University of Minnesota
Department of Computer Science
CSci 5103 - Fall 2016 (Instructor: Tripathi)
Midterm Exam 1 — Date: October 17, 2016 (4:00 – 5:15 pm)

(Time: 75 minutes) Total Points – 100

This exam contains five questions.

CLOSED BOOK/CLOSED NOTES – NO Laptops, PDAs, or Cell Phones

Please write your answer in the space provided with each question.

STUDENT NAME:
STUDENT ID:

Problem 1	Problem 2	Problem 3	Problem 4	Problem 5	Total
35	20	5	20	20	100

Problem 1 (35 points):

(a) (4 points) Select all of the correct choices as your answers for the following statements. (Note: negative points for wrong answers.)

1. The parameters to a system call are passed by the calling process to the kernel code by writing them to:
   (a) Kernel's stack
   (b) Process's stack
   (c) CPU register
   (d) Kernel memory

2. Given a set of jobs with the same priority level, ideally a CPU scheduling algorithm should pick the job which has
   (a) Longest CPU burst
   (b) Shortest CPU burst
   (c) Average CPU burst
   (d) Largest number of open files

(b) (5 points) Even when a process is swapped out to disk, some information has to be maintained for it in the process table which is always kept in the primary memory by the kernel. Identify at least six items of information that must always be maintained in the primary memory.

(c) (4 points) In this problem you have to compare the performance of two scheduling disciplines — FCFS and Round-Robin. Given is a fixed set of jobs in which all jobs have the same CPU service-time requirements. Assume that the time-quantum used in the Round-Robin scheme is much smaller (say, by at least one or two orders of magnitude) than the CPU requirement of a job.

Which of these two disciplines will result in higher average turn-around time?
Select Answer: FCFS Round-Robin

Which of these two disciplines will result in higher variance for the turn-around time?
Select Answer: FCFS Round-Robin

(d) (2 points) Do functions in the kernel code ever make system calls using trap instructions?
Select Answer: YES NO.

(e) (2 points) Consider a multithreaded process with threads implemented by a user-level library. Select the correct answer
Is there a separate kernel-level stack for each thread? NO YES
Is there a separate user-level stack for each thread? NO YES

(f) (2 points) Consider a multiprogrammed computer system with four processes. Suppose that each process spends $x\%$ of its time in performing I/O. What is the probability of CPU being idle?

(g) (4 points) On a single CPU system, under what conditions the use of a multithreaded server (with kernel level threads) is justifiable? Under what conditions would you use a single-threaded server?

(h) (4 points) A real-time system has four periodic events with periods 50, 100, 200, and 20 msec each. Suppose that the events require 35, 20, 10, and x msec of CPU time, respectively. What is the largest value of x for which the system is schedulable.

(i) (8 points) In this problem you are to compare reading a file using a single-threaded server and a multithreaded server. (Assume that threads are supported by the kernel.) Suppose that it takes 15 milliseconds to get a request for work, despatch it, and do the rest of the necessary processing, assuming that the data needed are in the main memory. If a disk operation is needed, as is the case one-third of the time, an additional 75 millisecond is required, during which time the thread sleeps.

Part A:(4 points) How many requests per second can this server handle if it is single-threaded?

Part B:(4 points) How many requests per second can this server handle if it is multi-threaded?

Problem 2: (20 points)

Five batch jobs *A, B, C, D* and *E* arrive at time 0 in this given order. They have estimated running times of 10, 8, 12, 4, and 6 seconds. Their externally determined priorities are 5, 1, 3, 4, and 2, respectively, with 5 being the highest priority. For each of the following four scheduling algorithms, determine the average turn-around time and average waiting time. Ignore process context-switch overhead.

(a) Round-Robin with processor-sharing, i.e. each jobs gets its fair share of the CPU.

(b) Priority based scheduling

(c) Shortest Job First

(d) FCFS

Answer for Problem 2

Problem 3 (5 points):

The following solution for the *two-process critical section problem* was presented in 1966. Later it was found to be incorrect. Show an example which violates the mutual exclusion requirement.

The two processes are numbered 0 and 1. The shared variables are:

```
  var flag: array[0..1] of boolean /* initially false */
      turn: 0..1;

Process (i)
    /* i can either 0 or 1, j is (i+1) mod 2  */
1.    repeat
         // ENTRY PROTOCOL
2.       flag[i] = true;
3.       while ( turn != i )  do   {
4.            while ( flag[j] ) do skip ;
5.            turn = i;
6.        }
7.        CRITICAL SECTION
         // EXIT PROTOCOL
8.       flag[i] = false;
         ...
10     until  false;
```

Problem 4 (20 points): Write a monitor called *ExchangeBox* that will be used by two asynchronous processes to swap two integer values.

Each process would call the monitor procedure *Swap(integer i)*.

Whichever process calls this procedure first would be blocked, waiting for the second process to make this call. When the other process calls Swap, their integer parameters would be swapped and then respectively returned as the result of the call.

For example, suppose that process A calls Swap(10) and process B calls Swap(20). Process A will return with result value 20, and B with value 10.

Problem 5 (20 points):

The following solution was presented and discussed in the lectures for the *Sleeping Barber Problem*. Assume that we have only one Barber process and any number of Customer processes. For your reference, the code is given below. The Barber process executes the procedure called *barber*. An arriving Customer process executes the procedure *customer*.

```
constant int   N  = 5   /*  Number of CHAIRS */
semaphore customerAvailable (initial value = 0);
semaphore barberAvailable (initial value = 0);
semaphore mutex  = 1  // for critical section implementation
int waiting = 0; //  number of customers waiting in the room

void barber() {
   while  ( TRUE ) {
        wait ( customerAvailable );
        wait ( mutex ) ;
        waiting =  waiting - 1;
        signal ( mutex );
        signal ( barberAvailable );
        cut_hair;
    }
}

void customer ( ) {
        wait (mutex )
        if ( waiting < N ) {
              waiting = waiting  +  1;
              signal ( mutex ) ;
              signal ( customerAvailable );
              wait ( barberAvailable );
        } else {
              signal ( mutex );    // shop is full with no empty chair to wait
        }
}
```

When the semaphores are weakly implemented, there is no guarantee that the customers would be served FCFS.

You are asked to rewrite the above code to ensure FCFS order for the customers to get haircuts.

Hint: Consider using one semaphore per waiting room chair, and impose a queuing order among the waiting customers, based on their arrivals.

Answer for Problem 5

Blank page, if you need one for writing answers or scratch work.