Computer Science 360
Assignment 2
Due July 5 th at 11:59 pm

Introduction

In this assignment you will learn to use three programming constructs provided by the POSIX pthread library: threads, mutexes and condition variables.

Your goal is to construct a simulation of an automated control system for a single lane bridge. You will use threads to simulate trains approaching the bridge from two different directions. Your software will ensure (among other things) that there is never more than one train on the bridge at any given time.

You may implement your solution in C or C++. You solution must run on the machine `linux.csc.uvic.ca`.

Manual Pages

Be sure to study the man pages for the various functions contained in the assignment. For example, the man page for `pthread_create` can be found by typing:

```
$ man pthread_create
```

At the end of this assignment you should be familiar with the following functions:

```
atoi
fopen
fclose
fgetc or fgets

pthread_create
pthread_join

pthread_mutex_lock
pthread_mutex_unlock
pthread_mutex_init

pthread_cond_wait
pthread_cond_init
pthread_cond_broadcast
```

It is absolutely critical that you read the man pages. The assignment specification does not discuss details of these functions. Your best source of information is the man page.

Trains

Each train will be simulated by a thread and the operation of each thread is defined by the function shown below.

```
void * Train ( void *arguments )
{
      TrainInfo   *train = (TrainInfo *)arguments;

      /* Sleep for awhile to simulate different arrival times */
      usleep (train->length*SLEEP_MULTIPLE);

      ArriveBridge (train);
      CrossBridge  (train);
      LeaveBridge  (train);

      free (train);
      return NULL;
}
```

You are responsible for implementing the functions `ArriveBridge` and `LeaveBridge`. The code for `CrossBridge` has been supplied.

The sample code supplied performs all output that is expected in the final solution. You may want to introduce additional output during debugging. See the sample code for an example of using conditional compilation to only produce output when debugging.

Step 1

Your program accepts two parameters on the command line. The first one is required, the second one is optional.

The first parameter is an integer, greater than 0, which is the number of trains to simulate.

The second parameter is optional: a filename to use as input data for the simulation. The format of the file is shown below.

You may assume :

- the file always contains data for at least the number of trains specified in the first parameter.
- during our testing the file specified on the command line will exist, and it will contain valid data.

If the second parameter is not specified (no filename is given) your program will randomly generate trains for the simulation. (This code is already given to you.)

Complete the implementation of `train.c` so that it correctly reads the input files.

Step 2

Complete the implementation of ArriveBridge and LeaveBridge so that the following conditions hold for the bridge:

- only one train is on the bridge at any given time
- trains do not overtake each other, trains cross the bridge in the order they arrive (subject to the requirement below)
- trains headed East have priority over trains going West
    - if there are trains waiting headed both East and West then two of the East bound trains should be allowed to cross for every West bound train allowed to cross.

Input file format

The input files have a simple format. Each line contains information about a single train. The files end with a blank line.

The first character on each line is one of the following four characters: 'e', 'E', 'w', or 'W'

The first two letters specify a train that is going East, the second two letters specify a train headed West.

Immediately following the letter specifying the direction is an integer that indicates the length of the train. There is no space between the direction character and the length.

The following file specifies three trains, two headed East and one headed West.

```
E10
w6
E3
```

Compilation

You've been provided with a Makefile that builds the sample code. It will produce an executable named `assign2`

Submission

Submit a tar archive named assign2.tar of your assignment using connex
You can create a tar archive of the current directory by typing:

```
tar cvf assign2.tar *
```

You should submit all the files required to create your solution (including the Makefile and any .c and .h files.)

Please do not submit .o or executable files. Erase them before creating the tar archive.

Note

This assignment is to be done individually. You are encouraged to discuss the design of the solution with your classmates, but each student must implement their own assignment. The markers will submit your code to an automated plagiarism detection service.

Testing

You've been provided with five files you can use to test your solution: t0.txt to t4.txt.

You've been given the expected output in the files t0_sol.txt to t4_sol.txt. To make things easier, the solution file only contains what order the trains go off the bridge.

To generate that output, you can run your program like this:

```
% ./assign2 3 t0.txt | grep OFF > o3.txt
```

And then compare your output to the sample solution like this:

```
% diff t0_sol o3.txt
```

Grading

Your solution will be run against five input files that are similar to the ones provided to you. You will receive two marks for each input file you generate the correct output for.