

<b>Sub:</b>	OPERATING SYSTEMS				<b>Code:</b>	15EC553			
<b>Date:</b>	20 / 09 / 2017	<b>Duration:</b>	90 mins	<b>Max Marks:</b>	50	<b>Sem:</b>	v	<b>Branch:</b>	ECE(A)/TCE (A &B)

**Note: Answer any five questions:**

1.	<p>Mention the operations of operating system. Discuss resource allocation and scheduling strategies and different computational structures with suitable examples.</p> <p><b>Mentioning 3 operations- 1M</b>  <b>Explanation about 2 types of resource allocation strategies- 5M</b>  <b>Explanation about 3 computational structures with suitable examples -4M</b></p> <ul style="list-style-type: none"> <li>• Fundamental goals of an operating system <ul style="list-style-type: none"> <li>– Programs</li> <li>– Resources</li> <li>– Security and protection</li> <li>– Scheduling</li> </ul> </li> <li>• Popular resource allocation strategies:</li> </ul> <p><b>Table 1.3 Resource Table for I/O Devices</b></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Resource name</th> <th style="text-align: left;">Class</th> <th style="text-align: left;">Address</th> <th style="text-align: left;">Allocation status</th> </tr> </thead> <tbody> <tr> <td>printer1</td> <td>Printer</td> <td>101</td> <td>Allocated to P<sub>1</sub></td> </tr> <tr> <td>printer2</td> <td>Printer</td> <td>102</td> <td>Free</td> </tr> <tr> <td>printer3</td> <td>Printer</td> <td>103</td> <td>Free</td> </tr> <tr> <td>disk1</td> <td>Disk</td> <td>201</td> <td>Allocated to P<sub>1</sub></td> </tr> <tr> <td>disk2</td> <td>Disk</td> <td>202</td> <td>Allocated to P<sub>2</sub></td> </tr> <tr> <td>cdw1</td> <td>CD writer</td> <td>301</td> <td>Free</td> </tr> </tbody> </table> <ul style="list-style-type: none"> <li>– <i>Resource partitioning</i> <ul style="list-style-type: none"> <li>• OS decides <i>a priori</i> what resources to allocate to each user program; divides system resources into <i>partitions</i> <ul style="list-style-type: none"> <li>– A resource partition is a collection of resources</li> </ul> </li> <li>• Resource table contains entries for partitions</li> <li>• Simple to implement, but lacks flexibility</li> </ul> </li> <li>– <i>Pool-based</i> <ul style="list-style-type: none"> <li>• OS allocates resources from a pool of resources <ul style="list-style-type: none"> <li>– Consults table and allocates the resource if it is free</li> </ul> </li> <li>• Less overhead of allocating and deallocating resources</li> <li>• Achieves more efficient use of resources</li> </ul> </li> </ul> <p>3 computational structures with suitable examples  Single programs  A sequence of single programs  Co-executing programs</p>	Resource name	Class	Address	Allocation status	printer1	Printer	101	Allocated to P <sub>1</sub>	printer2	Printer	102	Free	printer3	Printer	103	Free	disk1	Disk	201	Allocated to P <sub>1</sub>	disk2	Disk	202	Allocated to P <sub>2</sub>	cdw1	CD writer	301	Free	10M
Resource name	Class	Address	Allocation status																											
printer1	Printer	101	Allocated to P <sub>1</sub>																											
printer2	Printer	102	Free																											
printer3	Printer	103	Free																											
disk1	Disk	201	Allocated to P <sub>1</sub>																											
disk2	Disk	202	Allocated to P <sub>2</sub>																											
cdw1	CD writer	301	Free																											
2.	<p>Briefly explain the different classes of operating system, specifying the primary concerns and key concepts used. Explain the architectural support and techniques in multiprogramming systems.</p> <p><b>Explanation about classes of OS- 4M</b>  <b>Explanation about architectural support-3M</b>  <b>Explanation about techniques-3M</b></p>	10M																												

