1 Process Management-I

(Introduction, Process, Threads & CPU Scheduling)

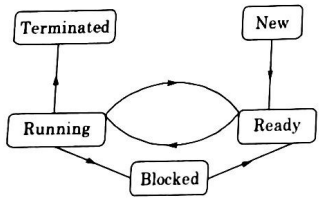
1.1 Which scheduling policy is most suitable for a time-shared operating systems?
(a) Shortest Job First
(b) Round Robin
(c) First come first serve
(d) Elevator **[1995 : 1 M]**

1.2 The sequence ..... is an optimal non-preemptive scheduling sequence for the following jobs which leaves the CPU idle for ..... unit(s) of time.

Job	Arrival time	Burst time
1	0.0	9
2	0.6	5
3	1.0	1

(a) {3, 2, 1}, 1 (b) {2, 1, 3}, 0
(c) {3, 2, 1}, 0 (d) {1, 2, 3}, 5
[1995 : 2 M]

1.3 The process state transition diagram in figure is representative of



(a) a batch operating system
(b) an operating system with a preemptive scheduler
(c) an operating system with a non-preemptive scheduler
(d) a uni-programmed operating system
[1996 : 1 M]

1.4 Which of the following is an example of spooled device?
(a) A line printer used to print the output of a number of jobs
(b) A terminal used to enter input data to a running program
(c) A secondary storage device in a virtual memory system
(d) A graphic display device **[1996 : 1 M]**

1.5 Four jobs to be executed on a single processor system arrive at time 0^+ in the order A, B, C, D. Their burst CPU time requirements are 4, 1, 8, 1 time units respectively. The completion time of A under round robin scheduling with time slice of one time unit is
(a) 10 (b) 4
(c) 8 (d) 9 **[1996 : 2 M]**

1.6 Which of the following is an example of a spooled device?
(a) The terminal used to enter the input data for the C program being executed
(b) An output device used to print the output of a number of jobs.
(c) The secondary memory device in a virtual storage system
(d) The swapping area on a disk used by the swapper. **[1998 : 1 M]**

1.7 Consider n processes sharing the CPU in a round-robin fashion. Assuming that each process switch takes s seconds, what must be the quantum size q such that the overhead resulting from process switching is minimized but at the same time each process is guaranteed to get its turn at the CPU at least every t seconds?

(a) $q \le \frac{t - ns}{n-1}$ (b) $q \ge \frac{t - ns}{n-1}$

(c) $q \le \frac{t - ns}{n+1}$ (d) $q \ge \frac{t - ns}{n+1}$

[1998 : 1 M]

1.8 Four jobs are waiting to be run. Their expected run times are 6, 3, 5 and x in what order should they be run to minimize the average response time? **[1998 : 2 M]**

1.9 System calls are usually invoked by using
(a) a software interrupt
(b) polling
(c) an indirect jump
(d) a privileged instruction **[1999 : 1 M]**

1.10 A multi-user, multi-processing operating system cannot be implemented on hardware that does not support
(a) Address translation
(b) DMA for disk transfer
(c) At least two modes of CPU execution (privileged and non-privileged)
(d) Demand paging **[1999 : 2 M]**

1.11 Which of the following actions is/are typically not performed by the operating system when switching context from process A to process B?
(a) Saving current register values and restoring saved register values for process B.
(b) Changing address translation tables.
(c) Swapping out the memory image of process A to the disk.
(d) Invalidating the translation look-aside buffer. **[1999 : 2 M]**

1.12 A processor needs software interrupt to
(a) Test the interrupt system of the processor
(b) Implement co-routines
(c) Obtain system services which need execution of privileged instructions
(d) Return from subroutine **[2001 : 1 M]**

1.13 A CPU has two modes-privileged and non-privileged. In order to change the mode from privileged to non-privileged
(a) a hardware interrupt is needed
(b) a software interrupt is needed
(c) a privileged instruction (which does not generate an interrupt) is needed
(d) a non-privileged instruction (which does not generate an interrupt) is needed **[2001 : 1 M]**

1.14 Consider a set of n tasks with known runtimes $r_1, r_2, ..., r_n$ to be run on a uniprocessor machine. Which of the following processor scheduling algorithms will result in the maximum throughput?
(a) Round-Robin
(b) Shortest-Job-First
(c) Highest-Response-Ratio-Next
(d) First-Come-First-Served **[2001 : 1 M]**

1.15 Which of the following scheduling algorithms is non-preemptive?
(a) Round Robin
(b) First-in-First-out
(c) Multilevel Queue Scheduling
(d) Multilevel Queue Scheduling with Feedback **[2001 : 1 M]**

1.16 Which of the following does not interrupt a running process?
(a) A device (b) Timer
(c) Scheduler process (d) Power failure **[2001 : 2 M]**

1.17 Which combination of the following features will suffice to characterize an OS as a multi-programmed OS? (i) more than one program may be loaded into main memory at the same time for execution. (ii) If a program waits for certain events such as I/O, another program is immediately scheduled for execution. (iii) If the execution of a program terminates, another program is immediately scheduled for execution
(a) (i) (b) (i) and (ii)
(c) (i) and (iii) (d) (i), (ii) and (iii) **[2002 : 2 M]**

1.18 Draw the process state transition diagram of an OS in which (i) each process is in one of the five states: created, ready, running, blocked (i.e. sleep or wait), or terminated, and (ii) only non-preemptive scheduling is used by the OS. Label the transitions appropriately. **[2002 : 2 M]**

1.19 A uni-processor computer system only has two processes, both of which alternate 10 ms CPU bursts with 90 ms I/O bursts. Both the processes were created at nearly the same time. The I/O of both processes can proceed in parallel. Which of the following scheduling strategies will result in the least CPU utilization (over a long period of time) for this system?
(a) First come first served scheduling
(b) Shortest remaining time first scheduling
(c) Static priority scheduling with different priorities for the two processes
(d) Round robin scheduling with a time quantum of 5 ms. **[2003 : 2 M]**

1.20 Consider the following statements with respect to user-level threads and kernel-supported threads
(i) Context switch is faster with Kernel-supported threads
(ii) For user-level threads, a system call can block the entire process
(iii) Kernel-supported threads can be scheduled independently
(iv) User-level threads are transparent to the Kernel
Which of the above statements are true?
(a) (ii), (iii) and (iv) only
(b) (ii) and (iii) only
(c) (i), and (iii) only
(d) (i) and (ii) only **[2004 : 1 M]**

Process	Arrival Time	Burst Time
P1	0	12
P2	2	4
P3	3	6
P4	8	5

The average waiting time (in milliseconds) of the processes is _____. **[2014 (Set-3) : 2 M]**

1.51 Consider a uniprocessor system executing three tasks T_1, T_2 and T_3, each of which is composed of an infinite sequence of jobs (or instances) which arrive periodically at intervals of 3, 7 and 20 milliseconds, respectively. The priority of each task is the inverse of its period, and the available tasks are scheduled in order of priority, with the highest priority task scheduled first. Each instance of T_1, T_2 and T_3 requires an execution time of 1, 2 and 4 milliseconds, respectively. Given that all tasks initially arrive at the beginning of the 1st millisecond and task preemptions are allowed, the first instance of T_3 completes its execution at the end of _____ milliseconds. **[2015 (Set-1) : 2 M]**

1.52 The maximum number of processes that can be in Ready state for a computer system with n CPUs is

(a) n (b) n^2
(c) 2^n (d) Independent of n

[2015 (Set-3) : 1 M]

1.53 For the processes listed in the following table, which of the following scheduling schemes will give the lowest average turnaround time?

