Name, SURNAME and ID ⇒

Middle East Technical University
Department of Computer Engineering

CENG 334

Section 2 and 3
Spring '2010-2011

Midterm

- **Duration: 120** minutes.
- **Exam:**
  - This is a **closed book**, **closed notes** exam. No attempts of cheating will be tolerated. In case such attempts are observed, the students who took part in the act will be prosecuted. The legal code states that students who are found guilty of cheating shall be expelled from the university for **a minimum of one semester**!
- **About the exam questions:**
  - The points assigned for each question are shown in parenthesis next to the question.
  - Whereever available, use the boxes to write down your answers.
- **This booklet consists of *8* pages including this page. Check that you have them all!**
- **GOOD LUCK !**

Question 1	
Question 2	
Question 3	
Question 4	
Question 5	
Question 6	
Total ⇒	

1 (15 pts)

(a) Write down one advantage and one disadvantage of using user-level threads to kernel-level threads.

(b) What's a trap? What generates it and what does the OS do when it gets one?

(c) What's a process and what constitutes its essential components of a process?

(d) What's preemption? What can a preemptive scheduler do that non-preemptive schedulers can't do?

(e) What's priority inversion? Give a scheduling policy in which priority inversion can happen? How can it be solved?

2 (15 pts)

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
main() {

  if (fork() == 0) {
    if (fork() == 0) {
      printf("3");
    }
    else {
      pid_t pid; int status;
      if ((pid = wait(&status)) > 0) {
        printf("4");
      }
    }
  }
  else {
    if (fork() == 0) {
      printf("1");
      exit(0);
    }
    printf("2");
  }

  printf("0");

  return 0;
```

Out of the 5 outputs listed below, circle only the valid outputs of this program. Assume that all processes run to normal completion.

Note that circling wrong outputs will be punished such that random guesses will receive zero point on average.

A. 2030401 B. 1234000 C. 2300140
D. 2034012 E. 3200410

3 (8 pts)

Consider the following synchronization code for two threads:

```
int turn = 0; // shared

void thread1(){
       while (turn !=0) ;
       critical_section();
       turn = 1;
}
void thread2(){
        while (turn != 1) ;
        critical_section();
        turn = 0;
}
```

Is this a good solution to the critical-section problem? Consider the requirements for synchronization, and discuss which ones it satisfy and which ones it doesn't?

4 (14 pts)

Consider 4 threads, called as A, B, C and D. Using semaphores, ensure that A completes its `job()` before any other process (B, C, or D) starts, and B completes its `job()` before C or D, but C and D may execute their `job()` concurrently.

You can declare and initialize semaphores such as `sem = semaphore(1);` and use them by calling two methods as `sem.up();` or `sem.down();`.

```
// declarations


void A(){









}
void B(){









}
void C(){









}
void D(){









}
```

5 (30 pts)

We are interested in cooking a pan of *menemen* which requires three *eggs* and one *tomato*. Create a Mesa-type monitor with methods `EggComes()` and `TomatoComes()`, which wait until a pan of menemen can be cooked. Don't worry about explicitly cooking menemen; just wait until three *egg* threads and one *tomato* thread can be grouped together. For example, if three *egg* threads call `EggComes()`, and then a fourth *tomato* thread calls `TomatoComes()`, the third thread should wake up the first three threads and they should then all return.

Use the following skeleton to write your code. Note that although a number of declaration are provided for you, you are free to declare and use more int's or condition variables as necessary. Assume that the functions `wait(cv)`, `notify(cv)`, and `notifyAll(cv)`, are available where `cv` is a declared condition variable.

```
monitor menemen{
    int wE= 0; // Number of waiting eggs
    int wT= 0; // Number of waiting tomatoes
    condition waitingE; // used by tomatoes
    condition waitingT; // used by eggs
```

```
    void eggComes(){




}
}
```

```
    void tomatoComes(){




    }
```

6 (18 pts)

Three processes are at the ready queue of the scheduler in the order A, B and E at time 0. If you use the FCFS policy the scheduling executed is as shown in the Gantt chart below. In the chart, A, B, and E represents the CPU use of these processes whereas a, b and e denote the I/O's of processes respectively.

Fill in the scheduling executed by Round robin (RR) with a timeslice of 1 unit, Shortest-Job-First (SJF), and Shortest-Remaining-Time-First (SRTF). For SJF and SRTF, assume that A, B and E uses CPU bursts of 5, 1 and 6 time units. If at a given time slice the CPU is being used by, say A, whereas the I/O requests of B and E are active, then fill in that time slice with Abe **vertically**. Also assume that the I/O device can handle multiple I/O requests concurrently. In SRTF, if the remaining CPU burst of the running process equals to the remaining CPU burst of another process in the ready list, then no context switch should occur. Grading: RR: 4pts; SJF: 6 pts; SRTF: 8 pts.

	0										1									
Time	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
FCFS	A					a					A				a		B		b	
RR																				
SJF																				
SRTF																				

	2										3									
Time	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
FCFS	b				E						e									
RR																				
SJF																				
SRTF																				