CHAPTER 1: **INTRODUCTION**

Why do performance analysis? There are two basic reasons:

[1] To improve productivity: more work is accomplished in the same amount of time.

[2] To add functionality: totally new functions will be performed that offer the potential for new productivity gains or new revenue.

Productivity has two components:

[1] Comparisons of different configurations of an existing system doing an established set of functions. Here are some illustrative examples:

- A file system is stored on a disk. Where should the files physically be located to provide acceptable access times for typical usage?
- An office contains three groups. Should there be one secretary per group, or a pool of three secretaries for all the groups?

[2] Changing the configuration of an existing system doing an established set of functions to improve performance. Here are some examples to illustrate this point:

- A computer system for program development is configured with one megabyte of memory. What information is needed to determine how much memory is really needed?
- A file system for billing inquiries is stored on two disk spindles. A single controller governs access to the two spindles. What information is needed to determine if a third disk spindle is needed? What about adding another controller?
- An application program that generates management reports reads one file repeatedly. What information is needed to determine if this file should be replicated and stored on more than one disk spindle, to improve performance?
- A packet switch has four communication links. Should there be a pool of buffers for all messages on all links, or should buffers be dedicated to different types of packets (e.g., control packets versus data packets, or so many buffers per line)
- A telephone switching system is controlled by a single processor. What information is needed to determine if this processor should be replaced by a single faster processor, or two processors of the same speed, in order to handle fifty per cent more telephone calls per hour?

Here are some examples of added functionality:

- An office already owns a computer controlled word processing system. What information is needed to determine if this system can support electronic mail?
- A bank wishes to install a network of automatic teller machines. What information is needed to determine what type of computer will be needed to handle this work?
- A CATV franchise wishes to supply voice communications over TV channels not used for entertainment. What type of computer system will be needed to handle billing for this purpose?

Coupled with any performance analysis must be an economic analysis to quantify benefits and costs. All too often costs are ignored altogether, for a variety of reasons. If costs are ignored, then, *de facto,* costs are assumed negligible. However, *costs are lost benefits.* If costs are studied, then the outcome must be worse than before the study, because now costs are **not** negligible. Costs quantify the benefits of different alternatives, which is one aspect of decision making.

Remember, issues of computer performance evaluation cannot be answered absolutely. They must be addressed relative to other factors, such as economic issues, political considerations, and many more. The aspects of performance dealt with here are still only a subset of all the factors that must be considered in evaluating a total system to determine if it is suitable for a given application.

Although we focus on performance analysis here, we can be more specific, and list some of these other considerations:

- Economic: What is the cost to the user and the provider for a set of services?
- Marketing: What are the long term strategies or goals? What products attempt to meet those goals?
- Service: What service must be provided by the vendor, and by the purchaser? At what cost?
- Human engineering and psychological: What do human beings see and hear and feel when they interact with the system?
- Logical correctness and functionality: What functions should the system perform? How do we know these functions are being correctly executed?
- Systems engineering: Should products be leased or purchased? From what vendors? What hardware configuration is cost effective? What software configuration?

1.1 An Example

The figure below shows a representative hardware block diagram of a computer system.

Figure 1.1.Computer System Hardware Block Diagram

Operators at terminals spend some time reading, thinking and entering commands. The commands are fed into the computer system, and executed, and the cycle repeats itself. In order to say *anything* at all quantitative concerning performance, we must describe *how* the system works. The simplest script for describing this would be to have only one type of job done by every operator. The script that each operator would follow might be

- Read, think and enter a command
- Wait for the response to the command

The process then repeats itself, again and again and again. This is summarized in the flow chart below. We measure the average time to do each step. There are two entities, an operator (at a terminal) and a computer system. The table below is a summary of what entities are needed at each step, and the average time per step:

Figure 1.2.Operator Interaction With Computer System Flow Chart

Table 1.1.Steps, Entities and Mean Time per Step

Step	*Operator*	*Computer*	*Time*
Read,Think,Type	1	0	T_{think}
Wait for Response	0	1	T_{system}

The times that are tabulated are measured with *no* contention, i.e., with only one operator at one terminal using the computer system. We will use the *no* contention measurements to extrapolate to system behavior *with* contention. But before doing so, we must take a diversion.

1.2 What Is a Resource?

The entities in the example above are concrete examples of what we call *resources.* In order to understand the performance of a computer or communication system, we must understand what *state* the system is in at any instant of time. It is often useful to summarize the system state by what jobs are holding what resources at each step of execution. What is a *resource?* **A resource is anything that can block subsequent execution of a job.** A job requires one or more resources at each stage of execution, and if it does not have them all, it is blocked from execution until the necessary resources are available. What are the resources in the previous example?

- Each operator is a resource
- Each terminal is a resource
- Each command is executed on a computer system, so the computer system is a resource

We could aggregate this set of resources in different ways. For example, in order for an operator to interact with the system, an operator needs a terminal, so we might combine these two resources and talk about operators and terminals interchangeably. We could disaggregate this set of resources in different ways. For example, we might look inside the computer system, and find a processor, memory, a disk controller handling one or more disk spindles, and so on, as well as operating system software and application programs, each of which might usefully be thought of as resources. Whatever description is most useful in a given situation will determine what level of aggregation and disaggregation is needed.

The *state* of the system at any instant of time is given by the number of operators at terminals actively reading, thinking, and typing, and hence holding a terminal resource, and the number of jobs inside the system in execution or waiting to be executed. More formally, $J_{operator}$ denotes the number of operators reading, thinking and typing, while J_{system} denotes the number of jobs inside the system. Each of these can take on integer values ranging from zero to N, the total number of terminals with operators, but there is a constraint that every job must be *somewhere,* either with an operator or inside the system, and hence

$$J_{operator} + J_{system} = N$$

The state space Ω is given by all ordered pairs $(J_{operator}, J_{system})$ such that the total number of jobs equals the total number of active operators:

$$\Omega = \{(J_{operator}, J_{system}) \mid J_{operator}, J_{system} = 0,1,...,N; J_{operator} + J_{system} = N\}$$

A useful picture of the system state is a *queueing network* block diagram. For our earlier example of terminals connected to a single system this is shown in the figure below:

Figure 1.3.Queueing Network Block Diagram

We see two queues, represented by a box with an open left end. Resources are shown by circles or servers: each terminal is represented by a circle, while the system is represented by one circle. Jobs circulate from the terminals into the system and back again.

Both physical and logical resources are required at each stage of execution of computer programs, and we must know these in order to say anything about performance. Resources can be

- Serially reusable--one job at a time can use a processor, a data link, a given secondary storage device, a protected file, a critical region of operating system code
- Shared concurrently--multiple buffers in an operating system allow different jobs to share the buffer resource concurrently
- Consumable--messages between software modules exemplify this resource type

In our previous example, the system is serially reusable, while the terminal resource can be thought of as shared concurrently: there is a pool of terminals, and we are not distinguishing anything about the work going on at each terminal (if we did, then each terminal would be a serially reusable resource). Because the notion of resource is essential to performance analysis, we will motivate this basic concept with additional examples in later sections.

1.3 Resource Allocation Policy

The choice of *policy* or *schedule* for allocation of resources is central to performance evaluation. We will focus on a variety of policies to allocate resources throughout this book, because the central issue in modern digital system design is the ability to allocate resource as demands change with time in a cost effective and flexible manner. Any scheduler will execute the highest priority job in the system at any one time. An illustrative flow chart of a scheduler is shown on page seven.

The details of *how* to set the *priorities* will concern us throughout the rest of this book. Again, we only have *finite* resources, due to economic concerns, that must be allocated amongst competing tasks in order to meet diverse and perhaps conflicting goals.

Figure 1.4.Scheduler Flow Chart

1.4 Likelihoods and Statistics

One problem with characterizing performance of computer communication systems is their *intrinsic* complexity. Even if we could know the state of every logic gate, which could be billions upon billions of states, what would we do with the information: it is simply too difficult to track and comprehend. On the one hand, for logically correct operation, we are interested in knowing *all* the states: is the system in a working state, or is it in an unacceptable state? On the other hand, for performance, we are not always interested in *all* the states, but only the most *likely* that the system would be operating in. Statistics are a natural way of summarizing this type of complexity. Statistics allow us to quickly draw cause and effect inferences. Computer communications systems do not operate in a *random* manner, which is one aspect of statistics. Our purpose in using statistics is to study the most *likely* behavior, averaged over a suitably long time interval. Finally, how long is long? This is a relative notion: we are assuming that the measurements and statistics we gather stabilize, and are well characterized by an average value or mean. Transients will be ignored until later.

1.5 Performance Measures

Our intent is to focus on some of the many facets of what is called *performance* of a computer or communication system. This had its origin in voice telephony, determining how many voice telephones could be connected to a switching system to provide acceptable service. The demands placed on such systems can generate *traffic* or *congestion* for different resources. We will refer to performance interchangeably with *traffic handling characteristics* of such systems from this point on.

Traffic handling measures are either quantitative or qualitative. Two types of measures arise: those oriented toward *customers* or *users* of the system, and those oriented toward the *system* as a whole. Each of these measures has its' own costs.

1.5.1 Customer Oriented Goals, Inputs and Outputs >From the point of view of a customer or user of a system, we might be interested in the time delay (both execution and waiting) for each step of each job: from arrival, to the initial wait for service, through service, and the final clean up prior to completion. Service might involve some work to set up a job (e.g., handle an appropriate interrupt of a processor), followed by execution of the job (e.g., processing followed by input/output to secondary storage followed by processing and so forth), followed by clean up (e.g., handle an appropriate interrupt of a processor). The time epochs of interest are the arrival or ready time of a job, the job completion time, the desired completion time or due time or window or deadline.

>From these statistics we could compute for each job its

- queueing or flow time--time from arrival until completion
- waiting or ready time--time to start processing less arrival time
- lateness--actual completion time minus desired completion time
- tardiness--maximum of zero and lateness

Can you think of any more? These are summarized in the figure below:

Figure 1.5.Illustrative Time Epochs for Processing a Job

For each measure or time interval, there would be a cost attached, which would reflect economic criteria. We could compute these for each step of each job submitted by each user. This can be quite complex, and instead we might compute a statistic, such as the *average* cost, averaged over all jobs by all users over time.

1.5.2 System Oriented Goals, Inputs and Outputs >From the point of view of the system as a whole, we might wish to record over a given time interval

- The fraction of time each resource is busy doing work
- The mean throughput rate of executing jobs: the mean number of jobs executed in a given time interval
- The fraction of time two resources are simultaneously busy (in order to see how many steps are being executed concurrently)

In our previous example, we would be interested in

- The fraction of time each operator is reading, thinking and typing
- The fraction of time the system is busy executing work
- The mean throughput rate of executing jobs
- The fraction of time at least two operators are busy
- The fraction of time the system plus at least one operator is busy

For each resource, we could attach a cost for being idle, and compute the cost per resource for each user, or compute a statistic such as the average cost due to a resource being idle, averaged over all jobs and all users and over time.

1.6 An Example (Revisited)

Let’s calculate as many performance measures as possible for our previous example: operators at terminals only doing one type of job, reading and thinking and typing followed by waiting for a response for the system. We will compute these *exactly* for one operator using the system, and then use this to *bound* performance with more than one operator, which is really the region of interest.

1.6.1 One Operator What is the response time for one operator? This is given by

$$\textit{mean response time for one terminal} = T_{response} = T_{system}$$

The rate at which work is executed by the system is the reciprocal of the mean time to completely cycle through a job. There are only two steps, and hence the mean throughput rate is given by

$$\textit{mean throughput rate for one terminal} = \frac{1}{T_{think} + T_{system}}$$

For example, if the mean time spent reading, thinking, and typing is fifteen seconds, while the mean response time is one second, then we see

$$\textit{mean response time for one terminal} = T_{system} = 1 \textit{ second}$$

$$\textit{mean throughput rate for one terminal} = \frac{1}{T_{think} + T_{system}}$$

$$= \frac{1}{15 \text{ sec} + 1 \text{ sec}} = 1 \textit{ command every } 16 \textit{ seconds}$$

What is the utilization of the operator, the fraction of time an operator is busy?

$$\textit{operator utilization} = \frac{T_{think}}{T_{think} + T_{system}} = \frac{15 \text{ sec}}{15 \text{ sec} + 1 \text{ sec}} = \frac{15}{16}$$

What is the utilization of the system, the fraction of time the system is busy?

$$\textit{system utilization} = \frac{T_{system}}{T_{think} + T_{system}} = \frac{1 \text{ sec}}{15 \text{ sec} + 1 \text{ sec}} = \frac{1}{15}$$

What is the fraction of time both the operator and the system are simultaneously busy? *Zero!*

1.6.2 Two Operators If two operators were using this system for command execution, what changes? The best the mean response time could ever be would be as if each operator were using the system separately. Hence, we see a *lower* bound on mean response time given by

$$\textit{mean response time with two terminals} \geq T_{system} = 1 \text{ sec}$$

The worst the mean response time could ever be would be to have one operator submit a job and immediately thereafter have the other job submitted, which gives us a *upper* bound on mean response time:

$$\textit{mean response time with two terminals} \leq 2T_{system} = 2 \text{ sec}$$

If we summarize all this, we find

$$T_{system} \leq T_{response} \leq 2T_{system} \quad \textit{two terminals}$$

The definition of mean throughput rate is simply the rate at which each operator is doing work, multiplied by the total number of operators. Each operator spends some time reading, thinking, and typing, and then waits for a response, so the mean throughput rate for each operator is

$$\textit{mean throughput rate per operator} = \frac{1}{T_{think} + T_{response}}$$

The upper bound on mean response time gives us a lower bound on mean throughput rate:

$$\textit{mean throughput rate per operator} \geq \frac{1}{T_{think} + 2T_{system}}$$

The lower bound on mean response time gives us an upper bound on mean throughput rate:

$$\textit{mean throughput rate per operator} \leq \frac{1}{T_{think} + T_{system}}$$

For the total system, with two operators we have twice the mean throughput rate of one operator:

$$\lambda_{lower} \leq \textit{total system mean throughput rate} \leq \lambda_{upper}$$

$$\lambda_{lower} = \frac{2}{T_{think} + 2T_{system}} = 2 \textit{ commands every } 17 \textit{ seconds}$$

$$\lambda_{upper} = \frac{2}{T_{think} + T_{system}} = 2 \textit{ commands every } 16 \textit{ seconds}$$

What is the utilization of each operator? The fraction of time each operator is busy is the mean rate each operator does work multiplied by the mean time spent reading, thinking and typing:

$$\textit{operator utilization} = \textit{mean throughput rate per operator} \times T_{think}$$

Since we have upper and lower bounds on mean throughput rate, we have upper and lower bounds on operator utilization. What is the system utilization?

$$\textit{system utilization} = \textit{total mean throughput rate} \times T_{response}$$

Since we have upper and lower bounds on mean throughput rate and response time, we have upper and lower bounds on system utilization.

