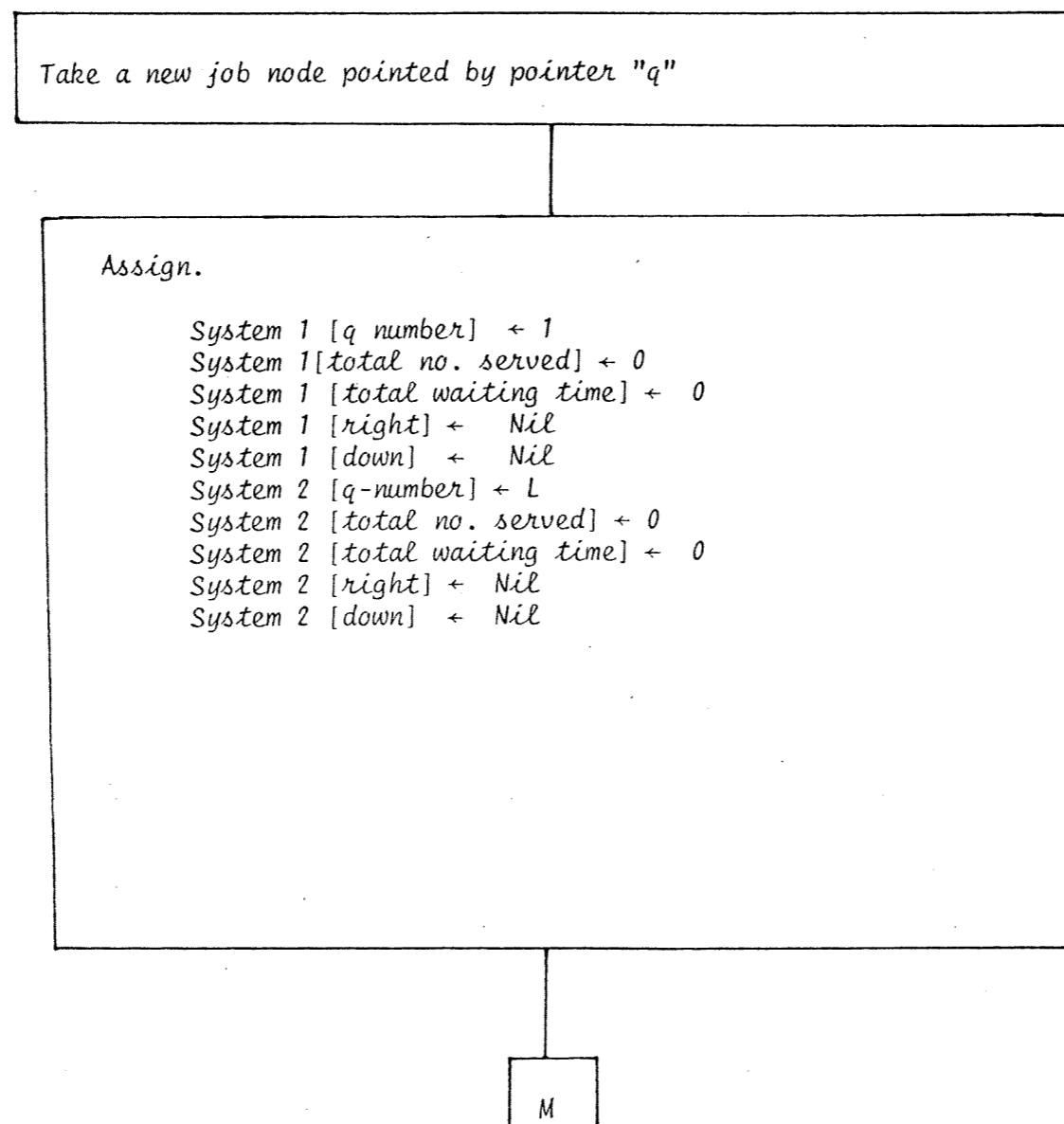


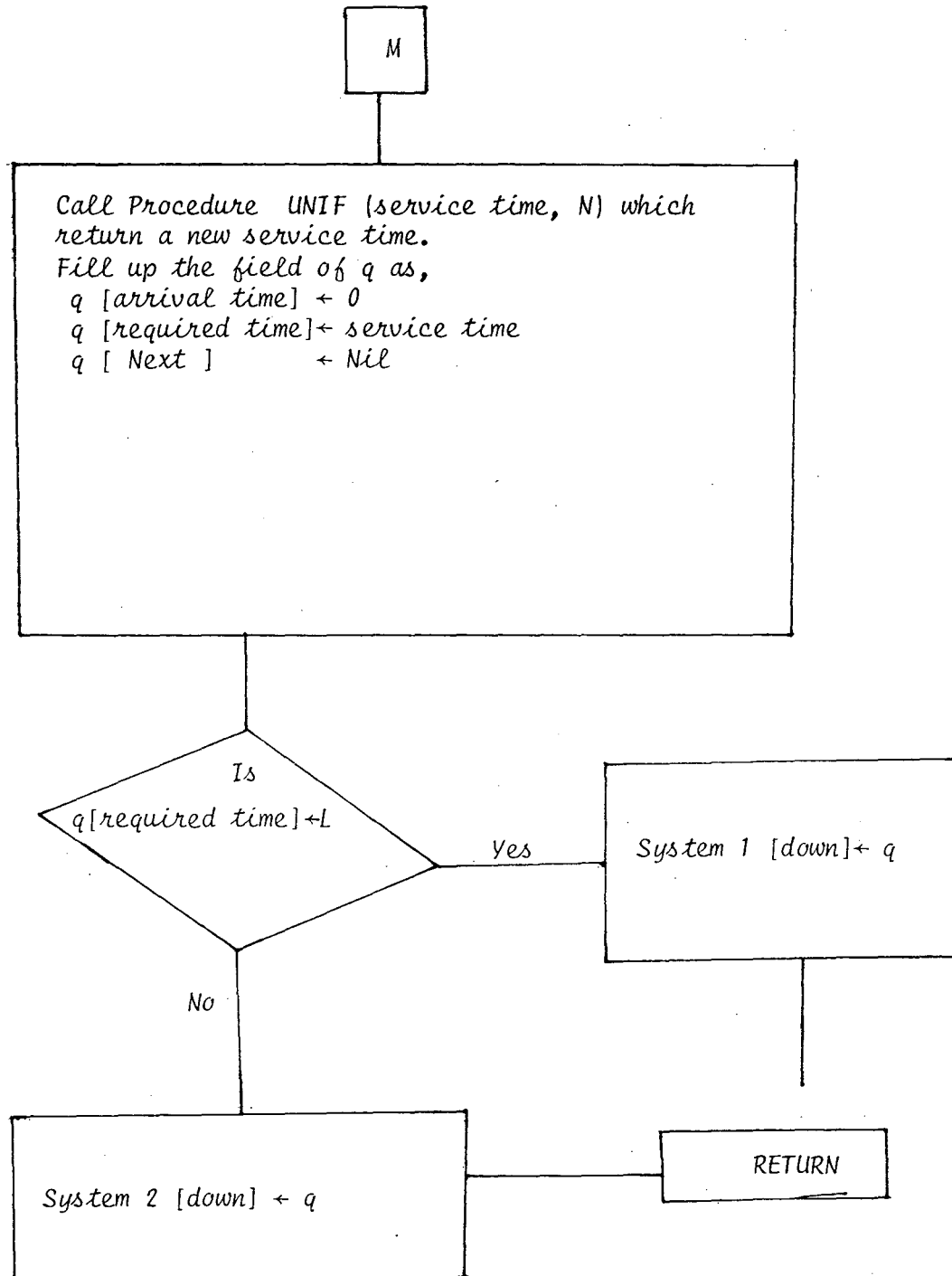


MODIFIED MULTILEVEL FEED BACK QUEUE

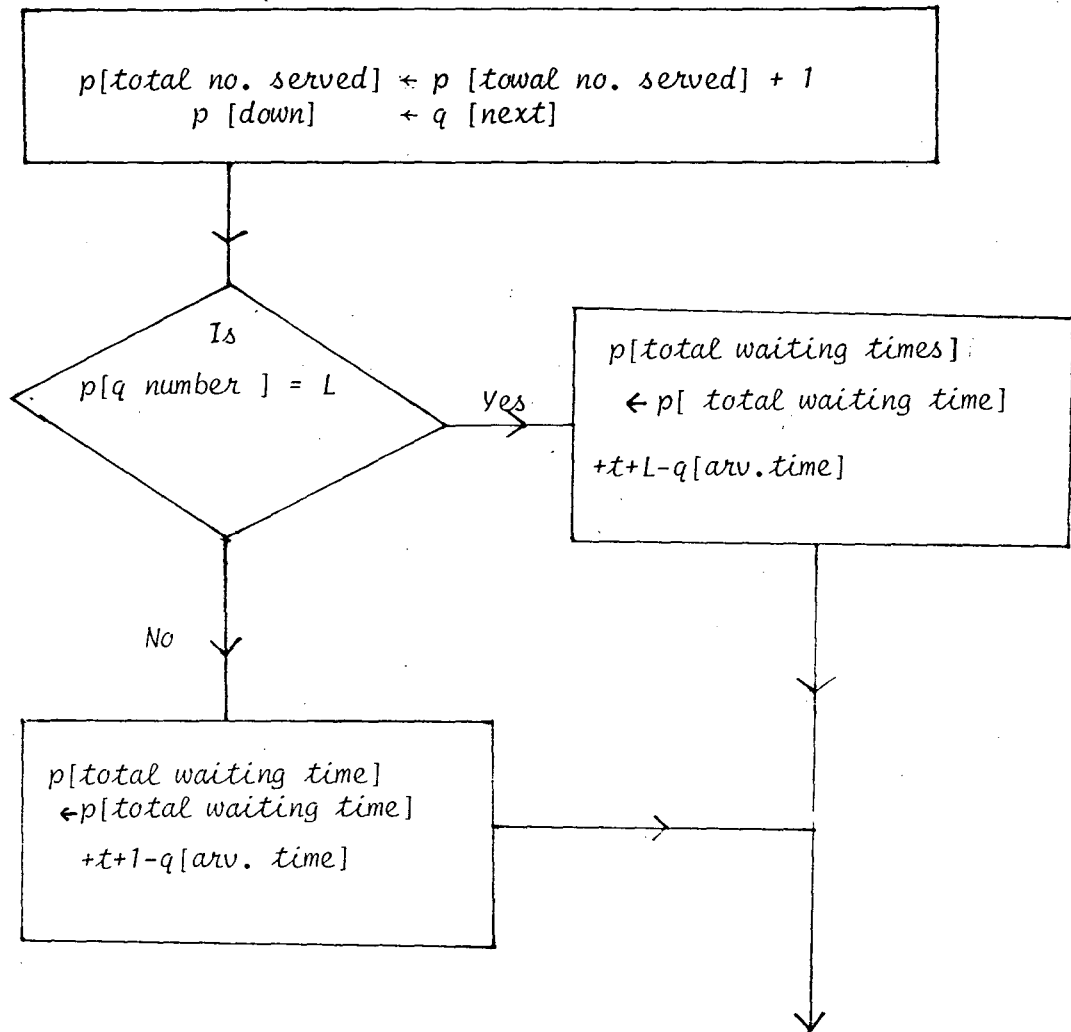
FLOW CHART 4.2.6.

Procedure Initialize (System 1, System 2)





Replace ; block marked with '*' of flow chart 4.2.3 by; (keeping all other parts same)