Process	Arrival Time	Processing Time
A	0	3
B	1	6
C	4	4
D	6	2

(a) First Come First Serve
(b) Non-preemptive Shortest Job First
(c) Shortest Remaining Time
(d) Round-Robin with Quantum value two

[2015 (Set-3) : 2 M]

1.54 Consider an arbitrary set of CPU-bound processes with unequal CPU burst lengths submitted at the same time to a computer system. Which one of the following process scheduling algorithms would minimize the average waiting time in the ready queue?

(a) Shortest remaining time first
(b) Round-robin with time quantum less than the shortest CPU burst
(c) Uniform random
(d) Highest priority first with priority proportional to CPU burst length

[2016 (Set-1) : 1 M]

1.55 Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is preemptive shortest remaining-time first.

Process	Arrival Time	Burst Time
P_1	0	10
P_2	3	6
P_3	7	1
P_4	8	3

The average turn around time of these processes is _____ milliseconds.

[2016 (Set-2) : 2 M]

1.56 Threads of a process share

(a) global variables but not heap
(b) heap but not global variables
(c) neither global variables nor heap
(d) both heap and global variables

[2017 (Set-1) : 1 M]

1.57 Consider the following CPU processes with arrival times (in milliseconds) and length of CPU bursts (in milliseconds) as given below:

Process	Arrival time	Burst time
P_1	0	7
P_2	3	3
P_3	5	5
P_4	6	2

If the pre-emptive shortest remaining time first scheduling algorithm is used to schedule the processes, then the average waiting time across all processes is _____ milliseconds.

[2017 (Set-1) : 1 M]

1.58 Which of the following is/are shared by all the threads in a process?

I. Program counter II. Stack
III. Address space IV. Registers

(a) I and II only (b) III only
(c) IV only (d) III and IV only

[2017 (Set-2) : 1 M]

1.59 Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority (0 is the highest priority) shown below. None of the processes have I/O burst time.

Process	Arrival Time	Burst Time	Priority
P_1	0	11	2
P_2	5	28	0
P_3	12	2	3
P_4	2	10	1
P_5	9	16	4

The average waiting time (in milliseconds) of all the processes using preemptive priority scheduling algorithm is _____.

[2017 (Set-2) : 2 M]

1.60 Consider the following C program is executed on a Unix/Linux system:

```
#include<unistd.h>
int main()
{
    int i;
    for (i = 0; i < 10; i++)
        if (i % 2 == 0) fork( );
    return 0;
}
```

The total number of child processes created is _____.

[2019 : 1 M]

1.61 Consider the following four processes with arrival times (in milliseconds) and their length of CPU burst (in milliseconds) as shown below:

Process	P_1	P_2	P_3	P_4
Arrival time	0	1	3	4
CPU time	3	1	3	Z

These processes are run on a single processor using preemptive Shortest Remaining Time First scheduling algorithm. If the average waiting time of the processes is 1 millisecond, then the value of Z is _____.

[2019 : 2 M]

1.62 Consider the following statements about process state transitions for a system using preemptive scheduling.

I. A running process can move to ready state.
II. A ready process can move to running state.
III. A blocked process can move to running state.
IV. A blocked process can move to ready state.

Which of the above statements are TRUE?

(a) I, II, III and IV
(b) II and III only
(c) I, II and III only
(d) I, II and IV only

[2020 : 1 M]

1.63 Consider the following set of processes, assumed to have arrived at time 0. Consider the CPU scheduling algorithms Shortest Job First (SJF) and Round Robin (RR). For RR, assume that the processes are scheduled in the order P_1, P_2, P_3, P_4.

Processes	P_1	P_2	P_3	P_4
Burst time (in ms)	8	7	2	4

If the time quantum for RR is 4 ms, then the absolute value of the difference between the average turnaround times (in ms) of SJF and RR (round off to 2 decimal places) is _____.

[2020 : 2 M]

1.64 Three processes arrive at time zero with CPU bursts of 16, 20 and 10 milliseconds. If the scheduler has prior knowledge about the length of the CPU bursts, the minimum achievable average waiting time for these three processes in a non-preemptive scheduler (round to nearest integer) is _____ milliseconds.

[2021 (Set-1) : 1 M]

1.65 Which of the following standard C library functions will always invoke a system call when executed from a single-threaded process in a UNIX/Linux operating system?

(a) exit (b) strlen
(c) sleep (d) malloc

[2021 (Set-1) : 1 M]

1.66 Which of the following statement(s) is/are correct in the context of CPU scheduling?

(a) The goal is to only maximize CPU utilization and minimize throughput.
(b) Turnaround time includes waiting time.
(c) Round-robin policy can be used even when the CPU time required by each of the processes is not known apriori.
(d) Implementing preemptive scheduling needs hardware support.

[2021 (Set-2) : 1 M]

1.67 Consider four processes P, Q, R, and S scheduled on a CPU as per round robin algorithm with a time quantum of 4 units. The processes arrive in the order P, Q, R, S, all at time $t = 0$. There is exactly one context switch from S to Q, exactly one context switch from R to Q, and exactly two context switches from Q to R. There is no context switch from S to P. Switching to a ready process after the termination of another process is also

considered a context switch. Which one of the following is NOT possible as CPU burst time (in time units) of these processes?

(a) P = 4, Q = 10, R = 6, S = 2
(b) P = 2, Q = 9, R = 5, S = 1
(c) P = 4, Q = 12, R = 5, S = 4
(d) P = 3, Q = 7, R = 7, S = 3

[2022 : 2 M]

1.68 Which one or more of the following need to be saved on a context switch from one thread (T1) of a process to another thread (T2) of the same process?

(a) Page table base register
(b) Stack pointer
(c) Program counter
(d) General purpose registers

[2023 : 1 M]

1.69 Which one or more of the following options guarantee that a computer system will transition from user mode to Kernel mode?

(a) Function call
(b) Malloc call
(c) Page fault
(d) System call

[2023 : 1 M]

1.70 Which one or more of the following CPU scheduling algorithms can potentially cause starvation?

(a) First-in First-Out
(b) Round Robin
(c) Priority Scheduling
(d) Shortest Job First

[2023 : 1 M]

1.71 Which of the following statements about threads is/are TRUE?

(a) Threads can only be implemented in Kernel space
(b) Each thread has its own file descriptor table for open files
(c) Threads belonging to a process are by default not protected from each other
(d) All the threads belonging to a process share a common stack

[2024 (Set-1) : 1 M]

1.72 Which of the following process state transitions is/are NOT possible?

(a) Waiting to Running
(b) Ready to Waiting
(c) Running to Terminated
(d) Running to Ready

[2024 (Set-1) : 1 M]

1.73 Consider the following two threads T1 and T2 that update two shared variables a and b. Assume that initially $a = b = 1$. Though context switching between threads can happen at any time, each statement of T1 or T2 is executed atomically without interruption.

T1	T2
$a = a + 1;$	$b = 2 * b;$
$b = b + 1;$	$a = 2 * a;$

Which one of the following options lists all the possible combinations of values of a and b after both T1 and T2 finish execution?

(a) $(a = 3, b = 4)$; $(a = 4, b = 3)$; $(a = 3, b = 3)$
(b) $(a = 4, b = 4)$; $(a = 4, b = 3)$; $(a = 3, b = 4)$
(c) $(a = 2, b = 2)$; $(a = 2, b = 3)$; $(a = 3, b = 4)$
(d) $(a = 4, b = 4)$; $(a = 3, b = 3)$; $(a = 4, b = 3)$

[2024 (Set-1) : 2 M]

1.74 Consider a process P running on a CPU. Which one or more of the following events will always trigger a context switch by the OS that results in process P moving to a non-running state (e.g., ready, blocked)?

(a) A timer interrupt is raised by the hardware.
(b) P tries to access a page that is in the swap space, triggering a page fault.
(c) An interrupt is raised by the disk to deliver data requested by some other process.
(d) P makes a blocking system call to read a block of data from the disk.

