Operating Systems

Class-Test 1

Date: 07/02/2019 Full Marks: 20 Time: 1 hour

Answer all the questions

1. Answer the following in brief. (4x2=8)

2 a) How can the processor mode changed from *user mode* to *supervisor mode*, and vice versa?

2 b) What are the differences between hardware interrupt and software interrupt? Give two examples of each.

2 c) In an actual implementation of the operating system, how exactly are the following process state transitions take place: (i) Running to Ready, (ii) Blocked to Ready?

2 d) Explain whether the following CPU scheduling algorithms discriminate in favour of short processes. FCFS ; SJF.

2. Answer the following: (3x2 = 6)

2 a) How many processes will be created by the following code segment?

```
    fork();
    for (i=0;i<6;i++)
       if (i%2 = = 0) fork();
```

2 b) What is a zombie process? How are such processes typically handled in Unix/Linux?

2 c) With respect to two concurrent processes, explain how race condition can occur with the help of an example.

3. A program contains a single loop that executes 50 times. The loop contains a computation that lasts 50 msec followed by an I/O operation that consumes 200 msec. This program is executed in a time-sharing system with 9 other identical programs. All programs start their execution at the same time. The scheduling overhead of the OS is 3 msec. Compute the average waiting time if

   a) The time slice is 50 msec.

   b) The time slice is 20 msec. (6)

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Mid-Spring Semester 2018-19

ate of Examination: 25/02/2019 Session: AN Duration: 2 hours Full Marks: 80

ubject No.: CS30002 Subject: Operating Systems

Department: Computer Science and Engineering

Instructions: Answer all the questions

1. Briefly answer the following. *[6 × 3 = 18]*
   a) Explain how SPOOLing helps in improving processor utilization.
   b) With the help of an example, clearly explain how race condition can occur when two concurrent processes **P1** and **P2** are updating a shared variable **X**.
   c) What is the role of the *dispatcher* in process scheduling? When is the dispatcher invoked?
   d) Suggest a method using which the next CPU burst time can be estimated in the shortest-job-first CPU scheduling algorithm.
   e) Write a C/C++ code segment to create four threads **T1**, **T2**, **T3** and **T4**. Thread **Ti** will print the message "**Hello, I am thread i**", wait for 5 seconds, and then terminate. After all the threads have terminated, print the message "**Good bye**".
   f) Show how Peterson's algorithm can fail to provide a correct solution to the critical section problem in modern computer architectures where out-of-order instruction execution is a possibility.

2. *State with clear justifications whether the following statements are true or false. No marks will be awarded if the justification is not correct.* [8 × 2 = 16]
   a) In a typical operating system, kernel routines are run as processes or threads, which can be invoked from user-level programs when needed.
   b) In Unix, a parent process does not terminate unless all of its child processes have terminated.
   c) The critical section problem does not arise when we use message passing for inter-process communication.
   d) Following are examples of privileged instructions: *disable interrupt, initialize timer, system call.*
   e) The highest-response-ratio-next CPU scheduling algorithm gives higher priority to processes with shorter CPU bursts as compared to those with longer CPU bursts.
   f) Context switching among processes takes more time as compared to that among threads.
   g) The following solution to the dining philosophers problem does not result in deadlock. Philosophers 0, 1 and 2 pick their left chopstick first and then the right chopstick; whereas philosophers 3 and 4 pick their right chopstick first and then the left chopstick.
   h) Binary semaphores cannot be used by more than two processes. For more than two processes, we need counting semaphores.

3. Answer the following. [6 + 3 + 3 + 3 = 15]

   6 a) Briefly explain the hardware support required by the operating system to handle the following situations:
      2 i) Implementation of round-robin scheduling.
      2 ii) Protecting process address spaces in memory (assuming contiguous allocation).
      2 iii) Implementation of system calls for requesting some service from the operating system.

   b) How many times will the message "**Hello**" be printed by the following C code segment?

   3
```
pid = fork();
if (pid == 0) { fork(); fork(); fork(); }
else { fork(); fork(); }
printf ("\n Hello");
```

3 c) Consider the following pseudo-code of two concurrent processes **P1** and **P2** that uses two binary semaphores **n** and **s**, initialized to 0 and 1 respectively. Examine whether there is any possibility of deadlock in the solution.

```
void P1()                      void P2()
{                              {
   while (1) {                    while (1) {
     ...                            wait (s);
     wait (s);                      wait (n);
     ...                            ...
     signal (s);                    signal (s);
     signal (n);                    ...
   }                              }
}                              }
```

3 d) With the help of examples, differentiate between *deadlock* and *starvation*.

4. Answer the following. **[5 + 4 + 6 = 15]**

5 a) *Show the typical state diagram of a process (including the swapped out states)*, clearly indicating the scenarios under which the various state transitions can take place.