MAIN PROGRAM

Following changes in the flow chart of 4.2.5 are required

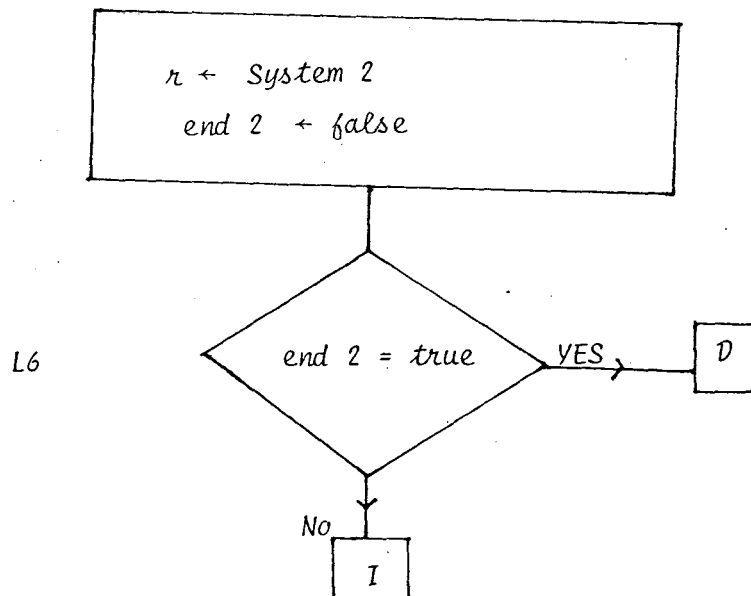
a) Replace block marked with (i) by,

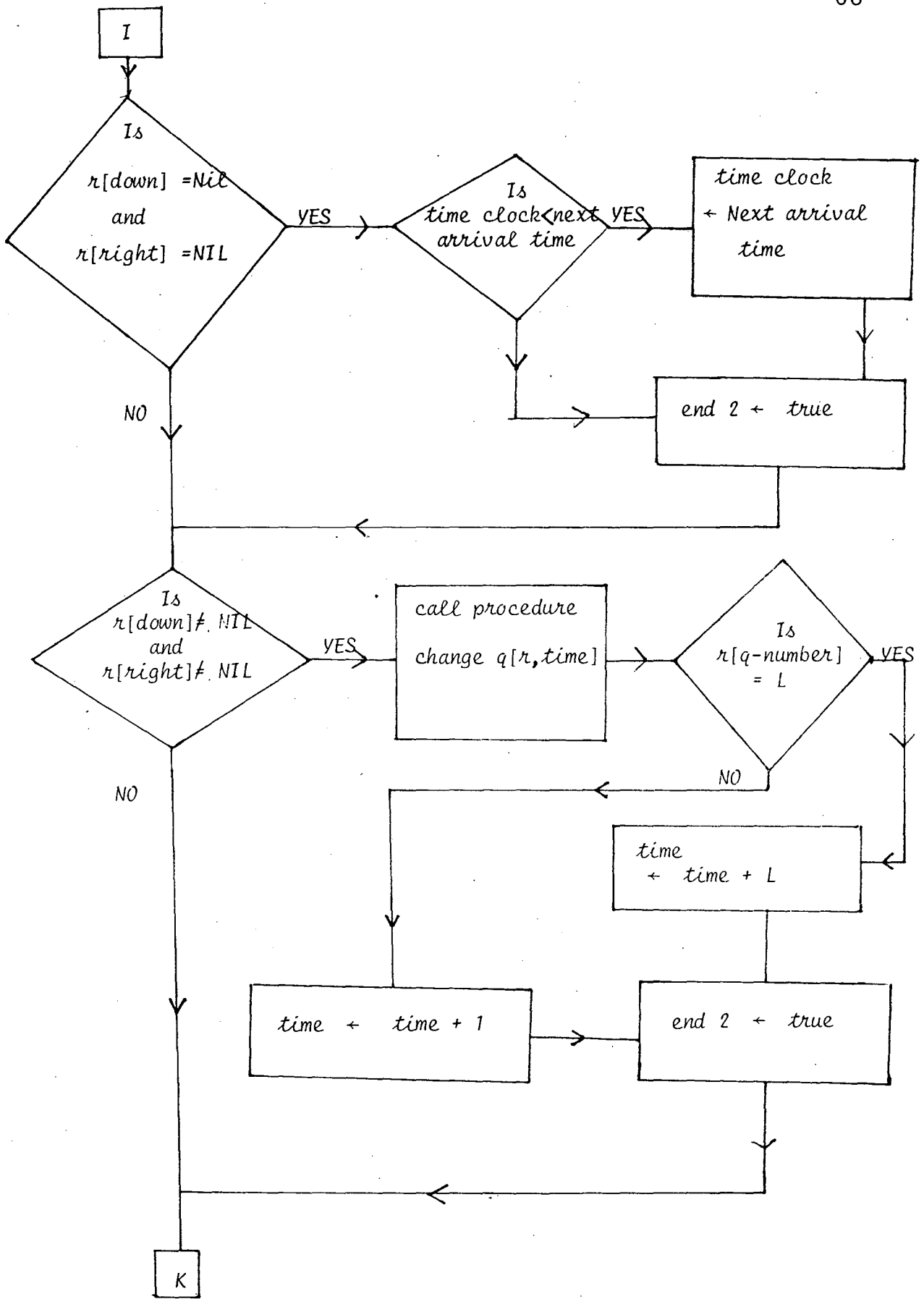
Take two queue nodes pointed by pointers
System 1 and System 2.