[2024 (Set-2) : 1 M]

1.75 Consider a multi-threaded program with two threads T1 and T2. The threads share two semaphores: s1 (initialized to 1) and s2 (initialized to 0). The threads also share a global variable x (initialized to 0). The threads execute the code shown below.

```
// code of T1          // code of T2
wait (s1);             wait (s1);
x = x + 1;             x = x + 1;
print(x);              print(x);
wait(s2);              signal(s2);
signal(s1);            signal(s1);
```

Which of the following outcomes is/are possible when threads T1 and T2 execute concurrently?

(a) T2 runs first and prints 1, T1 runs next and prints 2
(b) T2 runs first and prints 1, T1 does not print anything (deadlock)
(c) T1 runs first and prints 1, T2 runs next and prints 2
(d) T1 runs first and prints 1, T2 does not print anything (deadlock)

[2024 (Set-2) : 2 M]

1.76 Consider a single processor system with four processes A, B, C, and D, represented as given below, where for each process the first value is its arrival time, and the second value is its CPU burst time.

A(0, 10), B(2, 6), C(4, 3) and D(6, 7)

Which one of the following options gives the average waiting times when preemptive Shortest Remaining Time First (SRTF) and Non-Preemptive Shortest Job First (NP-SJF) CPU scheduling algorithms are applied to the processes?

(a) SRTF = 6, NP-SJF = 7
(b) SRTF = 7, NP-SJF = 7.5
(c) SRTF = 7, NP-SJF = 8.5
(d) SRTF = 6, NP-SJF = 7.5

[2024 (Set-2) : 2 M]

■■■■

Answers Process Management-I

1.1	(b)	1.2	(a)	1.3	(b)	1.4	(a)	1.5	(d)	1.6	(b)	1.7	(a)	1.8	Sol.	1.9	(a)		
1.10	(c)	1.11	(c)	1.12	(c)	1.13	(d)	1.14	(b)	1.15	(b)	1.16	(a)	1.17	(d)	1.18	Sol.		
1.19	(d)	1.20	(a)	1.21	(a)	1.22	(c)	1.23	(a)	1.24	(d)	1.25	(c)	1.26	(d)	1.27	(b)		
1.28	(b)	1.29	(b)	1.30	(b)	1.31	(a)	1.32	(a)	1.33	(d)	1.34	(b)	1.35	(c)	1.36	(b)		
1.37	(b)	1.38	(c)	1.39	(d)	1.40	(c)	1.41	(d)	1.42	(c)	1.43	(a)	1.44	(c)	1.45	(c)		
1.46	(b)	1.47	(d)	1.48	(7.2)	1.49	(1000)	1.50	(5.5)	1.51	(12)	1.52	(d)	1.53	(c)	1.54	(a)		
1.55	(8.25)	1.56	(d)	1.57	(3)	1.58	(b)	1.59	(29)	1.60	(31)	1.61	(2)	1.62	(d)	1.63	(5.25)		
1.64	(12)	1.65	(a, c)	1.66	(b, c, d)	1.67	(d)	1.68	(b, c, d)	1.69	(c, d)	1.70	(c, d)	1.71	(c)	1.72	(a, b)		
1.73	(d)	1.74	(b, d)	1.75	(a, d)	1.76	(d)												

Process	Priority	CPU Time	Arrival Time	TAT
P1	10	20 sec	00.00.05	20
P2	9	10 sec	00.00.03	30
P3	8	15 sec	00.00.00	45

So, Turn Around Time of P2 is 30.

Using non-preemptive priority scheduling.

Gantt chart:

P3	P1	P2

0 15 35 45

Process	Priority	CPU Time	Arrival Time	TAT
P1	10	20 sec	00.00.05	30
P2	9	10 sec	00.00.03	42
P3	8	15 sec	00.00.00	15

So, Turn Around Time of P2 is 42.

1.25 (c)

By following I and II conditions the process P_1 will get both the resouces R_1 and R_2.

If R_1 and R_2 are allocated to the Process P_1, then it will complete it's job and release it. After that process P_2 will get both the resources and complete it's Job.

1.26 (d)

Fork( ) returns 0 is child process and process ID of child process is parent process.

In child(x), a = a + 5

In paraent(u), a = a – 5. Therefore, x = u + 10

The physical addresses of parent and child processes will be different, v ≠ y.

1.27 (b)

Let three processes are P_1, P_2 and P_3

Process	Arrival Time	Burst Time
P_1	0	10
P_2	2	20
P_3	6	30

The Gantt chart for SRTF scheduling algorithm is

P_1	P_2	P_3

0 10 30 60

So there is only two context switches, at time unit 10 context switch from P_1 to P_2 and at time unit 30 context switch from P_2 to P_3.

1.28 (b)

If it were a preemptive scheduling then there would have been a transition from Running state to Ready state. So it is non-preemptive scheduling.

1.29 (b)

At t = 0 only P1 is present so execute it for 1 unit (**remaining CPU time of P1 = 0, now it will perform I/O**).

At t = 1 no process is available for CPU so CPU will remain idle and P1 will perform I/O (**remaining I/O time of P1 = 4**)

At t = 2 P2 arrives and is available for CPU so execute it for 1 unit and simultaneously P_1 will do I/O (**remaining CPU time of P2 = 2 & remaining I/O time of P1 = 3**).

At t = 3 P3 also arrives and it is having highest priority among all processes available for CPU so we can execute P3 for its complete CPU burst (i.e 2 unit) as all processes has arrived and simultaneously P1 will do I/O(**remaining CPU time of P3 = 0, now P3 will perform I/O & remaining I/O time of P1 = 1**).

At t = 5 only P2 is available for CPU so it will execute for 1 unit and P1 and P3 will perform I/O(**remaining CPU time of P2 = 1, remaining I/O time of P3 = 2, remaining I/O time of P1 = 0, now P1 will perform CPU**).

At t = 6 P2 and P1 are available for CPU but P1 will be selected because it is having highest priority then P2 so execute P1 for 1 unit (**remaining CPU time of P1 = 2 & remaining I/O time of P3 is 1**)

At t = 7 again P1 will be executed for 1 unit (**remaining CPU time of P1 = 1 & remaining I/O time of P3 = 0, now P3 will perform CPU**).

At t = 8 now P1, P2 & P3 are available for CPU so P3 will be selected based on highest priority and will be executed for 1 unit (**remaining CPU time of P3 = 0, so P3 completed at t = 9**).

At t = 9 P1 will be executed on CPU for 1 unit (**remaining CPU time of P1 = 0, so P1 completed at t = 10**).

At t = 10 only P2 is available for CPU so it will execute its remaining CPU burst (i.e 1 unit) (**remaining CPU time of P2 = 0, now P2 will perform I/O.**)

At t = 11 P2 will perform I/O for 3 units and CPU will remain IDLE.

At t = 14 P2 will perform CPU again and P2 **is completed at t = 15**

P1	IDLE	P2	P3	P2	P1	P1	P3	P1	P2	IDLE	P2
0	1	2	3	5	6	7	8	9	10	11	14 15

1.30 (b)

Process	Total burst time	I/O time	CPU time	I/O time
P1	10	2	7	1
P2	20	4	14	2
P3	30	6	21	3

The Gantt chart is

Idle	P1	P2	P3	Idle

0 2 9 23 44 47

Total time spent = 47

Idle time = 2 + 3 = 5

% of idle time = $\frac{5}{47} \times 100 = 10.6\%$

1.31 (a)

Process id	Arrival Time	CPU burst
P_0	0	2
P_1	0	4
P_2	0	8

The Gantt chart for LRTF CPU scheduling algorithm.