What about bounding other measures of concurrency?

- The fraction of time both operators are busy reading, thinking and typing and the system is idle
- The fraction of time one operator is reading, thinking and typing and the other operator is waiting for the system to respond
- The fraction of time both operators are waiting for the system to respond

EXERCISE: Explicitly calculate bounds on each of these measures.

1.6.3 N>2 Operators What happens if we increase the number of operators more and more? At some point the system will be completely busy doing work, and this will occur at

$$\textit{total mean throughput rate} = \frac{1}{T_{system}} = 1 \textit{ command every second} \quad N \rightarrow \infty$$

This is a *different* upper bound from the upper bound on mean throughput rate we just found. Our first upper bound was due to *how* the system was used by operators, while this upper bound is *intrinsic* to the system. One way to measure this is to *always* have a command ready to be executed. Once a command finishes execution, another command is immediately started in execution. Every system will have a maximum rate at which it can execute jobs, and this is a *key* performance measure. We combine both upper bounds in one expression:

$$\textit{mean throughput rate} \leq \lambda_{upper} = \min \left[\frac{1}{T_{system}}, \frac{N}{T_{think} + T_{response}} \right]$$

As we vary the number of active operators or terminals, N=1,2.., the upper bound on mean throughput rate exhibits a *breakpoint* (denoted $N_{breakpoint}$ terminals or users) as a function of N, moving from an *operator limited* regime to a *system limited* regime. The breakpoint marks that value of N for which the two upper bounds equal one another:

$$\frac{N_{breakpoint}}{T_{think} + T_{system}} = \frac{1}{T_{system}} \quad N_{breakpoint} = \frac{T_{think} + T_{system}}{T_{system}}$$

Below this breakpoint, the terminals and operators cannot generate enough work to drive the system to continuously execute commands. Above this breakpoint, the system is continuously busy, and the operators are experiencing delays. This suggests the notion of a *bottleneck:* a resource that is completely busy. The terminals and operators are a bottleneck, provided the number is less than the breakpoint, and otherwise the system is a bottleneck. This breakpoint is a different measure of *concurrency.*

The mean response time is related to the mean throughput rate via the *definition* of mean throughput rate. A job is either with an operator (who is thinking) or inside the system (being executed):

$$total\ mean\ throughput\ rate = \frac{N}{T_{think} + T_{response}}$$

If we solve for the mean response time (which is a function of the number of terminals, among other things), we find

$$T_{response} = \frac{N}{total\ mean\ throughput\ rate} - T_{think}$$

On the one hand, the worst the mean response time could be is to be the last job to be done, after all the other operators submit jobs:

$$T_{response} \leq NT_{system}$$

This gives us a *lower* bound on mean throughput rate:

$$total\ mean\ throughput\ rate \geq \frac{N}{T_{think} + NT_{system}}$$

On the other hand, the best the mean response time could be is to be equal to the response time for one terminal:

$$T_{response} \geq T_{system}$$

Since the mean throughput rate can be upper bounded by two different expressions, we have two possible *lower* bounds on mean response time, and we will choose the larger of the two:

$$T_{response} \geq \max \left[T_{system}, NT_{system} - T_{think} \right]$$

These upper and lower bounds on mean throughput rate are plotted in the figure below versus number of active operators. Figure 1.7 plots upper and lower bounds on mean response time versus number of active operators.

1.6.4 Stretching Factor Ideally, as we increase the number of terminals, the mean throughput rate increases, while the mean response time stays low. Unfortunately, at some point the mean response time will grow, and the mean throughput rate will saturate. This suggests writing the mean throughput rate as the product of two factors, one that is a dimensionless *stretching factor* and one that is the ideal mean throughput rate, N times the mean throughput rate for one terminal:

$$mean\ throughput\ rate = \frac{N}{T_{think} + T_{response}}$$

$$= stretching\ factor \times \frac{mean\ throughput\ rate\ \text{for}\ N>1}{N\ mean\ throughput\ \text{for}\ N=1}$$

By definition, we see

$$stretching\ factor = \frac{N\ mean\ throughput\ rate\ \text{for}\ N=1}{mean\ throughput\ rate\ \text{for}\ N\geq 1}$$

The upper and lower bounds on mean throughput rate for the system give us upper and lower bounds on the stretching factor:

Figure 1.6.Upper and Lower Bounds on Mean Throughput Rate

Figure 1.7.Upper and Lower Bounds on Mean Response Time

$$stretching\ factor \geq \frac{NT_{system}}{T_{think} + NT_{system}}$$

$$stretching\ factor \leq \begin{cases} 1 & N \leq N_{breakpoint} \\ \dfrac{NT_{system}}{T_{think} + T_{system}} & N \geq N_{breakpoint} \end{cases}$$

The upper bound on stretching factor grows *linearly* with the number of active terminals, i.e., it stretches proportional to N. Ideally, we want *no* stretching, i.e., a stretching factor of unity.

1.6.5 Additional Reading

[1] E.Arthurs, B.W.Stuck, *Upper and Lower Bounds on Mean Throughput Rate and Mean Delay in Queueing Networks with Memory Constraints,* Bell System Technical Journal, **62** (2), 541-581 (1983).

[2] H.Hellerman, T.F.Conroy, **Computer System Performance,** McGraw Hill, NY, 1975.

1.7 The Two Fundamental Approaches to Achieving Desirable Performance

The previous example shows the following: there are only two *fundamental* approaches to achieve a desired level of performance in a computer communication system.

1.7.1 Execution Time Speeding up one or more steps of a job, or reducing the execution time of one or more steps, is one way to impact performance. Recoding an application program, reducing the amount of disk input/output or terminal character input/output are examples of application program speed changes. Replacing a processor with a faster processor, or a data link with a faster data link are examples of hardware speed changes.

1.7.2 Concurrency and Parallelism At a given instant of time, there are a given number of jobs in the system that are partially executed, which are said to be *concurrently* in execution. In particular, a certain number of the jobs concurrently in execution can in fact be *actively* in execution, and these jobs are said to be executing *in parallel.* The greater the number of jobs *actively* in execution, the greater the impact on performance. This will be the central theme for the remainder of the book. Note that the execution time of each step of a job need *not* be reduced, so the total time to complete a job is the same, but the *total* number of jobs per unit time that can be executed can be increased. A single processor handling multiple terminals doing program development is an example of *logically* concurrent execution of multiple jobs: at any instant of time, only one job is running on the processor, but there can be many jobs partially executed in the system. An operating system that can support multiple disk spindles off multiple controllers doing independent *parallel* data transfers is an example of *parallelism* or *physical* simultaneous execution of multiple jobs.

1.8 Layered Models of Computer and Communication Systems

Layers accomplish two things:

- They allow us to deal with a subsystem internals in one way, and design this to meet a set of goals
- They allow us to deal with a subsystem externals (for handling input and output to other subsystems) in a different way from the internals and design this to meet a different set of goals

What layers and resources lie inside the box labeled *system?* One way to answer this is hierarchically, as a series of *layers* that are peeled away, with resources residing in each layer. Figure 1.8 shows an illustrative layered model of the computer system described earlier:

Figure 1.8.Layered Model of Computer System

In this model we see subsystems labeled

- Hardware--Terminals, processor, memory, input/output handlers, disk controller, disks, tape drives, printers and so forth
- Operating system--A set of programs that control the hardware and make use of it (i.e., the operating system programs reside in secondary disk storage and in main memory and require processor time to

run, and so forth)

- Application programs--A set of programs that make use of operating system functions to accomplish a specific application

What are the resources here? Examples might be

- Hardware resources--Processors, memory, disk controller, disk spindle, printer, input/output handler, terminal, cache, floating point accelerator, communications bus
- Operating system resources--Tables, files, events, signals, semaphores, messages, objects, processes, capabilities, pages, segments, partitions
- Application resources--Data base manager, application level transaction manager, communications manager, front end terminal manager, compilers, editors, report generators

Ultimately a resource involves using up *hardware,* but remember that *anything* that can block execution of a job is a resource. Many viewpoints of the same thing are often useful here.

1.9 Layers and Performance

Each layer is a candidate for performance analysis. First, we might wish to speed up each:

- Hardware might be too slow for certain jobs and can be replaced with faster hardware with little or no changes to the operating system and application software
- Operating system software might be too slow for certain jobs, and might be changed to speed up certain system calls that demand a lot of resources (either because they are called frequently or because they demand a lot or both)
- Application software might be too slow for certain jobs, and might be recoded to speed up certain areas, to take better advantage of the operating system and hardware

At present, application programs are typically done with serial flow of control, while operating system programs typically handle concurrent flow of control: to take better advantage of concurrency of the hardware and software resources, the application programs might be recoded to take better advantage of the operating system and hardware.

To improve performance, we might

- Keep the operating system and application software, and replace hardware with faster hardware
- Keep the operating system and hardware, and recode the application to take best advantage of the operating system with its hardware
- Keep the application software and hardware, and change the operating system (adding more file space, changing the scheduling algorithm, changing the buffering strategy)

We can aggregate and combine these different layers in talking about performance evaluation. At the hardware level, we might be interested in the maximum mean instruction execution rate, the memory access time per word, and so forth. At the operating system level we might be interested in system functions and time required to execute each on a given hardware configuration. At the application level, we might be interested in taking best advantage of operating system call implemented on a given hardware configuration.

We have fixed the evaluation interface and the subsystems underneath it in describing performance. This structures our data analysis and allows us to quickly draw cause and effect inferences about performance problems. There is also a need to evaluate each layer by itself, or to evaluate sets of layers together: this realm of testing and diagnostics is worthy of independent study by itself, and can be the subject of more than one book. Here we are suggesting a methodology for representing total system state for different purposes: many such state representations are fruitful.

1.10 An Application Program Development View

The figure below shows a flow chart of the types of activities involved in application program development.

Figure 1.9.Application Program Development Flowchart

First, a program is entered into a computer system using an interactive editor. Next, the program is parsed to see if the syntax matches the rules of the language. If errors are reported, the program is rewritten, and parsed again, until the program successfully passes this stage. Next, machine code is generated, and this may produce a variety of new error messages that were not checked by the parser. The program is rewritten, and now this may introduce parsing errors. If the program successfully is parsed and generates executable code, we may wish to make use of optimizing features of the compiler (to reduce the amount of storage and execution time). Again, this step may generate a variety of errors, and cause the program to be rewritten. Finally, we have a program that passes all the rules of parsing, code generation, and optimization. We still might have to rewrite this program, because although it does what we *say,* it might not do what we *mean.*

Figure 1.10 is a block diagram summarizing all of these actions.

Figure 1.10.Program Development Block Diagram

1.10.1 Steps and Resources We can construct a step resource table for each activity outlined above.

Table 1.2.Step/Resource Table

Step	*Programmer*	*Processor*	*Time*	
Type	*Required*	*Required*	*Symbol*	*Value*
Read, Think	1	0	T_{think}	15 sec
Edit Code	0	1	T_{edit}	1 sec
Parse Code	0	1	T_{parse}	1 sec
Generate Code	0	1	$T_{generate}$	30 sec
Optimize Code	0	1	$T_{optimize}$	30 sec

Illustrative values of time for each step are shown in the table. Note the great discrepancy between the time required to handle simple edits and parsing versus to handle code generation and optimization. To simplify our analysis, we will aggregate together the steps for parsing, code generation and optimization. Together we call these steps *compilation* with a total mean time of $T_{compile}$.

1.10.2 Resource Allocation We want to examine the consequences of different schedules or resource allocation policies about performance measures. We will study

- Executing jobs in the order of submission
- Round robin quantum scheduling

1.10.3 Execution in Order of Arrival There are two different job types, edits and compilations. We have a total of N terminals, so the following cases *might* occur:

- There are *no* jobs in the system, and so once a job is submitted it immediately runs to completion
- Every other terminal has submitted a job, and we are the last in line.

For the first case, the mean queueing time is simply T_{edit} or $T_{generate}$ or $T_{optimize}$, depending upon what type of job was submitted. For the second case, the mean queueing time is as small as NT_{edit} if all jobs are edits, or as big as $NT_{generate}$ if all jobs are compilations.

The variation in response time can be huge: if an edit command was last in line, and all the other commands were compilations, a relatively quickly executing command would have to wait for many very long commands, and this might be unacceptable!

1.10.4 Execution via Quantum Round Robin Scheduling This schedule executes a job for a maximum amount of time called a *quantum,* and either the job finishes or it goes to the end of the queue where it will get another shot, until eventually it finishes. This is shown in the queueing network block diagram below:

Figure 1.11.Round Robin Queueing Network Block Diagram

Let's look at two cases, as before:

- There are *no* other jobs in the system, and provided nothing more arrives, a job will run until it is executed
- All the other terminals have submitted commands, and we are last in line

For the first case, the mean response time is T_{edit} or $T_{compile}$, depending upon the command type. For the second case, caution is needed. If the quantum is set equal to T_{edit}, then if we have submitted an edit, it will complete at NT_{edit}, *even* if all the other commands were compilations; each of the other commands would only get one quantum of time. On the other hand, if the quantum is set to $T_{compile}$, then if we had an edit command and all the other commands were compilations we would wait for $(N-1)T_{compile} + T_{edit}$ for a response, much much greater than for our first choice of quantum.

1.10.5 Summary For this particular example, where we have a *huge* variation in the execution time of jobs, with short jobs being much more urgent than long jobs, and where we do *not* know whether a command will be short or long, quantum round robin scheduling with the quantum set at T_{edit} can offer short response to short commands, while long commands take a long time.

If we set the quantum to $T_{compile}$, then we are executing jobs in order of arrival, and encounter *huge* variations in response time.

Note that if all jobs took the *same* amount of time, then we would set the quantum equal to that amount, and execute jobs in order of arrival.

1.10.6 Additional Reading

[1] A.V.Aho, J.D.Ullman **Principles of Compiler Design,** Addison Wesley, Reading, Massachusetts, 1977.

[2] L.Kleinrock, **Queueing Systems, Volume II: Computer Applications,** Wiley, NY, 1976.

1.11 Local Area Network Interfaces

The hardware block diagram below shows one design for a *local area network interface* between a *local area network* and a *work station.* The interface resources are a local bus that interconnects the work station interface (called here the *bus interface unit* or *BIU)* to the network interface unit *(NIU),* a bus controller to determine which device gains access to the bus, plus a local processor with memory for buffering bursts of data (either from the work station to the network or vice versa).

Figure 1.12.Local Area Network Hardware Interface Block Diagram

The steps involved in transferring data from the work station out over the local area network are summarized in the figure below.