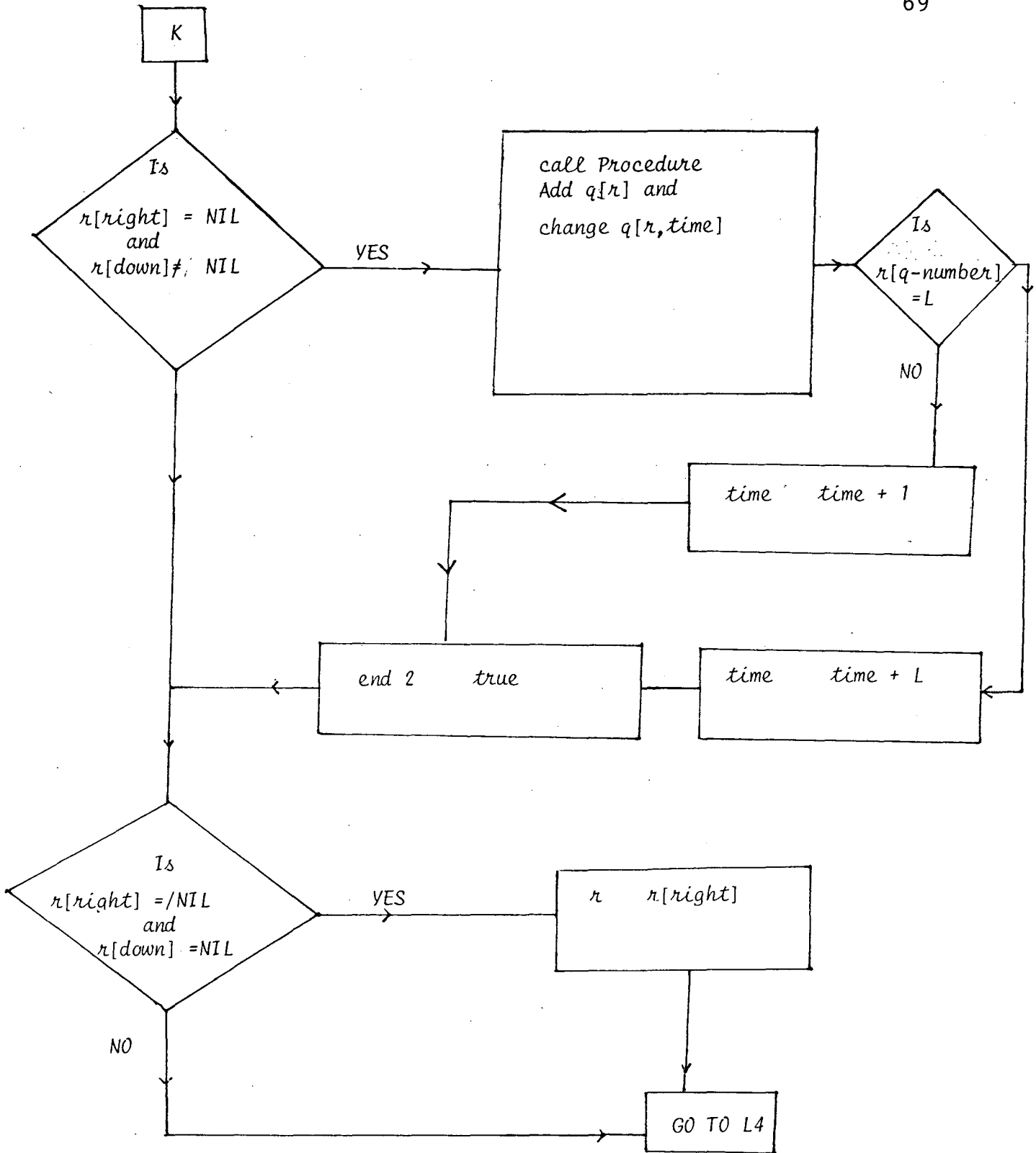
b) Replace block marked with (ii) by,

$p \leftarrow \text{System 1}$
 $\text{end 1} \leftarrow \text{false}$

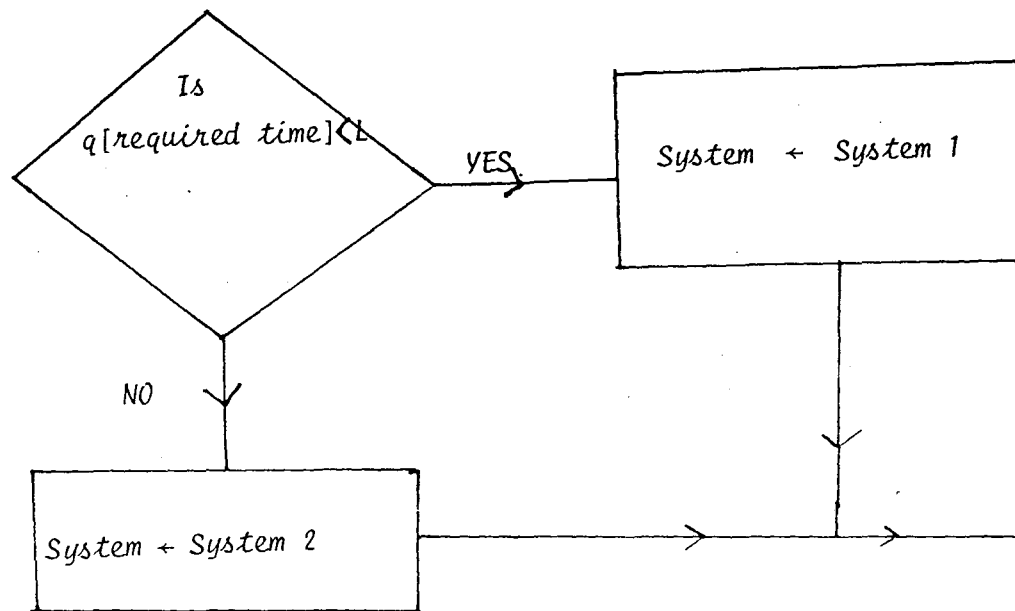
c) Replace block marked with (iii) by,







d) Add after block marked (iv),



All other portions remain same

4.3 Program Listing

Enclosed Programs

- 1] Tausworthe Generator
- 2] Kolmogrov - Smirnov Test
- 3] Lattice Test
- 4] Usual Feedback Queue Simulation
- 5] Modified Feedback Queue Simulation

