COMP3520 Operating Systems Internals Exercise 1 – River Ferry Problem

General Instructions

In this exercise, you will write a *pthreads* program that uses mutexes and condition variables to solve a simple problem involving a river ferry and some customers.

To help you get started, a template is provided. Some comments are included to explain various parts of the source code.

Please bear in mind that this exercise is intended to introduce you to concepts that you will need in order to complete COMP3520 Assignment 1.

This exercise will **not** be marked; assessment for COMP3520 Assignment 1 will be based only on your final source code and discussion document.

The Problem

A river ferry operates a **shuttle** service between the northern and southern banks of a river that flows in an easterly direction. A passenger may board the ferry at either river bank. The ferry has a capacity of one passenger. If there is more than one passenger, other passengers must wait for their turn. The ferry goes back and forth with a constant speed (i.e., takes a fixed time to go from one bank to the other) and will not wait if there is no passenger.

Write a program to solve this problem using condition variables and mutexes to provide synchronization between the ferry and the passengers. To assist you get started on this exercise, a program template ferry_template.c has been written for you. You need to fill in the sections marked with dots and write the thread routines *boat_routine()* and *passenger_routine()*. Note that the intended direction of travel for each passenger will be **randomly** determined at runtime.

In the *main()* function, the program asks the user to supply the following parameters as inputs:

- *no_of_passengers* – total number of passengers from both directions
- *boat_pace* – the time it takes for the ferry to perform a one-way trip
- *passenger_rate* – the duration between passenger arrivals

Your *boat_routine()* needs to print the following lines where *boat_pace* is a parameter supplied by the user as described earlier:

- "Boat starts working in pace [*boat_pace*]." – when the thread is started
- "Boat to move n2s." – whenever the ferry begins moving from north to south
- "Boat to move s2n." – whenever the ferry begins moving from south to north

Your *passenger_routine()* needs to print the following lines where *direction* and *id* correspond to the passenger's intended direction of travel and identification number respectively:

- "Passenger [*direction*] [*id*] arrives and waits to cross the river." – when the thread is started
- "Passenger [*direction*] [*id*] is now on the boat." – when the passenger has boarded the ferry

Appendix

Basic functions for mutexes

*int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);*

This function initializes *mutex* with attributes specified by *mutexattr*. If *mutexattr* is NULL, the default mutex attributes are used. Upon successful initialization, the state of *mutex* becomes initialized and unlocked.

*int pthread_mutex_lock(pthread_mutex_t *mutex);*

The *mutex* object shall be locked by calling this function. If *mutex* is already locked, the calling thread shall block until *mutex* becomes available.

*int pthread_mutex_trylock(pthread_mutex_t *mutex);*

This function is equivalent to *pthread_mutex_lock()*, except that if *mutex* is currently locked (by any thread, including the current thread), the call shall return immediately.

*int pthread_mutex_unlock(pthread_mutex_t *mutex);*

This function shall release *mutex*. If there are threads blocked on the *mutex* object when *pthread_mutex_unlock* is called, *mutex* shall become available, and the scheduling policy determine which thread shall acquire *mutex*.

*int pthread_mutex_destroy(pthread_mutex_t *mutex);*

This function destroys the *mutex* object.

Basic functions for condition variables

*int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);*

This function shall initialize the condition variable *cond* with attributes referenced by *attr*. If *attr* is *NULL*, the default condition variable attributes shall be used.

*int pthread_cond_signal(pthread_cond_t *cond);*

This function shall unblock one of the threads that are blocked on the condition variable *cond* if any threads are blocked on *cond*.

*int pthread_cond_broadcast(pthread_cond_t *cond);*

This function shall unblock all threads currently blocked on *cond*.

*int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);*

This function shall be called with *mutex* locked by the calling thread. The function atomically releases *mutex* and causes the calling thread to block on the condition variable *cond*.

*int pthread_cond_destroy(pthread_cond_t *cond);*

This function shall destroy the condition variable *cond*.

Note: A condition variable must always be used in conjunction with a *mutex* lock. Both *pthread_cond_signal()* and *pthread_cond_wait()* should be called after *mutex* is locked.