**Table 3.2 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

- An appropriate measure of performance of a multiprogramming OS is throughput
  - Ratio of the number of programs processed and the total time taken to process them
- OS keeps enough programs in memory at all times, so that CPU and I/O devices are not idle
  - Degree of multiprogramming: number of programs
  - Uses an appropriate program mix of CPU-bound programs and I/O-bound programs
  - Assigns appropriate priorities to CPU-bound and I/O-bound programs

#### Priority of Programs

In multiprogramming environments, an I/O-bound program should have a higher priority than a CPU-bound program

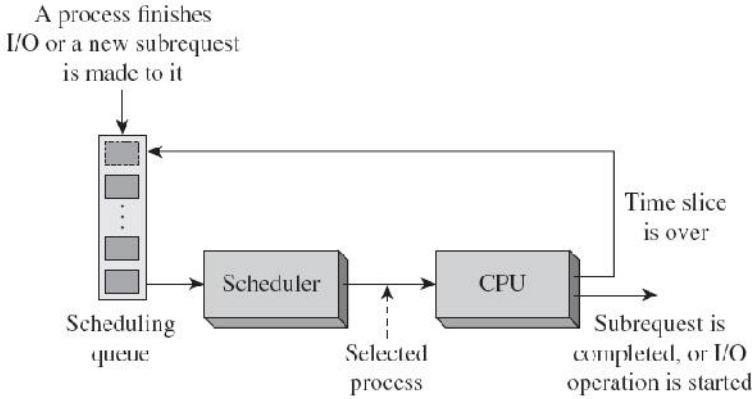
When an appropriate program mix is maintained, an increase in the degree of multiprogramming would result in an increase in throughput.

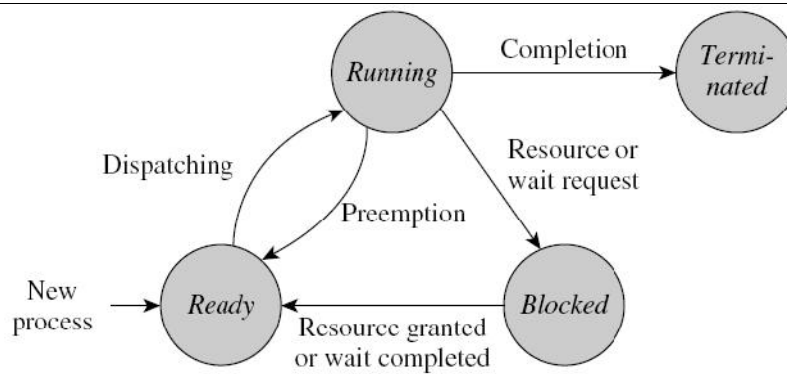
**Table 3.4 Architectural Support for Multiprogramming**

Feature	Description
DMA	The CPU initiates an I/O operation when an I/O instruction is executed. The DMA implements the data transfer involved in the I/O operation without involving the CPU and raises an I/O interrupt when the data transfer completes.
Memory protection	A program can access only the part of memory defined by contents of the <i>base register</i> and <i>size register</i> .
Kernel and user modes of CPU	Certain instructions, called <i>privileged instructions</i> , can be performed only when the CPU is in the kernel mode. A program interrupt is raised if a program tries to execute a privileged instruction when the CPU is in the user mode.

**Table 3.5 Techniques of Multiprogramming**

Technique	Description
Appropriate program mix	The kernel keeps a mix of CPU-bound and I/O-bound programs in memory, where <ul style="list-style-type: none"> <li>• A <i>CPU-bound program</i> is a program involving a lot of computation and very little I/O. It uses the CPU in long bursts—that is, it uses the CPU for a long time before starting an I/O operation.</li> <li>• An <i>I/O-bound program</i> involves very little computation and a lot of I/O. It uses the CPU in small bursts.</li> </ul>
Priority-based preemptive scheduling	Every program is assigned a priority. The CPU is always allocated to the highest-priority program that wishes to use it. A low-priority program executing on the CPU is preempted if a higher-priority program wishes to use the CPU.