PROGRAM 4.3.1

```

1      C  PROGRAM TAUSWORTHE GENERATOR
2      REAL*8      C(31),A2,B2
3      INTEGER      P,Q,PP,IA(31,31),IB(31),SBIT
4      INTEGER*8    K,S,IS,A(31),INT
5      PP=33
6      Q=13
7      P=31
8      K=2
9
10     DO 123 I=1,5
11     K=K*K
12     123 CONTINUE
13     K=K/2-1
14     A(1)=1
15     DO 10 I=2,P
16     10  A(I)=2*A(I-1)
17     DO 321 I=1,P
18     L=1
19     13  IF (A(I).EQ.1.AND.L.LT.P) GO TO 14
20     IF (A(I).EQ.1.AND.L.EQ.P) GO TO 33
21     IB(L)=MOD(A(I),2)
22     A(I)=A(I)/3
23     L=L+1
24     IF (L.LE.P) GO TO 13
25     GO TO 121
26     14  IB(L)=1
27     DO 17 K1=L+1,P
28     17  IB(K1)=0
29     GO TO 121
30     33  IB(P)=1
31     121 DO 555 M=1,P-Q
32     555 IA(I,M)=IB(Q+M)
33     DO 556 M=P-Q+1,P
34     556 IA(I,M)=IB(M-P+Q)
35     321 CONTINUE
36     SBIT=0
37     DO 41 J=1,PP
38     DO 80 I=1,P
39     DO 21 M=1,P
40     21  IB(M)=1
41     DO 22 M=Q+1,P
42     22  IB(M)=IA(I,M-Q)
43     CALL EXOR(IA,IB,P,I)
44     DO 29 NN=1,Q
45     29  IB(NN)=IA(I,P-Q+NN)
46     DO 30 NN=Q+1,P
47     3   IB(NN)=0
48     CALL EXOR(IA,IB,P,I)
49     S=IA(I,1)

```

```

50      DO 26 N1=7,P
51      IF (IA(I,N1).EQ.0)GO TO 26
52      IB=IA(I,N1)
53      DO 111 I1=1,N1-1
54 111  IS=IS*2
55      S=S+IS
56 26   CONTINUE
57      A2=S
58      B2=K
59      C(I)=A2/B2
60      IF(SBIT.EQ.0)GO TO 747
61      C(I)=C(I)*1000
62      INT=C(I)
63      C(I)=C(I)-INT
64      SBIT=0
65      GO TO 87
66 747  C(I)=C(I)*1000
67      INT=C(I)
68      C(I)=C(I)-INT
69      SBIT=1
70 87   CONTINUE
71      DO 997 I=1,P
72      IS=P*(J-1)+I
73      WRITE(6,40)IS,C(I)
74 40   FORMAT (IX,I4,10X,F14.12)
75 997  CONTINUE
76 40   CONTINUE
77      STOP
78      END
79      SUBROUTINE EXOR(IA,IB,M,I)
80      DIMENSION IA(31,31),IB(31)
81      DO 24 N=1,M
82      IF (IA(I,N).EQ.1.AND.IB(N).EQ.1) GO TO 25
83      IA(I,N)=IA(I,N)+IB(N)
84      GO TO 24
85 25   IA(I,N)=0
86 24   CONTINUE
87      RETURN
88      END

```

PROGRAM 4.3.2

```

1      C      *****
2      C      PROGRAM KOLNOGRPV-SMIRINOV TEST FOR TESTING RANDOMNES
3      C      *****
4      REAL *8      RND(1000),X,MAX,D(3),ALPHA(3)
5      N=575
6      READ (5,39) (RND(I),I=1,N)
7      39  FORMAT(7X,F9.7)
8      DO 149 I=1,N-1
9      X=RND(I+1)
10     J=I
11     75  J1=J+1
12     IF (X.GE.RND(J)) GO TO 75
13     RND(J1)=RND(J)
14     J = J -1
15     IF (J.EQ.0) GO TO 76
16     GO TO 75
17     76  J1=J+1
18     RAN(J1)=X
19     149 CONTINUE
20     I=1
21     MAX=ABS(FLOAT(I)/FLOAT(N)-RND(I))
22     DO 152 I=2,N
23     RNS(I)=ABS(FLOAT(I)/FLOAT(N)-RND(I))
24     IF (MAX.GE.RND(I)) GO TO 152
25     MAX=RND(I)
26     152 CONTINUE
27     WRITE(5,20)MAX
28     20  FORMAT ('MAX = ' F14.10)
29     D(1)=1.63/SORT(N*1.0)
30     D(2)=1.56/SORT(N*1.0)
31     D(3)=1.22/SORT(N*1.0)
32     WRITE (6,19) (D(I),I=1,3)
33     19  FORMAT(2X,F15.12)
34     ALPHA(1)=0.01
35     APLHA(2)=0.15
36     ALPHA(3)=0.11
37     DO 280 I=0,3
38     IF(MAX.GT.D(I))GO TO 201
39     WRITE(6,205)ALPHA(I)
40     205 FORMAT('AT ALPHA=',F6.4,'LEVEL OF SIGNIFICANCE' RANDOM
41     * NUMBERS GENERATED SATISFY K-S TEST OF UNIFORMITY')
42     GO TO 280
43     201 WRITE (6,206)ALPHA(I)
44     206 FORMAT ('AT ALPHA=',F6.4 'LEVEL OF SIGNIFICANCE RANDOM
45     * NUMBERS GENERATED FAIL TO SATISFY THE K-S TEST OF UNIFORMITY;')
46     280 CONTINUE
47     STOP
48     END

```

PROGRAM 4.3.3

```

1   C *****
2   C PROGRAM LATTICE TEST TO TEST N-SPACE UNIFORMITY
3   REAL*8 RND(1000),P(4,3),U,V,W
4   INTEGER*8 MODULO,L1,N1,G(500),D(500),REM,TEMP
5   MODULO=64
6   DO 88 K=1,500
7   88  D(K)=0
8       L2=5
9       N1=575
10      READ(5,13) (RND(I),I=1,N1)
11  13  FORMAT (2X,F9.7)
12      DO 999 N=1,3
13      M=N1/(N*(N+1))
14      K=1
15      J1=1
16      L=1
17  222 IF(L.GT.M.OR.K.GT.M)GO TO 777
18      DO 998 J=1,N+1
19      DO 998 I=1,N
20      I1=M*(N+1)*(L-1)+I+(J-1)*N
21      P(J,I)=RND(I1)
22  998  CONTINUE
23      DO 997 J=1,N
24      DO 997 I=1,N
25      D(J,I)=P(J+1,I)-P(J,I)
26  947  CONTINUE
27      GO TO (51,42,53)N
28  51  DET=P(1,1)
29      GO TO 54
30  52  DET=P(1,1)*P(2,2)-P(1,2)*P(2,1)
31      GO TO 54
32  53  U=P(2,2)*P(3,3)-P(2,3)*P(3,2)
33      V=P(2,1)*P(3,3)-P(3,1)*P(2,3)
34      W=P(2,1)*P(3,2)-P(3,1)*P(2,2)
35      DET=P(1,1)*U-P(1,2)*V+P(1,3)*W
36  54  DET=ABS(DET)
37      L1=MODULO*DET
38      IF (L1.GT.0) GO TO 55
39      L=L+1
40      GO TO 222
41  55  D(K)=L1
42      WRITE(6,2)K,D(K)
43  2   FORMAT('K=',18.10X,'D(K)=',18)
44      GO TO 94
45  95  WRITE(6,3)G(K)
46  3   FORMAT('G(K)=',18)
47      IF(G(K).EQ.1) GO TO 111
48      IF(G(K).EQ.G(K-1))GO TO 193
49      L=L+1
50      K=K+1

