INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Mid-Spring Semester 2019-20

Date of Examination: 17/02/2020 **Session: FN** **Duration: 2 hours** **Full Marks: 80**

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

Department: Computer Science and Engineering

Instructions: **Answer all the questions**
All parts of the same question must be answered together

1. Answer the following questions in brief. **[10 × 2 = 20]**

a) State two ways in which the processor mode can be changed from ***user mode*** to ***supervisor mode***.

(i) A program invoking a system call instruction

(ii) An interrupt signal arrives to the processor (timer, I/O, etc.)

b) With respect to the state transition diagram of a process, state two ways in which the process state can switch from ***Running*** to ***Ready***.

(i) The time slice of a process elapses in round-robin scheduling

(ii) An I/O (completion) interrupt arrives, for which the currently running process has to be suspended and the interrupt request serviced.

c) Justify with reasons which of the following is a privileged instruction:

i) Switch from user mode to supervisor mode.

This happens when a program executes the syscall instruction in user mode. This is not a privileged instruction, as otherwise user programs can never execute this.

ii) Initialize the timer value in round-robin scheduling.

This must be a privileged instruction, as otherwise a user level program can modify the value loaded in the timer, and hence tamper with the scheduling policy of the operating system.

d) The primary objective of multiprogramming is to minimize user response time, while the primary objective of time sharing is to maximize processor utilization. Justify or contradict.

The statement is FALSE.

In multiprogramming, a context switch from one program to another happens only when the first program gets blocked due to an I/O operation. As such, the primary objective is to maximize processor utilization; processor should not lie idle, no unnecessary context switches.

In time sharing system, there are a number of interactive users sitting on terminals. Minimizing the user response time is the main objective here.

e) What do you mean by ***CPU Burst*** time and ***I/O Burst*** time of a process?

During the execution of a process, the process can continue to use the CPU until it issues a system call to request for some service from OS; this typically will result in the process to move to Waiting state. This time is called the CPU burst time.

Similarly, when a process issues an I/O request, it moves to Waiting state and remains there while the I/O operation is going on. Upon completion, it moves to Ready state. This time is called I/O burst time.

f) State four events that may lead to process context switch in a time sharing operating system.

(i) A timer interrupt arrives

(ii) An I/O interrupt arrives

(iii) The process executes a syscall instruction

(iv) The process terminates execution

g) How many times will "Hello" be printed by the following code segment?

```
if (fork() && (!fork()))   // 2nd condition not evaluated if 1st is false
  if (fork() || fork())    // 2nd condition evaluated only if 1st is false
    fork();
printf ("\nHello");
```

The message "Hello" will be printed 7 times.

h) What is a ***zombie process*** in Linux? How are such processes cleaned by the operating system?

******

i) Justify with reasons whether the following CPU scheduling algorithms can result in process starvation: (A) first-come first-serve, (B) shortest-job next.

(A) The FCFS scheduling algorithm can lead to increased waiting time for processes if a long process starts executing, but there will be no starvation.

(B) The SJF algorithm on the other hand can lead to starvation, where a continuous stream of processes with shorter CPU bursts can prevent a process with longer CPU burst from getting the CPU.

j) Why is thread scheduling faster than process scheduling?

When context switch happens with process, all registers, code area, data area, stack area, open files, etc. must be saved and restored.

In threads, the code area, data area, open files, etc. are all shared among the threads. Thus during context switch, only registers and stack area need to be saved and restored.

For this reason, thread scheduling is faster than process scheduling.

2. Answer the following. **[4 + 3 + 4 + 4 = 15]**

a) Consider a variant of the round-robin scheduling algorithm in which the entries in the ready queue are pointers to the PCBs.

i) What would be the effect of putting two or more pointers in the ready queue to the same process PCB?

This way some process may get more CPU time per cycle in the round-robin scheduling. If a process has 3 pointers, then it will get 3 times more CPU time as compared to a process that has only 1 pointer.

ii) How would you modify the basic round-robin algorithm to achieve the same effect without using duplicate pointers?

The same effect can be implemented by incorporating priority values to the processes. When the despatchers selects a process for execution, it sets the timer value depending upon the process priority. A higher priority process will get a higher δ value as compared to a lower priority process.

b) Consider a time sharing operating system that uses the round-robin scheduling algorithm. Suppose there are **N** processes in the ready queue, with time quantum **Δ** and context-switch overhead of **δ**. Assume that the average CPU burst time of a process is **β**. Estimate the average waiting time for a process before it again gets chance to run on the CPU. Clearly state any assumptions you make.