The steps involved in data transfer are summarized in the table below:

Table 1.3.Work Station/Local Network Data Transfer

Step	*Resources*					*Time*
Number	*Bus*	*Processor*	*Memory*	*NIU*	*BIU*	*Interval*
WS->BIU	1	0	0	0	1	$T_{WS \rightarrow BIU}$
BIU->MEM	1	0	1	0	1	$T_{BIU \rightarrow MEM}$
Process	1	1	1	0	0	$T_{process}$
MEM->NIU	1	0	1	1	0	$T_{MEM \rightarrow NIU}$

Figure 1.13.Work Station to Local Area Network Data Transfer

As is evident, more than one resource must be held at each step of execution of this type of job.

What is the maximum rate of executing network accesses?

- If the work station is completely busy transferring data to the bus interface unit, then

$$\textit{maximum mean throughput rate} = \frac{1}{T_{WS \rightarrow BIU}}$$

- If memory is completely busy, then

$$\textit{maximum mean throughput rate} = \frac{1}{T_{BIU \rightarrow MEM} + T_{MEM \rightarrow NIU}}$$

- If the processor is completely busy, then

$$\textit{maximum mean throughput rate} = \frac{1}{T_{processor}}$$

- If the network is completely busy, then

$$\textit{maximum mean throughput rate} = \frac{1}{T_{MEM \rightarrow NIU}}$$

The smallest of these, the one that becomes completely busy first, we call the *bottleneck* resource. In any system, there will be *some* bottleneck; the design question is to *choose* where it should be, while the analysis question is to *find* it.

1.12 A Communications System Example

A different type of layering model that is being widely used is the *Open Systems Interconnection (OSI)* architecture of the *International Standards Organization,* shown in the figure below.

In the figure, we show the seven layers of the model, plus an illustrative example of the flow of control and data to move a file from one system to another. The layers are as follows:

[1] Hardware level--Specifications of voltages, currents, media, waveforms, timing, and packaging.

[2] Frame or message level--Specifications of the meaning of each bit within a frame or packet of bits, signifying either control (source and destination address, priority, error detecting or correcting code) and data.

Figure 1.14.Open Systems Interconnection Architecture

[3] Link level--Multiple transmitters and receivers send frames to one another over a logical abstraction of a *physical* circuit called a *virtual* circuit.

[4] Transport level--Multiple transmitters and receivers send packets or frames to one another over one or more logical circuits.

[5] Session level--Controlling initiation, data transfer, and completion clean up of a communication session.

[6] Presentation level--Conversion of an application program protocol to a session level protocol.

[7] Application level--Description of application program behavior.

In the illustrative example, a file is transferred from one system to another as follows:

[1] An application program is accessed from a terminal to begin the process

[2] A chain of commands is retrieved from a local disk to control communication over a link

[3] Communications software controls initial access to the link and sends a request over the link to the appropriate application in the other system

[4] The desired file is retrieved from secondary storage

[5] The file is transferred from one application to another via the link

[6] The application program notifies the operator the file has been retrieved

[7] The application program stores the file in local disk space

At each step, in order to say *anything* concerning performance, we would need to specify what resources, software and hardware, are required for each step, for what time interval, along with a policy for arbitrating contention for shared resources. Questions concerning the physical location of text and data can only be answered by describing both the system operation and the staff or people operations.

What are the resources for this system?

[1] Hardware Level--Processor, memory, and bus time

[2] Frame or message level--Source and destination address tables

[3] Link level--Packet sequence numbers with associated priorities

[4] Transport level--Network name tables, virtual circuit tables

[5] Session level--Number of sessions outstanding

[6] Presentation level--Table of each protocol conversion in progress

[7] Application level--Table of number of in progress communications

To be explicit, we suppose that there are two different computer systems, labeled *A* and *B* each with their own processor and disk for handling this scenario. The table below is a summary of the steps and resources required for the file transfer shown earlier:

Table 1.4.Step/Resource Summary

Step	*CPU A*	*DISK A*	*LINK*	*CPU B*	*DISK B*	*Time*
Start	1	0	0	0	0	T_{start}
Load	1	1	0	0	0	T_{load}
Request	1	0	1	1	0	$T_{request}$
Retrieve	0	0	0	1	1	$T_{retrieve}$
Transmit	1	1	1	1	1	$T_{transmit}$
Notify	1	0	0	0	0	T_{notify}
Store	1	1	0	0	0	T_{store}

What is the maximum mean throughput rate, denoted by λ_{upper}, of doing file transfers?

- If CPU A is a bottleneck, then the maximum mean throughput rate is given by

$$\lambda_{upper} = \frac{1}{T_{start} + T_{load} + T_{request} + T_{transmit} + T_{notify} + T_{store}}$$

- If disk A is a bottleneck, then the maximum mean throughput rate is given by

$$\lambda_{upper} = \frac{1}{T_{load} + T_{transmit} + T_{store}}$$

- If the link is a bottleneck, then the maximum mean throughput rate is given by

$$\lambda_{upper} = \frac{1}{T_{request} + T_{transmit}}$$

- If CPU B is a bottleneck, then the maximum mean throughput rate is given by

$$\lambda_{upper} = \frac{1}{T_{request} + T_{retrieve} + T_{transmit}}$$

- If disk B is a bottleneck, then the maximum mean throughput rate is given by

$$\lambda_{upper} = \frac{1}{T_{retrieve} + T_{transmit}}$$

Communication link data rates are only *one* component in controlling the *maximum* mean throughput rate: the processors and disks at either end can also be important.

EXERCISE: What happens if we have only *one* processor and *one* disk?

1.12.1 Additional Reading

[1] A.Tanenbaum, **Computer Networks,** pp.15-21, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[2] F.D.Smith, C.H.West, *Technologies for Network Architecture and Implementation*, IBM J.Research and Development, **27** (1) 68-78 (1983).

1.13 Operating System Performance

An *operating system* is a set of *logically* concurrently executing programs. An operating system runs on a given hardware configuration and provides services to application programs.

1.13.1 Hardware Configuration We assume that the hardware configuration consists of one processor with a fixed amount of memory, a set of terminals that are connected to the processor via a terminal handler, and one disk. The disk is used for two purposes, to store programs and data that the programs access, and for holding programs that cannot be held in main memory, so called *swapped* programs.

Figure 1.15.Hardware Block Diagram

1.13.2 Application Programs The operating system hides the details of the hardware operations of the processor, disk input/output, terminal input/output, and memory administration or management, from the application programs, by presenting a set of interfaces called *system calls* that allow these programs to interact without knowing the details of the system call implementation. The operating system *schedules* or manages or administers the physical and logical resources, which is the crux of performance. The programs are *virtual* processors, and for short are called *processes*. Each process can be in the following states: ready to run except that the processor is busy with higher priority work, idle, executing or running on the processor, blocked on terminal input/output (waiting for a terminal to respond), blocked on disk input/output (waiting for a disk to respond), or blocked on memory (waiting for memory to become available). This is summarized in Figure 1.17.

Figure 1.16.States of a Process

The operating system, as it schedules each step of each job, would migrate or move each program or process around, from one state to another, in the course of execution. The figure below shows one set of processes that might be managed by the operating system: For each state of a process, we can list

Figure 1.17.Illustrative Set of Processes Managed by Operating System

the resources required:

Table 1.5.State/Resource Summary

State	*Processor*	*Memory*	*Disk*	*Terminal*	*Mean Time*
Idle	0	0	0	0	T_{idle}
Run	1	1	0	0	T_{cpu}
Blocked:					
On Disk I/O	0	1	1	0	T_{disk}
On Terminal I/O	0	1	0	1	T_{term}
On Memory	0	0	0	0	T_{memory}

Different resources are held at different steps. Each job migrates through a network of queues, as shown in Figure 1.18.

Figure 1.18.Operating System Network of Queues

The resources of memory and processor time are allocated sequentially: a process must first be loaded into memory, and holds this while waiting for the processor, or while waiting for input/output from the disk.

Processes that are resident in main memory are candidates for being swapped to secondary disk storage. Processes that are swapped out of main memory are candidates for being loaded into main memory.

The policy for determining which processes are resident in main memory and which are not, and for determining which processes are executed by the processor and by the disk, are *separate* from the logically correct execution of each step of each program.

The maximum mean throughput rate of executing processes can be limited by memory, disk input/output, terminal input/output, or processor time. Suppose we gathered measurements on the *total* time the system processes were in each state over a given time interval.

- If memory is the bottleneck, then the time spent in the memory blocked state would be the greatest
- If the disk is the bottleneck, then the time spent in the disk blocked state would be the greatest
- If terminal handling is the bottleneck, then the time spent in the terminal blocked state would be the greatest
- If the processor is the bottleneck, then the time spent in the processor running state would be the greatest

EXERCISE: Suppose an operating system table contains a list of all active processes. What data should be gathered to determine if this is limiting performance by simply being set too small?

1.13.3 Additional Reading

[1] A.L.Shaw, **Principles of Operating Systems,** Prentice-Hall, Englewood Cliffs, NJ, 1974

[2] D.M.Ritchie, K.Thompson, *The UNIX^{TM} * Time-Sharing System,* Communications of the ACM, **17** (7), 365-374 (1974).

[3] K.Thompson, *UNIX Implementation,* Bell System Technical Journal, **57** (6), 1931-1946 (1978).

[4] D.M.Ritchie, *UNIX Retrospective,* Bell System Technical Journal, **57** (6), 1947-1870 (1978).

1.14 Processor Design

Many actual processors are made up of networks of smaller processors, as shown in the figure below:

A processor might consist of an arithmetic logic unit, that has three types of memory connected to it:

- General register or so called *scratch pad* memory connected via a special dedicated bus
- Program counter processor connected via both a special dedicated bus to the processor and via a general purpose switch to cache memory
- Cache memory that is much higher speed memory than main memory for hiding recently used segments of text and data on the assumption that it is highly likely they will be executed in the immediate future

The resources here are the arithmetic logic unit, the program counter and general registers, and cache memory, with the switch and switch controller necessary to allow communications between these different devices.

The flow chart below illustrates how the processor would execute an instruction.

The steps and resources required are summarized in the table below:

* UNIX is a trademark of Bell Laboratories

Figure 1.19.Processor Block Diagram

Figure 1.20.Processor Instruction Execution Flow Chart

Table 1.6.Steps/Resource Summary

Step	*PC*	*ALU*	*GR*	*Cache*	*Switch*	*Time*
Load PC	1	0	0	0	1	T_{PC}
Fetch Instruction	0	1	1	0	1	T_{fetch}
Decode Instruction	1	1	1	1	0	T_{decode}
Fetch Data	0	1	1	1	1	T_{data}
Execute Instruction	1	1	1	1	1	$T_{execute}$
Store Results	0	1	1	1	1	T_{store}

Each of these resources can execute at a maximum rate. The total system mean throughput rate is upper bounded by λ_{upper}: As we raise the number of executing programs, one resource will become completely busy first, which will be called the *bottleneck.*

- If the program counter is a bottleneck, then the maximum mean throughput rate is

$$\lambda_{upper} = \frac{1}{T_{PC} + T_{decode} + T_{execute}}$$

- If the ALU is a bottleneck, then the maximum mean throughput rate is

$$\lambda_{upper} = \frac{1}{T_{fetch} + T_{decode} + T_{data} + T_{execute} + T_{store}}$$

- The maximum rate at which the general registers can load and unload instructions is

$$\lambda_{upper} = \frac{1}{T_{fetch} + T_{decode} + T_{operand} + T_{execute} + T_{store}}$$

- The maximum rate at which the switch can transfer instructions is

$$\lambda_{upper} = \frac{1}{T_{PC} + T_{fetch} + T_{data} + T_{execute} + T_{store}}$$

Based on this analysis, we see that the bottleneck is the *switch.* This is because the switch is held for the long time per instruction execution.

EXERCISE: What can be changed to move the bottleneck to the ALU?

1.14.1 Additional Reading

[1] G.Radin, *The 801 Minicomputer,* IBM J.Research and Development, **27** (3), 237-246 (1983)

[2] G.J.Myers, **Advances in Computer Architecture, Second Edition,** Wiley, NY, 1982.

[3] H.Lorin, **Introduction to Computer Architecture and Organization,** Wiley, NY, 1982.

1.15 Text Outline

The problems and chapters in the text are organized along the lines described in the preface.

1.15.1 Introductory Material First, we survey results from *deterministic* or *static* scheduling theory. All of the work that must be executed is present at an initial time; we wish to study different configurations and schedules to see how long it takes to finish all the work. The intent is to get used to systematically described computer communication system operation: for each step, what resources are required, for how long. This occurs in practice in telephone call processing, discrete manufacturing, and data communications: a list of work is scanned at the start of a clock cycle and all the work must be finished before the clock begins its next cycle.

Following this, data analysis and simulation is surveyed. This is a mandatory step in verifying self consistency and having credibility in performance analysis. In order to summarize data with a model, measurements must be gathered, and evidence presented showing how well the data fits the model. Many would argue the issue of data interpretation is the crux of performance analysis.

Next, we study the long term time averaged behavior of systems that execute jobs requiring *multiple resources* at each step of execution. An example would be a job that requires both a processor and a given amount of memory and a given set of files. Both the mean throughput and mean delay per job are studied for different hardware and software configurations and for different scheduling policies. The concluding section reinforces this analysis with a series of examples drawn from office communication systems: case studies, starting from models of simple everyday situations and building up to more realistic and complex situations.

1.15.2 Jackson Networks Next additional modeling assumptions or restrictions are made beyond just means or averages, which lead to so called *Jackson* queueing network models. These assumptions yield sharper performance bounds on computer communication systems than mean values. Case studies are studied, starting from the simple and proceeding to the more realistic and complicated. Each case is analyzed first via mean values alone, and then refined using Jackson network models.

1.15.3 Priority Scheduling In a computer communication system typically one resource is limiting the maximum mean throughput rate of completing work, and this is called the *bottleneck.* Here we study in detail how to effectively schedule work at the bottleneck resource in order to ameliorate delay, using priority arbitration. We will make much stronger assumptions about the system statistical behavior here than we did in earlier sections, and we will display much greater variety and accuracy (not just means but variances and even statistical distributions) in the phenomena we will deal with.

1.16 Additional Reading

[1] D.Ferrari, **Computer System Performance Evaluation,** Prentice-Hall, Englewood Cliffs, N.J., 1978.

[2] U.Grenander, R.F.Tsao, *Quantitative Methods for Evaluating Computer System Performance: A Review and Proposals,* in **Statistical Computer Performance Evaluation,** W.Freiberger (editor), Academic Press, NY, 1972.

[3] H.Lorin, **Parallelism in Hardware and Software,** pp.3-44, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[4] M.Phister, Jr., **Data Processing Technology and Economics, Second Edition,** Digital Press, 1979.

