INB365 Assignment 2

Semester 2 – 2011  
Distributed Communication

**Author: Rod Howarth – N6294685**

Table of Contents

Statement of Completeness 3

Execution Instructions 4

Description of Data Structures Used 5

Description of Thread Execution 6

# Statement of Completeness

The program is complete to the full specifications. The program demonstrates a client server infrastructure by implementing parts of the HTTP specification using pthreads and sockets. To the best of my knowledge, there are no deadlocks or erroneous race conditions and the socket connections are cleaned up correctly when the application is killed.

This code was written by Rod Howarth alone.

# Execution Instructions

Extract the files to a directory on a Linux based machine.

To run the server use:

**make clean && make && sudo ./server 80**

Where 80 is the port you wish to listen on. You can omit sudo if you are already running with elevated privileges. Press Control + C to exit.

To run the client use:

**make clean && make && sudo ./client localhost 80**

Where localhost is the hostname you wish to connect to, and 80 is the port.

# Description of Data Structures Used

The application is setup in five files

**server.h** – The header file that defines the functions for the main server program

**server.c** – The source file that implements these functions   
**connection\_queue.h** – The connection queue data structure definition and functions that relate to it

**connection\_queue.c** – The implementation of these functions

**client.c** – The client application code

The connection\_queue data structure represents a queue of connections waiting to be read from. It is a simple implementation of a generic queue data structure, it has an array of integers representing connections, and has a first and last array index pointer for adding to and removing from the queue.

In addition to this, there is a mutex, queue\_mutex which is used to control access to the queue, and two semaphores, slots\_free and slots\_used, which act as counters for the amount of free spots and used spots in the queue. This provides a convenient way of signaling when there is a connection added to or removed from the queue.

An array of connection\_queue objects is created in server.c, when a new connection comes in this array is added to using functions from connection\_queue.c. This means that all semaphore and mutex use is handled by the connection\_queue itself, each thread just needs to call the functions defined in connection\_queue.h

Server.c defines the thread pool as an array of pthread\_t’s which are created in the main method. They simply poll the connection\_queue for new connections to process. When a connection is processed a simple character array is used to read and send data.

# Description of Thread Execution

In the server, the main method first creates a new connection queue and initializes it, and then it creates and stores the 30 threads that will be used to read connections from this queue.

After performing this setup, the socket is created on the requested port, and listening begins. Once a connection arrives it is immediately added to the connection queue, and the server waits for a new connection.

When a connection is added to the queue, the add\_connection and get\_next\_connection functions from connection\_queue.c will make use of the slots\_free and slots\_used semaphores to control the amount of free slots available in the connection queue. Access to modify the queue is protected via the queue\_mutex.

Once a new connection is added, the slots\_used semaphore will be updated, which will kick off the processing of this connection by one of the threads created in the thread\_pool. This thread will call the get\_next\_connection method to remove the connection from the queue and update the slots\_free semaphore.

The connection\_handler function is then called and is supplied with the connections file descriptor. The get\_line functions is used to read the first line of the request is read into a buffer, and this buffer is analyzed to determine that the request is a GET request for a certain file. The rest of the request is then read and discarded.

The file is then read from the disk and sent in a HTTP 200 response along with the server headers. If the file is missing, a 404 response is sent via the send\_line function. Once this has been done, the connection is closed and the thread begins waiting for a new connection to be added to the connection\_queue once more.

To close the server, control + c is pressed, which sends a SIGINT message. This is handled by the sigint\_handler function which properly closes the socket before exiting.

In the client program, the hostname and port to connect to are read from the command line. The host that this hostname refers to is resolved and a socket is crated. Once the socket is setup a GET request for the / path is sent to the server and a response is waited on. Once the response arrives, the output is written to the console and the program exits.