3.	<p>(a) Explain the operation and memory management in Time sharing systems with schematic diagrams.</p> <p><b>Scheduling with time slicing diagram -2M</b>  <b>Explanation-2M</b>  <b>Memory management using swapping diagram with explanation--3M</b></p> <ul style="list-style-type: none"> <li>• Provide a quick response to user subrequests <ul style="list-style-type: none"> <li>– <i>Round-robin scheduling with time-slicing</i> <ul style="list-style-type: none"> <li>• Kernel maintains a <i>scheduling queue</i></li> <li>• If <i>time slice ( )</i> elapses before process completes servicing of a subrequest, kernel preempts it, moves it to end of queue, and schedules another process</li> </ul> </li> </ul> </li> </ul> <p>Implemented through a timer interrupt</p>  <p><b>Figure 3.6</b> A schematic of round-robin scheduling with time-slicing.</p> <p><b>Definition 3.6 Swapping</b> The technique of temporarily removing a process from the memory of a computer system.</p> <p>Kernel performs <i>swap-out</i> and <i>swap-in</i> operations</p>	7M
4.	<p>(b) Discuss the types of Real Time operating systems</p> <ul style="list-style-type: none"> <li>• <i>A hard real-time system meets response requirements under all conditions</i> <ul style="list-style-type: none"> <li>– It is typically <i>dedicated</i> to processing real-time applications</li> </ul> </li> <li>• <i>A soft real-time system makes best effort to meet response requirement of a real-time application</i> <ul style="list-style-type: none"> <li>– Cannot guarantee that it will be able to meet it <ul style="list-style-type: none"> <li>• Meets requirements in a probabilistic manner</li> </ul> </li> <li>– E.g., multimedia applications</li> </ul> </li> </ul>	3M
4.	<p>Explain the 4 fundamental states of a process with a neat diagram. State the causes of fundamental state transition from Running → Terminated state and from Running → Ready state.</p> <p><b>Process state diagram with explanation-5M</b>  <b>Explanation about Running → Ready state-1M</b>  <b>Explanation about Running → Terminated-4M</b></p>	10M



**Figure 5.4** Fundamental state transitions for a process.

**Table 5.3** Fundamental Process States

State	Description
<i>Running</i>	A CPU is currently executing instructions in the process code.
<i>Blocked</i>	The process has to wait until a resource request made by it is granted, or it wishes to wait until a specific event occurs.
<i>Ready</i>	The process wishes to use the CPU to continue its operation; however, it has not been dispatched.
<i>Terminated</i>	The operation of the process, i.e., the execution of the program represented by it, has completed normally, or the OS has aborted it.

- A *state transition* for a process is a change in its state
  - Caused by the occurrence of some event such as the start or end of an I/O operation

*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.

**Table 5.4** Causes of Fundamental State Transitions for a Process

State transition	Description
<i>running</i> → <i>terminated</i>	<p>Execution of the program is completed. Five primary reasons for process termination are:</p> <ul style="list-style-type: none"> <li>• <i>Self-termination</i>: 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.</li> <li>• <i>Termination by a parent</i>: A process makes a “terminate <math>P_i</math>” system call to terminate a child process <math>P_i</math>, when it finds that execution of the child process is no longer necessary or meaningful.</li> <li>• <i>Exceeding resource utilization</i>: An OS may limit the resources that a process may consume. A process exceeding a resource limit would be aborted by the kernel.</li> <li>• <i>Abnormal conditions during operation</i>: 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.</li> <li>• <i>Incorrect interaction with other processes</i>: The kernel may abort a process if it gets involved in a deadlock.</li> </ul>