4 b) Explain how the multi-level feedback queue scheduling algorithm favours processes with short CPU bursts, and at the same time prevents starvation for processes with long CPU bursts.

6 c) Consider the following set of processes, with all times specified in milliseconds.

Process	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**
Arrival Time	0	1	3	3	5	6	8
CPU Burst Time	5	13	3	8	1	7	2

Compute the average waiting time when the processes are scheduled using the following scheduling algorithms:

i) Non-preemptive SJF

ii) Preemptive SJF

iii) Round-robin with time quantum of 3

5. Answer the following. **[4 + 6 + 6 = 16]**

a) Give an implementation of binary semaphore that tries to eliminate busy waiting. Hence discuss whether busy waiting is totally eliminated in the implementation.

*b) Consider the following solution to the bounded-buffer producer-consumer problem, where the buffer is implemented as an array of "**item**" of size **BUFSIZE**. Identify the problem, if any, in the solution and suggest suitable modifications to avoid it.*

```
In shared memory:
      item buff[BUFSIZE];
      int in = 0, out = 0, count = 0;

void producer()                    void consumer()
{                                  {
  while (1) {                        while (1) {
    next_prod = produce();             while (count == 0)
    while (count == BUFSIZE)              ;
       ;                               next_con = buff[out];
    buff[in] = next_prod;              out = (out + 1) % BUFSIZE;
    in = (in + 1) % BUFSIZE;           count --;
    count ++;                          consume(next_con);
  }                                  }
}                                  }
```

c) Suggest a method for implementing semaphores in multiprocessor systems.

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

End-Spring Semester 2018-19

Date of Examination: 29/04/2019 **Session: AN** **Duration: 3 hours** **Full Marks: 80**

Subject No.: CS30002 **Subject: Operating Systems**

Department: Computer Science and Engineering

Instructions: *Answer Q1 and any four of the rest*
All parts of the same question must be answered together

1. Justify with reasons whether the following statements are true or false. *No marks will be granted if the justification is not correct.* [10 × 2 = 20]

   a) Non-preemptive scheduling algorithms are more suitable for a time-sharing system.

   b) In round-robin CPU scheduling, a short time quantum allows the scheduler to cycle through more processes more quickly and is therefore preferred when the number of processes is large.

   c) A page fault must be preceded by a TLB miss, but a TLB miss does not necessarily mean a page fault.

   d) In typical computer architectures, 32-bit words are stored from memory addresses that are some multiple of 4 because instruction sizes can be made more compact.

   e) In segmentation, each row of the segment table typically contains the starting address of the segment in memory and additional flags like valid, dirty and reference bits.

   f) Dirty bit for a page in a page table helps avoid unnecessary writes on paging device.

   g) If an instruction takes x microseconds, a page fault takes an additional y microseconds, and on the average a page fault occurs every z instructions, then the average instruction execution time is $x + y/z$ microseconds.

   h) Using a larger block size in a file system leads to better disk throughput but poorer disk utilization.

   i) RAID level 5 provides the same level of fault tolerance as RAID level 1 but incurs more cost.

   j) Belady's anomaly may exhibit only in stack page replacement algorithms.

2. Answer the following. [4 + 6 + 5 + 5 = 20]

   a) *Clearly explain the difference between deadlock prevention and deadlock avoidance.*

   b) A system contains 3 resource types R1, R2 and R3, with the total number of resource instances being 7, 7 and 10 respectively. For three concurrent processes P1, P2 and P3, the current resource allocation state in the system is as follows:

7 7 10

	Allocated Resources			*Maximum Needs*		
	R1	R2	R3	R1	R2	R3
Process P1	2	2	3	3	6	8
Process P2	2 2	0 3	3 0	4	3	3
Process P3	1 2	2 3	4	3	4	4

Examine whether the current allocation state is *safe*. Hence examine whether the following requestes *that arrive in sequence* can be granted.

   i) *Process P1 requests (1, 1, 0)*

   ii) Process P3 requests (0, 1, 0)

   iii) Process P2 requests (0,.1, 0)

c) Consider a system consisting of m resources of the same type being shared by n processes. A process can request or release only one resource at a time. Show that the system will be free from deadlock if the following two conditions hold:

i) The maximum need of each process is between 1 and m.

ii) The sum of all maximum needs is less than $m+n$.

d) Consider the dining philosopher problem. Assume that some philosophers always pick up their left forks first (a *lefty*), and some philosophers always pick up their right forks first (a *righty*). Also assume that there is at least one lefty and one righty at the table. Can deadlock occur? Is starvation possible (assuming a fair scheduling policy)? Give proper justification in support of your answer.

