CMPUT 379 Assignment 4 Project Report

Name: Tymoore Jamal

Student ID: 1452978

LEC A1

Objectives:

This project had 2 main objectives. To familiarize us with threads and multithreaded programming, and also to familiarize us with methods of protecting a critical section of code using Mutexes and Semaphores. These objectives are closely related, and together teach us how to effectively create an application which uses several threads which safely interact with shared resources while maintaining good concurrency.

This provides us with meaningful experience interacting with the GNU operating system through making system calls to the Unix shell. This allows us to have significantly more freedom when programming in the Unix environment, as system calls are quite powerful and allow us to achieve results that we would not otherwise be able to do.

Specific functions that this assignment focused on were pthread_create, pthread_mutex_init, pthread_mutex_lock, pthread_mutex_unlock, and pthread_exit.

Design Overview:

The important features are outlined below:

a4tasks:

- Upon execution check the validity of the user's inputs
- Get the start time
- Process the input file
  - Store all tasks in a vector of task classes
  - Store all resources in a vector of resource structs
- Provide the monitor class all of its needed data
- Create a monitor thread and a thread for each task
- Continually check each thread to determine if it has completed all of its needed iterations
- If all threads are complete, acquire the printing mutex and print all of the needed information

task:

- When instructed by the main thread create a thread for the current task
    - Pass this thread a pointer to the current task
- Cast the task pointer to a pointer of the task class
- Start the main loop
- Set the state to WAIT
    - Do this inside the print mutex lock so nothing can print during a state transition
- Lock the resources mutex, once locked determine if we can acquire all the resources, we need
    - If we cannot acquire the resources, release the lock and sleep for the wait time, then restart the loop
- Acquire the needed resources, then unlock the resources mutex
- Set the state to RUN
    - Do this inside the print mutex lock so nothing can print during a state transition
- Sleep for the busy time
- Lock the resources mutex, then release the resources, then unlock the resources mutex
- Set the state to IDLE
    - Do this inside the print mutex lock so nothing can print during a state transition
- Sleep for the IDLE time
- Print the needed information inside of the printing mutex lock
- Increment the iterations completed
- If the iterations completed equals the total needed iterations exit the thread
    - Otherwise go back to the start of the loop

monitor:

- When instructed by the main thread create a thread for the monitor
    - Pass this thread a pointer to the monitor
- Cast the monitor pointer to a pointer of the monitor class
- Start the monitor loop
- Lock the printing mutex
- Print all tasks with state WAIT
- Print all tasks with state RUN
- Print all tasks with state IDLE
- Unlock the printing mutex
- Sleep for the monitor time

Project Status:

The project is now fully functional and complete. A summary of some of the difficulties encountered are mentioned below.

a4tasks:

Some difficulties included:

- Ensuring that we use the proper pointers and we do not reuse pointers that we want to pass to a thread

tasks:

Some difficulties included:

- Ensuring the thread got the correct pointer to the correct task
- Keeping track of all our mutexes and when they should be locked/unlocked
- Ensuring two tasks do not print over one another
- Keeping track of the amount of time waiting

Testing and Results:

I found that for this specific program testing has to be an ongoing and iterative process. This involved heavily testing each new aspect of the program as it was being developed.

For example, I first began implementing the task, resource, and monitor classes/structs before I even began to work with threads an mutexes, this allowed me to determine if the classes were behaving properly and contained the proper references to one another. (E.g. the monitor class contains pointers to tasks, and tasks contain pointers to resources). Once I had those completed and tested, I added in threads and tested all threads were executed correctly and contained the correct data. Finally, I fleshed out the work each thread did and added mutexes, while doing ongoing testing every step along the way.

Overall this approach was extremely beneficial for me as it allowed me to confirm that I was always on the right track and that the functionalities I implemented were working.

Acknowledgements:

I did this assignment individually and did not collaborate with anyone. I used the eclass page as a rough guide on how to implement some functionality involving mutextes.

Assumptions Made:

- a4tasks is provided correct values, this entails:
  - A valid .dat input file (e.g. test.dat)
  - A valid monitor time in milli seconds (greater than 0)
  - A NITER value greater than 0
- The input file will be correct in nature (same format as the examples provided)
- Resources will have a max availability of at least 1
- There will be at least one task and 1 resource
- Printing is done after IDLE
- A iteration is considered complete after IDLE finishes