P_0	P_1	P_2	P_2	P_2	P_2	P_2	P_2	P_1	P_2	P_1	P_2	P_0	P_1	P_2

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Turn around time for P_0 = 12 – 0 = 12

Turn around time for P_1 = 13 – 0 = 13

Turn around time for P_2 = 14 – 0 = 14

Average Turn around Time = $\frac{39}{3} = 13$

1.32 (a)

List-I

A. Gang Scheduling

B. Rate Monotonic Scheduling

C. Fair Show Scheduling

List-II

3. Thread Scheduling

2. Real-time Scheduling

1. Guaranteed Scheduling

1.33 (d)

In Kernel level threads, the blocking system call causes, the Kernel can schedule another thread in the application for execution. So statement (d) is false about Kernel level threads.

1.34 (b)

Process	Execution Time	Arrival Time
P_1	20	0
P_2	25	15
P_3	10	30
P_4	15	45

The Gantt chart for SRT Scheduling algorithm is

P_1	P_2	P_3	P_2	P_4

0 20 30 40 55 70

So the waiting time for

$P_2 = (20 - 15) + (40 - 30) = 5 + 10 = 15$

1.35 (c)

When time quantum used in round robin scheduling is more than maximum time required to execute any process then its behave like First Come First serve.

1.36 (b)

Statement (b) is not true because an ISR (Interrupt Service Routine) is invoked after completion of I/O in synchronous but not in asynchronous I/O.

1.37 (b)

fork( ) system call creates the child process initially number of processes is 0. After first fork( ), it creates a single process. After second fork( ), it creates one parent and two child processes. After n + 1 fork( ), the total number of processes is 2^n but we subtract the main process then total number of child processes is $2^n - 1$.

1.38 (c)

Transition C indicates that OS uses preemptive scheduling. II is also true.

1.39 (d)

In the shortest remaining time first scheduling, process will not get a chance if that one has the high remaining time so due to this, starvation take place.

Shortest remaining time first scheduling will cause starvation, is true because if a large CPU brust required process is there and if consider that all the processes which are coming have less CPU-brust then the former have to wait untill all finish execution gives starvation.

In preemptive scheduling, always one process preempt due to low priority then due to this starvation take place.
Response time = first response - process submission time
In the round robin every process first response gets after certain time quantum so due to this it give more better than FCFS in term of response time.

1.40 (c)

Process switching includes mode switching.
Context switching can occur only in Kernal mode.

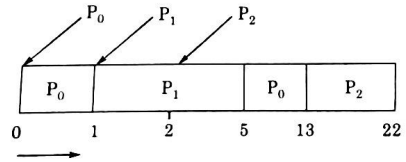
1.41 (d)

Interrupt from CPU temperature sensor is given top priority to protect system resources. When CPU temperature is too high, the BIOS initiate an interrupt and informs the Operating System. OS gives top priority to this interrupt and immediately shuts down the system.

1.42 (c)

(a) **False:** On per thread basics, OS maintains both CPU register state and stack. Hence option (a) is false, because only CPU register state mentioned.
(b) **False:** OS maintains a separate stack for each thread.
(c) **True:** OS does not maintain virtual memory state.
(d) **False:** OS does not maintain scheduling and accounting information.

1.43 (a)

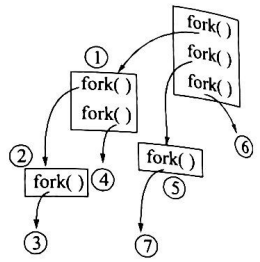


Average waiting time

$$= \frac{w_1 + w_2 + w_3}{3}$$

$$= \frac{(5-1)+0+(13-2)}{3} = 5.0 \text{ ms}$$

1.44 (c)



The number of child process created
$= 2^n - 1 = 2^3 - 1 = 7$
(where n is number of fork( ) statements)

1.45 (c)

RR Queue:

P1	P2	P1	P3	P2	P1	P3	P2
0	1	2	3	4	6	8	10

P1	P2	P1	P3	P2	P1	P3	P2

0 2 4 6 8 10 11 13 16

$\Rightarrow$ P1, P3, P2

FCFS: P1, P2, P3
Therefore option (c) is correct.
FCFS : P1, P2, P3
RR2 : P1, P3, P2

1.46 (b)

Lets take an example:

Process	Execution time	Arrival time
1	10	0
2	8	1
3	7	2
4	6	3

Consider scheduler schedule processes priority after S times units so order of execution will be $P_1\ P_2\ P_3\ P_4\ P_1\ P_2\ P_3\ P_4$ (S < execution time of any process) which is exactly same as round robin so answer is (b).

1.47 (d)

Kernel level threads shares the code segment.

1.48 (7.2)

	A.T	E.T
A	0	6
B	3	2
C	5	4
D	7	6
E	10	3

Using SRTF:

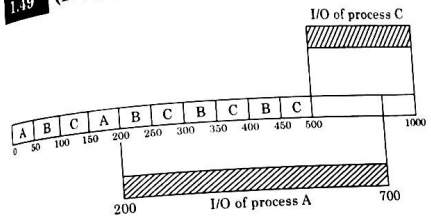
A	B	A	C	E	D

0 3 5 8 12 15 21

T.A.T (A) = 8 – 0 = 8
T.A.T (B) = 5 – 3 = 2
T.A.T (C) = 12 – 5 = 7
T.A.T (D) = 21 – 7 = 14
T.A.T (E) = 15 – 10 = 5

Average T.A.T $= \frac{8+2+7+14+5}{5} = 7.2$

1.49 (1000)



$\therefore$ At 1000 time units C completes its I/O.

1.50 (5.5)

SRTF:

P1	P2	P3	P4	P1

0 2 6 12 17 27

Waiting time of P1 = 17 – 2 = 15
Waiting time of P2 = 2 – 2 = 0
Waiting time of P3 = 6 – 3 = 3
Waiting time of P4 = 12 – 8 = 4

Average waiting time $= \frac{15+0+3+4}{4} = 5.5$

1.51 (12)

Periodic arrival times of T1: 0, 3, 6, 9, 12, 15, 18, 21,...

Priority of T1 $= \frac{1}{3}$, service time of $T_1 = 1$.

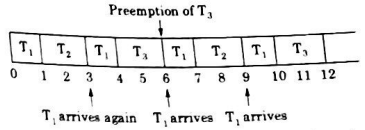
Periodic arrival times of T_2: 0, 7, 14, 21,...

Priority of $T_2 = \frac{1}{7}$, service time of $T_2 = 2$.

Periodic arrival times of T_3: 0, 20, 40,...

Priority of $T_3 = \frac{1}{20}$, service time of $T_3 = 4$.

T_1 has highest priority and T3 has lowest priority.



First instance of T_3 (4 units) completed at the end of 12 ms.

1.52 (d)

Maximum number of processes that can be in ready state is independent of number of processes (n).

1.53 (c)

Process	Arrival Time	B.T
A	0	3
B	1	6
C	4	4
D	6	2

(i) **FCFS:**

A	B	C	D

0 3 9 13 15

Avg TAT $= \frac{3+8+9+9}{4} = \frac{29}{4} = 7.25$

(ii) **Non-peemptive SJF:**

A	B	D	C

0 3 9 11 15

Avg TAT $= \frac{3+8+5+11}{4} = 6.75$

(iii) **SRTF:**

A	B	D	C	B

0 3 4 8 10 15

Avg TAT $= \frac{3+14+4+4}{4} = \frac{25}{4} = 6.25$

(iv) **RR:**

A	B	A	C	B	D	C	B

0 2 4 5 7 9 11 13 15

Avg TAT $= \frac{5+14+9+5}{4} = \frac{33}{4} = 8.25$

$\therefore$ SRTF has lowest turn around time.

1.54 **(a)**

To minimize the average waiting time, we need to select the shortest remaining time process first, because all are arriving at the same time, and they have unequal CPU burst times.

All other options will not minimize the waiting time. So, the answer is SRTF algorithm.

