University ofOttawa
School of Information Technology and Engineering

Course: CSI3310 – Operating Systems Principles
SEMESTER: Summer 2005

Professor: Stefan Dobrev
Date: June 9 2004
Hour: 10:30-12:30 (2 hours)
Room: STE G0103

Midterm Exam

120 minutes, closed book

The exam consists of three (3) parts:

Part 1	**Multiple Choice**	**6 points**
Part 2	**Short Answers**	**10 points**
Part 3	**Problem Solving**	**12 points**
Total		**28 points**

25 points represent 100% for the midterm (25% of the final grade)

Part I: Multiple choice (6 points):

1. One of the following is true about privileged instructions

    a. When a privileged instruction is executed, the CPU switches to kernel mode.
    b. When a privileged instruction is executed while the CPU is in kernel mode, an exception is raised and the CPU switches back to user mode.
    c. If a privileged instruction is executed while the CPU is in user mode, the user is prompted for the supervisor password.
    d. The only privileged instructions are the input/output instructions.
    e. Executing a privileged instruction while the CPU is in user mode causes exception.

2. One of the following actions does not necessarily occur during context switch from process P:

    a. saving the program counter of P
    b. updating the PCB of P
    c. placing P's PCB in the appropriate queue
    d. suspending P
    e. choosing the next process to be given the CPU

3. Mark all (there are several of them) resources that are **NOT** shared by threads within the same process:

    a. program counter
    b. open files
    c. the code segment
    d. the data segment
    e. the stack

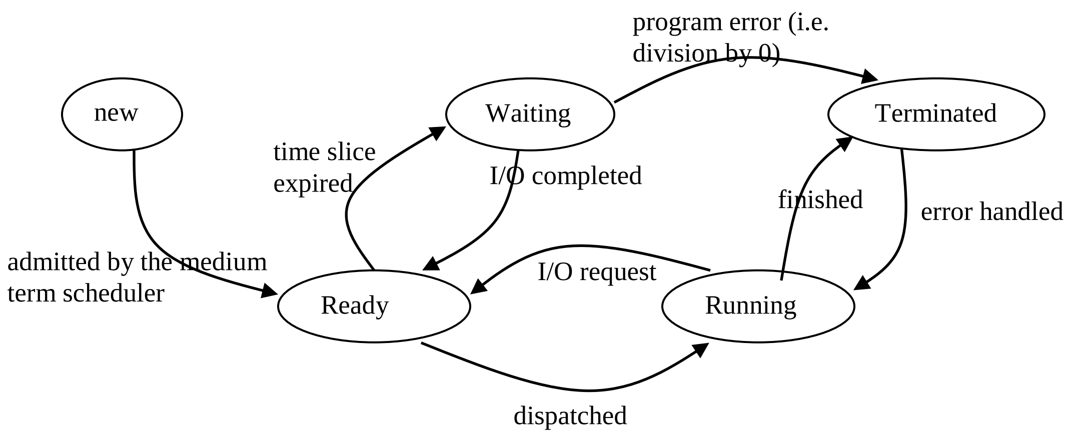
4. One of the following statements is incorrect about pure user threads
    a. Context switches between threads of the same process are done without the involvement of the OS kernel.
    b. On a multi-CPU computer, two threads of the same process can be executed on different CPUs.
    c. When one thread of a process blocks, no other thread of that process will be given the CPU.
    d. There can be many user threads that are mapped to single kernel thread
    e. The threads of the same process can be in different states.

5. One of the following statements is correct about monitors
    a. Mutual exclusion is ensured by the use of conditional variables.
    b. A call to cwait() is always blocking.
    c. With monitors, mutual exclusion must be programmed explicitly.
    d. A call to csignal() always wakes-up another process or thread.
    e. The shared variables to which the monitor guards access are called conditional variables.

6. (2 points) For each of the following statements mark whether they are true or false
    a. True [ ] False [ ] Mutual exclusion requirement means that at any moment, at most one process per CPU can be in the critical section
    b. True [ ] False [ ] Interrupt disabling ensures mutual exclusion on computers with one CPU.
    c. True [ ] False [ ] Progress requirement means that each process that enters the critical section must exit it before the next process enters the critical section.
    d. True [ ] False [ ] A time-shared system providing low-response time environment must use preemptive scheduler.
    e. True [ ] False [ ] Round-robin scheduler does not have the starvation problem
    f. True [ ] False [ ] Multilevel-feedback queue scheduler ensures efficient scheduling of processes with long CPU bursts, but processes with very short CPU bursts get starved.

PART II: Short answer questions (10 points)

Question 1 (4 points): Correct all errors in the following diagram of process states:



Question 2 (2 points): Mutual exclusion solutions

(1 point) Why use semaphores when we have hardware instructions (test&set, xchng)?

(1 point) Why use monitors when we have semaphores?

=============

Do not write past here, I don’t want novels.

Question 3 (4 points): Simulate Shortest Job First Scheduler

System specification:

- Each CPU burst is followed by an I/O operation taking 16 time units
- The I/O operations of different processes overlap and do not interfere, i.e. if P1 starts to wait at time 10 and P2 starts to wait at time 20, P1 will become ready at time 10+16 = 26 and P2 will become ready at time 20+16=36.

