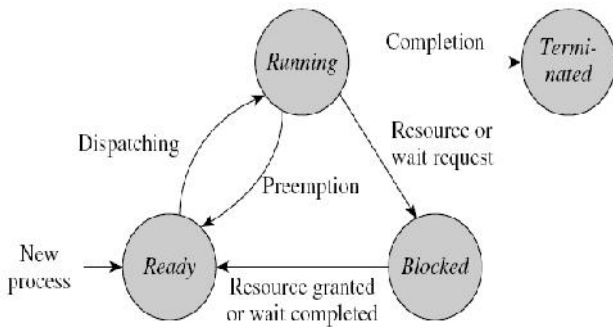


Sub:	OPERATING SYSTEMS				Code:	10EC65			
Date:	9 / 05 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	VI A & B	Branch:	ECE

Note: Answer any five questions:

1.	<p>(a) What is a process state? Explain the various states of a process giving process state transition diagram.</p> <p>Definition of process state: 1M Process state diagram with transitions: 4M Explanation about the various states : 3M</p> <p>Definition: An execution of a program using the resources allocated to it.</p> <div style="text-align: center;">  <pre> graph TD NP[New process] --> R[Ready] R -- Dispatching --> Run[Running] Run -- Preemption --> R Run -- "Resource or wait request" --> B[Blocked] B -- "Resource granted or wait completed" --> R Run -- Completion --> T[Terminated] </pre> </div> <p>Figure 5.4 Fundamental state transitions for a process.</p> <p>Table 5.3 Fundamental Process States</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #e0e0e0;"> <th style="text-align: left;">State</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td><i>Running</i></td> <td>A CPU is currently executing instructions in the process code.</td> </tr> <tr> <td><i>Blocked</i></td> <td>The process has to wait until a resource request made by it is granted, or it wishes to wait until a specific event occurs.</td> </tr> <tr> <td><i>Ready</i></td> <td>The process wishes to use the CPU to continue its operation; however, it has not been dispatched.</td> </tr> <tr> <td><i>Terminated</i></td> <td>The operation of the process, i.e., the execution of the program represented by it, has completed normally, or the OS has aborted it.</td> </tr> </tbody> </table>	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.	8M
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.											
	<p>(b) Mention any 4 exceptional conditions in message passing</p> <ul style="list-style-type: none"> • Some exceptional conditions: <ol style="list-style-type: none"> 1. Destination process mentioned in <i>send</i> doesn't exist 2. Source process does not exist (symmetric naming) 3. <i>send</i> cannot be processed (out of buffer memory) 4. No message exists for process when it makes <i>receive</i> 5. A set of processes become deadlocked when one is blocked on a <i>receive</i> 	2M										

2.

(a) (a) Explain the 5 primary reasons for process termination.

6M

Explanation about the 5 reasons- 6M**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"> • <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. • <i>Termination by a parent</i>: 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. • <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. • <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. • <i>Incorrect interaction with other processes</i>: The kernel may abort a process if it gets involved in a deadlock.

(b) Explain the benefits of child processes.

Mentioning 4 benefits-4M

4M

Table 5.2 Benefits of Child Processes

Benefit	Explanation
Computation speedup	Actions that the primary process of an application would have performed sequentially if it did not create child processes, would be performed concurrently when it creates child processes. It may reduce the duration, i.e., running time, of the application.
Priority for critical functions	A child process that performs a critical function may be assigned a high priority; it may help to meet the real-time requirements of an application.
Guarding a parent process against errors	The kernel aborts a child process if an error arises during its operation. The parent process is not affected by the error; it may be able to perform a recovery action.

3.

10M

Explain Interprocess communication in unix operating systems using pipes, message queues and sockets

Diagrams of pipes ,queues, and sockets each 1 mark-3M

Explanation of pipes-2M

Explanation of message queues -2M

Explanation of sockets with various system calls-3M

- Three interprocess communication facilities:
 - *Pipe*:
 - Data transfer facility
 - Unnamed pipes can be used only by processes that belong to the same process tree
 - *Message queue*: analogous to a mailbox
 - Used by processes within “Unix system domain”
 - Access permissions indicate which processes can send or receive messages
 - *Socket*: one end of a communication path
 - Can be used for setting up communication paths between processes within the Unix system domain and within certain Internet domains
- One common feature: processes can communicate without knowing each other’s identities