```

```
51      J1=1
52      GO TO 222
53 193   J1=J1+1
54      IF (J1.EQ.L2) GO TO 666
55      L=L+1
56      K=K+1
57      GO TO 999
58 777   WRITE(6,776)N
59 776   FORMAT('FOR',I2,'-SPACE UNIFORMITY THE TEST DIDN'T TERMINATE')
60      GO TO 222
61 111   WRITE(6,112)N
62 112   FORMAT ('FOR',I2,'-SPACE UNIFORMITY THE GENERATOR HAS SUCCESSFULLY
63      * PASSED THE TEST')
64      GO TO 997
65 666   WRITE(6,667)N
66 667   FORMAT ('FOR',I2,'SPACE UNIFORMITY THEGENERATOR HAS FAILED TO
67      * PASS THE TEST')
68      GO TO 999
69 94    IF(K.EQ.1)GO TO 26
70      I=2
71 27    I1=I-1
72      IF(D(I).GE.D(I1))GO TO 24
73      TEMP=D(I1)
74      D(I1)=D(I)
75      D(I)=TEMP
76 24    REM=DDD(D(I1),D(I))
77      IF(REM.EQ.0)GO TO 25
78      IF(REM.EQ.1000)GO TO 29
79      D(I1)=D(I)
80      D(I)=REM
81      GO TO 24
82 25    IF(I.GT.K)GO TO 26
83      I=I+1
84      GO TO 22
85 29    G(K)=1
86      GO TO 26
87 26    G(K)=D1(K)
88 28    GO TO 94
89 999   CONTINUE
90      STOP
91      END
```


PROGRAM 4.3.4

```

PROGRAM DFBSimulation(INPUT,OUTPUT,RANDU,D3,D4):
Type
  pntdtype2= ^ndtype2;
  ndtype2=RECORD
    arvtime:real;
    reptime:integer;
    next:ptndtype2
  end;
  pntdtype1=^ndtype1
  ndtype1=RECORD
    onumber:integer ;
    totalns:integer ;
    totalwt:real ;
    right:ptndtype1 ;
    down:ptndtype2
  end;

VAR
  Lamda,Sum1,Sum2,Throughput,time,timelimit,nextarvtime:real;
  ProcessorUtilization,IntarvTime,wtime,avwtime:real;
  system,r,p:ptndtype1;
  m,q:ptndtype2
  endi:Boolean;
  N,X,ServiceTime:integer;
  D4,RANDU,D3:TEXT;

Procedure Expo(var intarvtime,Lamda:real);
VAR
  P,X, Random:real;
Begin
  Readln (RANDU,Random);
  IntArvTime:=-1.0*LN(random)/Lamda;
end;

Procedure Unif(Var ServiceTime,N:integer):
VAR
  P,X,Random:real;
Begin
  Readln(RANDU,Random);
  p:=1.0/N;
  x:=Random/P;
  servicetime:=Trunc(X)+1;
End;

procedure Initialize(system:ptndtype1);
VAR
  q:ptndtype2;
begin
  New(q);
  system^.onumber:=1
  system^.totalns:=0;
  system^.totalwt:=0.0;
  system^.right:=Nil
  system^.down:=q;

```

```

q.arvtime:=0.0;
  Unif(ServiceTime,N);
  q^.reqtime:=Servicetime;
  p^.next :=NIL
end;
procedure addq(q:ptndtype1) ;
  VAR
    q:ptndtype1 ;
  begin
    NEW(q);
    q^.onumber:=p^.onumber+1;
    q^.totalns:=0;
    q^.totalwt:=0.0;
    q^.right :=NIL;
    q^.down :=NIL;
    p^.right :=q
  end;
Procedure CHANGEq(p:ptndtype1;t:real);
  VAR
    q:ptndtype2;
    r:ptndtype1;
    s:ptndtype2;
  Begin
    q:=p^.down;
    if(q^.reqtime=p^.onumber)Then
      Begin
        p^.totalns :=p^.totalns +1;
        p^.totalwt = p^.totalwt + t+1-q^.arvtime;
        p^.down :=q^.next;
        Dispose(q);
      end
    Else
      Begin
        p^.down:=q^.next;
        q^.next:=NIL;
        r:=p^.right;
        If(r^.down=NIL) Then r^.down:=q
        else
          Begin
            s:=r^.down ;
            While (s^.next <> NIL) Do
              s:=s^.next;
              s^.next:=q;
            End;
          End;
        End;
      End;
  End;
procedure HeadReport(Var Laada,TimeLimit:real;Var N:integer);
  VAR
    I:integer;
  Begin
    Writeln(D3,'1');Writeln(D3);Writeln(D3);Writeln(D3);

```

```

Writeln(D3, ' ':18, 'SIMULATION RESULTS FOR A USUAL FEEDBACK QUEUE');
Writeln(D3);
Writeln(D3, ' ':8, 'InterArrival rate Exponential With parameter Lamda =');
Writeln(D3, Lamda:7:3); Writeln(D3);
Writeln(D3, ' ':3, 'Service rate discrete Uniform With parameter N =');
Writeln(D3, N:3); Writeln(D3);
Writeln(D3, ' ':15, 'Simulation time:=', TimeLimit:6:3);
Writeln(D3);
Write(D3, ' ':16); For i:=1 to 95 do Write(D3, '*'); Writeln(D3);
Writeln(D3, '*:11, *:17, *:25, *:75);
Write(D3, ' ':10, '*', ' ':4, 'JOB SIZE', ' ':4, ' ', ' ':4);
Write(D3, 'NO. OF JOB SERVED', ' ':4, '*', ' ':4, 'AV. WAITING TIME');
Writeln(D3, ' ':4, '*', ' ':4, 'WT.AV.WT.TIME', ' ':4, '*');
Writeln(D3, '*:11, *:17, *:27, *:25, *:25);
Write(D3, ' ':100, For i:=1 to 95 do Write(D3, '*'); Writeln(D3);
      End;
Procedure Report(r:ptndtype1; Var SUM1, SUM2:real);
      Var
            WtTime, AvwTime, WavWtTime:real; T:integer;
Begin
      Writeln(D3, '*:11, *:17, *:27, *:25, *:25, *:25);
      While (r^.right <> nil) do
      begin
            Sum1:=Sum1+r^.totalns;
            Write(D3, ' ':10, '*', ' ':7, r^.onumber:2, ' ':7, '*');
            Write(D3, ' ':11, r^.totalns:4);
            If r^.totalns=0 Then AvwTime:=0.0
      Else
            begin
                  WtTime:=r^.totalwt*1.0;
                  AvwTime:=WtTime/r^.totalns;
            end;
            WavwTime:=AvwTime/r^.onumber;
            Write(D3, ' ':11, '*', ' ':6, AvwTime:12:7, ' ':6);
            Writeln(D3, '*', ' ':6, WavwTime:12:7, ' ':6, '*');
            Writeln(D3, '*:11, *:17, *:27, *:25, *:25);
            Sum2:=Sum2+WavwTime*r^.totalns;
            r:=r^.right;
            Write(D3, ' ':10); For i:=1 to 95 do Write(D3, '*');
            Writeln(D3);
      end;
      End;
Begin
      rewrite(D3); reset(D4);
      For X:=1 to 4 do
      begin
            reset(RANDU);
            Time:=0.0; TimeLimit:=20000.00;
            Sum1:=0.0; Sum2:=0.0
            Readln(D4, Lamda, N);

