Indian Institute of Information Technology Allahabad

Mid-Semester Examination (September 2019)

Third semester B.Tech (IT): Section A & B

Course Name	**Course Code**	**Date of Exam**	**MM**	**Time**
Operating Systems	IOPS332C	Sept. 20, 2019	30	1 Hrs

Important Instructions: *All questions are compulsory. Answer all questions in strict order as is given in the question paper.*

1. **($4 * 1 = 4$ marks)** Answer True/False
   (*Note for marking: no partial marking*):

   (a) A context switch takes place at every system call. **F**

   (b) The POSIX `execve` system call creates a new process **F**

   (c) Using mutual exclusion ensures that a system avoids deadlock. **F**

   (d) The value of a semaphore can never be negative. **T**

2. **($6 * 1 = 6$ marks)** Answer the following. Every question can have multiple correct choices.
   (*Note for marking: No partial marking*)

   (a) When a process is first launched, the operating system does not know the size of this segment

   ```
   (i)    text     (ii)  data
   (iii)  bss      (iv)  heap
   ```

   Solution: (iv)

   (b) With a legacy PC BIOS, the Master Boot Record

   i. Identifies type of file system on the disk and loads the operating system.
   ii. Contains the first code that is run by the computer when it boots up.
   iii. Contains a list of operating systems available for booting.
   iv. Contains a boot loader to load another boot loader located in the volume boot record.

   Solution: (iv)

   (c) Which of the following does NOT cause a trap?

   i. A user program divides a number by zero.
   ii. The operating system kernel executes a privileged instruction.
   iii. A programmable interval timer reaches its specified time.
   iv. A user program executes an interrupt instruction.

   Solution: (ii)

   (d) Which of the following is not a system call?

   i. Duplicate an open file descriptor.
   ii. Get the current directory.
   iii. Decrement a semaphore.
   iv. Create a new linked list.

   Solution: (ii)

   (e) A process exists in the zombie (also known as defunct) state because:

   i. It is running but making no progress.
   ii. The user may need to restart it without reloading the program.
   iii. The parent may need to read its exit status.

   Solution: (iii)

   (f) What information is stored in a thread control block (TCB)?

i. List of open files.
ii. Stack pointer.
iii. Memory map.
iv. Thread owner ID.

Solution: (ii)

3. **(8 marks)** Answer the following in brief ***(Note for marking: No partial marking)***

   (a) What is the difference between a cpu mode switch and a context switch? **(1 marks)**
   Solution: Cpu mode switch: change CPU execution mode from one privilege level to another e.g., user `->` kernel via a trap or syscall.
   Context switch: save one process execution context & restore that of another process.

   (b) Answer with explanation that how many times does this code print “hello”? **(2 marks)**

```
void main(int argc, char **argv) {
        int i;
        for (i=0; i < 3; i++) {
                execl("/bin/echo", "echo", "hello", 0);
        }
}
```

   Solution: The execl system call overwrites the current process by loading the program /bin/echo. The for loop is gone! Therefore, the answer is 1

   (c) A CPU scheduling algorithm determines an order for the execution of its scheduled processes. Given n processes to be scheduled on one processor, how many possible different schedules are there? Give a formula in terms of n. **(1 marks)**
   Solution: $n!$

   (d) Consider the following program: **(2+2=4 marks)**

```
main() {
        int fd;
        fd = open("outfile", O_RDWR);
        fork();
        write(fd, "hello", 5);
        exit();
}
```

   Assume all system calls finish successfully on a uniprocessor system. Also, assume that a system call cannot be interrupted in the middle of its execution.

   i. What will be the contents of the “outfile” file, after all processes have successfully exited? Explain briefly.
   Solution: hellohello
   Fork will create two processes with the same `fd` pointing to the given file. Two “hello” writes will be observed by the same `fd` (seek value or `pos` will be shared) and because of the given assumption, interleaving/mixing of these two writes is not possible.

   ii. If open is now called after fork (not before fork as in the previous question). What will be the contents of the “outfile” file, after all processes have successfully exited? Explain briefly.
   Solution: hello
   Value of `fd` will be same for both the process. But open file entry corresponding to both this `fd` are different ie., there are two copies of writeoffset. After open system call, write offset in each of the opened file entry is initialized to zero and second write will over write the first one.

