# CSE4/521 Introduction to Operating System
# Spring 2010
# Project #2
# Multi-Threaded(MT) Programming: Communication and Synchronization

Bina Ramamurthy

February 20, 2010

## 1 Objective

To familiarize the students with:

- Thread creation and control (create, join, termination, and synchronization).
- Designing and implementing a multi-threaded application.

## 2 Problem Statement

1. (25 points) **Bounded-buffer Producer/Consumer problem** Implement a solution to the bounded-buffer producer/consumer problem. Implement the solution using synchronization primitives (pthread_mutex_t for mutual exclusion and sem_t for minding the buffer size) provided by the POSIX threads. Your implementation will consist of a control program that (i) initializes the buffer and the synchronization variables and (ii) creates and terminates the threads for the producer and the consumer. The producer generates printable characters and places them into the buffer and the consumer pulls the characters out of the buffer one at a time and prints them out. Let the control program simulate all possible conditions: buffer empty (consumer waiting), buffer full (producer waiting), buffer not empty (both producer and consumer working). Assume that production and consumption per character take the same amount of time.

2. (25 points) **You identify the IPC pattern** A student majoring in anthropology and minoring in computer science has embarked on a research project to see if African baboons can be taught about deadlocks. He locates a deep canyon and fastens a rope across it, so the baboons can cross hand-over-hand. Several baboons can cross at the same time, provided that they are all going in the same direction. If eastward moving and westward moving baboons ever get onto the rope at the same time, a deadlock will result (the baboons will get stuck in the middle) because it is impossible for one baboon to climb over another one while suspended over the canyon. If a baboon wants to cross the canyon, it must check to see that no other baboon is currently crossing in the opposite direction. Write a program using semaphores that avoids deadlock. Do not worry about a series of eastward moving baboons holding up the westward moving baboons indefinitely.

3. (50 Points) **Barrier synchronization primitive**

   (a) Implement a simple barrier primitive using the posix thread routines. Barrier synchronization is a means for more than two threads to synchornize. Barrier is specified through the barrier.h file below. It defines the method signatures for your implementation. The class consists of four items.

The Barrier constructor will allow you create a barrier of a particular capacity (size). The Barrier::barrierSynch method will cause each thread invoking the barrier to block until the thread capacity is met. For example, if a barrier is created with a size of 3 elements, the first two threads that call the barrierSynch() method will block, and the third will release the first two and continue. The order for release should be the same as the order for arrival. A Barrier::print() method prints list of threads blocked within the barrier as well as the capacity information. You will also define a destructor, which will release any blocked thread back into the system. You will define the necessary private data and implement all methods contained within the barrier.cc file. Provide the appropriate tests in order to demonstrate the success of your simple barrier.

```
#ifndef BARRIER_H
#define BARRIER_H

class Barrier {
  public:
    Barrier(int size);
    ~Barrier();
    void barrierSynch();
    void print();

  private:
    // data for implementation of barrier
};
#endif // BARRIER_H
```

(b) Implement a test program that uses the user-level barriers: Use barriers to coordinates three types of processes in the problem described `Santa-Elf-Reindeer` IPC problem. Santa Claus sleeps in his shop at the North pole and can only be awakened by either (1) all 2 reindeer being back from their vacation, or (2) by some of the elves having difficulty in making toys. Elves can wake Santa up when three of them have problems. When three elves are having their problems solved, any other elves wishing to visit Santa must wait for the three elves to return. "Three elves waiting" has higher priority than all reindeer getting back. That is because without any toys Santa cannot go on his gift giving spree. All the elves' difficulty should be solved before Santa leaves.

# 3 Material to be Submitted

1. Submit the **source code** for the programs. Use meaningful names for the file so that the contents of the file is obvious from the name. You may zip all the source files into a single file. Also provide a Pr2README file that explains the contents of the zip file.
2. Use internal documentation to explain your design.
3. Test runs: It is very important that you show that your program works for all possible inputs. Submit **a single script** that shows for each program the working for correct input as well as graceful exit on error input.
4. Submit your makefile.

# 4 Due date

3/28/2010 submit on-line before mid-night.