USN

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A++ GRADE BY NAAC

Internal Assessment Test 1 – April 2023

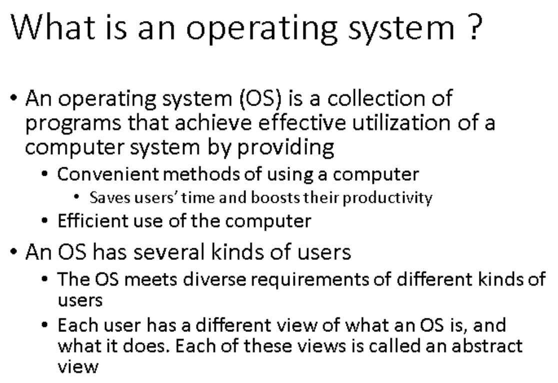
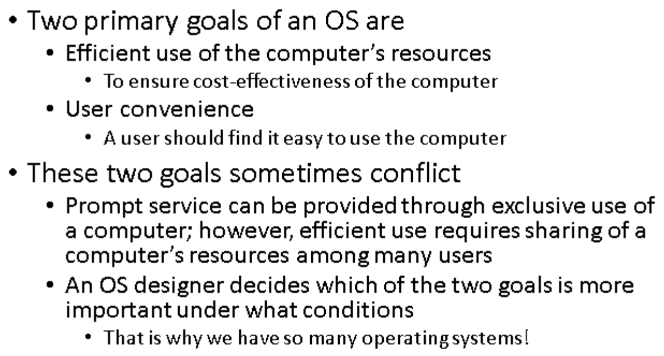
Sub:	Operating System					Sub Code:	18 EC 641	Branch:	ECE
Date:	25-04-23	Duration:	90 min's	Max Marks:	50	Sem / Sec:	6 – A B C D		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	Define Operating System? What are the goals of Operation systems? Explain the key concern of an operating system.	[10]	CO1	L1
2.	Explain partition based & pool based resource allocation strategies with a neat diagram.	[10]	CO1	L2
3a.	Explain the measures of Efficiency, System Performance and User Service	[06]	CO1	L1
3b.	Discuss in brief about the benefits of Distributed Operating Systems	[04]	CO1	L1
4.	Explain important properties of Time sharing processing and Real Time OS with neat sketch and explain their advantages and Disadvantages.	[10]	CO1	L2
5.	Explain briefly, the different classes of an OS with primary concern and key concepts.	[10]	CO1	L1
6a.	With a neat diagram explain the turnaround time in batch processing system.	[06]	CO1	L2
6b.	Discuss various computations in an operating system.	[04]	CO1	L2
7	Explain the state transition for a process with state transition diagram	[10]	CO2	L1

SCHEME OF VALUATION

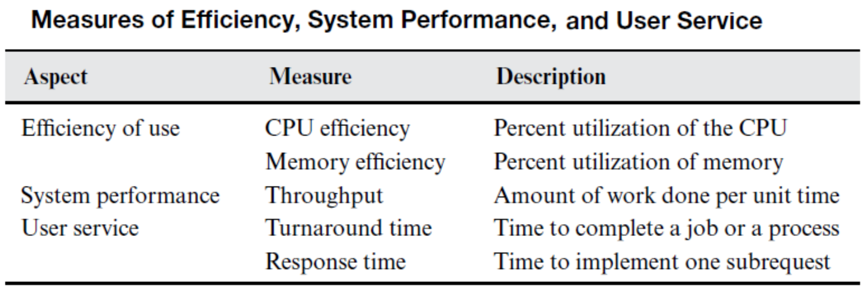
Q. no.	Questions	Marks
1.	Definition	2M
	Goals & Key Concerns of an OS	8M
2.	Partition based resource allocation strategy with diagram	5M
	Pool based resource allocation strategy with diagram	5M
3a.	Efficiency, System Performance and User Service	6M
3b.	Benefits of Distributed Operating Systems	4M
4.	Time sharing system Properties and explanation	5M
	Real time system OS Properties and explanation	5M
5.	Classes of Operating system	6M
	Primary concerns and key concepts	4M
6a.	Turnaround Time in Batch processing system	6M
6b.	Various Computations in an OS	4M
7.	State Transition for a process diagram	6M
	Explanation	4M