Problems

1) A data communications system consists of a single transmitter, a noisy link, and a single receiver. The transmitter sends ones and zeros (bits) over the link. Each bit is equally likely to be transmitted correctly a given fraction of time, $1-P$, and to be incorrectly transmitted a given fraction of time P, due to noise corrupting the transmission. The diagram below summarizes all this:

Figure 1.21.Bit Transmission Probabilities

Determine the data transmission rate, measured in successfully transmitted bits per second, of each of the following communications systems

A. A three hundred bit per second data link that requires retransmission of one bit in every ten thousand (P=0.0001) due to noise; repeat for retransmission of one bit in every one hundred due to noise

B. A ninety six hundred bit per second data link that requires retransmission of one bit in every ten thousand due to noise; repeat for retransmission of one bit in every one hundred due to noise

C. A courier delivery service that takes a twenty four hundred foot magnetic tape containing sixteen hundred bits per inch of tape and will deliver it five thousand miles away in twenty four hours; repeat if the tape contains six thousand two hundred and fifty bits per inch

D. A courier delivery service that takes a one hundred megabyte magnetic disk pack and will deliver it five thousand miles away in twenty four hours; repeat for a five hundred megabyte magnetic disk pack

2)* Imagine you have trained your St.Bernard, Bernie, to carry a box of three floppy disks, each containing a quarter of a million bytes. The dog can travel to your side, wherever you are, at eighteen kilometers per hour. For what range of distances does Bernie have a higher data rate than a twelve hundred bit per second noiseless data link? Repeat for the case where Bernie can carry a single ten megabyte disk pack.

* A.S.Tanenbaum, **Computer Networks,** p.30, Prentice-Hall, Englewood Cliffs, NJ, 1981.

3) A computer system consists of three identical processors, each processor being capable of executing a one second job in one second. The system must execute nine jobs, with the execution times for these jobs as follows: (3, 2, 2, 4, 4, 2, 9, 4, 4).

A. If the jobs are selected for execution in the above order, how long does it take to completely execute all jobs?

B. Can you find an order of execution which reduces the time to completely execute all the jobs? If so, how long is it?

4) A computer system consists of a pipeline of N identical processors, with the input job stream going into the first processor, the output of the first processor feeding the input of the second processor, and so on:

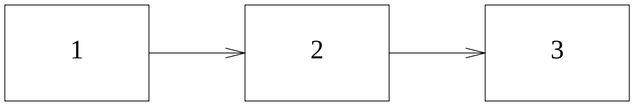


Figure 1.22.Three Stage (N=3) Processor Pipeline

$T(K)$ denotes the time, which is constant, required to complete the Kth step of executing a job. Derive the following results:

A)The time required to completely process one job, T_1:

$$T_1 = \sum_{K=1}^{N} T(K)$$

B)The time required to completely process M jobs, T_M:

$$T_M = \sum_{K=1}^{N} T(K) + (M-1)\max_K T(K) = T_1 + (M-1)T_{\max} \qquad T_{\max} \equiv \max_K T(K)$$

C)The mean throughput rate of executing or processing M jobs in time T_M:

$$mean\ \ throughput\ \ rate = \frac{M}{T_M}$$

D)The long term time averaged mean throughput rate $M \rightarrow \infty$:

$$long\ \ term\ \ time\ \ averaged\ \ mean\ \ throughput\ \ rate \equiv$$

$$\lim_{M \rightarrow \infty} mean\ \ throughput\ \ rate \text{ for } M\ \ jobs = \frac{M}{T_M}$$

5) The figure below is a hardware block diagram of a multiple processor multiple memory computer system: Two steps occur for each job:

[1] A processor executes code

[2] Data is fetched from memory

This is summarized in the table below:

Table 1.7.Job Step/Resource Summary

Step	*Processor*	*Memory*	*Time*
1	1	0	$T_{execute}$
2	0	1	T_{fetch}

Figure 1.23.Multiple Processor Multiple Memory Hardware Block Diagram

The hardware configuration for this system consists of three processors and four memories.

A. Draw a queueing network block diagram of this system

B. A total of M jobs can be held in the system at any one time. Plot upper and lower bounds on mean throughput rate and mean delay versus M for $T_{execute}=T_{fetch}=1$

C. Repeat the above for $T_{execute}=3T_{fetch}=1$

D. What is the stretching factor for both cases?

6) The hardware configuration for a word processing system consists of four terminals, a central processor, a disk controller, two disks, a high speed printer and a low speed printer. One terminal is dedicated to express or rush jobs, and uses the high speed printer. The other terminals are dedicated to normal jobs, and use the low speed printer or, if available, the high speed printer.

A. What are the hardware resources in this system?

B. Make a flow chart for each step required for a normal job. Repeat for a rush job.

C. What are the resources required for each step of a normal job? A rush job?

D. With no contention, what is the mean time to execute a normal job? A rush job?

E. What is the maximum mean throughput rate to execute J normal jobs? J rush jobs?

F. Suppose there are only normal jobs. Plot upper and lower bounds on mean throughput rate and mean delay versus the total number of normal jobs. Repeat if there are only rush jobs.

G. Suppose we fix the *mix* or fraction of jobs of each type, rush and normal, that the system can execute at any one time: F_{rush} and F_{normal} are the fraction of rush and normal jobs. If we submit J total jobs and allow $J\rightarrow\infty$, what resource will become completely busy first?

H. Suppose jobs are executed either in order of arrival or with a quantum round robin scheduler. What are the ranges on mean response time and mean throughput rate for each schedule?

7) The hardware configuration for an order entry system consists of sixteen terminals, four cluster controllers (one for each four terminals), a dedicated link to a computer, and a computer. The computer hardware configuration consists of a processor, memory, a disk controller, two disks connected to the disk controller, a printer, and an interface to a data link to a second inventory control computer system. The system operates as follows:

- Clerks enter orders at terminals and then wait for the system to respond
- The order is sent by the cluster controller over the data link to the computer
- The computer does some initial processing on the order, and if it is valid, stores it in the secondary disk storage
- The computer returns control to the operator at the terminal
- At a later point in time, the orders stored in secondary disk storage are retrieved and sent over a communications line to the inventory control system

Answer the following questions:

A. Make a flow chart of this job work flow

B. Make a table showing the resources required at each step of execution

C. If we ignore the last step of retrieving orders from disk and communicating them to an inventory control system, what is the fastest rate at which orders can be executed?

8) We want to study electronic mail pending performance at the hardware level in more detail. An operator at a terminal submits a query, making use of the hardware configuration shown in Figure 1.24.

Figure 1.24.Hardware Block Diagram

The steps involved in mail query are as follows:

- The terminal controller seizes the switch and alerts the processor
- The processor transfers the query from the terminal via the terminal controller through the switch to memory
- The processor executes the commands associated with mail pending on the data in memory via the switch
- The processor seizes the switch and accesses the mail pending information which is stored on secondary storage in a moving head disk
- The processor executes the code associated with this mail pending activity while holding the bus and memory
- The terminal is notified of the mail pending by the processor seizing the switch and demanding the information be transferred from memory to the terminal controller

- The tape controller is seized by the processor via the bus in order to transfer information logging the mail pending activity
- At a later point in time the processor will transfer data from the tape to memory via the switch
- The processor will generate a management report on mail activity and transfer this to the printer via the switch from memory

Answer the following questions:

A. What are the hardware resources?

B. Make a table showing what resources are required at each step and for how long.

C. What is the maximum rate at which mail pending can be executed, assuming nothing else is executed?

9) We want to study electronic mail pending performance at a higher level in more detail. The steps involved in checking to see if electronic mail is pending are summarized in the figure below.

Figure 1.25.Electronic Mail Pending Flow Chart

In words, these steps are:

[1] The terminal interacts with the input/output handling routines to generate operating system activity

[2] The operating system generates application program activity in turn

[3] The application program retrieves the mail pending from the data base manager

[4] The operating system is fed the list of outstanding mail items

[5] The input/output handling routines generate the list of outstanding mail items on the terminal

Answer the following questions:

A. Make up a flow chart for mail pending.

B. The resources available here are hardware, operating system, data base management, and application. Verify that the table below is a candidate for showing the resources held at each step:

Table 1.8.Mail Pending Step Resource Summary

Step	*Resource*				*Time*
Number	*Hardware*	*Op Sys*	*Data Base*	*Application*	*Interval*
1	1	0	0	0	$T_{hardware}$
2	0	1	0	0	$T_{op\ sys}$
3	0	0	0	1	T_{applic}
4	0	0	1	0	$T_{data\ base}$
5	0	0	0	1	T_{applic}
6	0	1	0	0	$T_{op\ sys}$
7	1	0	0	0	$T_{hardware}$

C. What is the mean response time with only one terminal using the system? What is the mean throughput rate with only one terminal using the system?