1.55 **(8.25)**

Gantt chart

P_1	P_2	P_3	P_2	P_4	P_1
0	3	7	8	10	13	20

Process Number	Arrival time	Burst time	Completion time	TAT
1	0	10	20	20
2	3	6	10	7
3	7	1	8	1
4	8	3	13	5
Average turn around time : 33/4 = 8.25				

Average turn around time is 8.25.

1.56 **(d)**

Generally every thread of a process have their own PC and stack. Both heap and global variables are shared by every thread of a process.

1.57 **(3)**

Using premptive SRTF algorithm Gantt chart will be,

P_1	P_2	P_4	P_1	P_3
0	3	6	8	12	17
		P_2	P_4	P_1	P_3

Turnaround time

$P_1 \rightarrow 12 - 0 = 12$

$P_2 \rightarrow 6 - 3 = 3$

$P_3 \rightarrow 17 - 5 = 12$

$P_4 \rightarrow 8 - 6 = 2$

Waiting time

$P_1 \rightarrow 12 - 7 = 5$

$P_2 \rightarrow 3 - 3 = 0$

$P_3 \rightarrow 12 - 5 = 7$

$P_4 \rightarrow 2 - 2 = 0$

$$\text{Average Waiting Time} = \frac{5+0+7+0}{4} = 3.0$$

1.58 **(b)**

All the threads share address space but other entities like, stack, PC, resisters are not shared and every thread will have its own.

1.59 **(29)**

Process	Arrival Time	Burst Time	Priority	C.T.	T.A.T.	W.T.
P_1	0	11	2	49	49	38
P_2	5	28	0 (high)	33	28	0
P_3	12	2	3	51	39	37
P_4	2	10	1	40	38	25
P_5	9	16	4	67	58	42
					Total	145

P_1	P_4	P_2	P_4	P_1	P_3	P_5
0	2	5	33	40	49	51	67

$$\text{Average waiting time} = \frac{145}{5} = 29$$

1.60 **(31)**

Number of child process created using n fork() calls $= 2^n - 1$

- fork(); statement inside the loop is executed when $i = 0, 2, 4, 6, 8$. So $n = 5$ times fork() is executed.

So, number of child processes created $= 2^5 - 1 = 31$

1.61 **(2)**

Assume, $Z = 2$

P_1	P_2	P_1	P_1	P_4	P_3
0	1	2	3	4	6	9

	Arrival time	CPU time	Completion time	Waiting time
P_1	0	3	4	1
P_2	1	1	2	0
P_3	3	3	9	3
P_4	4	$Z = 2$	6	0

$$\text{Average waiting time (WT)} = \frac{1+0+3+0}{4} = 1 \text{ ms}$$

Hence, $Z = 2$

1.62 **(d)**

Statement I, II and IV are correct.

1.63 **(5.25)**

SJF:

P_3	P_4	P_2	P_1
0	2	6	13	21

Turn Around Time (TAT) = (21 – 0) + (13 – 0) + (2 – 0) + (6 – 0)

$$\text{Average TAT} = \frac{42}{4} = 10.5$$

RR:

P_1	P_2	P_3	P_4	P_1	P_2
0	4	8	10	14	18	21

Turn Around Time (TAT) = (18 – 0) + (21 – 0) + (10 – 0) + (14 – 0)

= 18 + 21 + 10 + 14

$$\text{Average TAT} = \frac{63}{4} = 15.75$$

Hence, |SJF (TAT) – RR(TAT)|

= |10.5 – 15.75| = 5.25

1.64 **(12)**

WT	CT	P. No.	AT	BT
0	10	A	0	16
26	46	B	0	20
10	26	C	0	10

C	A	B
0	10	26	46

$$\text{Average writing time} = \frac{0+26+10}{3} = \frac{36}{3} = 12$$

1.66 **(b, c, d)**

- Goal is to maximize CPU utilization and maximize the throughput. So, statement (a) is false.
- Statement (b) is true, because turnaround time = completion time – arrival time and waiting time is included in this.
- Statement (c) is true because using time quantum, we can run the processes even if burst time is not known initially in round-robin.
- True for example, round robin scheduling requires hardware support which is timer.

1.67 **(d)**

P	Q	R	S	Q	R	Q

The above gantt chart satisfies all conditions given in the question the option (a), (b), (c) but not option (d). So, answer is option (d).

1.68 **(b, c, d)**

The status of a process or thread is saved via context switching. This makes it possible for the procedure to continue running later. During context switches, the registers (integer and floating-point), programme counter, condition registers (status register or flag register), and any other thread execution state must be stored. Registers, the stack pointer, and the programme counter must all be reloaded from the TCB of the new thread.

When the threads are from different processes, it is necessary to load the pointer for the top-level page table of the new address space in addition to saving and restoring the information mentioned above. (Notice that the process control block (PCB) already contains this top-level page table pointer from the previous process, so it is not necessary to preserve it.)

1.69 **(c, d)**

Page fault and system call required to change mode but others may not required.

1.70 **(c, d)**

In First Come First Serve (FCFS), if a process with a very large burst time comes before other processes, the other processes will have to wait for a very long time, but it is obvious that other processes will definitely get their chance to execute, so it is not likely that it will suffer from starvation.

There is no starvation in Round Robin because there is a defined time duration and every process will have a chance to run.

If higher priority processes keep coming in priority-based scheduling, low priority processes will starve.

If processes with short process times continue to arrive in Shortest Job First (SJF), processes with longer burst times will have to wait and starve.

■■■■

2 Process Management-II
(IPC, Synchronization and Concurrency)

2.1 A critical section is a program segment.
(a) Which should run in certain specified amount of time
(b) Which avoids deadlocks
(c) Where shared resources are accessed
(d) Which must be enclosed by a pair of semaphore operations, P and V
[1996 : 1 M]

2.2 A solution to the Dining Philosophers Problem which avoids deadlock is
(a) ensure that all philosophers pick up the left fork before the right fork
(b) ensure that all philosophers pick up the right fork before the left fork
(c) ensure that one particular philosophers picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork
(d) None of the above
[1996 : 2 M]

2.3 Each process Pi, i = 1 ... 9 is coded as follows
repeat
P(mutex)
{critical section}
V(mutex)
forever
The code for P_{10} is identical except that it uses V(mutex) in place of P(mutex). What is the largest number of processes that can be inside the critical section at any moment?
(a) 1 (b) 2
(c) 3 (d) None of these
[1997 : 2 M]

2.4 When the result of a computation depends on the speed of the processes involved there is said to be
(a) cycle stealing (b) race condition
(c) a time lock (d) a deadlock
[1998 : 1 M]

2.5 A counting semaphore was initialized to 10. Then 6 P (wait) operations and 4V (signal) operations were completed on this semaphore. The resulting value of the semaphore is
(a) 0 (b) 8
(c) 10 (d) 12 **[1998 : 1 M]**

2.6 Let m[0]...m[4] be mutexes (binary semaphores) and P[0]...P[4] be processes. Suppose each process P[i] executes the following:
wait (m[i]);wait (m[(i+1) mode 4]);
.....
release (m[i]); release (m[(i+1)mod 4]);
This could cause
(a) Thrashing
(b) Deadlock
(c) Starvation, but not deadlock
(d) None of the above **[2000 : 1 M]**

2.7 Suppose a processor does not have any stack pointer register. Which of the following statement is true?
(a) It cannot have subroutine call instruction
(b) It can have subroutine call instruction, but no nested subroutine calls
(c) Nested subroutine calls are possible, but interrupts are not
(d) All sequences of subroutine calls and also interrupts are possible **[2001 : 1 M]**

2.8 Where does the swap space reside?
(a) RAM (b) Disk
(c) ROM (d) On-chip cache
[2001 : 1 M]

2.9 Consider Peterson's algorithm for mutual exclusion between two concurrent processes i and j. The program executed by process is shown below:
repeat
flag[i]=true;
turn=j;
while (P) do no-op;
Enter critical section, perform actions, then exit critical section
flag[i]=false;
Perform other non-critical section actions.
Until false;
For the program to guarantee mutual exclusion, the predicate P in the while loop should be
(a) flag[j] = true and turn = i
(b) flag [j] = true and turn = j
(c) flag [i] = true and turn = j
(d) flag [i] = true and turn = i **[2001 : 2 M]**

Common Data for Q. 2.10 & 2.11:
Suppose we want to synchronize two concurrent processes P and Q using binary semaphores S and T. The code for the processes P and Q is shown below.

```
Process P:          Process Q:
while (1) {         while (1) {
  W:                  Y:
  print '0';          print '1';
  print '0';          print '1';
  X:                  Z:
}                   }
```

Synchronization statements can be inserted only at points W, X, Y and Z.