IAT-1 SOLUTION

Q. no.	Solution	Marks
1.	Definition  What is an operating system ? • An operating system (OS) is a collection of programs that achieve effective utilization of a computer system by providing • Convenient methods of using a computer • Saves users' time and boosts their productivity • Efficient use of the computer • An OS has several kinds of users • The OS meets diverse requirements of different kinds of users • Each user has a different view of what an OS is, and what it does. Each of these views is called an abstract view	2M
	Goals & Key Concerns of an OS  • Two primary goals of an OS are • Efficient use of the computer's resources • To ensure cost-effectiveness of the computer • User convenience • A user should find it easy to use the computer • These two goals sometimes conflict • Prompt service can be provided through exclusive use of a computer; however, efficient use requires sharing of a computer's resources among many users • An OS designer decides which of the two goals is more important under what conditions • That is why we have so many operating systems! With Key Concerns of an OS	8M
2.	Partition based resource allocation strategy with diagram Two resource allocation strategies are popular. In the resource partitioning approach, the OS decides a priori what resources should be allocated to each user program, for example, it may decide that a program should be allocated 1 MB of memory, 1000 disk blocks, and a monitor. It divides the resources in the system into many resource partitions, or simply partitions; each partition includes 1 MB of memory, 1000 disk blocks, and a monitor. It allocates one resource partition to each user program when its execution is to be initiated. To facilitate resource allocation, the resource table contains entries for resource partitions rather than for individual resources as in Table 1.3. Resource partitioning is simple to implement, hence it incurs less overhead; however, it lacks flexibility. Resources are wasted if a resource partition contains more resources than what a program needs. Also, the OS cannot execute a program if its requirements exceed the resources available in a resource partition. This is true even if free resources exist in another partition.	5M
	Pool based resource allocation strategy with diagram In the pool-based approach to resource management, the OS allocates resources from a common pool of resources. It consults the resource table when	5M

a program makes a request for a resource, and allocates the resource if it is free. It incurs the overhead of allocating and deallocating resources when requested. However, it avoids both problems faced by the resource partitioning approach—an allocated resource is not wasted, and a resource requirement can be met if a free resource exists.

3a. Efficiency, System Performance and User Service — 6M



Measures of Efficiency, System Performance, and User Service

Aspect	Measure	Description
Efficiency of use	CPU efficiency	Percent utilization of the CPU
	Memory efficiency	Percent utilization of memory
System performance	Throughput	Amount of work done per unit time
User service	Turnaround time	Time to complete a job or a process
	Response time	Time to implement one subrequest

3b. Benefits of Distributed Operating Systems — 4M

A distributed operating system permits a user to access resources located in other computer systems conveniently and reliably. To enhance convenience, it does not expect a user to know the location of resources in the system, which is called *transparency*. To enhance efficiency, it may execute parts of a computation in different computer systems at the same time. It uses *distributed control*; i.e., it spreads its decision-making actions across different computers in the system so that failures of individual computers or the network does not cripple its operation.

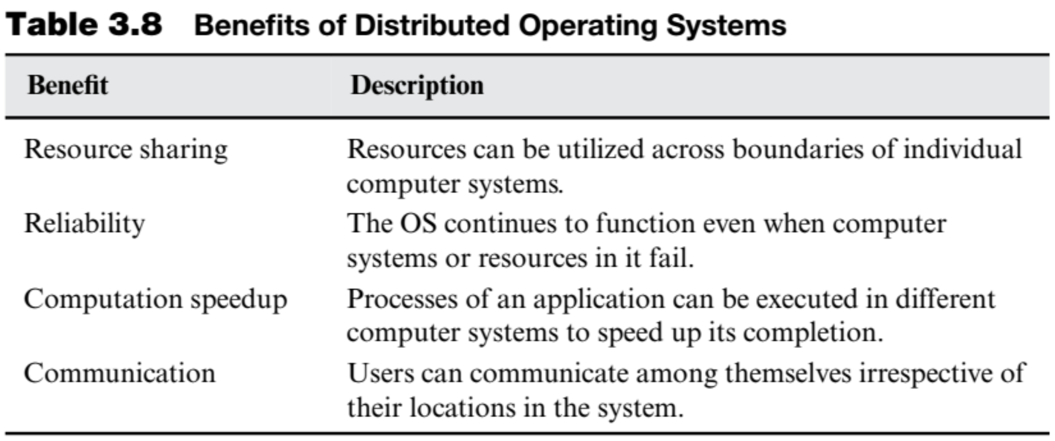


Table 3.8 Benefits of Distributed Operating Systems

Benefit	Description
Resource sharing	Resources can be utilized across boundaries of individual computer systems.
Reliability	The OS continues to function even when computer systems or resources in it fail.
Computation speedup	Processes of an application can be executed in different computer systems to speed up its completion.
Communication	Users can communicate among themselves irrespective of their locations in the system.