D. Assuming only mail pending queries are made, what is the maximum mean throughput rate for executing this command?

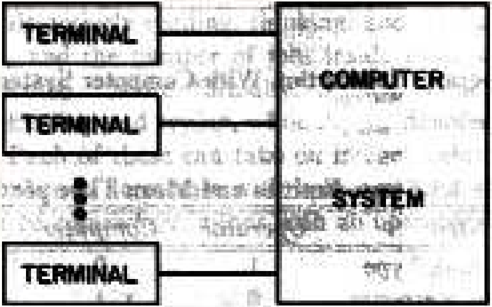


Figure 1.1.Computer System Hardware Block Diagram

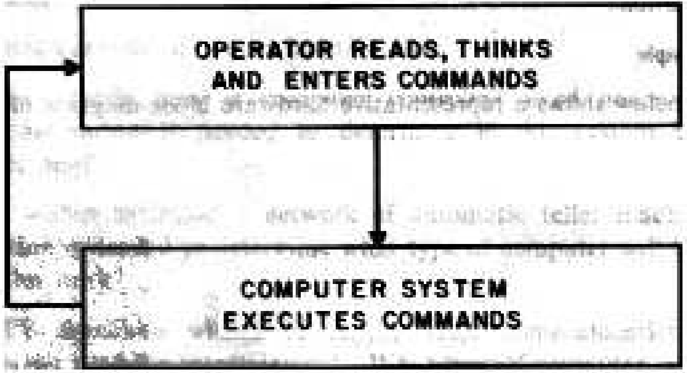

OPERATOR READS, THINKS AND ENTERS COMMANDS
COMPUTER SYSTEM EXECUTES COMMANDS

Figure 1.2.Operator Interaction With Computer System Flow Chart

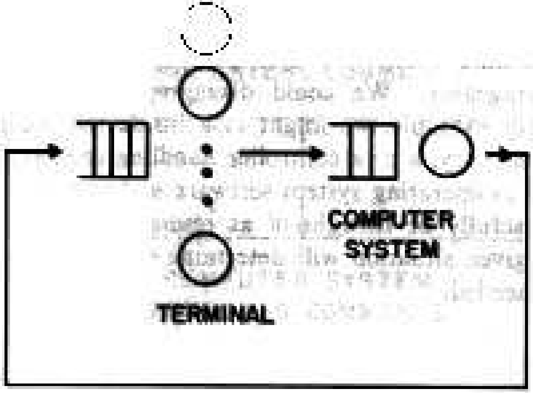


Figure 1.3.Queueing Network Block Diagram

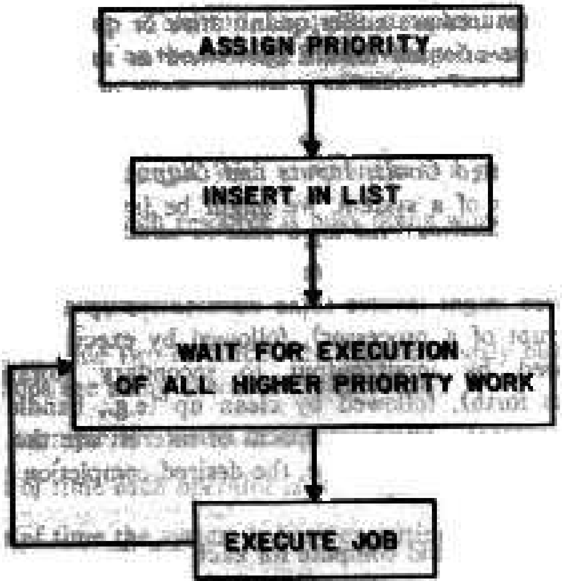


Figure 1.4.Scheduler Flow Chart

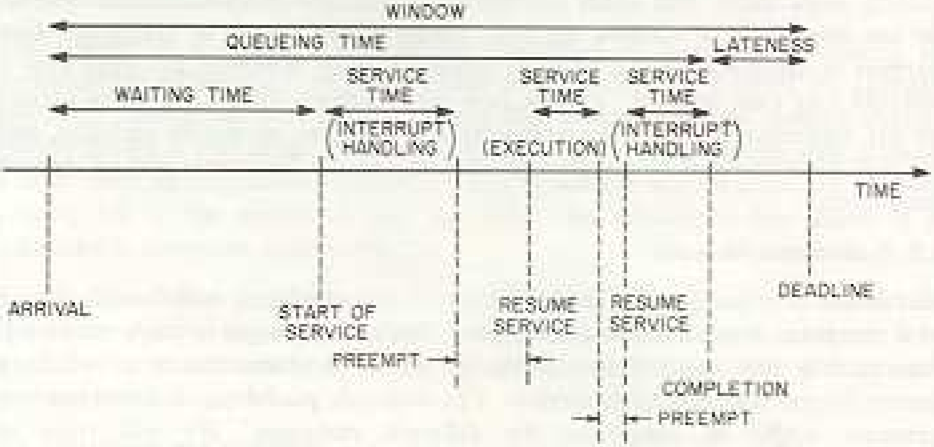


Figure 1.5. Illustrative Time Epochs for Processing a Job

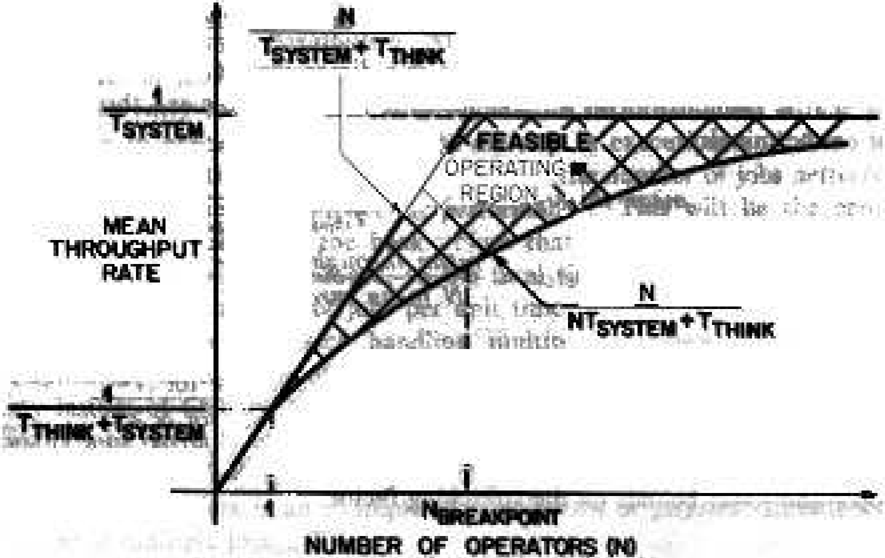


Figure 1.6.Upper and Lower Bounds on Mean Throughput Rate

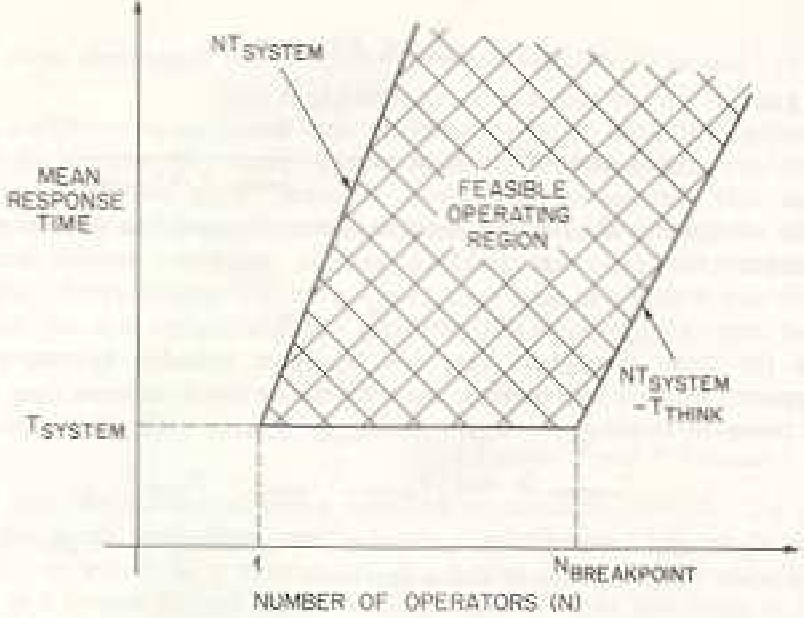


Figure 1.7.Upper and Lower Bounds on Mean Response Time

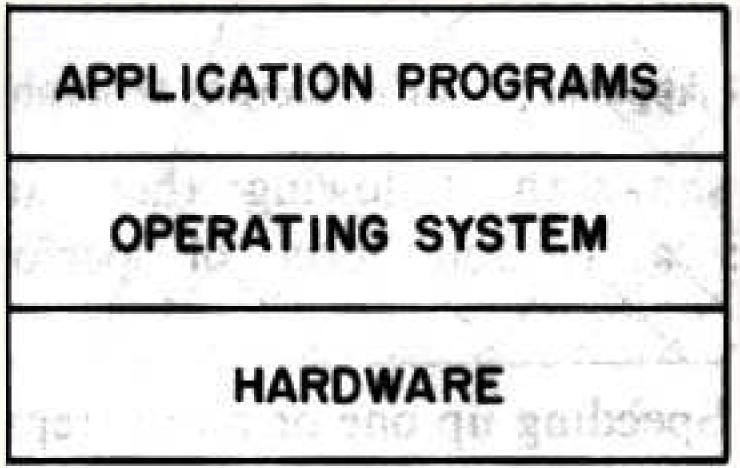


Figure 1.8.Layered Model of Computer System

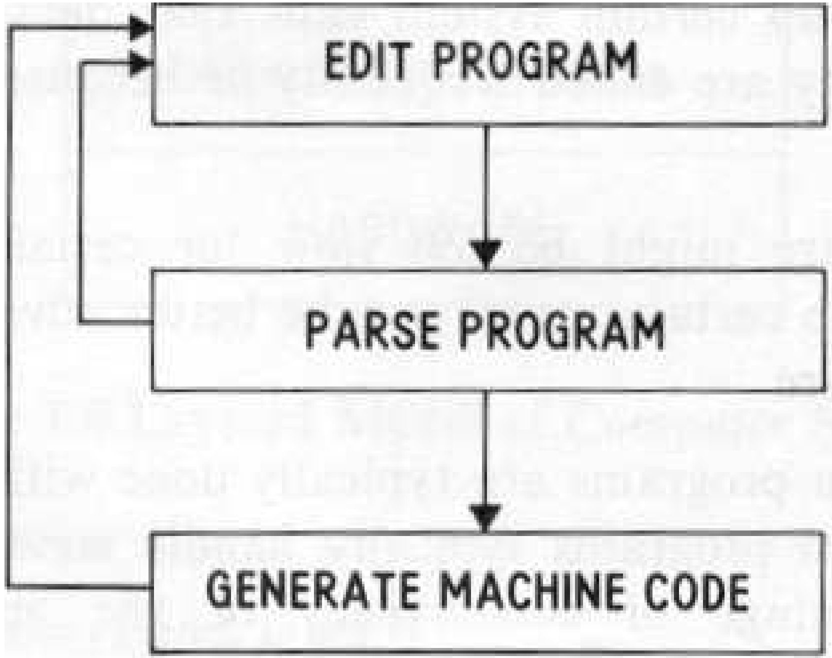


Figure 1.9.Application Program Development Flowchart

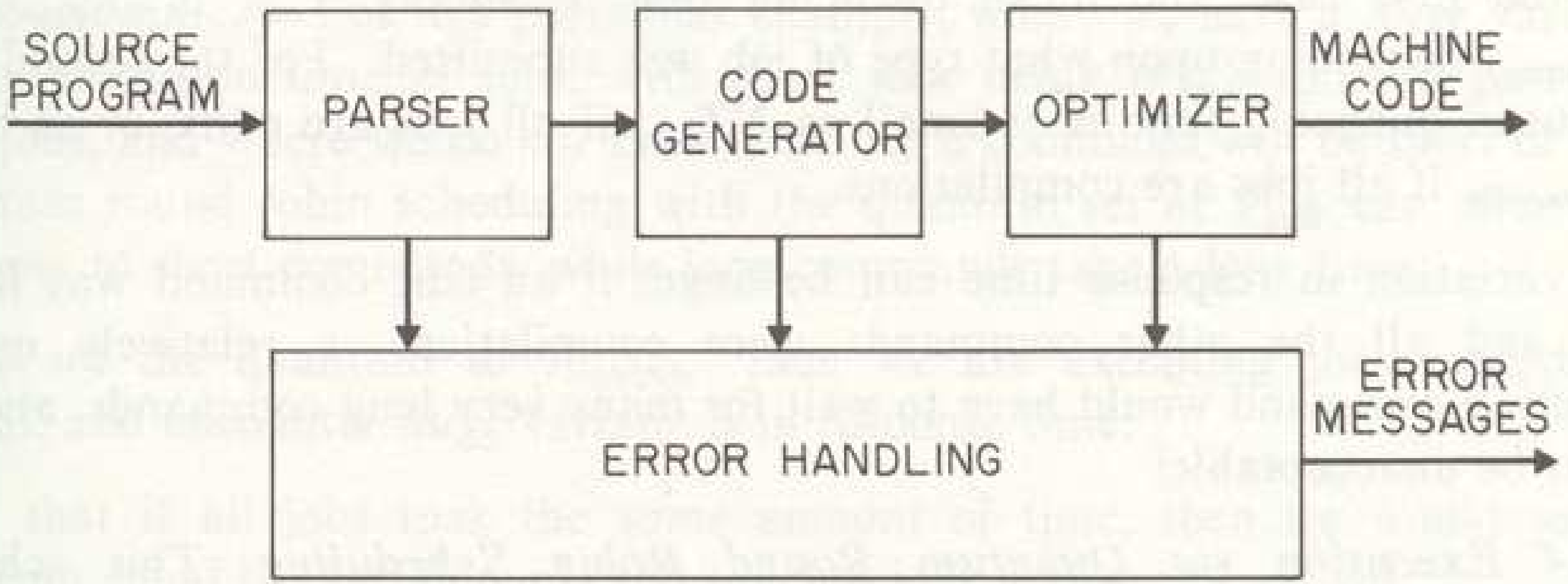


Figure 1.10.Program Development Block Diagram

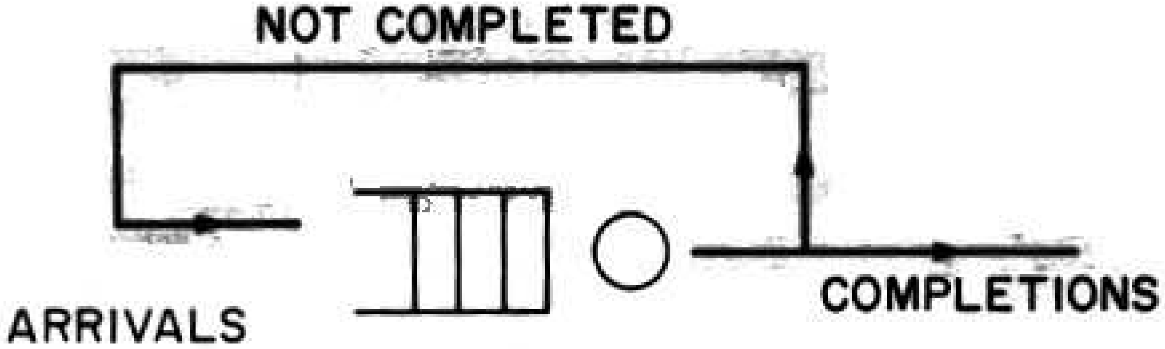


Figure 1.11.Round Robin Queueing Network Block Diagram

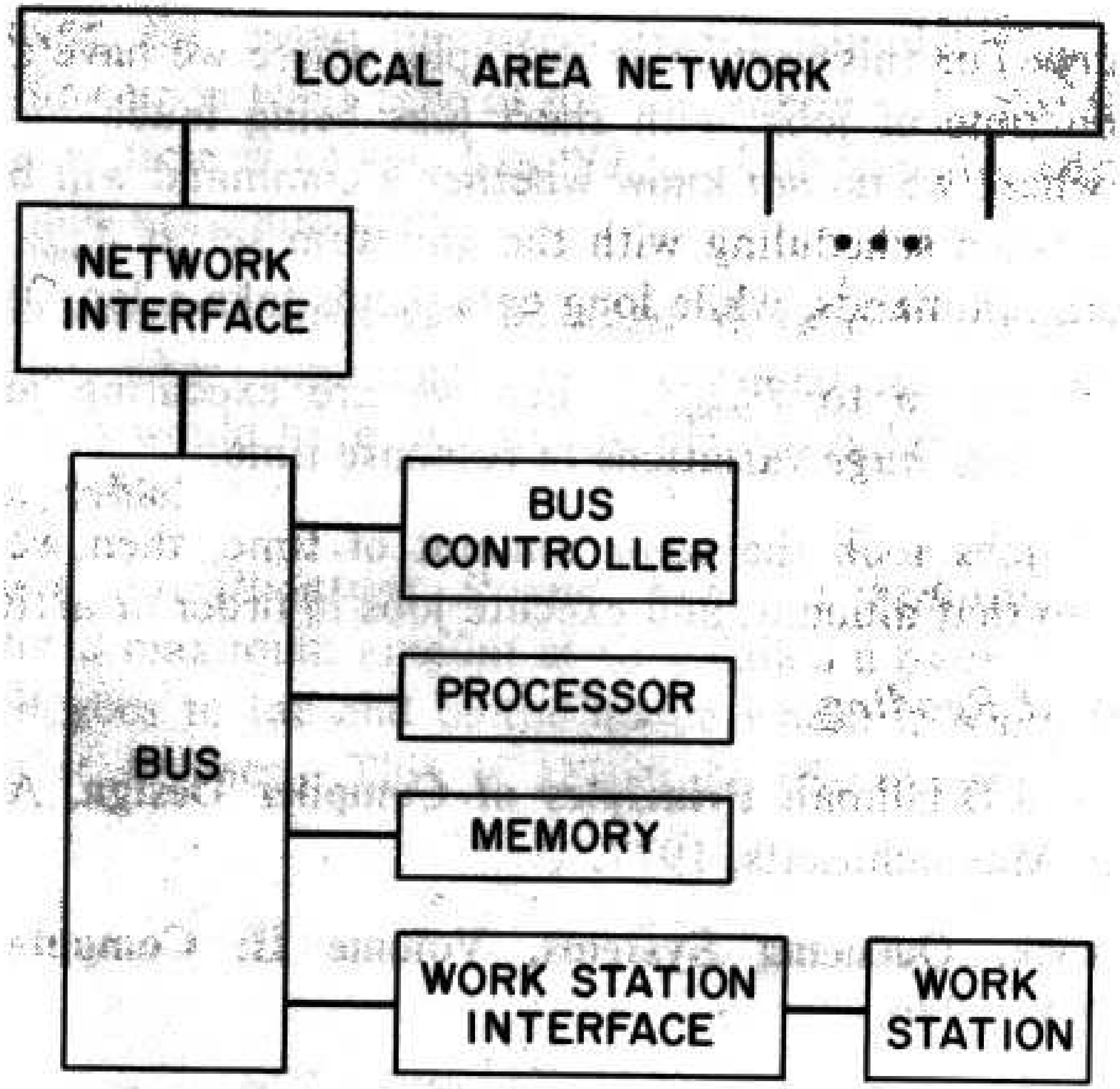
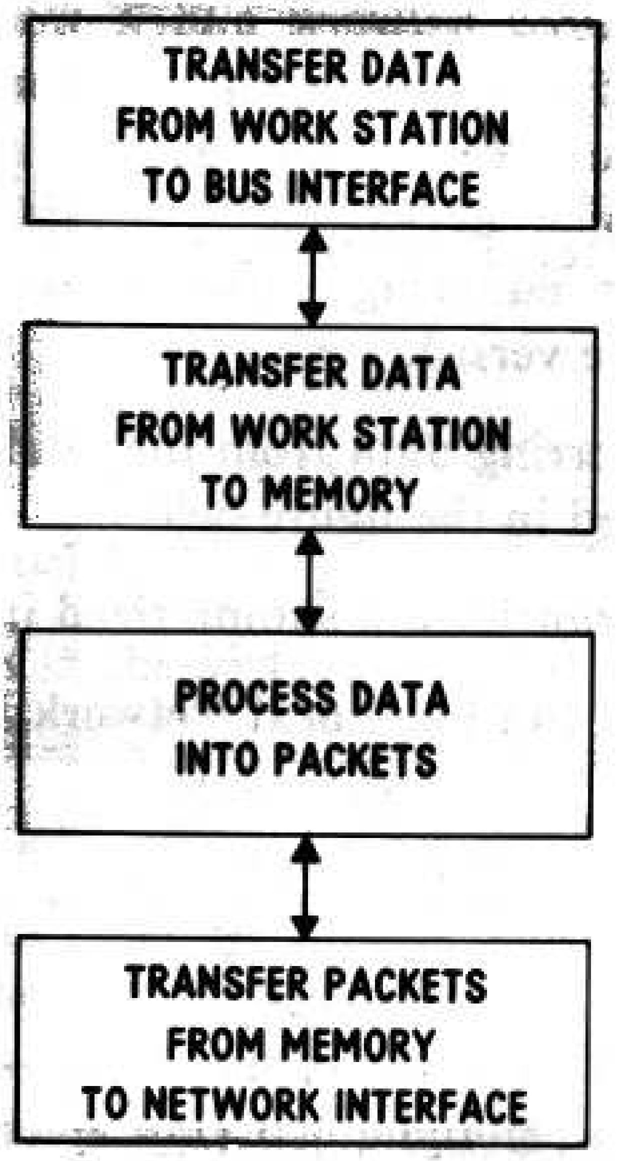


