ASSIGNMENT 4 REPORT

Ajiitesh Gupta
1562557

Objectives:

This programming assignment is intended to give experience in providing hands-on experience in using Linux System API, such as, multithreaded programming, thread synchronization, and deadlock prevention.

How to use:

Makefile commands (assignment spec):

- Executing 'make a4w22' produces the a4w22 executable file.
- Executing 'make clean' removes unneeded files produced in compilation. - Executing 'make tar' produces the 'submit.tar.gz' archive.

Running the program:

The program can be invoked using the command line as follows:

"./a4w22 inputFile monitorTime NITER"
where inputFile is a file describing the system, monitorTime is an integer (in millisec) that specifies how often a monitor thread runs, and NITER represents when the simulator finishes when each job in the system executes NITER times.

Design Overview:

I made small functions that do one thing each to increase understandability of the functions. Code is all compiled in one big file, which is:

1. *main.cpp*: Main file which compiles all the header, and all my methods, and executes the code.
2. *Makefile*: The file which makes the code into an executable, having the option of also making a compressed file in the tar.gz form, and has the ability to delete any unneeded files.

- Each job has its own thread, and the "worker" in my implementation is referring to each job.
- Two global mutexes to lock resources pool and job state.
- Two global objects to track resources and job state.
- Resources have two states, "BUSY" and "FREE".
- Tasks have three states, `WAIT`, `RUN`, and `IDLE`.
- Lock resource pool before acquiring and releasing the resource in job.
- Lock job state before updating job state.
- Lock job state before printing job state from monitor thread.

Project Status:

All functionality delivered to match example test files and specification. All required features are implemented with error handling for most important functions; however, some edge cases may not be handled properly.

Testing and Results:

The programs were compared with the example inputs and the outputs were equivalent.

Acknowledgments:

https://en.cppreference.com/w/cpp/thread/lock_guard