2.10 Which of the following will always lead to an output string with '001100110011'?
(a) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1
(b) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S initially 1, and T initially 0
(c) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1
(d) P(S) at W, V(T) at X, P(T) at Y, V(T) at Z, S initially 1, and T initially 0 **[2003 : 2 M]**

2.11 Which of the following will ensure that the output string never contains a substring of the form 01^n0 or 10^n1 where n is odd?
(a) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1
(b) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1
(c) P(S) at W, V(S) at X, P(S) at Y, V(S) at Z, S initially 1
(d) V(S) at W, V(T) at X, P(S) at Y, P(T) at Z, S and T initially 1 **[2003 : 2 M]**

2.12 Consider two processes P_1 and P_2 accessing the shared variables X and Y protected by two binary semaphores S_X and S_Y respectively, both initialized to 1. P and V denote the usual semaphore operators, where P decrements the semaphore value, and V increments the semaphore value. The pseudocode of P_1 and P_2 is as follows:

```
P1:                 P2:
while true do {     while true do{
  L1 : .....          L3 : .....
  L2 : .....          L4 : .....
  X = X + 1;          Y = Y + 1;
  Y = Y - 1;          X = X - 1;
  V(SX);              V(SY);
  V(SY); }            V(SX); }
```

In order to avoid deadlock, the correct operators at L_1, L_2, L_3 and L_4 are respectively
(a) $P(S_Y)$, $P(S_X)$, $P(S_X)$, $P(S_Y)$
(b) $P(S_X)$, $P(S_Y)$, $P(S_Y)$, $P(S_X)$
(c) $P(S_X)$, $P(S_X)$, $P(S_Y)$, $P(S_Y)$
(d) $P(S_X)$, $P(S_Y)$, $P(S_X)$, $P(S_Y)$
[2004 : 2 M]

2.13 The semaphore variables full, empty and mutex are initialized to 0, n and 1, respectively. Process P_1 repeatedly adds one item at a time to a buffer of size n, and process P_2 repeatedly removes one item at a time from the same buffer using the programs given below. In the programs, K, L, M and N are unspecified statements.

```
P1 : while (1)
{
   K;
   P(mutex);
   Add an item to the buffer;
   V(mutex);
   L;
}
P2 :  while (1)
{
   M;
   P(mutex);
   Remove an item from the buffer;
   V(mutex);
   N;
}
```

The statements K, L, M and N are respectively
(a) P(full), V(empty), P(full), V(empty)
(b) P(full), V(empty), P(empty), V(full)
(c) P(empty), V(full), P(empty), V(full)
(d) P(empty), V(full), P(full), V(empty)
[2004 : 2 M]

2.14 Given below is a program which when executed spawns two concurrent processes:

```
semaphore X : = 0 ;
/* Process now forks into concurrent processes
P1 & P2 */
P1: repeat forever      P2: repeat forever
V(X);                       P(X);
Compute;                    Compute;
P(X);                       V(X);
```

Consider the following statements about processes P1 and P2:
I. It is possible for process P1 to starve.
II. It is possible for process P2 to starve.
Which of the following holds?

(a) Both I and II are true.
(b) I is true but II is false.
(c) II is true but I is false
(d) Both I and II are false. **[2005 : 2 M]**

2.15 Two concurrent processes P1 and P2 use four shared resources R1, R2, R3 and R4, as shown below:

P1:	P2:
Compute:	Compute;
Use R1;	Use R1;
Use R2;	Use R2;
Use R3;	Use R3;
Use R4;	Use R4;

Both processes are started at the same time, and each resource can be accessed by only one process at a time. The following scheduling constraints exist between the access of resources by the processes:

- P2 must complete use of R1 before P1 gets access to R1.
- P1 must complete use of R2 before P2 gets access to R2.
- P2 must complete use of R3 before P1 gets access to R3.
- P1 must complete use of R4 before P2 gets access to R4.

There are no other scheduling constraints between the processes. If only binary semaphores are used to enforce the above scheduling constraints, what is the minimum number of binary semaphores needed?

(a) 1 (b) 2
(c) 3 (d) 4 **[2005 : 2 M]**

2.16 The atomic *fetch-and-set x, y* instruction unconditionally sets the memory location x to 1 and fetches the old value of x in y without allowing any intervening access to the memory location x. Consider the following implementation of P and V functions on a binary semaphore S.

```
void P(binary_semaphore *S) {
    unsigned y;
unsigned *x = & (S → value);
    do {
        fetch-and-set x, y;
    } while (y);
}
void V (binary_semphore *S) {
    S → value = 0;
}
```

Which one of the following is true?

(a) The implementation may not work if context switching is disabled in P
(b) Instead of using fetch-and-set, a pair of normal load/store can be used
(c) The implementation of V is wrong
(d) The code does not implement a binary semaphore **[2006 : 2 M]**

Common Data for Q. 2.17 & 2.18:

Barrier is a synchronization construct where a set of processes synchronizes globally i.e. each process in the set arrives at the barrier and waits for all others to arrive and then all processes leave the barrier. Let the number of processes in the set be three and S be a binary semaphore with the usual P and V functions. Consider the following C implementation of a barrier with line numbers shown on the left.

```
void barrier (void) {
 1 :  P (S) ;
 2 :  process_arrived + +;
 3 :  V (S) ;
 4 :  while (process_arrived ! = 3) ;
 5 :  P (S) ;
 6 :  process_left + + ;
 7 :  if (process_left = = 3){
 8 :        process_arrived = 0;
 9 :        process_left = 0;
10 :  }
11 :  V (S) ;
12 :  }
```

The variables process_arrived and process_left are shared among all processes and are initialized to zero. In a concurrent program all the three processes call the barrier function when they need to synchronize globally.

2.17 The above implementation of barrier is incorrect. Which one of the following is true?

(a) The barrier implementation is wrong due to the use of binary semaphore S
(b) The barrier implementation may lead to a deadlock if two barrier invocations are used in immediate succession
(c) Lines 6 to 10 need not be inside a critical section
(d) The barrier implementation is correct if there are only two processes instead of three **[2006 : 2 M]**

2.18 Which one of the following rectifies the problem in the implementation?

(a) Lines 6 to 10 are simply replaced by process_arrived

(b) At the beginning of the barrier the first process to enter the barrier waits until process_arrived becomes zero before proceeding to execute P(S)
(c) Context switch is disabled at the beginning of the barrier and re-enabled at the end
(d) The variable process_left is made private instead of shared **[2006 : 2 M]**

2.19 Processes P1 and P2 use critical_flag in the following routine to achieve mutual exclusion. Assume that critical_flag is initialized to FALSE in the main program.

```
get_exclusive_access ( )
{
   if (critical _flag = = FALSE)
   {
      critical_flag = TRUE ;
      critical_region ( ) ;
      critical_flag = FALSE;
   }
}
```

Consider the following statements.
(i) It is possible for both P1 and P2 to access critical_region concurrently.
(ii) This may lead to a deadlock.
Which of the following holds?
(a) (i) is false and (ii) is true
(b) Both (i) and (ii) are false
(c) (i) is true and (ii) is false
(d) Both (i) and (ii) are true **[2007 : 1 M]**

2.20 Two processes, P1 and P2, need to access a critical section of code. Consider the following synchronization construct used by the processes:

```
/* P1 */
while (true) {
 wants1 = true;
 while (wants2 = = true);
 /*    Critical
       Section */
 wants 1 = false;
}
/* Remainder section */
```

```
/* P2 */
while (true) {
 wants2 = true;
 while (wants1 = = true);
 /*    Critical
       Section */
 wants 2 = false;
}
/* Remainder section */
```

Here, wants1 and wants2 are shared variables, which are initialized to false. Which one of the following statements is TRUE about the above construct?

