COMP3520 Operating Systems Internals
Programming Exercise 1
Student id Assignment

General Instructions

In this exercise, you will write a *pthreads* program that uses mutexes and condition variables **only** to solve a simple synchronization problem involving a teacher and some students.

Please bear in mind that this exercise is intended to introduce the 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

There are *N* students in a class. The teacher is to assign a distinctive id to each student. The student id is a natural number ranged from 1 to *N*. **Note:** each student thread is assigned a student id **by the teacher thread** (**not** by the main default thread when the student threads are created).

Write a program to solve this student id assignment problem using condition variables and mutexes **only** to provide synchronization between the teacher and the students. You need to write two thread routines *teacher_routine()* and *student_routine()*, in addition to the *main()* function.

In the *main()* function, the program asks the user to supply one parameter as input:

- *n* – the total number of students in the class

Your *teacher_routine()* needs to print the following status message wherever appropriate:

- "Teacher: student id [*id*] for next student." – when the teacher is to give a student id to a student

Your *student_routine()* needs to print the following status message wherever appropriate::

- "Student: My student id is [*id*]." – when a student is obtained an id.

When every student thread has been assigned a student id and terminated, the main default thread must print the following status message and then terminate:

- "Main thread: All students have obtained their student ids. The simulation will end now."

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.