| | |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| Tot | |

Name ________________________________

Roll no. ________________________________

**CS 347M: Operating Systems Minor**
**Mid-sem Examination. 26 February 2016**

---

**Notes:** 1. Do not detach pages.
2. Answers must be self-explanatory and complete.
3. Verify that this booklet has questions on pages 1-8.
4. **An answer may be continued overleaf.**

**Q.1** ( 15 + 15 Marks )
Programs $P_1, P_2, P_3$ are to be executed in an OS immediately after it is booted. We refer to the moment of booting completion as **time zero. The OS overheads are negligible.** Each program has a loop that runs 3 times. Each iteration of the loop has computations that consume $t_{cpu}$ milliseconds followed by an I/O operation that lasts for $t_{io}$ milliseconds. The characteristics of the programs are as follows:

| Program | $t_{cpu}$ | $t_{io}$ |
|---|---|---|
| $P_1$ | 5 milliseconds | 100 milliseconds |
| $P_2$ | 200 milliseconds | 100 milliseconds |
| $P_3$ | 25 milliseconds | 50 milliseconds |

(a) Answer the following questions if the OS is a time sharing system and programs are entered in the scheduling queue in the order $P_1, P_2, P_3$ at time zero.

(i) Draw a timing chart to show execution of the programs in the OS until all of them complete their operations.

| **Execution on CPU** | | | |
|---|---|---|---|
| Process | Start | End | Remarks |
| $P_1$ | 0 | 5 | I/O from 5 to 105 msec |
| $P_2$ | 5 | 25 | |
| $P_3$ | 25 | 45 | |
| $P_2$ | 45 | 65 | |
| $P_3$ | 65 | 70 | I/O from 70 to 120 msec |
| $P_2$ | 70 | 90 | |
| $P_2$ | 90 | 110 | |
| $P_1$ | 110 | 115 | I/O from 115 to 215msec |
| $P_2$ | 115 | 135 | 80 msec exec so far |
| $P_3$ | 135 | 155 | |
| $P_2$ | 155 | 175 | 100 msec so far |
| $P_3$ | 175 | 180 | I/O from 180 to 230 msec |
| $P_2$ | 180 | 200 | 120 msec so far |
| $P_2$ | 200 | 220 | 140 msec so far |
| $P_1$ | 220 | 225 | I/O from 225 to 325 msec; $P_1$ ends |
| $P_2$ | 225 | 245 | 160 msec so far |
| $P_3$ | 245 | 265 | |
| $P_2$ | 265 | 285 | 180 msec so far |
| $P_3$ | 285 | 290 | I/O from 290 to 340 msec; $P_3$ ends |
| $P_2$ | 290 | 310 | I/O from 310 to 410 msec |
| $P_2$ | 410 | 610 | I/O from 610 to 710 msec |
| $P_2$ | 710 | 910 | I/O from 910 to 1010 msec; $P_2$ ends |

**Evaluation scheme:**

**i. 5 marks for each process, if everything is correct.**

ii. **Deduct 5 marks if preemption at end of time slice not done properly. (Deduct 2 marks for each such instance, deduct max 5 marks.)**

iii. **Deduct 5 marks if time slice remaining when a process starts I/O is wasted (any other process that is available should be scheduled) (Deduct 2 marks for each such instance, deduct max 5 marks.)**

iv. **Deduct 5 marks if preempted process is not added at END of scheduling queue. (Deduct 2 marks for each such instance, deduct max 5 marks.)**

(ii) Give the times when programs $P_1, P_2$, and $P_3$ would complete their operations.

**End times are $P_1$: 325 milliseconds, $P_2$: 1010 milliseconds, $P_3$: 340 milliseconds**

(b) Answer the following questions if the OS is a multiprogramming system, and all programs are started at time zero.

(i) What should be the relative priorities of programs $P_1, P_2, P_3$?

**$P_1$ should have the highest priority, $P_3$ should have intermediate priority and $P_2$ should have the lowest priority.**

(ii) Draw a timing chart to show execution of the programs in the OS until all of them complete their operation.