```

```

        If( (Lamda*(N+1)) <= 2:0) then
begin
    Intarvtime:=0.0;ServiceTime:=0;
    New(System);
    Initialize(System);
    Expo(IntarvTime,Lamda);
    NextArvTime:=Time+IntArvTime;
    While(Time<TimeLimit)Do
    Begin
        While (Time<NextArvTime)do
        Begin
            p:=system;
            endl:=false;
            Repeat
            If(p^.down=NIL)and(p^.right=NIL)then
            Begin
                Times:=NextarvTime;
                Endl:=true
            End;
            If(p^.down<>NIL) and (p^.right<> Nil) then
            Begin
                Changeq(p,time);
                time:=time+1;endl:=True
            End;
            If(p^.down <> NIL) and (p^.right=NIL)Then
            Begin
                Addq(p);Changeq(p,time);
                time:=time+1;endl:=true
            End;
            If(p^.down=Nil) And (p^.right <>Nil) Then p:=p^.right;
                until endl=true;
            End;
            New(q);
            q^.ArvTime:=NextArvTime;
            Unif(ServiceTime,N);
            q^.ReqTime :=ServiceTime ;
            q^.Next :=Nil;
            If(System^.down=Nil) then System^.down:=0
            Else
                Begin
                    m:=System^.down;
                    while(m^.next<>Nil) do
                        m:=m.next; m^.next:=0
                    end;
                    Expo(IntarvTime,Lamda);
                    NextArvTime:=NextArvTime + IntArvTime
                End;
            HeadReport(Lamda.TimeLimit.N);
            Report(System,SUM1,SUM2);
            Throughput:=Sum1/Sum2;Writeln(D3);

```

```
Writeln(D3, ' ':16, 'Throughput:=', 'Throughput:10:6);  
      ProcessorUtilization:= Lamda*(N+1)/2.0;Writel3);  
Writeln(D3, ' ':16, 'ProcessorUtilization:=', Processorutilization:10  
Writeln(D3):Writeln(D3, ' ':40, '**TIME UNIT = 1 QUANTUM **');  
      end;  
end;  
END.
```

PROGRAM 4.3.5

```

PROGRAM      MFBSimulation(INPUT,OUTPUT,RANDU,D3,D4)      ;
TYPE
    ptndtype2=^ndtype2;
                ndtype2=RECORD
                arvtime   :   real           ;
                reqtime   :   integer        ;
                next      :   ptndtype2      ;
                end;
    ptndtype1=^ndtype1;
                ndtype1=RECORD
                onumber   :   integer        ;
                totalsns :   integer        ;
                totalwt   :   real          ;
                right     :   ptndtype1     ;
                down      :   ptndtype2     ;
                end;
VAR
    N,i,X,servicetime,L           :   integer      ;
    time,nexttarvtime,Lamda,sum1,sum2 :   real      ;
    timelimit,throughput          :   real      ;
    processorutilization,intarvtime :   real      ;
    t,system1,system2,T,p         :   ptndtype1   ;
    m,q                             :   ptndtype2   ;
    end1,end2                       :   Boolean    ;
    RANDU,D3,D                      :   TEXT      ;
Procedure Expo(Var intarvtime,landa:real);
    VAR
        Random:real;
    Begin
        Readln(RANDU,random)
        Intarvtime:=-1.0*LN(Random)/Lamda
    End
Procedure Unif(VAR ServiceTime,N:integer);
    VAR
        p,x,Random:real;
    Begin
        Readln(RANDU,Random)
        p:=1.0/N
        X:=Random/p
        Servicetime:=Trunc(x)+1;
    End
Procedure Initialize(system1,system2:ptndtype1);
    VAR
        q:ptndtype2;
    Begin
        NEW(q)
        system1^.onumber:=1
        system1^.totalsns:=0
        system1^.totalwt:=0.0
        system1^.right:=Nil

```

```

system1^.down:=Nil ;
system2^.onumber:=1 ;
System2^.totalns:=0 ;
system2^.totalwt:=0.0 ;
system2^.right:=Nil ;
system2^.down:=Nil ;
q^.arvtime:=0.0 ;
Unif(servicetime,n ;
q^.reqtime := servicetime ;
q^.next :=NIL ;
If(q^.reqtime<L) Then System^.down:=q
Else
System2^.down:=q
End;
Procedure Addq(p:ptndtype1);
VAR
q:ptndtype1;
Begin
NEW(q);
q^.onumber:=p^.onumber+1 ;
q^.totalns:=0 ;
q^.totalwt:=0 ;
q^.right:=Nil ;
q^.down :=Nil ;
p^.right:=q ;
End;
Procedure CHANGEq(p:ptndtype1;t:real);
VAR
q:ptndtype2;
r:ptndtype1;
s:ptndtype2;
Begin
q:=p^.down;
If (q^.reqtime=p^.onumber) Then
Begin
p^.totalns := p^.totalns + 1 ;
p^.down := q^.next ;
If(p^.onumber=L) Then
p^.totalwt:= p^.totalwt +t+1-q^.arvtime;
else
p^.totalwt:= p^.totalwt +t+1-q^.arvtime;
Dispose(q);
End
Else
Begin
p^.down:=q^.next;
q^.next:=NIL ;
r:=p^.right ;
If (r^.down = NIL ) Then r^.down:=q
Else

```

```

Begin
    s:=r^.down;
    While (s.next (<) NIL) Do
        s:=s^.next;
        S^.next:=q;
    End;
End;
End;
End;
Procedure Headreport (Var Lamda,TimeLimit:real;Var N:integer);
Begin
    Writeln(D3,' ');Writeln(D3);Writeln(D3);Writeln(D3);
    Writeln(D3,' ':11,'SIMULATION RESULTS FOR A MODIFIED FEEDBACK QUEUE ');
    Writeln(D3);
    Write(D3,' ':3,'InterArrival rate Exponential With parameter Lamda =');
    Writeln(D3,' ':8,'Service rate discrete Uniform With parameter N =');
    Write(D3,N:3);Writeln(D3);
    Writeln(D3,' ':15,'Simulation time:=',TimeLimit:6);
    Writeln(D3);
    Writeln(D3,' ':10,'Job Greater than or Equal to');
    Writeln(D3,L:2,'@'s is a Long job');
    Writeln(D3,'*':11,'*':17,'*':27,'*':25,'*':25);
    Write(D3,' ':10,'*',' ':4,'JOB SIZE',' ':4,'*',' ':4);
    Write(D3,'NO. OF JOBS SERVED',' ':4,'*',' ':4,'AV WAITING TIME');
    Writeln(D3,' ':4,'*',' ':4,'WT.AV.WT.TIME',' ':,'*');
    Writeln(D3,' ':11,' ':17,'*':27,'*':25,'*':25);
    Writeln(D3,' ':1);For i:1 to 95 do write(D3,'*');Writeln(D3);
END
Procedure Report (r:ptndtype1);
VAR
    WtTime,AvWtTime,WavWtTime:real;I:integer;
Begin
    Writeln(D3,'*':11,' ':17,'*':27,'*':25,'*':25);
    While (r^.right(<) Nil)do
    begin
        Sum1:=Sum1+r^.totalns;
        Write(D3,' ':10,'*',' ':7,r^.onumber:2,' ':7,'*');
        Write(D3,' ':11,r^.totalns:4);
        If r^.totalns=0 Then AvWtTime:=0.0
        Else
            begin
                WtTime:=r^.totalwt*1.0;
                AvWtTime:=WtTime/r^.totalns;
            end;
        WavWtTime :=AvWtTime/r^.onumber;
        Write(D3,' ':11,'*',' ':6,AvWtTime:12:7,' ':6);
        Writeln(D3,'*',' ':6,WavWtTime:12:7,' ':6'*');
        Writeln(D3,'*':11,'*':17,'*':27,'*':25,'*':25);
        Sum2:=Sum2+WavWtTime*r^.totalns;
        r:=r^.right;
    end;
END