4. Time sharing system Properties and explanation — 5M

In an interactive computing environment, a user submits a computational requirement—a subrequest—to a process and examines its response on the monitor screen. A time-sharing operating system is designed to provide a quick response to subrequests made by users. It achieves this goal by sharing the CPU time among processes in such a way that each process to which a subrequest has been made would get a turn on the CPU without much delay.

The scheduling technique used by a time-sharing kernel is called round-robin scheduling with time-slicing. It works as follows (see Figure 3.6): The kernel

	maintains a scheduling queue of processes that wish to use the CPU; it always schedules the process at the head of the queue. When a scheduled process completes servicing of a subrequest, or starts an I/O operation, the kernel removes it from the queue and schedules another process. Such a process would be added at the end of the queue when it receives a new subrequest, or when its I/O operation completes. This arrangement ensures that all processes would suffer comparable delays before getting to use the CPU. However, response times of processes would degrade if a process consumes too much CPU time in servicing its subrequest. The kernel uses the notion of a time slice to avoid this situation. We use the notation δ for the time slice. If the time slice elapses before the process completes servicing of a subrequest, the kernel preempts the process, moves it to the end of the scheduling queue, and schedules another process. The preempted process would be rescheduled when it reachesthe head ofthe queue once again. Thus, a process may haveto be scheduled several times before it completes servicing of a subrequest. The kernel employs a timer interrupt to implement time-slicing	
	Real time system OS Properties and explanation n a class of applications called real-time applications, users need the computer to perform some actions in a timely manner to control the activities in an external system, or to participate in them. The timeliness of actions is determined by he time constraints of the external system. Accordingly, we define a real-time application as follows: If the application takes too long to respond to an activity, a failure can occur in the external system. We use the term response requirement of a system to indicate the largest value of response time for which the system can function perfectly; a timely response is one whose response time is not larger than the response requirement of the system. Consider a system that logs data received from a satellite remote sensor. The satellite sends digitized samples to the earth station at the rate of 500 samples per second. The application process is required to simply store these samples in a file. Since a new sample arrives every two thousandth of a second, i.e., every 2 ms, the computer must respond to every “store the sample” request in less than 2 ms, or the arrival of a new sample would wipe out the previous sample in the computer’s memory. This system is a real-time application because a sample must be stored in less than 2 ms to prevent a failure. Its response requirement is 1.99 ms. The deadline of an action in a real-time application is the time by which the action should be performed. In the current example, if a new sample is received from the satellite at time t, the deadline for storing it on disk is t + 1.99 ms. Examples of real-time applications can be found in missile guidance, command and control applications like process control and air traffic control, data	5M

sampling and data acquisition systems like display systems in automobiles, multimedia systems, and applications like reservation and banking systems that employ large databases. The response requirements of these systems vary from a few microseconds or milliseconds for guidance and control systems to a few seconds for reservation and banking systems.