| **Execution on CPU** | | | |
|---|---|---|---|
| Process | Start | End | Remarks |
| $P_1$ | 0 | 5 | I/O from 5 to 105 msec |
| $P_3$ | 5 | 30 | I/O from 30 to 80 msec |
| $P_2$ | 30 | 80 | 50 msec execution so far |
| $P_3$ | 80 | 105 | I/O from 105 to 155 msec |
| $P_1$ | 105 | 110 | I/O from 110 to 210 msec |
| $P_2$ | 110 | 155 | 95 msec so far |
| $P_3$ | 155 | 180 | I/O from 180 to 230 msec; $P_3$ ends at 230 |
| $P_2$ | 180 | 210 | 125 msec so far |
| $P_1$ | 210 | 215 | I/O from 215 to 315 msec; $P_1$ ends at 315 |
| $P_2$ | 215 | 290 | I/O from 290 to 390 msec |
| $P_2$ | 390 | 590 | I/O from 590 to 690 msec |
| $P_2$ | 690 | 790 | I/O from 790 to 990 msec; $P_2$ ends at 990 |

**Evaluation scheme:**

i. **5 marks for each process, if everything is correct.**
ii. **Deduct 7.5 marks if priorities not correctly assigned.**
iii. **Deduct 5 marks if priorities not correctly implemented. (Deduct 2 marks for each such instance, deduct max 5 marks.)**

(iii) Give the times when programs $P_1, P_2$, and $P_3$ would complete their operations.

**End times are: $P_1$: 315 milliseconds, $P_2$ 990 milliseconds, $P_3$ 230 milliseconds.**

**Q.2** ( 10 + (10 + 10) Marks )

(a) A process wishes to perform an I/O operation on an I/O device that has already been allocated to it. ou are required to list **in chronological order** all actions that occur in (i) computer hardware, and (ii) in various routines of the OS, until the OS hands over the CPU to a user program. Give all relevant information for each action.

**Evaluation scheme: 1 mark for each of the following steps.**

1. **Process makes a system call with an appropriate number**
2. **System call number is saved in IC field and program interrupt takes place**
3. **PSW is saved**
4. **New PSW of program interrupt is loaded, so control goes to interrupt processing routine for program interrupt**
5. **Program context is saved**
6. **Event handling routine for I/O Request is called**
7. **I/O is started**
8. **state of process is changed to Blocked**
9. **Control goes to scheduler, it picks another program**
10. **Program is dispatched**

(b) Describe **all actions** the OS should perform in each of the following situations concerning parent and child processes. Give all relevant details. (*Note:* Only OS actions are asked.)

(i) A process P wishes to create a child process Q.

**Evaluation scheme:**

1. **OS saves context of process P (1 Mark)**
2. **OS changes state of process P to *ready* (1 Mark)**
3. **OS sets id of Q as id of a child in P's PCB. (1 Mark)**
4. **Creation of Q's PCB: (Max marks 6: 1.5 mark for each of the following.)**
   - **OS creates PCB for process Q.**
   - **OS enters start address of Q's code in PC field of CPU state of Q.**
   - **OS sets state of Q to *ready*.**
   - **OS sets id of P as id of parent in Q's PCB.**
5. **Scheduling is performed (0.5 marks)**
6. **Dispatching is performed (0.5 marks)**

(ii) Process P wishes to sleep until child process Q terminates.

**Evaluation scheme**

1. **When P makes a system call to sleep until Q terminates (Max marks 5)**
   - **Changes state of P to** *ready* **(1 mark ONLY IF none of following steps are mentioned)**
   - **OS takes Q's id and accesses its PCB (2 marks)**
   - **If Q is not already terminated, it changes state of P (in its PCB) to** *blocked* **(2 marks)**
   - **Scheduling followed by dispatching (1 mark)**
2. **When Q terminates (Max marks 5)**
   - **Changes state of Q to** *ready* **(1 mark ONLY IF none of following steps are mentioned)**
   - **OS takes parent's id from Q's PCB (2 marks)**
   - **If parent is** *blocked* **for child termination, changes state of the parent to** *ready* **(2 marks)**
   - **Scheduling followed by dispatching (1 mark)**

**Q.3** ( 12 + 4 + 4 Marks )
A pseudo-code is given for a concurrent program that consists of two processes called P and Q. Process Q has two phases in its execution called phase A and phase B (Each phase contains some computations and I/O operations; however, their details are irrelevant here.) After executing phase A, process Q should execute phase B only after process P has sent it a user defined signal.

*Note:* install_signal_handler(), send_signal(), and sleep() are system calls. signal1 is the name of a user defined signal. x is a local variable of process Q. **while (condition) {** < **body** > **}** is a while loop ==> if the conditions is true, its body is executed and the condition is checked once again.

| Process P | Process Q |
|---|---|
| ... | x := 0; |
| ... | ... |
| ... | { Code of phase A } |
| ... | ... |
| ... | install_signal_handler(signal1, alpha()) |
| ... | ... |
| send_signal(Q, signal1); | **while** (x = 0) |
| ... | { sleep (5 seconds); } |
| ... | ... |
| ... | { Code of phase B } |
| ... | ... |
| ... | function alpha() |
| ... | x := 1; |
| ... | return(); |

(a) Does the pseudo-code implement the required functionality correctly under all conditions? If not, suggest changes that would ensure correctness under all conditions. In either case, give a detailed justification for your answer.