Figure 1.12.Local Area Network Hardware Interface Block Diagram



TRANSFER DATA FROM WORK STATION TO BUS INTERFACE
TRANSFER DATA FROM WORK STATION TO MEMORY
PROCESS DATA INTO PACKETS
TRANSFER PACKETS FROM MEMORY TO NETWORK INTERFACE

Figure 1.13. Work Station to Local Area Network Data Transfer

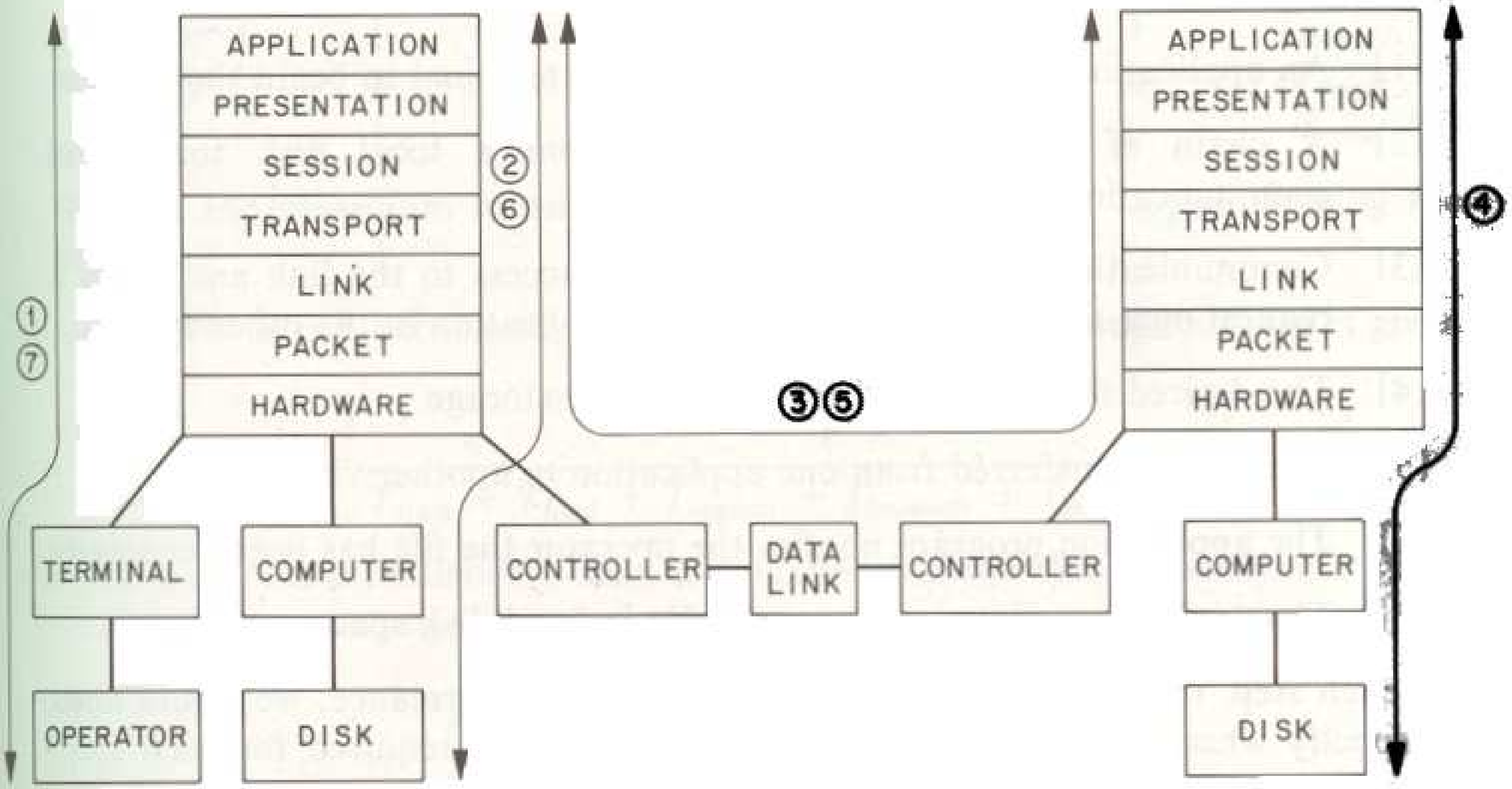


Figure 1.14.Open Systems Interconnection Architecture

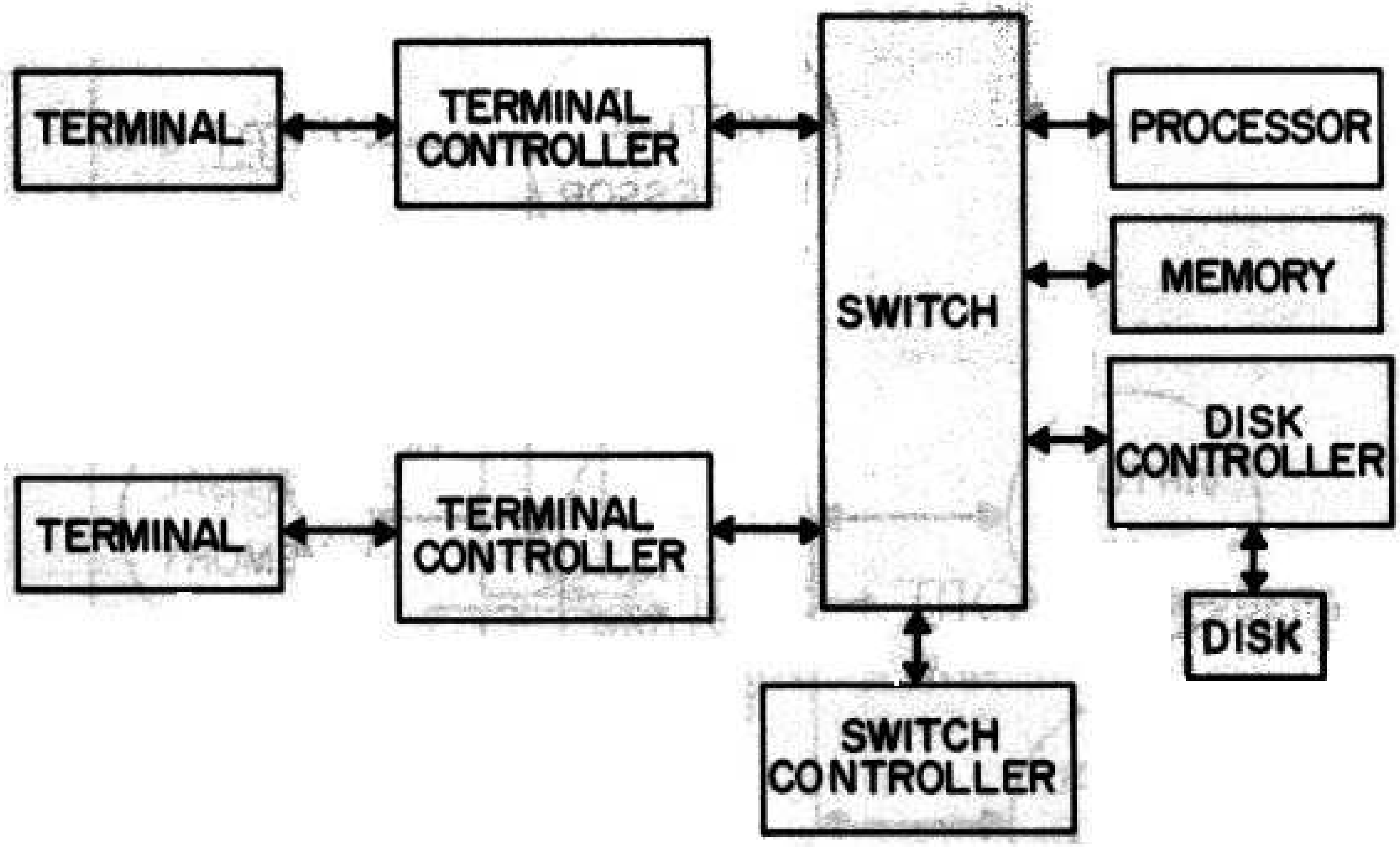


Figure 1.15.Hardware Block Diagram

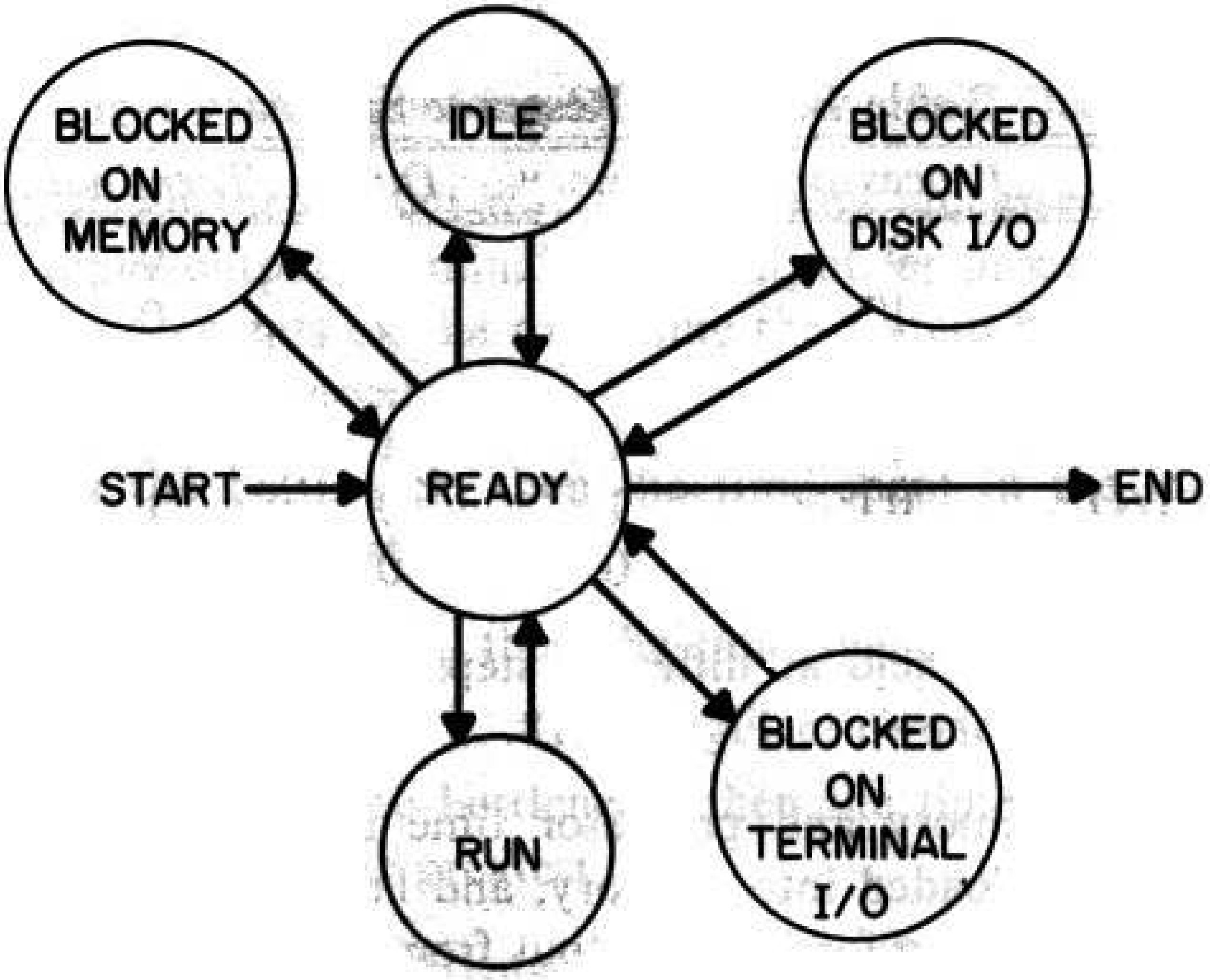


Figure 1.16.States of a Process