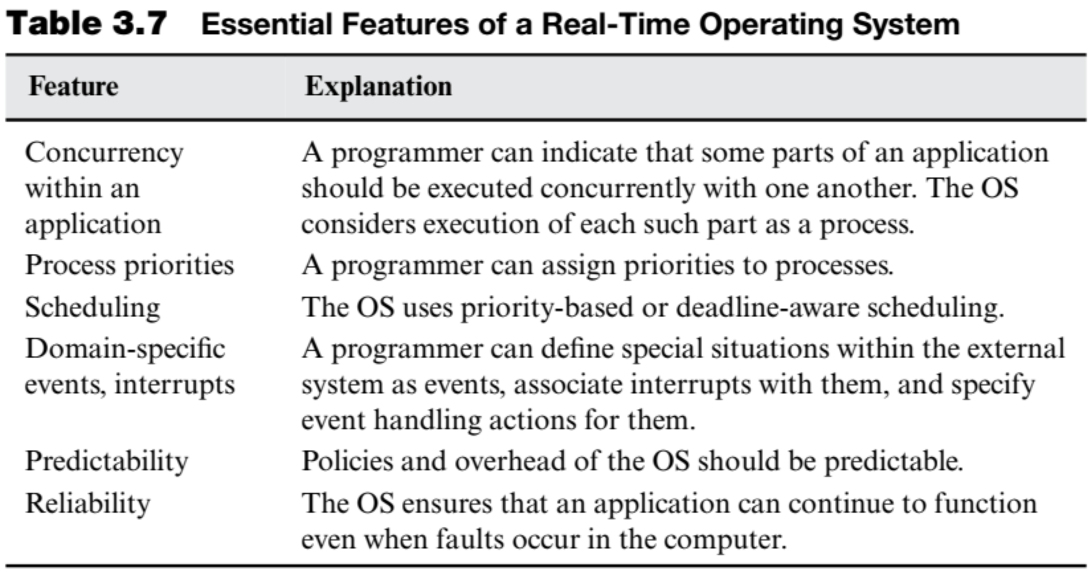


Table 3.7 Essential Features of a Real-Time Operating System

Feature	Explanation
Concurrency within an application	A programmer can indicate that some parts of an application should be executed concurrently with one another. The OS considers execution of each such part as a process.
Process priorities	A programmer can assign priorities to processes.
Scheduling	The OS uses priority-based or deadline-aware scheduling.
Domain-specific events, interrupts	A programmer can define special situations within the external system as events, associate interrupts with them, and specify event handling actions for them.
Predictability	Policies and overhead of the OS should be predictable.
Reliability	The OS ensures that an application can continue to function even when faults occur in the computer.

5. Classes of Operating system — 2M

Batch Processing

Multiprogramming

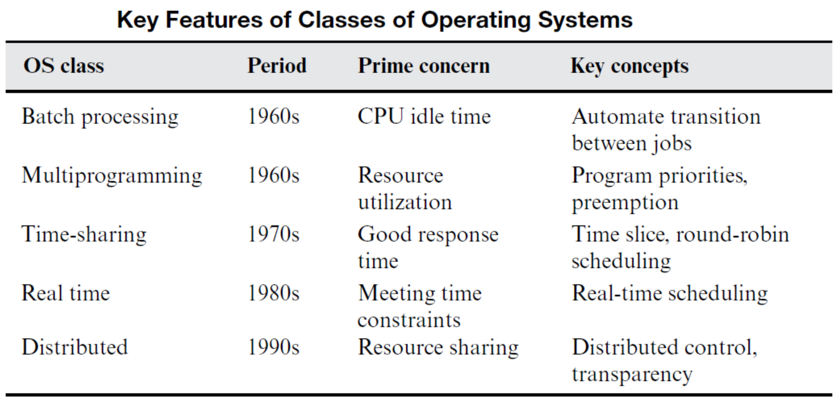
Time Sharing

Real Time

Distributed

Primary concerns and key concepts — 8M

Classes of operating systems have evolved over time as computer systems and users' expectations of them have developed; i.e., as computing environments have evolved. As we study some of the earlier classes of operating systems, we need to understand that each was designed to work with computer systems of its own historical period; thus we will have to look at architectural features representative of computer systems of the period.



Key Features of Classes of Operating Systems

OS class	Period	Prime concern	Key concepts
Batch processing	1960s	CPU idle time	Automate transition between jobs
Multiprogramming	1960s	Resource utilization	Program priorities, preemption
Time-sharing	1970s	Good response time	Time slice, round-robin scheduling
Real time	1980s	Meeting time constraints	Real-time scheduling
Distributed	1990s	Resource sharing	Distributed control, transparency

With Explanation

6a. Turnaround Time in Batch processing system — 6M

A batch is a sequence of user jobs formed for processing by the operating system. A computer operator formed a batch by arranging a few user jobs in a sequence and inserting special marker cards to indicate the start and end of the batch. When the operator gave a command to initiate processing of a batch, the batching kernel set up the processing of the first job of the batch. At the end of the job, it initiated execution of the next job, and so on, until the end of the batch. Thus the operator had to intervene only at the start and end of a batch

