COMP 2240 ASSIGNMENT 2 REPORT

C3238805 NI ZENG

- Problem 1 (WAR controlling) solving:

  The java program for Problem 1 has a static class "warRunnable" which implements Runnable interface. The design of solving this problem is to make each of the war as a single Thread. This java program used "Semaphore semaphore = new Semaphore (1, true)".
  All threads run concurrently and share only 1 semaphore permit (only one war can cross the intersection at a time). After all thread at their running state, each of the thread will then try to get the semaphore permit in order to execute its critical section, each acquire() blocks until a permit is available and the given fairness setting FCFS. IF a thread can successfully require an available permit, then the thread will start crossing the intersection. After the war crossed the intersection, then release one permit so the other threads can continues to enter its critical section.

  ## Case considered (in order to test the program's ability):

  Case 1: when input has no war input in each of the direction (e.g. N=0 S=0 E=0 W=0).
  Case 2: when input has only war input in either track 1 or track 2 (e.g. N=1 S=1 E=0 W=0) OR (e.g. N=0 S=0 E=0 W=3).

  ## Behaviour of the algorithm on those cases:

  case 1, the java program can handle this case and will just exit the program without interruption if run input N=0 S=0 E=0 W=0.
  case 2, the java program can also handle this case and will just exit the program if either track 1 or track 2 reached the track limit.

- Problem 2 (Monitor Colour and Monochrome Printing) solving:

  To solve the Problem 2, the java program I wrote have two threads and use Monitor to enforced mutual exclusion. Thread 1 mainly to find the suitable job to print next following the assignment specification's condition. Using wait() while all three heads are currently in use or next job require mode switch. Thread2 wait() if no available job to be print. A notifyAll() is called after each thread has execute its critical section to prevent Deadlock.

  ## Case considered (in order to test the program's ability):

  Case 1: when the first job is colour print and the rest of the job are Monochrome print.
  Case 2: when the first two jobs are Monochrome print, the rest of the jobs are colour print.

  ## Behaviour of the algorithm on those cases:

  case 1, the java program can handle the case and update the current time correctly, in this case, only one printer head is needed to print the colour job, the rest of the head is remain idle until the first colour print job is finished.
  case 2, the java program can also handle this case, head 1 and head 2 can print concurrently and head 3 remain idle until the first two jobs are finished, then start to print the remaining colour jobs.

- Problem 3 (Colour and Monochrome Printing with semaphore) solving:
  To solve Problem 3, I have use "Semaphore semaphore = new Semaphore (3, true)". to enforce

the mutual exclusion. Each available permit is matched to each print head. There are 2 threads used in this java program, thread 1 is mainly for job to require its permit to enter a “Ready to be print LinkedList”. Thread 2 is mainly to get the jobs from “Ready to be print LinkedList” and release 1 permit after execution.

Case considered (in order to test the program’s ability):

Case 1: when input only 2 jobs (1 colour job and 1 monochrome job).

Case 2: when input only 2 jobs (2 colour job).

Behaviour of the algorithm on those cases:

case 1, the java program can handle this case, head 1 start on the first colour job and finished while the other 2 printer head remain idle. After the first job finished, head 1 then start on the monochrome job. The current time is correctly updated.

case 2, the java program can also handle this case and head 1 and head 2 start to print the fist tow colour jobs concurrently.