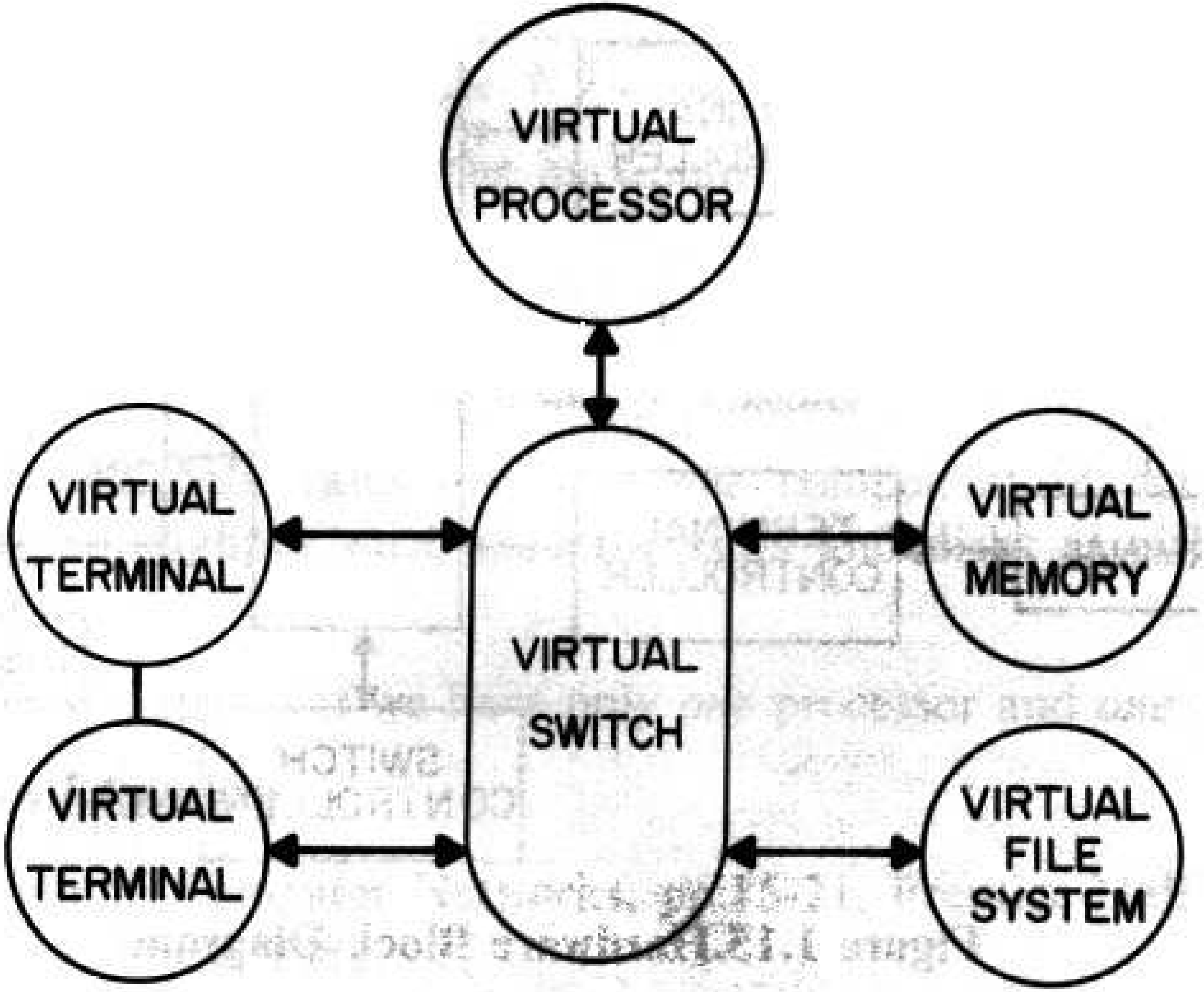


Figure 1.17.Illustrative Set of Processes Managed by Operating System

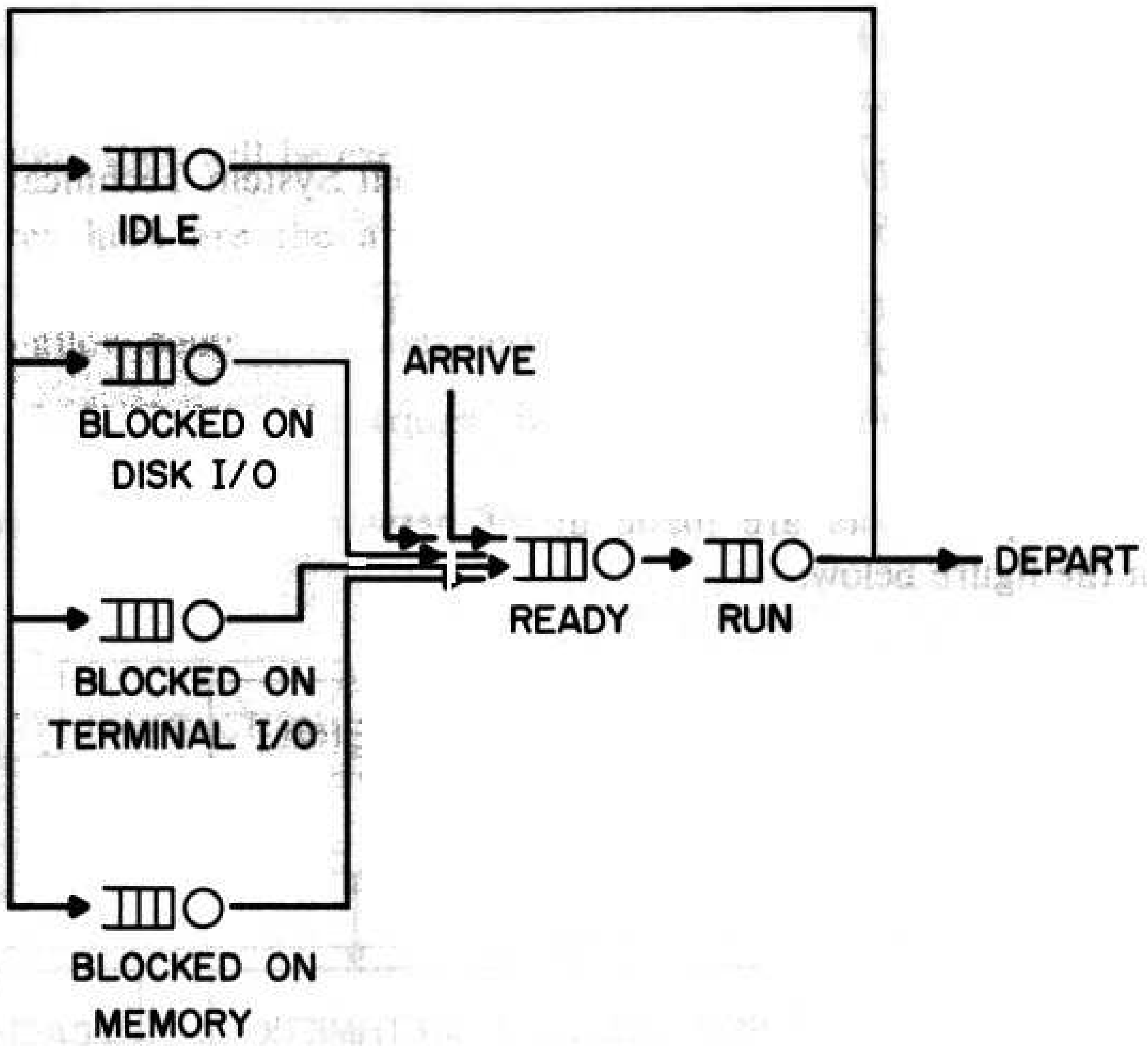


Figure 1.18.Operating System Network of Queues

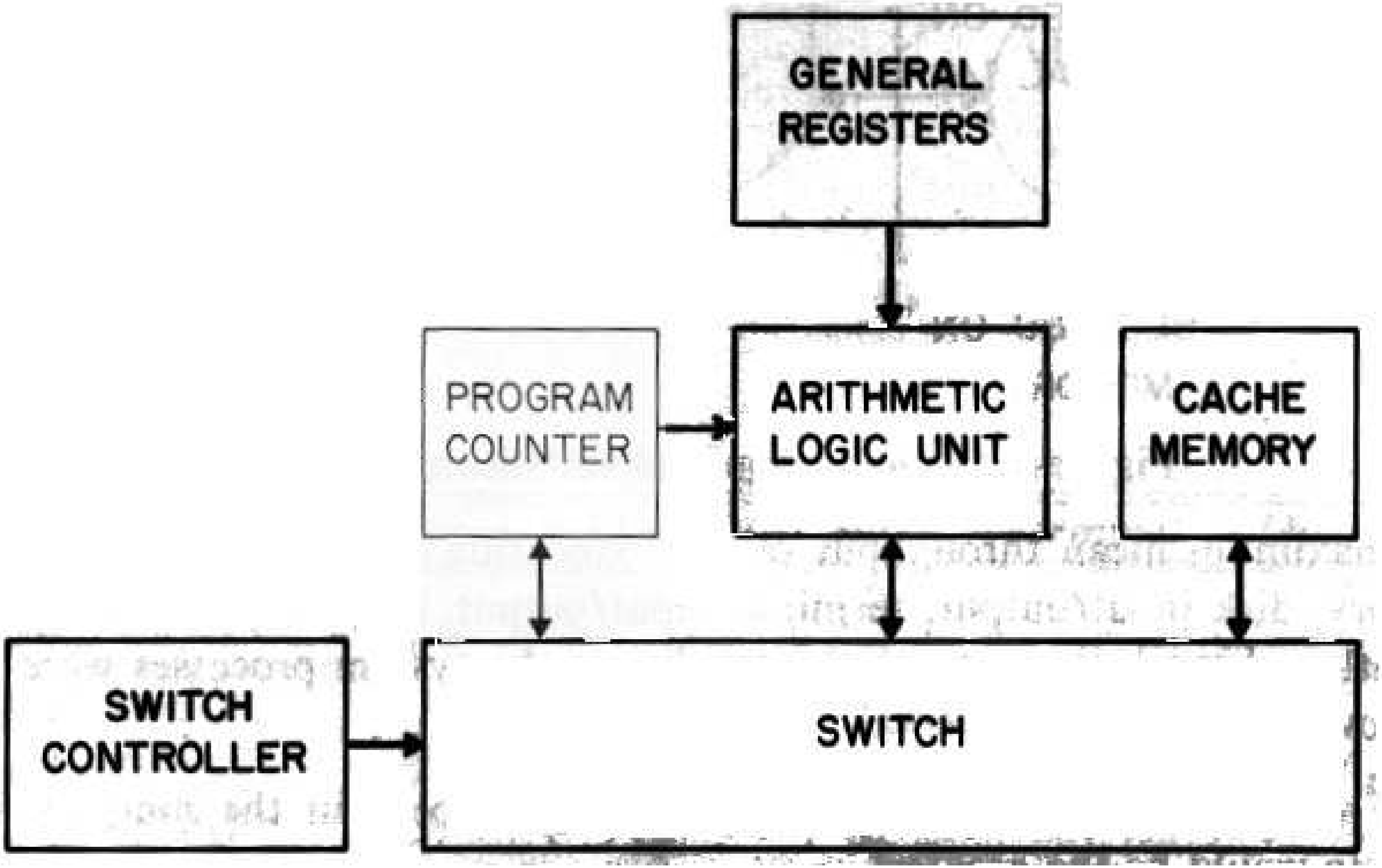


Figure 1.19.Processor Block Diagram

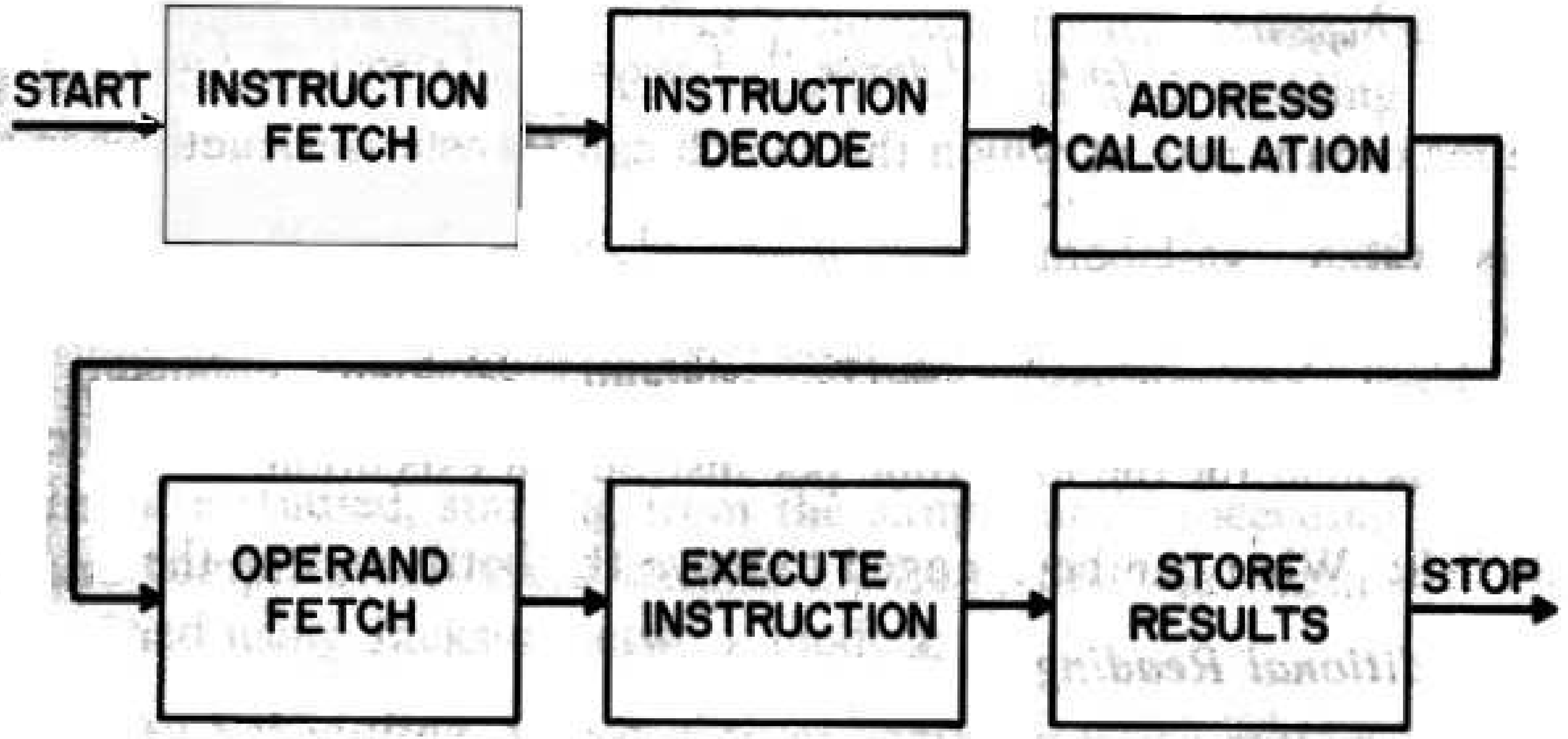


Figure 1.20.Processor Instruction Execution Flow Chart

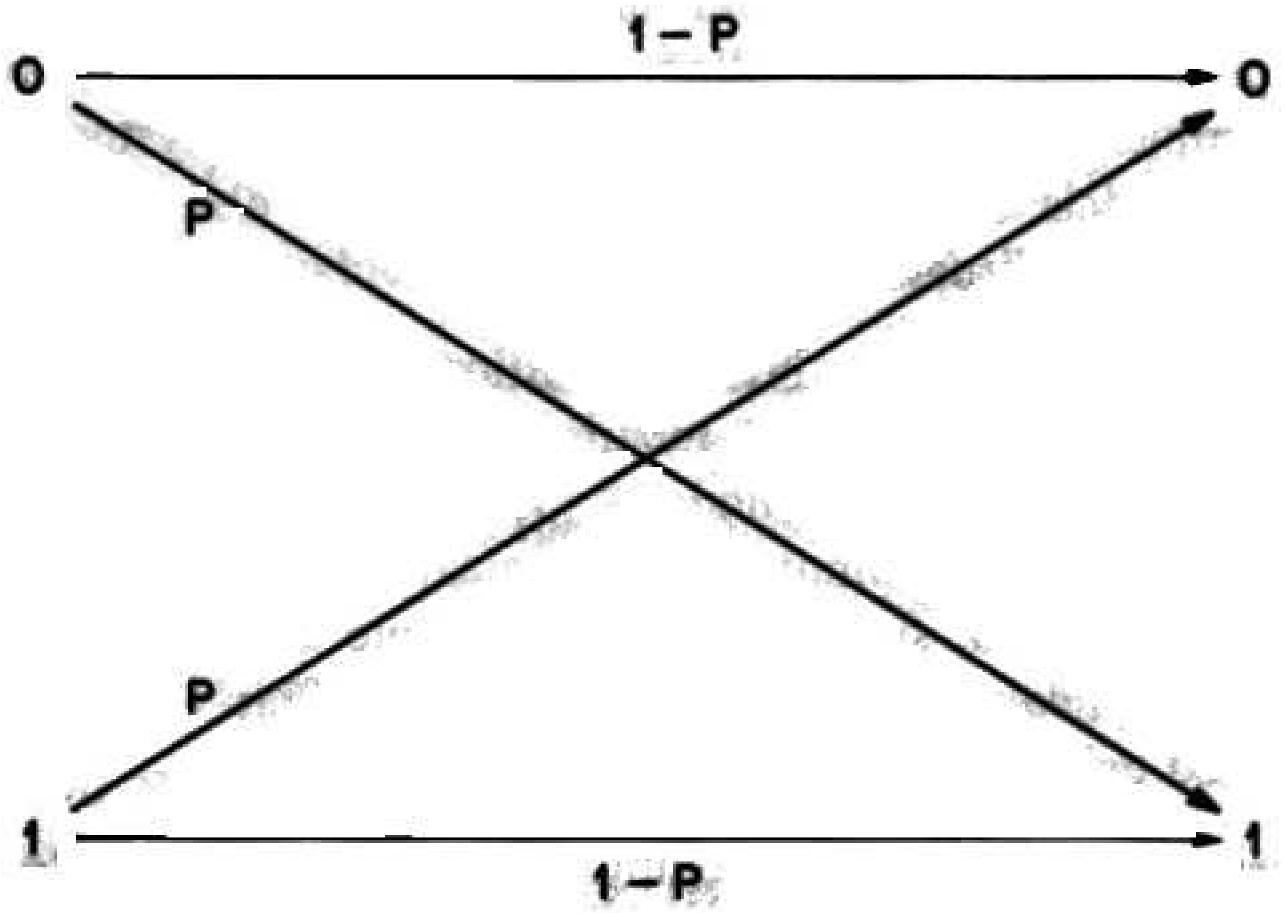


Figure 1.21.Bit Transmission Probabilities

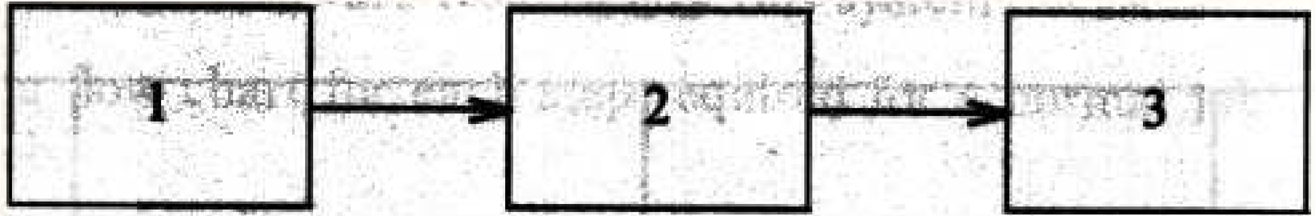


Figure 1.22.Three Stage (N=3) Processor Pipeline