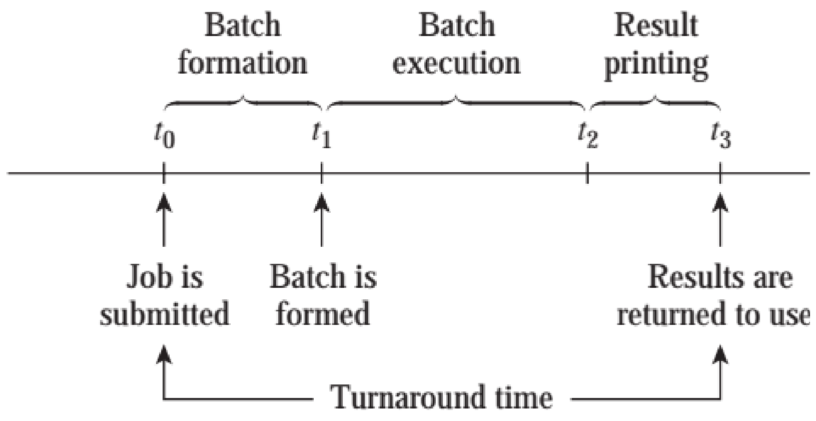
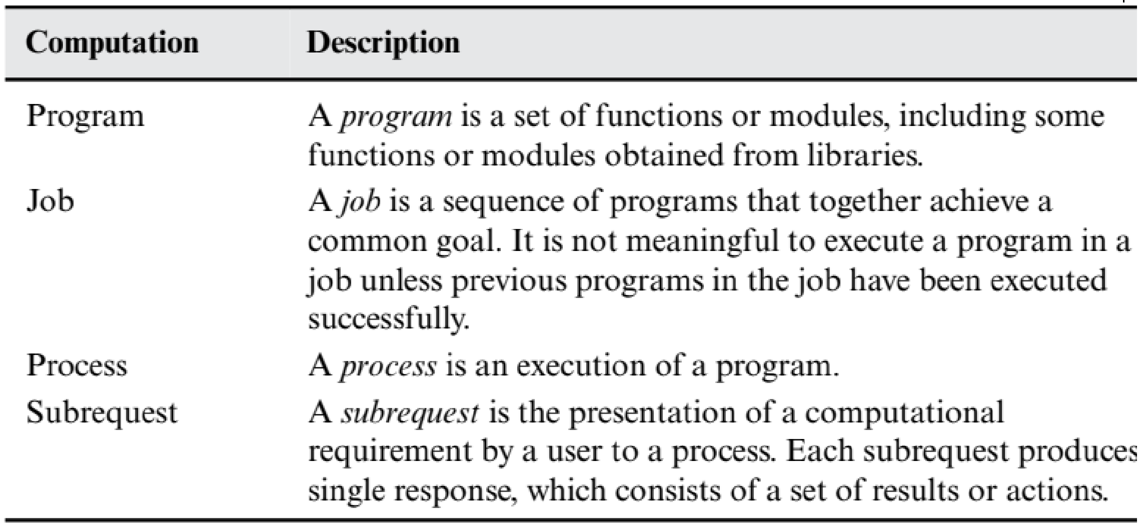


Figure 3.1 Turnaround time in a batch processing system.

6b. Various Computations in an OS — 4M

A *computing environment* consists of a computer system, its interfaces with other systems, and the services provided by its operating system to its users and their programs. Computing environments evolve continuously to provide better quality of service to users; however, the operating system has to perform more com plex tasks as computer systems become more powerful, their interfaces with I/O devices and with other computer systems become more complex, and its users demand new services.



Computation	Description
Program	A *program* is a set of functions or modules, including some functions or modules obtained from libraries.
Job	A *job* is a sequence of programs that together achieve a common goal. It is not meaningful to execute a program in a job unless previous programs in the job have been executed successfully.
Process	A *process* is an execution of a program.
Subrequest	A *subrequest* is the presentation of a computational requirement by a user to a process. Each subrequest produces single response, which consists of a set of results or actions.