**Evaluation scheme**

1. **It does not work in the following situation (6 Marks)**
   - **P sends a signal before Q installs signal handler (3 Marks)**
   - **Correct explanation (3 Marks)**
2. **Improvement (6 Marks)**
   - **Let Q install the signal handler immediately after setting x :=0 OR let Q install signal handler before setting x := 0 (3 Marks)**
   - **Correct explanation (3 Marks)**

**Grace marks:** 2 Marks if an answer says that the scheme works and gives a complete explanation of the one situation in which it would work.

(b) Comment on execution efficiency of the pseudo-code. Is there any way you could improve the execution efficiency? Give a detailed justification for your answer.

**Evaluation scheme (4 Marks)**

**(a) The loop in Q iterates every 5 seconds. To improve efficiency its period could be increased to a large value. (3 Marks)**

**(b) However, the delay would increase. (1 Mark)**

(c) Comment on the delay between sending of the signal by process P and execution of phase B in process Q. Is there any way you can reduce or eliminate the delay? Give a detailed justification for your answer.

**Evaluation scheme (4 Marks)**

**(a) The delay is caused by the sleep in Q's loop. The sleep period could be reduced to reduce delay. (3 Marks)**

**(b) However, the overhead would increase. (1 Mark)**

**Q.4** ( 10 + 10 Marks )

(a) A program is initially coded as a sequential program. The user then realizes that the program contains many activities that are independent of one another, so (s)he decides to convert the program into a concurrent program by simply adding "create_thread" kind of system calls **without making any other changes.** (S)he realizes that all threads are I/O bound in nature.

The program is to be executed in an OS that uses priority-based scheduling by assigning the highest priority to it. If the purpose of concurrentizing the program is to reduce its elapsed time, should the user use kernel-level threads (KLTs) or user-level threads (ULTs)?

**Give a detailed justification for you answer, mentioning why you would choose one alternative and why you would reject the other alternative.**

**Evaluation scheme**

**KLT (Max marks 5)**

- **Switching overhead higher than in ULTs.(1 Mark)**
- **If one thread performs blocking I/O, another thread can be scheduled.(2 Marks)**
- **Hence overlap between I/O of one thread with computations of another thread.(2 Marks)**
- **Hence elapsed time reduces.(1 Mark)**

**ULT (Max marks 5)**

- **Switching overhead lower than in KLTs.(1 Mark)**
- **However, when one thread performs I/O, the entire process blocks.(2 Marks)**
- **Hence no overlap between computations and I/O, or I/O and I/O (2 Marks)**
- **Hence no reduction in elapsed time.(1 Mark)**

**Hence KLTs preferred. (1 Mark)**

**Grace marks:** 2 marks if the only correct relevant thing in an answer is that ULTs have 100x low overheads while KLTs have 10x low overheads.

(b) A process P has been in execution in an OS for some time, during which time it executed on the CPU a few times. At some time instant T, we check the state of process P and find that it is in the *ready* state. In each of parts (i) and (ii) below, you are required to explain how and why the process would have entered the *ready* state by mentioning some recent events in the hardware and the OS. **Give a detailed explanation and clear justifications.**

(i) If the OS is a multiprogramming OS.

**Evaluation scheme** (Max marks 5)

(Many situations are possible; however, it is enough to mention any one situation. Each situation would be evaluated along the following lines.)

- All higher priority processes are blocked. State change Ready → Running when scheduled (2 Marks)
- An interrupt occurs and OS concludes that a higher priority process can come out of the blocked state. So a higher priority process becomes ready (2 Marks)
- Process state changed to ready and the higher priority process is scheduled. (1 Mark)

(ii) If the OS is a time-sharing OS.

**Evaluation scheme** (Max marks 5)

(Many situations are possible; however, it is enough to mention any one of them. Each situation would be evaluated along the following lines.)

- The process is at the head of the scheduling queue. Hence State change Ready → Running when scheduled (2 Marks)
- An interrupt occurs indicating that the time slice of the process has elapsed. (2 Marks)
- Hence process is preempted and its state is changed to Ready. (2 Marks)

**Grace marks:** 2 marks in each part if an answer does not mention relevant events but merely mentions "Higher priority process becomes active" or "Time slice ends".

— Paper Ends —