(a) It does not ensure mutual exclusion.
(b) It does not ensure bounded waiting.
(c) It requires that processes enter the critical section in strict alternation.
(d) It does not prevent deadlocks, but ensures mutual exclusion. **[2007 : 2 M]**

2.21 Synchronization in the classical readers and writers problem can be achieved through use of semaphores. In the following incomplete code for readers-writers problem, two binary semaphores mutex and wrt are used to obtain synchronization

```
wait (wrt)
writing is performed
signal (wrt)
wait (mutex)
readcount = readcount + 1
if readcount = 1 then S1
S2
reading is performed
S3
readcount = readcount – 1
if readcount = 0 then S4
Signal (mutex)
```

The values of S1, S2, S3, S4, (in that order) are
(a) signal (mutex), wait (wrt), signal (wrt), wait (mutex)
(b) signal (wrt), signal (mutex), wait (mutex), wait (wrt)
(c) wait (wrt), signal (mutex), wait (mutex), signal (wrt)
(d) signal (mutex), wait (mutex), signal (mutex), wait (mutex) **[2007 : 2 M]**

2.22 The P and V operations on counting semaphores, where s is a counting semaphore, are defined as follows:

```
P(s): s = s – 1;
      if s < 0 then wait;
V(s): s = s + 1;
      if s ≤ 0 then wakeup a process waiting on s;
```

Assume that P_b and V_b, the wait and signal operations on binary semaphores are provided. Two binary semaphores X_b and Y_b are used to implement the semaphore operations P(s) and V(s) as follows:

```
P(s):   Pb(Xb);
        s = s – 1;
        if (s < 0) {
           Vb(Xb);
           Pb(Yb);
        }
        else Vb(Xb);
```

V(s): $P_b(X_b)$;
s = s + 1;
if (s ≤ 0) $V_b(Y_b)$;
$V_b(X_b)$;

The initial values of X_b and Y_b are respectively
(a) 0 and 0 (b) 0 and 1
(c) 1 and 0 (d) 1 and 1
[2008 : 2 M]

2.23 The enter_CS ( ) and leave_CS ( ) functions to implement critical section of a process are realized using test-and-set instruction as follows:

```
Void enter_CS (X)
{
    while (test-and-set (X));
}
Void leave_CS (X)
{
    X = 0 ;
}
```

In the above solution, X is a memory location associated with the CS and is initialized to 0. Now consider the following statements
I. The above solution to CS problem is deadlock-free.
II. The solution is starvation free.
III. The processes enter CS in FIFO order.
IV. More than one process can enter CS at the same time.
Which of the above statements are TRUE?
(a) I only (b) I and II
(c) II and III (d) IV only
[2009 : 2 M]

2.24 Consider the methods used by processes P1 and P2 for accessing their critical sections whenever needed, as given below. The initial values of shared boolean variables S1 and S2 are randomly assigned.

Method used by P1	Method used by P2
while (S1==S2);	while (S1!=S2);
Critical Section	Critical Section
S1=S2;	S2=not (S1);

Which one of the following statements describes the properties achieved?
(a) Mutual exclusion but not progress
(b) Progress but not mutual exclusion
(c) Neither mutual exclusion nor progress
(d) Both mutual exclusion and progress
[2010 : 1 M]

2.25 The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0 = 1, S1 = 0, S2 = 0.

Process P0	Process P1	Process P2
while (true) {	wait (S1);	wait (S2)
wait (S0);	release(S0);	release(S0);
print '0'		
release (S1);		
release (S2);		
}		

How many times will process P0 print '0'?
(a) At least twice
(b) Exactly twice
(c) Exactly thrice
(d) Exactly once
[2010 : 2 M]

2.26 Fetch_And_Add(X, i) is an atomic Read-Modify-Write instruction that reads the value of memory location X, increments it by the value i, and returns the old value of X, it is used in the pseudocode shown below to implement a busy-wait lock. L is unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

```
AcquireLock ( L ){
    while ( Fetch_And_Add( L, 1))
    L = 1;
}
ReleaseLock ( L ){
    L = 0;
}
```

This implementation
(a) fails as L can overflow
(b) fails as L can take on a non-zero value when the lock is actually available
(c) works correctly but may starve some processes
(d) works correctly without starvation
[2012 : 2 M]

2.27 Three concurrent processes X, Y and Z execute three different code segments that access and update ceratin shared variables. Process X execute the P operation (i.e., wait) on semaphores a, b and c; process Y executes the P operation on semaphores b, c and d; process Z executes the P operation on semaphores c, d and a before entering the respective code segments. After completing the execution of its code segments, each process invokes the V operation (i.e., signal) on its three semaphores. All semaphores are binary semaphores initialized to one. Which one of the following represents a deadlock-free order of invoking the P operations by the processes?
(a) X : P(a)P(b)P(c) (b) X : P(b)P(a)P(c)
Y : P(b)P(c)P(d) Y : P(b)P(c)P(d)
Z : P(c)P(d)P(a) Z : P(a)P(c)P(d)
(c) X : P(b)P(a)P(c) (d) X : P(a)P(b)P(c)
Y : P(c)P(b)P(d) Y : P(b)P(c)P(d)
Z : P(a)P(c)P(d) Z : P(c)P(d)P(a)
[2013 : 1 M]

2.28 A shared variable x, initialized to zero, is operated on by four concurrent process W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory, and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory and then terminates. Each process before reading x invokes the P operations (i.e., wait) on a counting semaphore S and invokes the V operation (i.e., signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum possible value of x after all processes complete execution?
(a) – 2 (b) – 1
(c) 1 (d) 2 **[2013 : 2 M]**

2.29 A certain computation generates two arrays a and b such that a[i] = f for 0 ≤ i ≤ n and b[i] = g(a[i]) for 0 ≤ i < n. Suppose this computation is decomposed into two concurrent processes X and Y such that X computes the array a and Y computes the array b. The processes employ two binary semaphores R and S, both initialized to zero. The array a is shared by the two processes. The structure of the processes are shown below:

Process X
```
private i;
for (i = 0; i < n; i++)
{
    a[i] = f(i);
    ExitsX (R, S);
}
```

Process Y
```
private i;
for (i = 0; i < n; i++)
{
    EntryY(R, S);
    b[i] = g(a[i]);
}
```

Which one of the following represents the **CORRECT** implementations of ExitsX and EntryY?

