Hannah Desmarais
Hdesmara
CMPUT 379
April 7th, 2020

# Assignment 4 Project Report

## Objectives

Through this assignment, we can gain experience working with multiple threads. It helps teach us about how to properly handle execution of threads when they need access to the same type of resource by using either semaphores or mutex's. This project also helps us learn how to gather time and thread information about the program as it runs as well as prevent deadlocks.

## Design Overview

Things to note:

- I chose to use hash maps in order to handle resources allocation for simplicity and better efficiency since searching a HashMap is constant and I didn't have to use another loop for it.
- Only one set of resources may be specified in a data file.
- All times entered on the command line are assumed to be in milliseconds
- I chose to use one binary semaphore that only allows one thread at a time in to lock resources when a job was trying to determine if there was enough resources available for it to take. This was done so that multiple threads wouldn't try to take the same resource at the same time.
- I chose to use a counting semaphore to run jobs. This was done so that if a thread had passed the waiting stage (meaning it acquired the resources needed) then it could run at the same time as other threads for better concurrency.
- I also used a binary semaphore for whenever a function wished to print in order to prevent any text from overlapping.
- Lastly, my program remakes the monitor thread every time the monitor time is up. This was because I got some weird behaviour if I didn't do it this way with the jobs in the monitor thread not being updated as they should.

## Project Status

As of now, my project works as intended and described in the assignment 4 handout. Most of the difficulty I had was figuring out how to put my semaphores in my program in a way that made sense as well as ensuring that a resource couldn't be picked at the same time and was properly decremented or incremented at the right time. I also struggled a little with the times of the threads as I was getting some funny numbers but it was due to other bugs.

## Testing and Results

- Monitor: The monitor thread was tested by running the program and making sure it was printing however fast it was meant to. I also tested that the semaphore around the print statements was working by running the program and making sure no text was mixed with text from anywhere else like jobs. I also tested the status of the job by looking at what jobs had printed that they'd finished and slowed down some of the times. If a job had printed right before the monitor, then I knew it should be in the idle state, if it had printed a little further up then I knew it should be waiting and if it had been a while, then I knew it should've been in run.
- Deadlocks: The dead locks were tested for by running the program with the data file given in the assignment directions. Because there are overlapping resources, I knew it could end up in a deadlock if it wasn't executing right and if when the monitor printed, all the jobs were remaining in wait due to improper resource allocation.
- Resource allocation: This was tested by checking the monitor to make sure that no two jobs who needed the same resource were ever in run at the same time (because in the datafile only one of each resource type was available).
- Job threads: Job threads were checked that they were running correctly by checking the monitor to make sure states were changing and then checking the termination messages. I would make sure that it had ran the correct amount of times and that the timing of each job being finished and the total wait period was a sensible number.
- Termination: Lastly, I made sure that the termination would only ever print when all jobs were done by running the program with different values and ensuring that the termination prints always came last and that all jobs had printed the total amount of iterations they were meant to.

## Acknowledgements


- For some supplementary information on semaphores in C I watched these videos:
    - https://www.youtube.com/watch?v=ukM_zzrIeXs&ab_channel=JacobSorber
    - https://www.youtube.com/watch?v=YSn8_XdGH7c&ab_channel=CodeVault
    - https://www.youtube.com/watch?v=l6zkaJFjUbM&ab_channel=CodeVault
- I also consulted the class slides on semaphores
- I looked at this website for some information on how to use nanosleep(): https://stackoverflow.com/questions/1157209/is-there-an-alternative-sleep-function-in-c-to-milliseconds
- I also used the documentation on semaphores, specifically sem_wait(), by linux to figure out how to error check: https://man7.org/linux/man-pages/man3/sem_timedwait.3.html
- Lastly, I consulted the example on eclass for the split function: http://webdocs.cs.ualberta.ca/~cmput379/W22/379only/lab-tokenizing.html