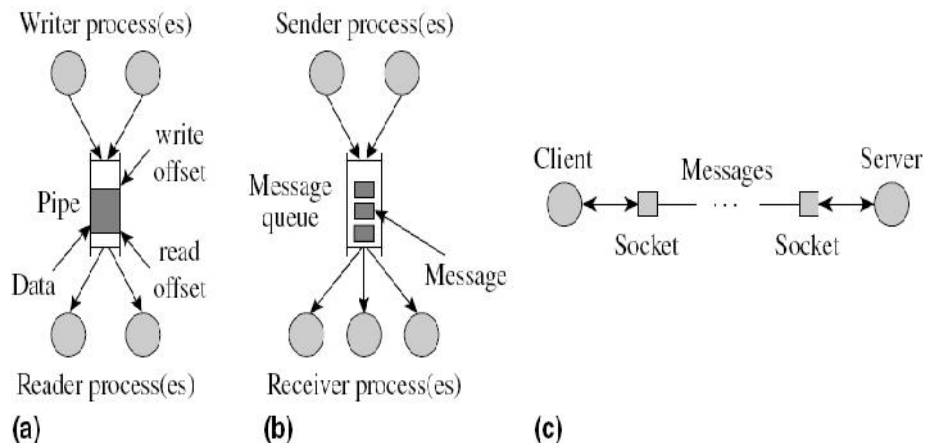


Figure 9.10 Interprocess communication in Unix: (a) pipe; (b) message queue; (c) socket.

Message Passing in Unix: Pipes

- FIFO mechanism for data transfer between processes called reader and writer processes
- Usually implemented in file system
 - But, data put into a pipe can be read only once
 - Removed from pipe when it is read by a process
- Two kinds of pipes: named and unnamed
 - Created through the system call *pipe*
 - A named pipe has an entry in a directory
- Like a file, but size is limited and kernel treats it as a queue

Message Passing in Unix: Message Queues

- Analogous to a mailbox
- Created and owned by one process
 - Creator specifies access permissions for send/receive
- Size specified at the time of its creation

Msgget(key,flag)
 Msgsnd(msgqid,msg_struct_ptr,count,flg)
 Msgrcv(msgqid,msg_struct_ptr,maxcount,type,flag)

Message Passing in Unix: Socket

A socket is one end of a communication path

- Can be used for interprocess communication within the Unix system domain and in the Internet domain
- Server can set up communication paths with many clients simultaneously
 - Typically, after *connect* call, server forks a new process to handle the new connection
 - Leaves original socket created by server process free to accept more connections
- Indirect naming: address (domain) used instead of process ids

- 1.S=SOCKET(DOMAIN,TYPE,PROTOCOL)
- 2.BIND(S, ADDR...)
- 3.LISTEN(S,...)
4. CONNECT(CS,SERVER_SOCKET_ADDR,SERVER_SOCKET_ADDRLLEN)
- 5.NEW_SOC=ACCEPT(S,CLIENT_SOCKET_ADDR,CLIENT_SOCKET_ADDRLLEN)
6. COUNT=SEND(S,MESSAGE,MESSAGE_LENGTH,FLAGS)

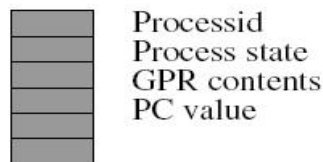
RECEIVE CALL

COUNT=RECV(S,MESSAGE,MESSAGE_LENGTH,FLAGS)
 7.CLOSE(S)

4. With the help of diagrams, discuss the contents of Process Control Block(PCB) and brief the events related to scheduling.

10M

Explaining 9 fields of PCB with diagram -5M
Explaining events related to scheduling with diagram-5M



Process control block (PCB)

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.

- Scheduling is the activity of selecting the next request to be serviced by a *server*
 - In an OS, a *request* is the execution of a job or a process, and the *server* is the CPU

