COMP3520 Operating Systems Internals Exercise 2 – Toy Playing Problem

General Instructions

In this exercise, you will write a *pthreads* program that uses mutexes and condition variables to solve a simple problem involving two children playing a toy.

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 get you to explore 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

Three children *child_0* and *child_1* take turns to play with a toy **in rotation**. Each child plays with the toy for a specified amount of time. Afterwards, they must pass the toy to the other child.

Initially, *child_0* plays with the toy. The children take turns to play with the toy in the following order until "Mum" has called: *child_0* → *child_1* → *child_0* → *child_1* → *child_0* …

Write a program to solve this problem using condition variables and mutexes to provide synchronization between the children. To assist you in getting started on this exercise, a program template *toy_playing_template.c* has been written for you. You need to fill in the sections marked with dots and write the thread routine *child_routine()*.

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

- *total_time* – total amount of time allowed for children to play with the toy
- *child_0_play_time* – amount of playing time per turn for *child_0*
- *child_1_play_time* – amount of playing time per turn for *child_1*

Your *child_routine()* needs to print the following lines where *child_0_play_time*, and *child_1_play_time* are parameters supplied by the user as described earlier:

- "child 0: I get to play with the toy for *child_0_play_time* units of time." – whenever *child_0* receives the toy
- "child 0: I now give the toy to child 1." – whenever *child_0* is obliged to pass the toy to *child_1*
- "child 1: I get to play with the toy for *child_1_play_time* units of time." – whenever *child_1* receives the toy
- "child 1: I now give the toy to child 1." – whenever *child_1* is obliged to pass the toy to *child_1*

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.