Konrad Schultz V00761540 CSC360 2.2
Changes in red

1. There will be one thread for each train loading, and a main controller thread that keeps track of timings and prints to stdout. Schedules the trains.

2. There is an overall controller thread ~~that keeps a queue of train departure order~~. This follows the manager-worker threaded program design pattern.

3. There is one mutex to lock each train list (eastbound_lock)(westbound_lock) and one mutex to keep track of number of trains adding to lists (adding_trains_lock).

4. No the main thread will not be idle. It will be responsible for dispatching the trains and sending them on their way. It will also spawn a thread for each train and read the input file.

5. Stations will be represented by a linked list. The each train thread will be responsible for inserting itself based on priority order (o(n)) and the main thread will be responsible for sending the next available train on the track (o(1)).

6. I can think of going about this two ways, either each of the threads will have access to the list and it will be guarded by a mutex/convar or the thread will return some sort of ready signal to the main controller thread that will add it to the list of trains (I'll probably use the former). There are 3 mutexes, the eastbound and westbound lists have a mutex, and there is a mutex to ensure the trains_adding variable is threadsafe.

7. From my understanding of conditional variables I don't think I will require any. Thread locking / mutual exclusion should be all the logic I require. The only conditions will be MAIN_TRACK_STATUS and TRAIN_LIST_STATUS which will be either locked or unlocked, and a thread should wait until it's unlocked and then perform its action. I didn't use any condition variables

8. In 15 lines or less briefly sketch the overall algorithm you will used.
Create all relevant mutexes and data structures
Read input file and spawn a thread for each train.
Sleep for train's loading time
Lock mutex; increment trains_adding; unlock mutex;
Once a train is loaded display output to screen and acquire list mutex
Add train to the proper place in the list based on its priority and other factors
Unlock the list mutex
Lock mutexs; decrement trains_adding; unlock mutex;
DISPATCHER
- Wait for something to be in a list
- Wait for trains_adding to be 0
- Lock both list mutexes
- Pop the next train to go off its respective list
- Unlock list mutexes
- Sleep for train's crossing time
- Unlock the track mutex

REPEAT UNTIL ALL TRAINS CROSSED