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, and run:

**make clean && make && ./terminal**

Follow the on screen instructions to run the program.

# Description of Data Structures Used

The application is setup in three files:

**terminal.h** – The header file that defines constants and functions for the main terminal program

**terminal.c** – The source file that implements these functions   
**ship.h** – The ship data structure definition

The ship data structure is a simple struct which contains a character array for the name of the ship, a timeval to represent the time it was docked, and an integer to indicate which berth number it was stored in. This is defined as ship\_t for ease of use.

Timeval is from <sys/time.h>, it represents a time as seconds and microseconds since the epoch, this is used as the data structures in the standard <time.h> do not provide the ability to specify times as milliseconds, and the specification requires we display seconds to two decimal places. Gettimeofday() is used to return this timeval structure. A function, get\_elapsed\_seconds is defined in terminal.h and implemented in terminal.c which returns the time in seconds that has passed since a specified timeval.

In terminal.c, the ‘wharf’ variable represents an array of pointers to ship\_t instances in memory. This array is defined as being of size WHARF\_SIZE, which is 10 in our instance. Each index in this array specifies a berth position, and when a ship is docked, the ship\_t is referenced from this array, and is updated with the position number and time. When the ship is removed, its position is set to NULL.

Two semaphores, berths\_free and berths\_used act as counters of the amount of free spots, and amount of used spots in the wharf. This provides a convenient way of signaling when there is a ship added or removed from the wharf. berths\_free is initialized to 10, berths\_used is set to 0.

A ‘struct timespec’ is used in terminal.c to specify the amount of time for a call to nanosleep, this is used because we want to sleep for 0.5 seconds, so nanosleep from <time.h> is required for this operation.

# Description of Thread Execution

The main function creates three threads, one for ships coming into the port ‘producer’, one for ships leaving the port ‘consumer’ and one for monitoring the keyboard input from the user ‘monitor’. It creates these threads, along with the semaphores and mutex’s and then waits for them to execute.

Two semaphores and two mutex’s are created to keep the threads synchronized. One mutex is created called seaway\_mutex, which is used to ensure that only one threads is adding or removing ships from the wharf at a time. Running\_check\_lock is used to ensure that as soon as the user presses q, the threads all stop executing. This is done by obtaining the lock, setting a boolean, and releasing the lock, the other threads wait on this lock and check the value of this boolean (keep\_running) before adding or removing ships from the wharf.

The semaphores, berths\_free and berths\_used are used to control the buffer between the producer and consumer threads. The consumer thread waits until the berths\_used semaphore is greater than 0 before it attempts to remove ship from the dock. Conversely, the producer thread waits until the berths\_free semaphore is greater than 0 before adding a ship.

The interaction goes like this:

* check the semaphore is greater than 0, if not, output the text and block until it is
* decrement the semaphore
* Check that the keep\_running variable hasn’t been changed, if it has, increment the opposite semaphore to avoid a deadlock on exit.
* generate a random number in order to work out wether you can add or remove a ship
* If so, acquire the seaway\_mutex and add or remove the ship
* Increment the opposite semaphore, to indicate you’ve added or removed a ship
* Release the seaway\_mutex
* If the random number wasn’t within the range required to add or remove a ship, increment the semaphore that was incremented in the second step, so as to say that we didn’t actually add or remove a ship. Ideally this would be avoided by doing the semaphore acquisition after you’ve done the random number generation, but this could result in the text being output more than one time.
* Sleep and repeat

The monitor thread acquires the seway\_mutex lock when it is outputting the status of the berths, so as to ensure that it has accurate information.