```



```

End;
Begin
  rewrite(D3);reset(D4);
  For x:=1 to 50 do
    begin
      Reset(Randu);
      Time:=0.0;TimeLimit:=20000.00;
      Readln(D4,Lamda,N,L);
      IF((Lamda*(N+1))<=2.0)then
        Begin
          intarvtime:=0.0;servicetime:0 ;
          Sum1:=0.0;Sum2:=0.0 ;
          New(System1);New (System2) ;
          Initialize (System1,System2) ;
          Expo(Intervtime,Lamda) ;
          NextArvTime:=Time+IntArvTime ;
          While ( Time < TimeLimit) Do
            Begin
              While (Time<NextArvTime)Do
                Begin
                  p:=system1;
                  end1:=False;
                  Repeat
                    If(p^.down=NIL) and(p^.right=Nil)then
                      Begin
                        r:=system2;
                        end2:=False;
                        Repeat
                          If(r^.down=Nil) And (r^.right=Nil) then
                            begin
                              If(Time<NextarvTime) THEN Time:=NextarvTime;
                              End2:=true
                              End;
                              If(r^.down<>Nil) And (r^.right<>Nil) then
                                begin
                                  Changeq(r,time);
                                  If(r^.onumber=L) Then
                                    time:=time+L else
                                      time:=time+1;End2:=true End;
                                  If(r^.down<>Nil)And (r^.right=Nil) then
                                    begin
                                      Addq(r);Changeq(r,time);If(r^.onumber=L)Then
                                        time:=time+L else
                                          time:=time+1;End2:=true
                                          End;
                                      If(r^.down=Nil) And (r^.right<>Nil) then r:=r^.right;
                                      Until End2=true;
                                      End1:=true
                                      End;
                                  If (p^.down<>NIL) and (p^.right<>NIL) Then

```

```

begin
  Changeq(p,time);
  time:=time+1;
  endl:=true
end;
If(p^.down<>Nil) And (p^.right<>Nil) Then
begin
  Addq(p);changeq(p,time);
  time:=time+1;endl:=true
End;
If(p^.down=Nil) And (p^.right<>Nil) Then r:=p^.right;
until endl=true;
end;
New(q);
q^.ArvTime:=NextArvTime ;
Unif(Servicetime,N) ;
q^.ReqTime:=ServiceTime ;
q^.Next :=Nil ;
If(q^.reqtime<L) Then t:=system1
else t:=system2;
If(t^.down=Nil) then t^.down:=0
else BEGIN
m:=t^.down;
while(m^.next<>Nil)do
m:=m^.next;m^.next:=q END;
Expo(Intarvtime,Lamda);
NextArvTime:=NextArvTime + IntArvTime;
End;
HeadReport (Lamda,TimeLimit,N);
Report(system1);
Report(system2);
Write(D3,' ':10;For i:=1 to 95 do Write (D3,'*');Writeln(D3);
Throughput:=Sum1/Sum2;Writeln(D3);
Writeln(D3,' ':10;'Throughput :=',Throughput:10:6);
Processor Utilization:=Lamda*(N+1)/2.0;Writeln(D3);
Writeln(D3,' ':10;'Processor Utilization:='processorutilization:10:6);
Writeln(D3);Writeln(D3,' ':40,'**TIME UNIT =1 QUANTUM**');
End;
End;
END

```

TABLE 4.4.1

SIMULATION RESULTS FOR A USUAL FEEDBACK QUEUE
 InterArrival rate exponential with parameter $\lambda = 0.050$
 service rate discrete Uniform with parameter $N = 10$
 Simulation time := 20000.00

* JOB SIZE *	* NO. OF JOBS SERVED *	* AV. WAITING TIME *	* WT. AW. WT. TIME *	* Analytical Results *
* 1 *	* 104 *	* 1.2086580 *	* 1.2086588 *	* 1.144737 *
* 2 *	* 114 *	* 2.5367341 *	* 1.2683670 *	* 1.332451 *
* 3 *	* 97 *	* 4.0912477 *	* 1.3637492 *	* 1.281756 *
* 4 *	* 113 *	* 5.5119171 *	* 1.3779793 *	* 1.245020 *
* 5 *	* 96 *	* 6.4728843 *	* 1.2945769 *	* 1.310693 *
* 6 *	* 115 *	* 8.0061391 *	* 1.3343566 *	* 1.366263 *
* 7 *	* 122 *	* 9.5013115 *	* 1.3573332 *	* 1.421583 *
* 8 *	* 101 *	* 12.0374824 *	* 1.5046853 *	* 1.465724 *
* 9 *	* 85 *	* 13.9497953 *	* 1.5499773 *	* 1.495886 *
* 10 *	* 99 *	* 16.6654566 *	* 1.6665452 *	* 1.510014 *

Throughput := 0.718010

ANALYTICAL THROUGHPUT : 0.736695

Processor Utilization := 0.275000

TIME UNIT = 1 QUANTUM

TABLE 4.4.2

SIMULATION RESULTS FOR A MODIFIED FEEDBACK QUEUE

InterArrival rate exponential with parameter := 0.050

Service rate discrete Uniform with parameter := 10

Simulation time:= 20000.00

Job Greater than or Equal to 6Q"s is a Long job

JOB SIZE	NO. OF JOBS SERVED	AV. WAITING TIME	WT. AV. WT. TIME	Analytical Results
1	104	1.41186514	1.4186584	1.525641
2	114	2.7419972	1.3709986	1.300107
3	97	4.07725800	1.3375293	1.240281
4	113	4.9358984	1.2464749	1.217313
5	96	5.8458009	1.6191602	1.202616
6	115	7.0626612	1.1771103	1.130607
7	122	9.5013115	1.3573302	1.421583
8	101	12.0574824	1.5046853	1.465724
9	85	13.9467953	1.5499773	1.495886
10	99	16.6654516	1.6665452	1.510014

Throughput := 0.724718

Processor Utilization:= 0.275000

ANALYTICAL THROUGHPUT : 0.740203

TIME UNIT = 1 QUANTUM

TABLE : 4.4.3
 SIMULATION RESULTS FOR A USUAL FEEDBACK QUEUE
 InterArrival rate exponential with parameter Lamda := 0.100
 Service rate discrete Uniform with parameter N := 5
 Simulation time:= 20000.00

* JOB SIZE *	* NO. OF JOBS SERVED *	* AV. WAITING TIME *	* WT. AN. WT. TIME *	* Analytical Results *
* 1 *	* 412 *	* 1.1816342 *	* 1.1816342 *	* 1.166667 *
* 2 *	* 401 *	* 2.0678938 *	* 1.3039469 *	* 1.271382 *
* 3 *	* 396 *	* 4.2077286 *	* 1.4025762 *	* 1.333547 *
* 4 *	* 417 *	* 5.6716648 *	* 1.4179162 *	* 1.372622 *
* 5 *	* 381 *	* 7.7505437 *	* 1.5501037 *	* 1.529365 *

Throughput := 0.729269

ANALYTICAL THROUGHPUT : 0.74925

Processor Utilization := 0.300000

TIME UNIT = 1 QUANTUM

TABLE 4.4.4

SIMULATION RESULTS FOR A MODIFIED FEEDBACK QUEUE
 InterArrival rate exponential with parameter Lamda =0.100
 Service rate discrete Uniform with parameter N := 5
 Simulation time:= 20000.00
 Job Greater than or Equal to 3Q's is a Long job

* JOB SIZE *	* NO. OF JOBS SERVED *	* AV. WAITING TIME *	* WT. AW. WT. TIME *	* Analytical Results *
* 1 *	* 412 *	* 1.3329934 *	* 1.3329934 *	* 1.343750 *
* 2 *	* 401 *	* 2.5707866 *	* 1.2853933 *	* 1.214760 *
* 3 *	* 396 *	* 3.5201529 *	* 1.1733843 *	* 1.151930 *
* 4 *	* 417 *	* 5.6716650 *	* 1.4179162 *	* 1.372622 *
* 5 *	* 381 *	* 7.7505437 *	* 1.5501087 *	* 1.529365 *

