Name ____________________

Student Number ____________________

Instructor: S. Qiao

CS/SE 3SH3

Day Class
Duration of examination: 50 minutes
McMaster University Midterm Examination February 2013

This examination paper includes **5** pages and **8** questions. You are responsible for ensuring that your copy of the paper is complete. Bring any discrepancy to the attention of your invigilator.

SPECIAL INSTRUCTIONS: This paper must be returned with your answers. Use of McMaster standard (Casio-FX991) calculator only is allowed.

1. (2 marks) Which of the following statements is *false*? If you think all the choices are true, you may answer none.
A time-sharing operating system

   (a) allows many users to share the computer simultaneously

   (b) provides short response time

   (c) is a logical extension of multiprogramming

   (d) does not provide direct communication between the user and the system

   Answer: d

2. (5 marks) List five entries in a PCB (process control block):

   (a) id

   (b) stack pointer

   (c) status

   (d) open files

   (e) address space

3. (3 marks) A process can be in one of the five states: *finish* (*terminated*), *new*, *ready*, *running*, and *wait*. What are the possible state(s) following the state *running*?

Answer: finish, ready, wait

4. (2 marks) Which of the following statements is *false*? If you think all the choices are true, you may answer none.
When a process terminates, it must

(a) close open files

(b) notify its parent

(c) notify its siblings (processes having the same parent)

(d) deallocate memory

Answer: C

5. (3 marks) Including the initial parent process, how many processes are created by the following program?

```
#include <stdio.h>
#include <unistd.h>

int main() {
    /* fork a child process */
    fork();

    /* fork another child process */
    fork();

    /* and fork another */
    fork();

    return 0;
}
```

Answer: 8

6. (3 marks) The following multithreaded C program using the Pthreads API computes the summation $sum = \sum_{i=1}^{N} i$. The upper bound N is provided on the command line, argv[1].

```
#include <pthread.h>
#include <stdio.h>

int sum;                /* shared by the thread(s) */
void *runner(void *param);

int main(int argc, char *argv[]) {

    pthread_t tid;       /* thread id */
    pthread_attr_t attr; /* thread attributes */

    /* set the default attributes */
    pthread_attr_init(&attri);
    /* create a thread */
    pthread_create(&tid, &attri, runner, argv[1]);
    /* wait for the thread to exit */
    pthread_join(tid, NULL);

    printf("sum = %d\n", sum);
}

void *runner(void *param) {

    int i, upper = atoi(param);
    sum = 0;

    for (i = 1; i <= upper; i++)
        sum += i;

    pthread_exit(0);
}
```

Which of the following statements is true? If you think all the choices are false, you may answer none.

(a) the child thread tid computes and prints the result

(b) the parent thread main computes the prints the result

(c) the child thread tid computes the result and the parent thread main prints the result

(d) the parent thread main computes the result and the child thread tid prints the result

Answer: C

next page ...

7. (4 marks) Suppose in a certain operating system two processes called OBSERVER and REPORTER share a variable called *count*. When OBSERVER observes an event, it increments *count*. Periodically, REPORTER is run to print out the number of events that OBSERVER has observed since the last time REPORTER was run and reset *count* to 0. Initially, *count* is 0. The code for each process is:

```
         OBSERVER:  while (true) {
    01                 observe an event
    02                 lw  $7, count     % load count into register 7
    03                 add $7, $7, one   % add one to register 7
    04                 sw  $7, count     % store register 7 in count
                    }

         REPORTER:  while (true) {
    R0                 print(count)
    R1                 lw  $6, count
    R2                 mv  $6, zero      % move zero to register 6
    R3                 sw  $6, count
                    }
```

Will the number of events observed necessarily be reported accurately? If so, why? If not, give an execution sequence that causes an inaccurate report.

Answer: No. Consider the following execution sequence:

01, 02, 03, 04, 01, 02, 03, R0, 04, R1, R2, R3

Two events are observed, but only one is reported.

next page ...

8. (6 marks) Given the Nachos 4.02 semaphore constructor:

```
Semaphore::Semaphore(char* debugName, int initialValue)
{
    name = debugname;
    value = initialValue;
    queue = new List<Thread *>;
}
```

Complete the implementation of the semaphore operation P():

```
void
Semaphore::P()
{
    Interrupt *interrupt = kernel->interrupt;
    Thread *currentThread = kernel->currentThread;

    // disable interrupts
    IntStatus oldLevel = interrupt->SetLevel(IntOff);

    if (value <= 0) {                  // semaphore not available

        queue ->Append(currentThread);
        currentThread -> Sleep(FALSE);

    } else {

        value --;

    }

    // re-enable interrupts
    (void) interrupt->SetLevel(oldLevel);
}
```

END