After a process gets time to run on the CPU, it will be waiting for all the remaining (N-1) processes to run before it gets back the CPU again. On the average, a process will be using the CPU for min(Δ, β) time before it relinquishes the CPU.

Hence, average waiting time = (N – 1) * [δ + min(Δ, β)]

c) Suggest a CPU scheduling algorithm that tries to reduce the waiting time of the processes and at the same time avoids starvation. Comment on whether the algorithm will give preference to processes with short CPU bursts or long CPU bursts.

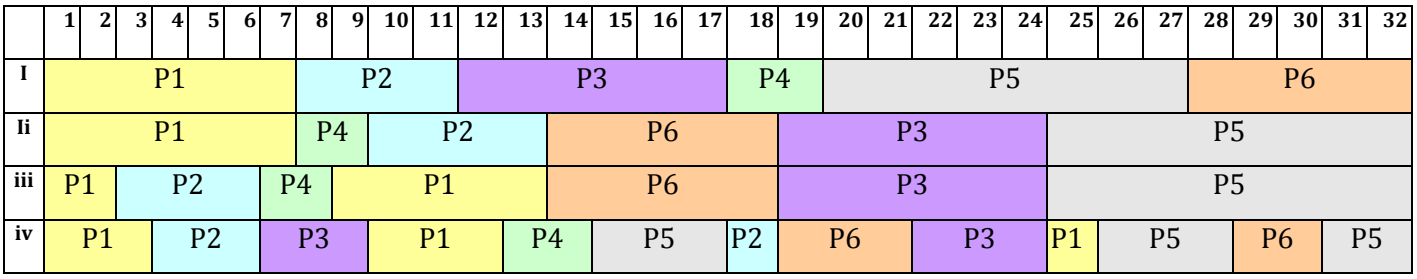
We can use the Highest Response Ratio Next scheduling algorithm. In this algorithm, the priority of a process increases as its waiting time increases. Also, the rate of increase of priority is faster for short CPU burst processes as compared to long CPU burst processes.

Hence, the algorithm gives preference to short CPU burst processes.

d) Consider the following set of processes. Calculate the ***average waiting time*** and ***average turnaround time*** for the following scheduling algorithms: (i) FCFS, (ii) non-preemptive SJF, (iii) pre-emptive SJF, and (iv) round-robin with time quantum of 3 msec.

Process	P1	P2	P3	P4	P5	P6
Arrival Time (msec)	0	2	3	5	6	8
CPU Burst (msec)	7	4	6	2	8	5

The Gantt chart for the four scheduling algorithms are shown below:



(i) FCFS

AWT = (0 + 5 + 8 + 12 + 13 + 19) / 6 = 9.5

ATT = (7 + 9 + 14 + 14 + 21 + 24) / 6 = 14.83

(ii) Non-preemptive SJF

AWT = (0 + 7 + 15 + 2 + 18 + 5) / 6 = 7.83

ATT = (7 + 11 + 21 + 4 + 26 + 10) / 6 = 13.17

(iii) Preemptive SJF

AWT = (6 + 0 + 15 + 1 + 18 + 5) / 6 = 7.5

ATT = (13 + 4 + 21 + 3 + 26 + 10) / 6 = 12.83

(iv) Round-robin

AWT = (18 + 12 + 15 + 7 + 18 + 17) / 6 = 14.5

ATT = (25 + 16 + 6 + 9 + 26 + 22) / 6 = 17.33

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

a) Write a code segment in C/C++ where a parent process forks three child processes. Each child process prints a welcome message "**`I AM A NEW CHILD`**", waits for a random delay between 1 and 10 seconds, and then terminates. The parent process will check the termination status of the child processes, and print messages like "**`CHILD WITH PID *** TERMINATED`**", before finally printing the message "**`PARENT PROCESS EXITING …`**".

***

b) Write a code segment in C/C++ where a parent process creates a shared memory segment, allocates an integer array **`X`** of size 10, and populates it with random integers. It then creates two child processes **`C1`** and **`C2`**. **`C1`** will update the array by subtracting the average of the numbers from each of the numbers (i.e. **`X[i] = X[i] – average(X)`**). **`C2`** will wait for 1 second, and then print the contents of the array **`X`**. Finally, the parent process will remove the shared memory segment.

***

c) Clear explain the differences between the following with the help of examples: (i) System call, (ii) Exception, (iii) Internal hardware interrupt.