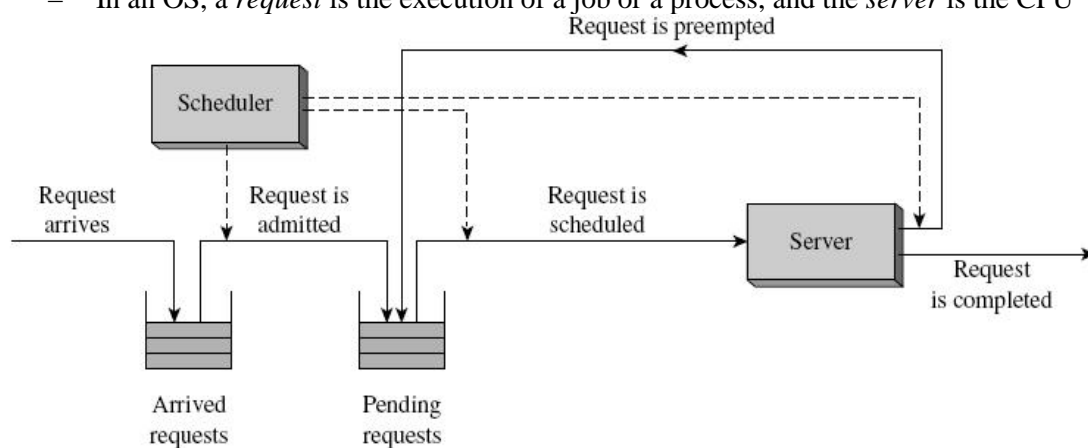


Figure 7.1 A schematic of scheduling.

5. Determine mean turnaround time and mean weighted turnaround time for following set of processes with appropriate graphs for **Shortest Request Next(SRN)** and **Highest Response Ratio Next(HRN)** scheduling algorithms.

10M

Processes	P ₁	P ₂	P ₃	P ₄	P ₅
Admission time	0	2	3	4	6
Service Time	3	3	5	2	3

SRN

Time	id	ta	w	Process in system	Scheduled process
0	-	-	-	P1	P1
3	P1	3	1	P2,P3	P2
6	P2	4	1.33	P3,P4	P4
8	P4	4	2	P3,P5	P5
11	P5	5	1	P3	P3
16	P3	13	2.60	-	-

Mean turnound time $t_a=5.8s$

Mean weighted turnound time=1.79

HRN

Time	id	ta	w	Response Ratios of the process					Scheduled process
				P1	P2	P3	P4	P5	
0	-	-	-	1.0					P1
3	P1	3	1		1.33	1.0			P2
6	P2	4	1.33			1.60	2.0		P4
8	P4	4	2.0			2.0		1.0	P3
13	P3	10	2.0					2.67	P5
16	P5	10	2.67						-

Mean turnound time $t_a=6.2s$

Mean weighted turnound time=1.93

6. Explain implementing Interprocess message passing by buffering and delivery of messages with neat diagrams.

10M

Explanation about the buffering of messages with diagram-5M
Explanation about the delivery of messages-2M

Sender process
Destination process
Message length
Message text or address
IMCB pointer

Figure 9.3 Interprocess message control block (IMCB).

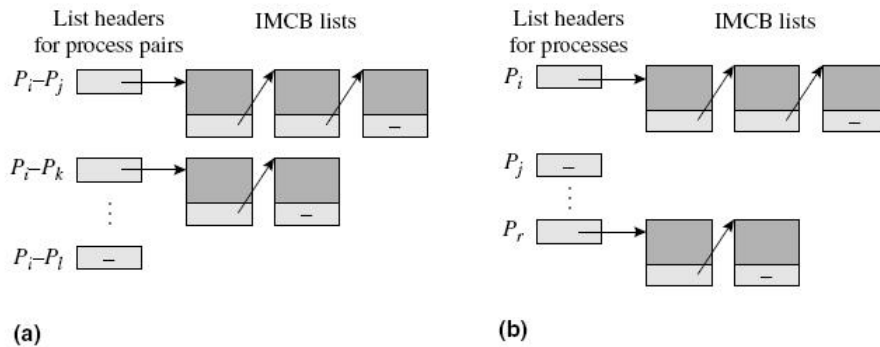


Figure 9.4 Lists of IMCBs for blocking sends in (a) symmetric naming; (b) asymmetric naming.

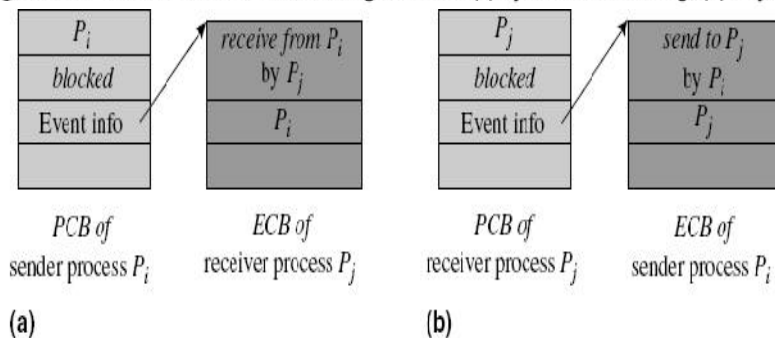


Figure 9.5 ECBs to implement symmetric naming and blocking sends (a) at send; (b) at receive.