5. Explain the process environment and Process Control Block (PCB) with neat diagrams.

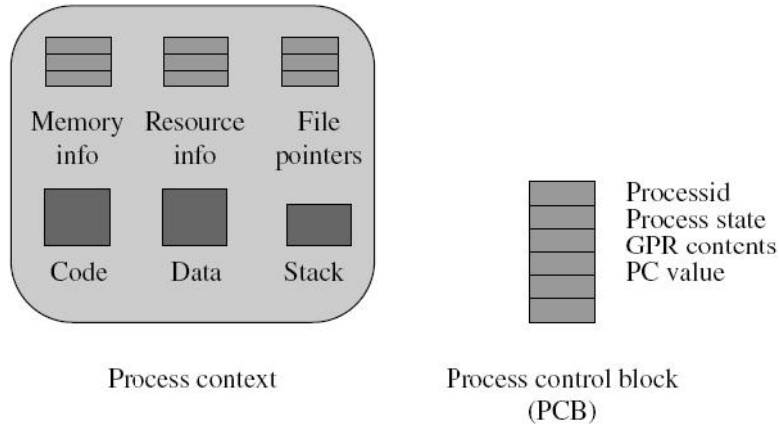
10M

**Process environment with PCB diagram-4M**

**Explanation-3M**

**PCB contents/Event handling -3M**

- Kernel allocates resources to a process and schedules it for use of the CPU
  - The kernel’s view of a process is comprised of the process context and the process control block



**Figure 5.6** Kernel's view of a process.

**Table 5.6** Fields of the Process Control Block (PCB)

PCB field	Contents
Process id	The unique id assigned to the process at its creation.
Parent, child ids	These ids are used for process synchronization, typically for a process to check if a child process has terminated.
Priority	The priority is typically a numeric value. A process is assigned a priority at its creation. The kernel may change the priority dynamically depending on the nature of the process (whether CPU-bound or I/O-bound), its age, and the resources consumed by it (typically CPU time).
Process state	The current state of the process.
PSW	This is a snapshot, i.e., an image, of the PSW when the process last got blocked or was preempted. Loading this snapshot back into the PSW would resume operation of the process. (See Fig. 2.2 for fields of the PSW.)
GPRs	Contents of the general-purpose registers when the process last got blocked or was preempted.
Event information	For a process in the <i>blocked</i> state, this field contains information concerning the event for which the process is waiting.
Signal information	Information concerning locations of signal handlers (see Section 5.2.6).
PCB pointer	This field is used to form a list of PCBs for scheduling purposes.

- Context save function:
  - Saves CPU state in PCB, and saves information concerning context
  - Changes process state from *running* to *ready*
- Scheduling function:
  - Uses process state information from PCBs to select a *ready* process for execution and passes its id to dispatching function
- Dispatching function:
  - Sets up context of process, changes its state to *running*, and loads saved CPU state from PCB into CPU
  - Flushes address translation buffers used by MMU

6. Consider two processes  $P_1$  and  $P_2$  whose CPU burst times are 16 and 30 ms respectively, while the I/O bursts are 110 and 60 ms, with time slice 10ms. Illustrate the operation of time sharing system with necessary timing diagram. Also explain time sharing operating systems with respect to memory management.

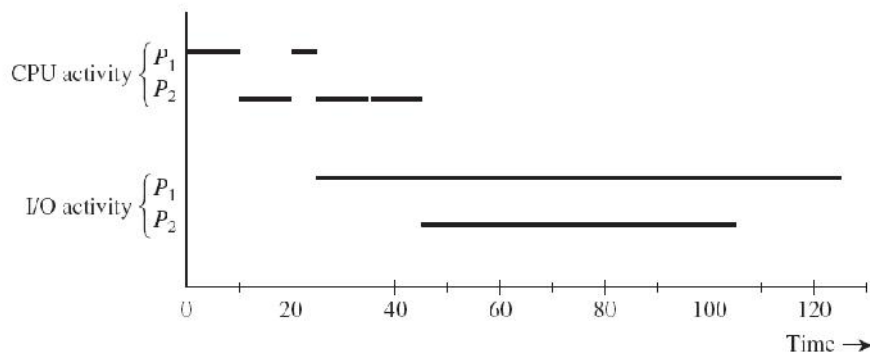
6M

**Solving the instance with table-2M**  
**Timing diagram-4 M**

Operation of Processes  $P_1$  and  $P_2$  in a time sharing system

Time	Scheduling List	Scheduled Program	Remarks
0	$P_1, P_2$	$P_1$	$P_1$ is preempted at 10ms
10	$P_2, P_1$	$P_2$	$P_2$ is preempted at 20ms
20	$P_1, P_2$	$P_1$	$P_1$ starts I/O at 26ms
26	$P_2$	$P_2$	$P_2$ is preempted at 36ms
36	$P_2$	$P_2$	$P_1$ starts I/O at 46ms
46	-	-	CPU is idle