(a)
```
ExitsX (R, S) {
  P(R);
  V(S)
}
EntryY(R, S) {
  P(S);
  V(R);
}
```

(b)
```
ExitsX (R, S) {
  V(R);
  V(S);
}
EntryY(R, S) {
  P(R);
  P(S);
}
```

(c)
```
ExitsX (R, S) {
  P(S);
  V(R)
}
EntryY(R, S) {
  V(S);
  P(R);
}
```

(d)
```
ExitsX (R, S) {
  V(R);
  P(S);
}
EntryY(R, S) {
  V(S);
  P(R);
}
```

[2013 : 2 M]

2.30 Consider the procedure below for the Producer-Consumer problem which uses semaphores:

```
semaphore n = 0;
semaphore s = 1;
void producer()
{
  while(true)
  {
    produce();
    semWait(s);
    addToBuffer();
    semSignal(s);
    semSignal(n);
  }
}
void consumer()
{
  while (true)
  {
    semWait (s);
    semWait (n);
    removeFromBuffer ( ),
    semSignal (s);
    consume( );
  }
}
```

Which one of the following is TRUE?
(a) The producer will be able to add an item to the buffer, but the consumer can never consume it.
(b) The consumer will remove no more than one item from the buffer.
(c) Deadlock occurs if the consumer succeeds in acquiring semaphore s when the buffer is empty.
(d) The starting value for the semaphore n must be 1 and not 0 for deadlock-free operation.
[2014 (Set-2) : 2 M]

2.31 The following two functions P1 and P2 that share a variable B with an initial value of 2 execute concurrently.

```
P1()
{
    C = B – 1;
    B = 2 * C;
}
```

```
P2()
{
    D = 2 * B;
    B = D – 1;
}
```

The number of distinct values that B can possibly take after the execution is _____.
[2015 (Set-1) : 1 M]

2.32 Two processes X and Y need to access a critical section. Consider the following synchronization construct used by both the processes

```
Process X
/* other code for process X */
while (true)
{   varP = true;
    while (varQ = = true)
    { /* Critical Section */
      varP = false;
    }
}
/* other code for process X */
Process Y
/* other code for process Y */
while (true)
{   varQ = true;
    while (varP = = true)
    { /* Critical Section */
      varQ = false;
    }
}
/* other code for process Y */
```

Here, varP and varQ are shared variables and both are initialized to false. Which one of the following statements is true?

(a) The proposed solution prevents deadlock but fails to guarantee mutual exclusion
(b) The proposed solution guarantees mutual exclusion but fails to prevent deadlock
(c) The proposed solution guarantees mutual exclusion and prevents deadlock
(d) The proposed solution fails to prevent deadlock and fails to guarantee mutual exclusion

[2015 (Set-3) : 1 M]

2.33 Consider the following proposed solution for the critical section problem. There are n processes: $P_0 \ldots P_{n-1}$. In the code, function $pmax$ returns an integer not smaller than any of its arguments. For all i, $t[i]$ is initialized to zero.

Code for P_i :

```
do
{
   c[i]=1; t[i] = pmax (t[0],..., t[n-1]) +1; c[i] = 0;
   for every j ≠ i in {0,..., n - 1}
   {  while (c[j]);
      while (t[j] != 0 && t[j] <= t[i]);
   }
   Critical Section; t[i] = 0;
   Remainder Section;
} while (true);
```

Which one of the following is TRUE about the above solution?

(a) At most one process can be in the critical section at any time
(b) The bounded wait condition is satisfied
(c) The progress condition is satisfied
(d) It cannot cause a deadlock

[2016 (Set-1) : 2 M]

2.34 Consider the following two-process synchronization solution:

```
Process 0
    Entry: loop while (turn == 1);
    (critical section)
    Exit: turn = 1;
Process 1
    Entry : loop while (turn == 0);
    (critical section)
    Exit: turn = 0;
```

The shared variable turn in initialized to zero. Which one of the following is **TRUE**?

(a) This is a correct two-process synchronization solution
(b) This solution violates mutual exclusion requirement
(c) This solution violates progress requirement
(d) This solution violates bounded wait requirement **[2016 (Set-2) : 2 M]**

2.35 Consider a non-negative counting semaphore S. The operation P(S) decrements S, and V(S) increments S. During an execution, 20 P(S) operations and 12 V(S) operations are issued in some order. The largest initial value of S for which at lest one P(S) operation will remain blocked is _____. **[2016 (Set-2) : 2 M]**

2.36 A multithreaded program P executes with x number of threads and uses y number of locks for ensuring mutual exclusion while operating on shared memory locations. All locks in the program are non-reentrant, i.e., if a thread holds a lock l, then it cannot re-acquire lock l without releasing it. If a thread is unable to acquire a lock, it blocks until the lock becomes available. The minimum value of x and the minimum value of y together for which execution of P can result in a deadlock are:

(a) $x = 1,\ y = 2$ (b) $x = 2,\ y = 1$
(c) $x = 2,\ y = 2$ (d) $x = 1,\ y = 1$

[2017 (Set-1) : 2 M]

2.37 Consider the following solution to the producer-consumer synchronization problem. The shared buffer size is N. Three semaphores empty, full and mutex are defined with respective initial values of 0, N and 1. Semaphore empty denotes the number of available slots in the buffer, for the consumer to read from. Semaphore full denotes the number of available slots in the buffer, for the producer to write to. The placeholder variables, denoted by P, Q, R and S, in the code below can be assigned either empty or full. The valid semaphore operations are: wait ( ) and signal ( ).

Producer:	Consumer

do {
wait (P);
wait (mutex);
//Add item to buffer
signal (mutex);
signal (Q);
} while (1);
	do {
	wait (R);
	wait (mutex);
	//Consume item from buffer
	signal (mutex);
	signal (S);
	} while (1);

Which one of the following assignment to P, Q, R and S will yield the correct solution?

(a) P: full, Q: full, R: empty, S; empty
(b) P: empty, Q: empty, R: full, S: full
(c) P: full, Q: empty, R: empty, S: full
(d) P: empty, Q: full, R: full, S: empty

[2018 : 2 M]

2.38 Consider three concurrent processes P1, P2 and P3 as shown below, which access a shared variable D that has been initialized to 100.

P1	P2	P3
D = D + 20	D = D - 50	D = D + 10

The processes are executed on a uniprocessor system running a time-shared operating system. If the minimum and maximum possible values of D after the three processes have completed executed are X and Y respectively, then the value of Y – X is _____. **[2019 : 1 M]**

2.39 Consider the following snapshot of a system running n concurrent processes. Process i is holding X_i instances of a resource R, $1 \le i \le n$. Assume that all instances of R are currently in use. Further, for all i, process i can place a request for at most Y_i additional instances of R while holding the X_i instances it already has. Of the n processes, there are exactly two processes p and q such that $Y_p = Y_q = 0$. Which one of the following conditions guarantees that no other process apart from p and q can complete execution?

(a) $X_p + X_q < \text{Min}\ \{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$
(b) $\text{Min}\ (X_p, X_q) \ge \text{Min}\ \{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$
(c) $\text{Min}\ (X_p, X_q) \le \text{Max}\ \{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$
(d) $X_p + X_q < \text{Max}\ \{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$

[2019 : 2 M]

2.40 Each of a set of n processes executes the following code using two semaphores a and b initialized to 1 and 0, respectively. Assume that count is a shared variable initialized to 0 and not used in CODE SECTION P.

```
CODE SECTION P
wait(a); count = count + 1;
if (count == n) signal(b);
signal(a); wait(b); signal(b);
CODE SECTION Q
```

What does the code achieve?

(a) It ensures that at most two processes are in CODE SECTION Q at any time.
(b) It ensures that all processes execute CODE SECTION P mutually exclusively.
(c) It ensures that no process executes CODE SECTION Q before every process has finished CODE SECTION P.
(d) It ensures that at most $n - 1$ processes are in CODE SECTION P at any time.

[2020 : 2 M]

2.41 Consider the following pseudocode, where S is a semaphore initialized to 5 in line#2 and counter is a shared variable initialized to 0 in line#1. Assume that the increment operation in linc#7 is not atomic.

```
1.  int counter = 0;
2.  Semaphore S = init(5);
3.  void parop(void)
4.  {
5.      wait(S);
6.      wait(S);
7.      counter++;
8.      signal(S);
9.      signal(S);
10. }
```

If five threads execute the function parop concurrently, which of the following program behavior(s) is/are possible?

(a) The value of counter is 1 after all the threads successfully complete the execution of parop.
(b) The value of counter is 0 after all the threads successfully complete the execution of parop.
(c) There is a deadlock involving all the threads.