ANALYTICAL THROUGHPUT : 0.756152

Throughput := 0.739667

Processor Utilization := 0.300600

TIME UNIT = 1 QUANTUM

TABLE 4.4.6

SIMULATION RESULTS FOR A MODIFIED FEEDBACK QUEUE

InterArrival rate exponential with parameter Lamda =0.200

Service rate discrete Uniform with parameter := 5

Simulation time:= 20000.00

Job Greater than or Equal to 30"s is a Long job

* JOB SIZE *	* NO. OF JOBS SERVED *	* AV. WAITING TIME *	* WT. AW. WT. TIME *	* Analytical Results *
* 1 *	* 818 *	* 1.6888487 *	* 1.6888487 *	* 1.717391 *
* 2 *	* 762 *	* 2.8911768 *	* 1.4455884 *	* 1.475790 *
* 3 *	* 785 *	* 4.4279088 *	* 1.4759686 *	* 1.421909 *
* 4 *	* 813 *	* 9.8127822 *	* 2.4531966 *	* 2.488636 *
* 5 *	* 765 *	* 16.4495640 *	* 3.2899123 *	* 3.268182 *

Throughput := 0.483595

Processor Utilization := 0.600000

TIME UNIT = 1 QUANTUM

ANALYTICAL THROUGHPUT : 0.4820713

TABLE 4.4.5

*SIMULATION RESULTS FOR A USUAL FEEDBACK QUEUE
 InterArrival rate exponential with parameter Lamda =0.200
 Service rate discrete Uniform with parameter := 5
 simulation time:= 20000.00

* JOB SIZE *	* NO. OF JOBS SERVED *	* AV.WAITING TIME *	* WT.AW.WT.TIME *	* Analytical Results *
* 1 *	* 818 *	* .3965748 *	* 1.3965748 *	* 1.375000 *
* 2 *	* 762 *	* .0254551 *	* 1.5127275 *	* 1.574219 *
* 3 *	* 785 *	* .2879567 *	* 2.0959856 *	* 2.076122 *
* 4 *	* 813 *	* 9.8127872 *	* 2.4531968 *	* 2.488636 *
* 5 *	* 765 *	* 16.4495646 *	* 3.2899128 *	* 3.268182 *

ANALYTICAL THROUGHPUT : 0.463729

Throughput := 0.465186
 Processor Utilization:= 0.600000
 TIME UNIT = 1 QUANTUM

Table below give system throughput values obtained from usual and modified feed back queue simulation for different arrival rates λ and service distribution parameter N.

Simulation Time : 20000 Quantums

Table 4.4.7

$\lambda = 0.03$

N	Usual Feed Back	Modified Feed Back	Value of L
10	0.818270	0.832019	6
20	0.631018	0.661047	16
30	0.474836	0.496152	20
40	0.284554	0.307129	30

Table 4.4.8

$\lambda = 0.05$

N	Usual Feed Back	Modified Feed Back	Value of L
5	0.844285	0.854284	3
10	0.718070	0.724718	6
15	0.522434	0.545788	8
20	0.371738	0.396979	16
25	0.207080	0.222511	16
30	0.101218	0.123743	20
35	0.061388	0.08778	25

Table 4.4.9

$\lambda = 0.1$

N	Usual Feed Back	Modified Feed Back	Value of L
5	0.729269	0.739667	3
10	0.360124	0.376124	
15	0.104368	0.108154	

Table 4.4.10

Table shows system throughput values obtained from modified feed back simulation at different values of L for fixed arrival rate and service distribution parameter

		L	N = 20
0.03		8	0.651763
		10	0.654420
		12	0.659232
		14	0.649021
		16	0.660147
0.05		8	0.385207
		10	0.386710
		12	0.385873
		14	0.386786
		16	0.396979

Chapter - V

INFERENCE AND CONCLUSION

Simulation results clearly show an improvement in system throughput, when prior information about the execution times of the job is utilised in feed back algorithm. A careful scrutiny of the simulation results clearly reveal that change in the average waiting time has taken place, only for the job whose execution time is less than or equal to L quanta, whereas the jobs requiring more than L quanta of run time have their average waiting time in the system same in both usual feed back and modified feed back scheduling algorithms. This fact is also supported by analytical results. (Table 4.4.1 - 4.4.6).

The results also show that there is no improvement in the average waiting time for very small jobs. In fact, jobs with very small run-time requirements have more waiting time in modified feed back algorithm compared to what it was in case of usual feed back algorithm. But, as the run-time requirements of the job increase, significant improvement in waiting time starts taking place. This improvement continues till the job size reaches L quanta, after which the average waiting time become same in both the algorithms. In order to justify this behaviour, let us try to find out, where exactly a short job gains or losses time in modified feed back algorithm compared to usual feed back scheduling algorithm. The expected time to finish the quantum in

progress $[E'(S_r)]$ is more in modified feedback scheduling algorithm in comparison to usual feed back algorithm. This is because, when the processor is serving a job at the first queue level in the Phase II of the modified feed back system, it is allocated L quanta. Hence, a new arrival may have to wait for as may as L quanta before it becomes a candidate for getting the processor, whereas in case of usual feed back algorithm the new arrival will have to wait at most for one quantum before it becomes a candidate for getting the processor. This is where small jobs loose time in modified feed back algorithm. The gain achieved in the modified feed back queue is due to the sequencing of jobs in such a way that long jobs join the queue only in the Phase II of the system and hence do not interfere with short jobs in Phase I. If not having a long job before it saves k quanta for a job of run time requirement k quanta. Hence, gain is more as k increase and this improvement in gain continues till $k=L$, after which both the systems become equivalent. This explains why as the job size increases significant improvement starts taking place.

Another important observation from the results is that choice of L i.e., the criterion of classifying the jobs into two categories (short job and long job) is also important (Table 4.4.10) For Example, when $\lambda = 0.05$ is the arrival rate and $N = 20$ we find that best throughput is obtained when $L = 16$. For our analysis, the choice of L is arbitrary.

Thus we conclude by saying that under the circumstances where it is exactly known to which category the job belongs, the modification suggested in the feedback algorithm gives better results.

BIBLIOGRAPHY

1. BRINCH HANSEN, P. "Short term scheduling in Multiprogramming Systems", Third ACM Symposium on Operating System Principles, Stanford University, Oct 1971, pp. 103-105.
2. DEITEL, H.M. An Introduction to Operating System, Ed.1, Reading: Addison - Wesley, 1984
3. DENNING, P.J. and COFFMAN, E. G. Operating Systems Theory, Englewood Cliffs, Prentice Hall, 1973.
4. DONOVAN, John J and MADNICK, STUART, E. Operating Systems, McGraw Hill, 1981.
5. GOON, A.M., GUPTA, M.K. and DASGUPTA, B. Outline of Statistic, Vol. 2, Calcutta, The World Press Private Limited, 1980.
6. KENNEDY, W.J., Jr. and GENTLE, J.E. Statistical Computing Marcel Dekker, New York, 1980.
7. KLIENROCK, L and NILSSON, A. 'On Optimal Scheduling Algorithms for Time shared Systems' JACM, Vol. 28, No. 3, July 1981, pp.456-477.
8. LUCAS, H. Performance Evaluation and Monitoring, ACM computing surveys, Vol. 3, No. 3, Sept. 1971, pp.79-91.
9. TENENBAUN, M AARON and AUGENSTEIN, MOSHE J. Data structures using Paxal. Prentice Hall, Englewood Cliffs, 1981.