7.	State Transition for a process diagram Process state The indicator that describes the nature of the current activity of a process. The kernel uses process states to simplify its own functioning, so the number of process states and their names may vary across OSs. However, most Oss use the four fundamental states described in Table 5.3. The kernel considers a process to be in the blocked state if it has made a resource request and the request is yet to be granted, or if it is waiting for some event to occur. A CPU should not be allocated to such a process until its wait is complete. The kernel would change the state of the process to ready when the request is granted or the event for which it is waiting occurs. Such a process can be considered for scheduling. The kernel would change the state of the process to running when it is dispatched. The state would be changed to terminated when execution of the process completes or when it is aborted by the kernel for some reason. A conventional computer system contains only one CPU, and so at most one process can be in the running state. There can be any number of processes in the blocked, ready, and terminated states. An OS may define more process states to simplify its own functioning or to support additional functionalities like swapping.	6M
	Explanation Process State Transitions A state transition for a process Pi is a change in its state. A state transition is caused by the occurrence of some event such as the start or end of an I/O operation. When the event occurs, the kernel determines its influence on activities in processes, and accordingly changes the state of an affected process. When a process Pi in the running state makes an I/O request, its state has to be changed to blocked until its I/O operation completes. At the end of the I/O operation, Pi's state is changed from blocked to ready because it now wishes to use the CPU. Similar state changes are made when a process makes some request that cannot immediately be satisfied by the OS. The process state is changed to blocked when the request is made, i.e., when the request event occurs, and it is changed to ready when the request is satisfied. The state of a ready process is changed to running when it is dispatched, and the state of a running process is changed to ready when it is preempted either because a higher-priority process became ready or because its time slice elapsed (see Sections 3.5.1 and 3.6). Table 5.4 summarizes causes of state transitions. Figure 5.4 diagrams the fundamental state transitions for a process. A new process is put in the ready state after resources required by it have been allocated. It may enter the running, blocked, and ready states a number of times as a result of events described in Table 5.4. Eventually it enters the terminated state.	4M

Table 5.4 Causes of Fundamental State Transitions for a Process

State transition	Description
ready → *running*	The process is dispatched. The CPU begins or resumes execution of its instructions.
blocked → *ready*	A request made by the process is granted or an event for which it was waiting occurs.
running → *ready*	The process is preempted because the kernel decides to schedule some other process. This transition occurs either because a higher-priority process becomes *ready*, or because the time slice of the process elapses.
running → *blocked*	The process in operation makes a system call to indicate that it wishes to wait until some resource request made by it is granted, or until a specific event occurs in the system. Five major causes of blocking are: • Process requests an I/O operation • Process requests a resource • Process wishes to wait for a specified interval of time • Process waits for a message from another process • Process waits for some action by another process.
running → *terminated*	Execution of the program is completed. Five primary reasons for process termination are: • *Self-termination:* The process in operation either completes its task or realizes that it cannot operate meaningfully and makes a "terminate me" system call. Examples of the latter condition are incorrect or inconsistent data, or inability to access data in a desired manner, e.g., incorrect file access privileges. • *Termination by a parent:* A process makes a "terminate P_i" system call to terminate a child process P_i, when it finds that execution of the child process is no longer necessary or meaningful. • *Exceeding resource utilization:* An OS may limit the resources that a process may consume. A process exceeding a resource limit would be aborted by the kernel. • *Abnormal conditions during operation:* The kernel aborts a process if an abnormal condition arises due to the instruction being executed, e.g., execution of an invalid instruction, execution of a privileged instruction, arithmetic conditions like overflow, or memory protection violation. • *Incorrect interaction with other processes:* The kernel may abort a process if it gets involved in a deadlock.

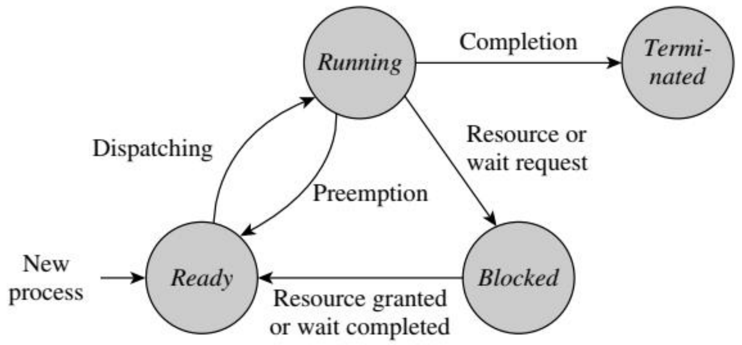


Figure 5.4 Fundamental state transitions for a process.