Thus the response times are 136ms and 106ms respectively



**Figure 3.7** Operation of processes  $P_1$  and  $P_2$  in a time-sharing system.

(b) Discuss the 3 fundamental functions to control processes by the kernel when an Event occurs

4M

**Explanation -2M**

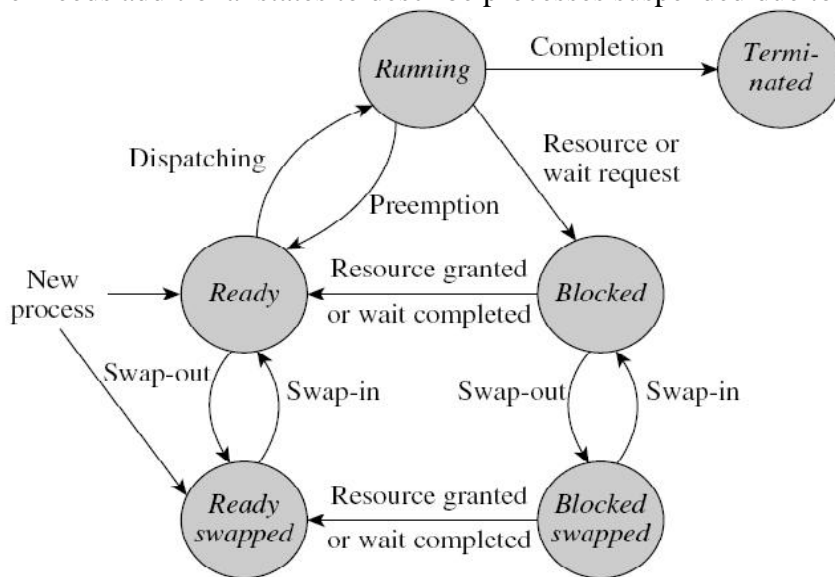
**Diagram-2M**

- Context save function:
  - Saves CPU state in PCB, and saves information concerning context
  - Changes process state from *running* to *ready*
- Scheduling function:
  - Uses process state information from PCBs to select a *ready* process for execution and passes its id to dispatching function
- Dispatching function:
  - Sets up context of process, changes its state to *running*, and loads saved CPU state from PCB into CPU
  - Flushes address translation buffers used by MMU.

(a) Explain briefly the suspended processes with a neat diagram.

- A kernel needs additional states to describe processes suspended due to swapping

4M



**Figure 5.5** Process states and state transitions using two swapped states.

7.

- (b) Define Turnaround Time and Throughput. Calculate the Mean turnaround time and Mean weighted turnaround time for the following processes using FCFS scheduling algorithm with suitable graph.

Process	P1	P2	P3	P4	P5
Arrival time	0	2	3	5	9
Service Time	3	3	2	5	3

**Definitions-2M**

**Final answer -4M**

**Throughput:** The average number of jobs, programs, processes or subrequests completed by a system in unit time.

**Turnaround time:** Time interval between submission of a job and its completion.

TIME	ID	TA	W	PROCESS IN SYSTEM	SCHEDULED
0	-	-	-	{P1}	P1
3	P1	3	1.0	{P2,P3}	P2
6	P2	4	1.33	{P3,P4}	P3
8	P3	5	2.50	{P4}	P4
13	P4	8	1.60	{P5}	P5
16	P5	7	2.33	{}	-

**Turn around Time=5.40 sec**

**Weighted turnaroung time=1.75**

6M



--	--	--