4. **(4 marks)** Consider a system running ten 1/O-bounds tasks and one CPU-bound task. Assume that the I/O bound tasks issue an I/O operation once for every millisecond of CPU computing

and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1 millisecond and all processes are long-running tasks. What is the CPU utilization for a round-robin scheduler when:

(a) The time quantum is 1 millisecond. **(2 marks)**

(b) The time quantum is 10 millisecond. **(2 marks)**

Solution: (a) The time quantum is 1 millisecond: Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context-switching cost for every context-switch. This results in a CPU utilization of 1/1.1 * 100 = 91%.

(b) The time quantum is 10 milliseconds: The I/O-bound tasks incur a context switch after using up only 1 millisecond of the time quantum. The time required to cycle through all the processes is therefore $10 * 1.1 + 10.1$ (as each I/O-bound task executes for 1 millisecond and then incur the context switch task, whereas the CPU-bound task executes for 10 milliseconds before incurring a context switch). The CPU utilization is therefore 20/21.1 * 100 = 94%.

(Note for marking: No partial marking)

5. **(2 marks)** Consider the following program fragment:

```
P(s1);
a++;
P(s2);
v++;
V(s2);
V(s1);
```

(`s1`, `s2` are semaphores). All variables are automatic. Now, consider two threads running this fragment of code simultaneously, can there be a deadlock? Why, or why not?

Solution: Both semaphores will be acquired in the same order by both threads. Since this is a linear order, there can be no situation where a deadlock could occur.
(Note for marking: No partial marking, award full two marks if answer is correct, otherwise zero.)

6. **(6 marks)** The code below is written to provide one possible solution to the READERS/WRITERS problem.

```
int readCount=0, writeCount=0;
semaphore mutex1=1, mutex2=1;
semaphore readBlock=1, writeBlock=1
```

```
reader() {
  while(TRUE) {
    <other computing>;

      P(readBlock);
        P(mutex1);
          readCount++;
          if(readCount == 1)
            P(writeBlock);
        V(mutex1);
      V(readBlock);

      access(resource);
    P(mutex1);
      readCount--;
      if(readCount == 0)
        V(writeBlock);
    V(mutex1);
  }
}
```

```
writer() {
  while(TRUE) {
    <other computing>;

    P(mutex2);
      writeCount++;
      if(writeCount == 1)
        P(readBlock);
    V(mutex2);
    P(writeBlock);
      access(resource);
    V(writeBlock);

    P(mutex2);
      writeCount--;
      if(writeCount == 0)
        V(readBlock);
    V(mutex2);
  }
}
```

Answer the following questions based on the code given above.

(a) Explain the role of each semaphore used in this code. **(4 marks)**

Solution:
mutex1: to provide mutually exclusive access to the shared variable ”readcount” by multiple readers.
mutex2: to provide mutually exclusive access to the shared variable ”writecount” by multiple writers.
readBlock: if a reader arrives while a writer is inside the critical section (CS) and other writers are waiting, this semaphore will cause that reader to get blocked until the last writer leaves the CS.
writeBlock: This semaphore causes any writer to get blocked if there is one writer or reader(s) currently accessing the CS.
(Note for marking: Each mutex definition gets one mark. No partial marking)

(b) Does this code provide a correct and fair solution for the readers/writers problem? **(2 marks)**

Solution: Even though this code is correct, it does not provide a fair solution. With the way it is written, there is a possibility that writers may starve readers if they arrive one after another. In other words, readers may never get a chance to enter the critical section (i.e. access the shared resource) if the writers don't take a break.
(Note for marking: No partial marking, award full two marks if answer is correct, otherwise zero.)