A system call or syscall is a machine level instruction that cause the mode to switch from user to supervisor, and transfer control to a service routine (typically inside the OS).

An exception is an interrupt that is caused due to the execution of an instruction. Examples include: trying to execute a privileged instruction in user mode, divide by zero, memory access violation, etc.

Interrupt hardware interrupt in a hardware interrupt that is originating from within the processor. An example in the timer interrupt in round-robin scheduling.

4. Answer the following. **[4 + 3 + 4 + 4 = 15]**

a) Three concurrent processes P1, P2 and P3 are concurrently updating a shared variable **xyz** (with initial value of 100) as follows:

P1: `xyz = xyz + 10;`

P2: `xyz = xyz – 20;`

P3: `xyz = xyz * 2;`

What will be the maximum and minimum values of **xyz** after execution of the three processes?

Each instruction will translate into a sequence of machine-level instructions. For example,

```
P1:  LOAD    R1,0(xyz)
     ADD     R1,R1,10
     STORE   R1,0(xyz)
P2:  LOAD    R2,0(xyz)
     SUB     R2,R2,20
     STORE   R2,0(xyz)
P1:  LOAD    R3,0(xyz)
     MUL     R3,R3,2
     STORE   R3,0(xyz)
```

Inconsistency may occur when a process P loads xyz into the register, and then there is a context switch to another process that uses the previous value of xyz to update it. After P resumes, it uses the old value of xyz to carry out the updation.

Minimum Value: **P2 loads xyz, then P1 & P3 finishes, then P1 resumes.**
The final value of xyz will be 80.

Maximum Value: **P1 finishes, P3 loads xyz, then P2 finishes, then P3 resumes.**
The final value of xyz will be 220.

b) Consider the following pseudo-code for a process **Pi**, where "**shared boolean flag[2]**" is a variable declared in shared memory, initialized as: **flag[0] = flag[1] = FALSE;**

```
P (int i) {        // i=0 for P0, i=1 for P1
  while (TRUE) {
    flag[i] = TRUE;
    while (flag[(i+1) % 2) == TRUE);
      < Critical Section >
    flag[i] = FALSE;
      < Remainder Section >
  }
}
```

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

This solution may result in a deadlock between the two processes, where neither P0 nor P1 can proceed to enter their critical section.

This will happen when P0 makes `flag[0] = TRUE`, and then there is a context switch. P1 makes `flag[1] = TRUE`. Now both P0 and P1 will be waiting indefinitely in the while loop before entry to the critical section.

c) The classical Peterson's algorithm for providing mutual exclusion among processes apply to a 2-process system only. Extend the algorithm for providing mutual exclusion between three concurrent processes.

***

d) Suppose that an instruction **swap(int R, int *mem)** is added to a processor that swaps the contents of a register **R** and a memory location **mem** in a single indivisible step. Suggest a solution to the critical section problem using this instruction.

Suppose a memory location "lock" is used to implement the lock, where lock=1 indicates that some process is already inside the critical section.

The entry and exit sections of a process may look like this:

```
do {
        lock = 1;
        swap (R, &lock);
} while (R == 1);
< Critical Section>
lock = 0;
```

5. Justify with reasons whether the following statements are true or false. **[5 x 3 = 15]**

   a) The kernel routines are written as functions that are called from user-level programs whenever some sevice is required from the operating system.

   FALSE: The kernel routines are written as interrupt handlers, that are invoked when a user-level program executes the syscall instruction or there is an interrupt or exception.

   b) We need special support from the hardware to provide mutual exclusion among a number of concurrent processes.

   TRUE: We need some special atomic instruction that carries out a READ-WRITE operation on a memory word in a single indivisible step. Examples are test-and-set and compare-and swap instructions.

   c) The SJF algorithm minimizes the average waiting time of a set of running processes, assuming that all processes arrive at time t = 0.

   TRUE: This can be proved by contradiction. Suppose we start with a SJF schedule. If we swap any pair of processes in the schedule, it is easy to show that the average waiting time can never decrease.

   d) The **`Swapped-Out`** states in a process state transition diagram are required to keep track of the processes that do not require CPU time.

   FALSE: The Swapped-Out states are used to temporarily move some processes in the Ready or Waiting states to the corresponding Swapped-Out states in disk, when the memory space to accommodate the processes is deemed insufficient.

   e) Race condition is possible when we use pipes to communicate between two or more processes.

   FALSE: Pipes uses message communication among processes, and as such the read and write operations in the pipe are atomic operations. There can be no race conditions here.