- An *event control block* (ECB) has three fields:
 - Description of the anticipated event
 - Id of the process that awaits the event
 - An ECB pointer for forming ECB lists

7. (a) With a diagram explain the working of long, medium and short term scheduling in a time sharing system.

Diagram-3M
 Explanation-2M

- These schedulers perform the following functions:
 - *Long-term*: Decides when to admit an arrived process for scheduling, depending on:
 - Nature (whether CPU-bound or I/O-bound)
 - Availability of resources
 - Kernel data structures, swapping space
 - *Medium-term*: Decides when to swap out a process from memory and when to

- *Short-term*: Decides which *ready* process to service next on the CPU and for how long
 - Also called the *process scheduler*, or *scheduler*

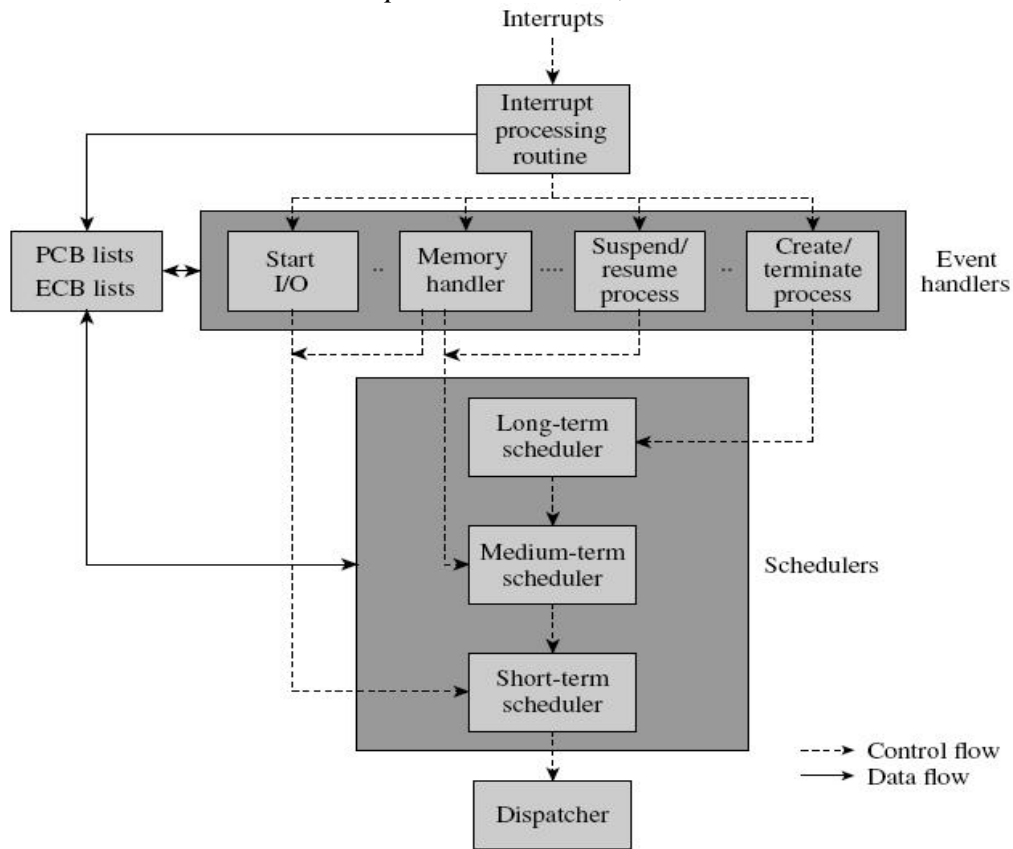


Figure 7.9 Event handling and scheduling.

(b) Determine the mean turnaround time and mean weighted turnaround time for following set of processes with appropriate graphs for **Round Robin (RR)** scheduling algorithm with time slice one second.

Processes	P ₁	P ₂	P ₃	P ₄	P ₅
Admission time	0	2	3	4	8
Service Time	3	3	5	2	3

5M

process	Completion	ta	w
P1	4	4	1.33
P2	9	7	2.33
P3	16	13	2.60
P4	10	6	3.00
P5	15	7	2.33

Mean turnound time ta=7.4s

Mean weighted turnound time=2.32