**ECE 465: Computer Networking Protocols**
**Prof. B.-P. Paris**
**Homework 6**
**Due: March 20, 2007**

**Reading** Chapter 2 in Kurose and Ross, handouts on programming with C sockets.

## Experiments

1. You are to write a multi-threaded program with the following functionality. The main thread creates and starts two additional threads, a SenderThread and a ReceiverThread.

   The SenderThread uses sockets to periodically (every 3 seconds) send a message(of your choice) via UDP to a port (of your choice) on the local host (IP address 127.0.0.1).

   The ReceiverThread monitors the port you chose above for incoming messages and prints any messages it receives on the screen. Additionally, it counts the number of received messages and once three messages have been received, it notifies the main thread of this fact.

   The main thread, after creating the two other threads, suspends its operation until it is notified by the ReceiverThread as described above. Once it is waken up, it prints an informative message to the screen, signals the other two threads to exit and waits for them to finish (i.e., joins them).

   This assignment explores some additional functionality beyond what we discussed in class and which is highly relevant for your project. In particular, the ability to exchange information between threads is examined here. Specifically, the sender and receiver thread communicate via sockets whereas the receiver and the main thread use a different way to exchange information.

   To complete this assignment you will likely read more on Posix threads. A great resource for this purpose is available from Livermore Labs at `http://www.llnl.gov/computing/tutorials/pthreads/`. Focus on the section on *Thread Management* to learn how to create and join threads. The section on *Condition Variables* describe an ideal way to facilitate communications between

the receiver and the main thread that naturally supports suspending and waking up the main thread as required. I recommend that you start with the example in the section entitled Waiting and Signaling on Condition Variables and simply modify the functionality of the threads in the example.

Turn in the code you developed and a trace showing the messages printed on the screen during program execution. Make sure that each print statement identifies the thread from which it originates.