COMP 2004
Instructor: Dr Vinicius Prado da Fonseca
Assignment 5

1. (10%) The pseudocode below illustrates an array-based stack's basic push() and pop() operations. If this algorithm could be used in a concurrent environment, answer the following questions:

    a. What data have a race condition?

    b. Describe how could the race condition be fixed.

```
push(item) {
   if (top < SIZE) {
        stack[top] = item;
        top++;
   }
   else
        ERROR
}

pop() {
   if (!is_empty()) {
        top--;
        return stack[top];
   }
   else
        ERROR
}

is_empty() {
   if (top == 0)
        return true;
   else
        return false;
}
```

2. (10%) Discuss the trade-off between fairness and throughput of operations in the readers-writers problem. Propose a method for solving the readers-writers problem without causing starvation.

3. (80%) Assume that a finite number of resources of a single resource type must be managed. Processes may ask for a number of these resources and will return them once finished. For example, many commercial software packages provide a given number of licenses, indicating the number of applications that may run

concurrently. When the application is started, the license count is decremented. When the application is terminated, the license count is incremented. If all licenses are in use, requests to start the application are denied. Such a request will be granted only when an existing license holder terminates the application and a license is returned.

The following program segment manages a finite number of instances of an available resource. The maximum number of resources and the number of available resources are declared as follows:

```
#define MAX_RESOURCES 5
int available_resources = MAX_RESOURCES;

/** When a process wishes to obtain a number of resources, it invokes the
* decrease_count() function: */
/* decrease available_resources by count resources */
/* return 0 if sufficient resources available, */
/* otherwise return -1 */
int decrease_count(int count) {
          if (available_resources < count)
             return -1;
          else {
             available_resources -= count;
          return 0;
          }
}

/** When a process wants to return a number of resources, it calls the
increase_count() function: /*
/* increase available_resources by count */
int increase_count(int count) {
          available_resources += count;
          return 0;
}
```

The preceding program segment produces a race condition. Do the following:

a. (15%) Identify the data involved in the race condition.
b. (15%) Identify the location (or locations) in the code where the race condition occurs.
c. (50%) Using a mutex lock, fix the race condition. It is permissible to modify the decrease_count() and increase_count() functions so that the calling process is blocked until sufficient resources are available.

   The code for this solution must read a set of integers from the command line and create threads that perform the decrease and increase count operations. Each integer will be negative to decrease resources and

positive to increase resources. As indicated, decrease_count() can be modified but must return -1 if the requested resources > MAX_RESOURCES. The output should be the results from each thread. For example, a MAX_RESOURCES of 5 with three calls would result in the following:

```
./a.out -4 -2 2

0, -1, 0
```

Edit the code provided (a5q3_sample.c) that has a framework you can use to solve this question.