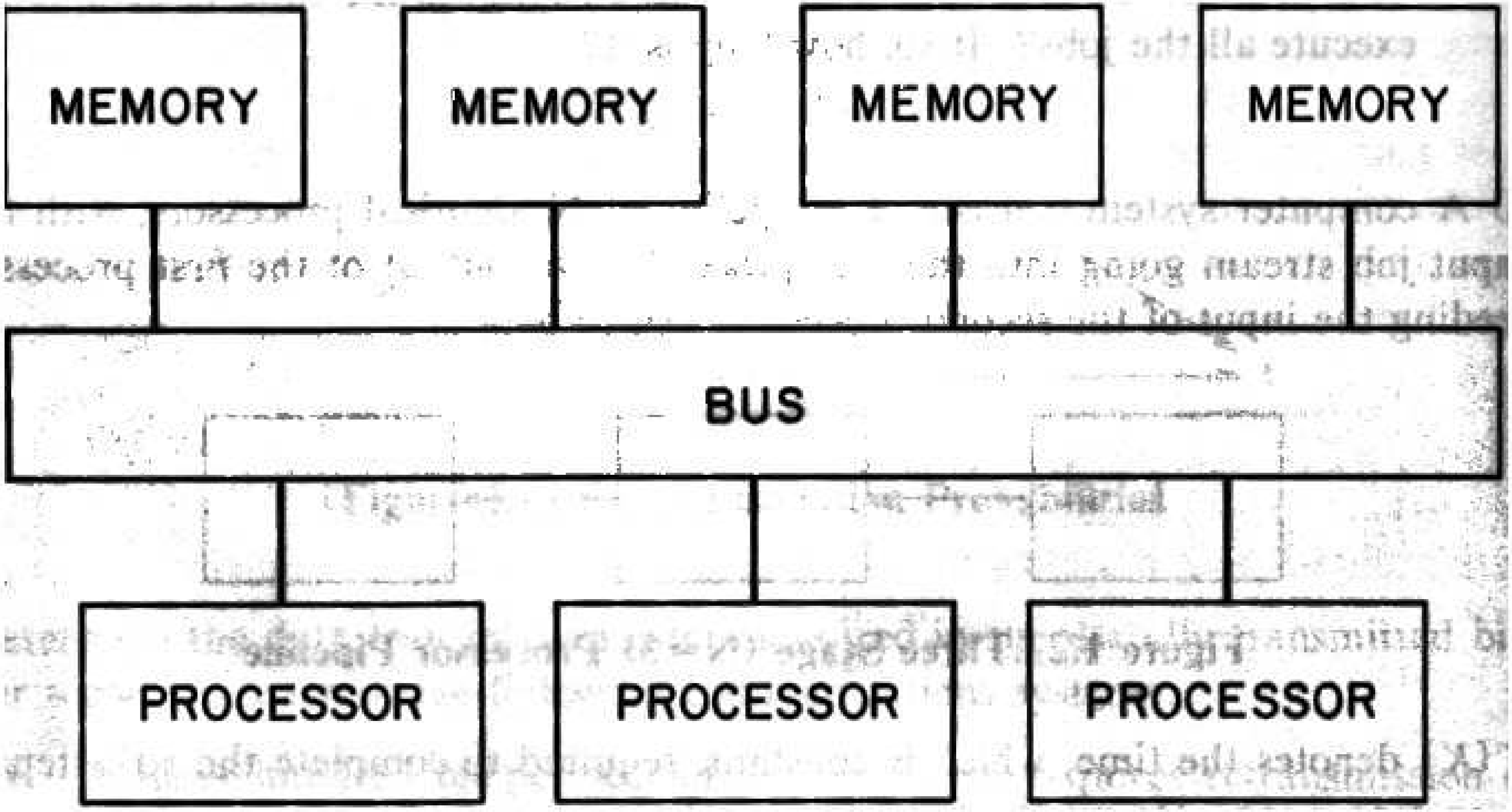


Figure 1.23.Multiple Processor Multiple Memory Hardware Block Diagram

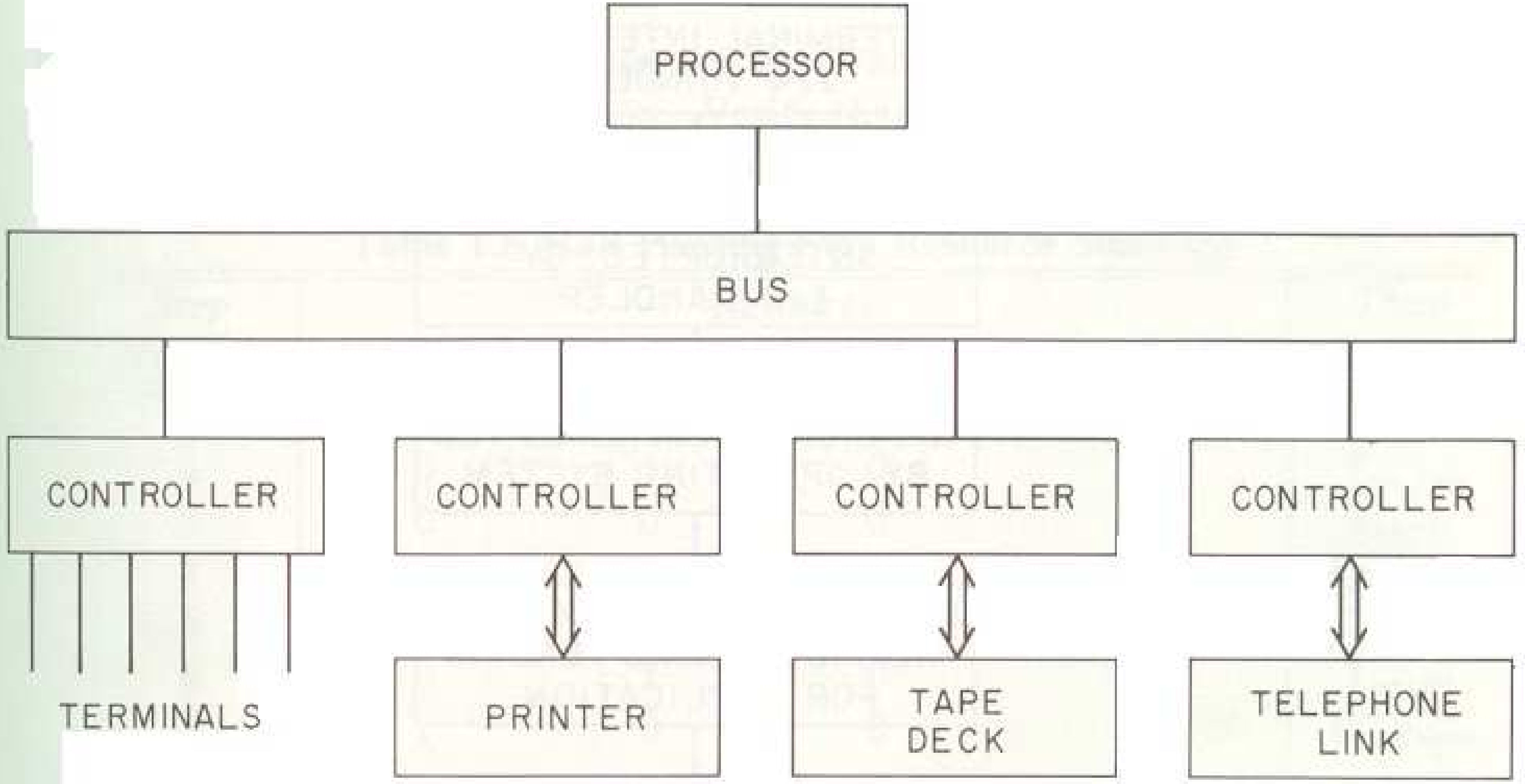


Figure 1.24.Hardware Block Diagram

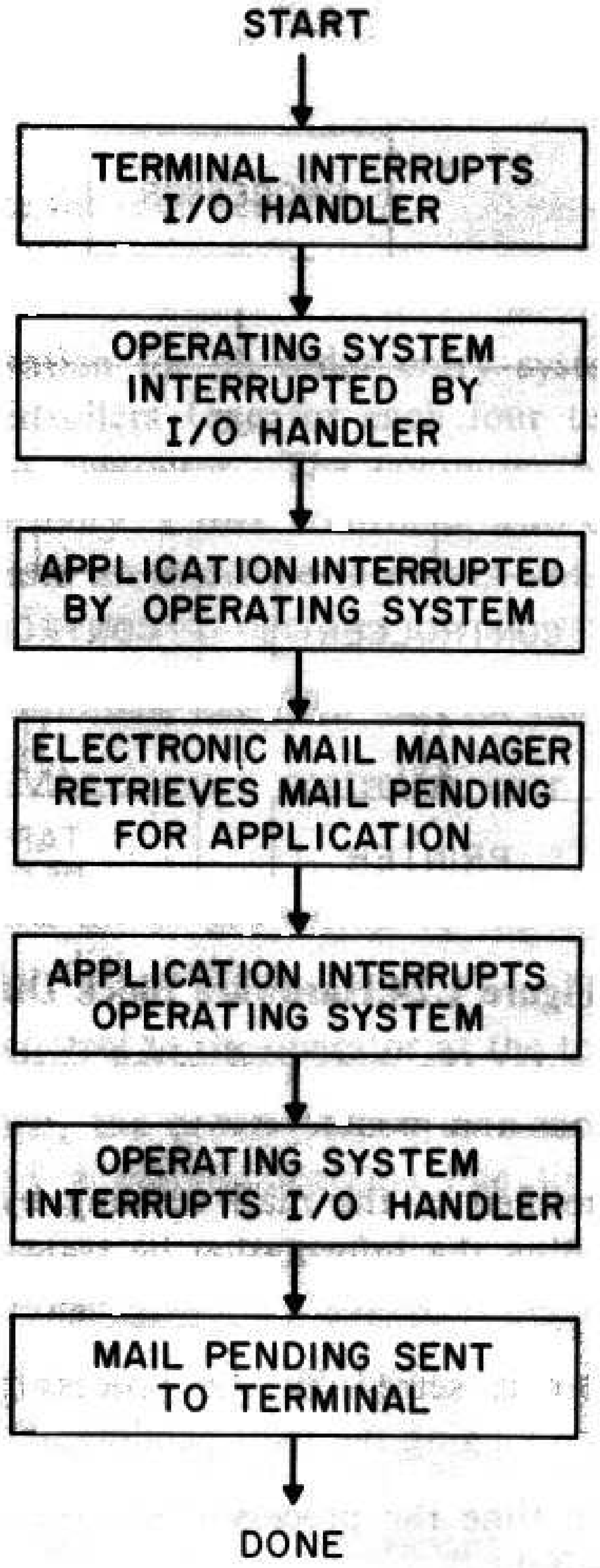

START
TERMINAL INTERRUPTS I/O HANDLER
OPERATING SYSTEM INTERRUPTED BY I/O HANDLER
APPLICATION INTERRUPTED BY OPERATING SYSTEM
ELECTRONIC MAIL MANAGER RETRIEVES MAIL PENDING FOR APPLICATION
APPLICATION INTERRUPTS OPERATING SYSTEM
OPERATING SYSTEM INTERRUPTS I/O HANDLER
MAIL PENDING SENT TO TERMINAL
DONE

Figure 1.25.Electronic Mail Pending Flow Chart