Ayan Abbas
1630145

CMPUT 379
Assignment 4 Report

OBJECTIVES

The main objective of Assignment 4 was to gain experience with multithreaded programming, synchronization, and deadlock prevention. I wrote a C++ program, a4w23, which uses threads and mutex to simulate the execution of a set of tasks. All the resource units are non-sharable and non-preemptable, meaning that once the resource has been acquired, it cannot be taken away by the system until it has finished executing. The focus on this assignment is on threads and I have used mutexes to maintain synchronization between the threads.

DESIGN OVERVIEW

- The program, a4w23 is a task simulator which obtains all the parameters using an input file. The input file is parsed, and all the required values are stored in a map or struct. I have used getline() to read the input file line by line and skipped the empty or commented lines. The resource and task lines are handled accordingly and stored. For my implementation, I have assumed that all the resource and task lines have a value.
- I have used the thread library to implement my program. I created a thread for each of the tasks and a monitor thread which prints the details of the task after a delay period entered by the user.
- The simulator() function is used to simulate the tasks by using a thread and three mutexes: t_mutex(task), res_mutex(resource), and mon_mutex(monitor). Initially, it places a lock of the task to update the status to wait while we are acquiring the resources. Once the resource is acquired, the task goes into run state and then idle state. There is a delay produced in the run and idle state to simulate a running task or an idle task. Wait time is also recorded. Mutex locks and unlocks are used when required to protect the critical sections.
- The monitor() function is used to print the information about task by using a thread. It prints the information about the tasks in the different states every time after a wait period specified by the user.
- The information about the resources and tasks and the total run time was the printed at the end of the program. Join() was used to ensure all the threads have finished executing before the termination of the program.

Makefile

The executable for the program can be obtained using the Makefile.
“make” or “make a4w23” creates an executable file for the program.
“make clean” removes all the unnecessary files from the directory.
“ make tar” compresses the desired files.

Running Instructions

Make the executable files using make or make a4w23. Then the simulator can be invoked by using “./a4w23 inputFile monitorTime NITER” where inputFile is used to define the parameters of the simulator, monitorTime specifies the delay after which the output needs to be printed and NITER is the number of iterations. I have tried to handle some invalid inputs but not all cases are covered. To run the program, use the input arguments stated above.

PROJECT STATUS

The project is complete according to me and compiles perfectly. All the specifications of the projects are met. It prints out all the details required listed in the specifications. However, due to the nature of the project, no exact matches can be made from the sample output but the information about the threads, resources, tasks, and total run time are being displayed when the program is invoked. For some reason, when I was using the tms struct to calculate the wait time, it was always zero or some large random value so at the very end I had to change my code. I used chrono high resolution clock to calculate the wait time. One little issue which I have come across is that while printing the task information for the first time, the wait time is a random large value sometimes. If, you run the program again then you will see the correct value but this rarely happens. I had very little experience working with threads and mutexes before this assignment so I had some difficulty figuring out how to use locks and unlocks but I think I have met all the specifications of the assignment.

TESTING AND RESULTS

The program was tested by using the input file given in the assignment description and the output was matched against the sample output. The output given by my assignment matched with the sample output. An exact match is not possible for this assignment, but the correctness can be verified by comparing the output with the sample output. Also, I tried to write and test small parts of the code used in the assignment separately and verified their correctness. I tried to run the program with different iteration and monitor time values and with different input files to verify the working of my assignment. After parsing the input file, I used print statements to ensure if all the resources and tasks were stored correctly.

ACKNOWLEDGEMENTS

1. https://stackoverflow.com/questions/14733761/printf-formatting-for-hexadecimal
2. https://en.cppreference.com/w/cpp/utility/hash
3. CodeVault YouTube video on mutexes: https://youtu.be/oq29KUy29iQ
4. https://en.cppreference.com/w/cpp/thread
5. https://www.geeksforgeeks.org/multithreading-in-cpp/
6. https://linuxhint.com/cpp-mutex-lock/
7. https://stackoverflow.com/questions/14733761/printf-formatting-for-hexadecimal
8. https://man7.org/linux/man-pages/man2/nanosleep.2.html
9. https://cplusplus.com/reference/chrono/high_resolution_clock/now/
10. https://en.cppreference.com/w/cpp/thread/mutex/try_lock
11. https://en.cppreference.com/w/cpp/thread/unique_lock
12. https://www.geeksforgeeks.org/processing-strings-using-stdistringstream/