# COSC1112/1114: Operating Systems Principles

## Tutorial 05 (week 06)

1. Race conditions are possible in many computer systems. Consider a banking system with two methods: deposit(amount) and withdraw(amount). These two methods are passed the amount that is to be deposited or withdrawn from a bank account. Assume that a husband and wife share a bank account and that concurrently the husband calls the withdraw() method and the wife calls deposit(). Describe how a race condition is possible and what might be done to prevent the race condition from occurring.

2. 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 —once finished—will return them. As an 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 requests will be granted only when an existing license holder terminates the application and a license is returned. The following program segment is used to manage 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. Identify the data involved in the race condition.
b. Identify the location (or locations) in the code where the race condition occurs.
c. Using a semaphore, fix the race condition.

3. Describe two kernel data structures in which race conditions are possible. Be sure to include a description of how a race condition can occur.

4. Discuss the tradeoff between fairness and throughput of operations in the readers-writers problem. Propose a method for solving the readers-writers problem without causing starvation.

5. Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.