3. Answer the following. [4 + 4 + 6 + 6 = 20]

a) Consider that the average size of a process is 8 Kbytes, and each page table entry is of size 4 bytes. What will be the optimum page size that minimizes the total memory overhead? Derive any expression you use.

b) What is the difference between local and global page allocation? What are their respective merits and demerits?

c) Explain the second-change page replacement algorithm that uses *reference bits* with the help of an example. How can the performance of the method be improved by additionally considering the *modify bits*?

d) Consider a small system where the virtual memory page size is 2K (2048 bytes), and main memory consists of 4 page frames. Now consider a process that requires 8 pages of storage. At some point during its execution, the page table is as shown below:

Virtual page	*Valid*	*Physical page*
0	No	
1	No	
2	Yes	1
3	No	
4	Yes	3
5	No	
6	Yes	0
7	Yes	2

i) List the virtual address ranges (in hexadecimal) for each virtual page.

ii) Give the main memory (physical) addresses ***in hexadecimal*** for each of the following virtual addresses (given in decimal), mentioning which of these will result in page faults: (i) 8500, (ii) 14000, (iii) 5000, (iv) 2100.

4. Answer the following. [5 + 5 + 6 + 4 = 20]

a) Show a schematic diagram of the logical to physical address mapping scheme for segmentation combined with demand paging.

b) In a computer system the page tables are stored in main memory that has an access time of 100 nsec. The TLB can hold 8 page table entries and has an access time of 10 nsec. During the execution of a program, it is found that 85% of the time the page being referenced is found in TLB, and only 2% of the memory references lead to page faults. The average time for page replacement is 2 msec. Compute the average memory access times for:

i) A single-level page table organization

ii) A three-level page table organization

c) Consider the following code fragment that processes data stored in an array `A`, stored in row-major order, where every array element is a 32-bit floating-point number. The variables `i`, `j` and `temp` are stored in processor registers, and hence do not need any memory reference to access them. The main memory is word addressable and uses demand paging, where the page size is 1024 words and one word is 4 bytes. Page replacement policy is LRU. The number of page frames allocated to the process to store the array elements is 16. Determine the number of page faults generated when the given program fragment is executed.

```
for (j=0; j<1024; j++) {
  temp = 0;
  for (i=0; i<25; i++) {
    temp = temp + A[i][j];
  }
  cout << temp / 25;
}
```

d) Assume that you have a page-reference string for a process with *m* frames (initially all empty). The page reference string has length *p*, and *n* distinct page numbers occur in it.

i) What is a lower bound on the number of page faults?

ii) What is an upper bound on the number of page faults?

5. Answer the following. [6 + 6 + 2 + 6 = 20]

3600 → 7200
1 → 2 20t
0.5 sec → 120t

a) Consider the following sequence of disk track requests: 27, 129, 110, 186, 147, 41, 10, 64, 120. Assume that initially the head is at track 30 and is moving in the direction of decreasing track number. Compute the number of tracks the head traverses using (i) FCFS, (ii) SSF, and (iii) elevator algorithms.

b) ✓ A hard disk spins at 7200 rpm, and each track contain 100 sectors of size 512 bytes each. The average seek time between any pair of tracks in 6 msec. Compute the ***worst-case time*** required to read a file of size 40 Kbytes:

i) if the blocks of the file occupy consecutive sectors on one track;

ii) if the file is stored in eight 5 Kbyte chunks, where the chunks are randomly distributed on the disk and each chunk consists of contiguous blocks of the same track.

In the second case, assume that the read is done in the order the blocks appear in the file.

c) How many disk accesses are needed to read a small file (i.e. smaller than the size of a block) from hard disk in a Unix-like file system? Explain your answer.

d) ✓ A Unix-style i-node has 10 direct pointers, and one single, one double and one triple indirect pointers. Disk block size is 1 Kbyte, disk block address is 32 bits, and the file size is stored as a 48-bit integer in the i-node. What is the maximum possible file size?

6. Answer the following. [5 + 4 + 5 + 6 = 20]

a) Consider the following pseudo-code for a process `Pi`:

```
shared boolean flag[2];
flag[0] = FALSE;   flag[1] = FALSE;
P (int i) {
  while (TRUE) {
    while (flag[(i+1) % 2] == TRUE);
    flag[i] = TRUE;
    < Critical Section >
    flag[i] = FALSE;
    < Remainder Section >
  }
}
```

Explain whether this code solves the critical section problem for two processes `P0` and `P1`.

b) Most operating systems are designed for general-purpose computation. A proposal has been put forward for an OS that is optimized for running math-intensive programs. In MathOS, the kernel includes system calls for many useful mathematical operations. These system calls are written in highly optimized assembly language for maximum performance. Is this concept a good idea> Briefly justify.

c) Discuss the merits and demerits of the FAT-based approach and the indexed approach (like Unix i-node) for keeping track of the data blocks in a file.

d) Briefly explain how memory protection can be achieved in a memory management scheme that uses:

i) Contiguous allocation

ii) Segmentation

iii) Paging