Scheduler specification:

- Non-preemptive shortest-job-first scheduler
- The initial estimated CPU burst length τ_0 for each process is 6 time units
- $\alpha = ½$ (the exponential averaging parameter)

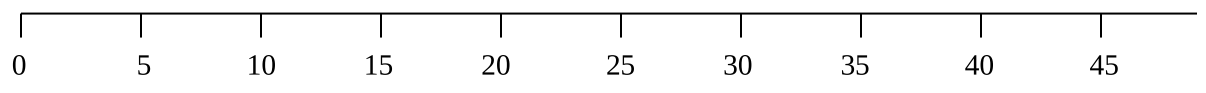
Process specification:

- There are four processes P0, P1, P2 and P3
- Arrival time of processes are as follows:
    - P0: 0, P1: 2, P2: 20, P3: 24
- The real CPU burst times of processes are as follows:

  P0: 8, 9, 10
  P1: 14, 14, 10
  P2: 2, 4, 2, 1, 4, 3
  P3: 6, 4, 7, 4

Your task:

- Simulate the Shortest Job First scheduler scheduling these 4 processes for the first 45 time steps
- In the following line, mark the times of the context switches and mark which process is executing at a given moment:



PART III: Problem questions (12 points)

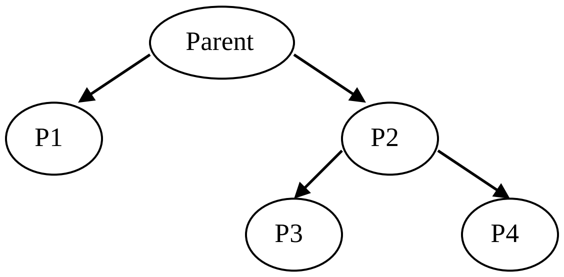
Problem 2 (7 points): Consider the following code snippet (you may assume that fork() never fails)

```
if ((p1_pid = fork())== 0)
    printf("I am P1 and I am proud of it.");
} else {
    if ((p2_pid = fork()) == 0) {
        p3_pid == fork();
        printf("I am P3. I like it.");
        p4_pid == fork();
        printf("I am P4. Get used to it.");
    } else
        printf("I am the parent process, obey or die!");

}
```

a) (1 point) How many processes (excluding the parent) will be created?

b) (2 points) Correct the above code so that the following scheme captures the process tree:
(1 point) Modify the code to make sure that P4 is launched only after P1 has terminated.



c) (2 points) Modify the code so that there is a pipe going from P1 to P3. Make sure that P1 has open only the write end of the pipe, P3 has open only read end of the pipe and nobody else has anything open.

   (1 point) Have the standard output of P1 redirected to the write end of that pipe, and the standard input of P3 redirected to the read end of that pipe. The P1's printing "I am P1 and I am proud of it" should already be sent to the pipe.

Problem 3 (5 points): Consider the problem of amusement ride in an airplane (as discussed in the review lecture): There is one airplane of capacity CAPACITY passengers providing amusement rides. The passengers are arriving to the airport, wait until the plane is ready for boarding, then board (one by one). When the plane is full, it takes off for the ride, then it lands. After the lane landed, the passengers leave (no leaving while in air!). When all passengers left, new passengers can start boarding.

Consider the following (pseudocode) solution for this problem:

```
int onboard = 0;

Semaphore mutex = new Semaphore(1); /* initialized to 1 */
Semaphore boarding = new Semaphore(0);
Semaphore full = new Semaphore(0);
Semaphore empty = new Semaphore(0);
Semaphore leave = new Semaphore(0);
```

Airplane code:
```
while(true) {
    boarding.signal(); /* ready for boarding */
    full.wait();       /* wait until full */
    /* takeoff and fly, then land */
    leave.signal();
    empty.wait();
    /* all passengers have left */
}
```

Passenger code:
```
while(true) {
    /* arrive to the airport, wait for the airplane */
    boarding.wait();
    mutex.wait();
    onBoard++; /* board the airplane */
    if (onBoard == CAPACITY) full.signal();
    else boarding.signal();
    mutex.signal();
    /* enjoying the ride */
    leave.wait();
    mutex.wait();
    onBoard--; /* leave the airplane */
    if (onBoard > 0) leave.signal();
    else empty.signal();
    mutex.signal();
    /* enjoy the solid ground, wait until the bowels settle */
}
```

a) (2 points) Is it possible that a passenger enjoys the ride before the plane has taken off? If no, justify why. If yes, correct the code – you might need to introduce a new semaphore or so. (No need to fully rewrite the code here, you may write/insert into the code provided on the previous page.)

b) Assume there are VIPs who would also like to take a ride. However a VIP is big, fat and ugly and needs bodyguards as well, so it takes 5 spaces instead of 1. Moreover, VIPs don't like to wait: if VIP boards, the airplane should take off even if it is nowhere near full. If there are less then 5 spaces left, the VIP cannot fit and the airplane must take-off (without the VIP) so it lands sooner and the VIP does not wait unnecessarily.

Your task: (2 points) Write the code for the VIP. (1 point) Make sure that even if there are plenty of waiting passengers and the VIP process does not have higher priority in being waken-up by signal , it boards the airplane first.