

# CSc 360: Operating Systems (Fall 2010)

## Programming Assignment 2 (P2): Multi-Thread Scheduling (MTS)

Spec Out: October 21, 2010  
Code Due: November 12, 2010

### 1 Introduction

In P0 (Simple Shell Interpreter, SSI) and P1 (Realistic Shell Interpreter, RSI), you have built a shell environment to interact with the host operating system. Great job! But very soon you find that both SSI and RSI are missing one of the key features in a real multi-process or multi-thread operating system: scheduling, i.e., all processes or threads created by your RSI are still scheduled by the host operating system, not yours!

Interested in building a multi-thread scheduling system for yourself? In this assignment, you will learn how to use the three programming constructs provided by the POSIX `pthread` library:

1. threads
2. mutexes
3. condition variables (convars)

to do so. Your goal is to construct a simulator of an automated control system for the railway track shown in Figure 1 (i.e., to emulate the scheduling of multiple threads sharing a common resource in a real operating system).

As shown in Figure 1, there are two stations (for high and low priority trains) on each side of the main track. At each station, one or more trains get loaded with commodities. Each train in the simulation commences its loading process at a common start time 0 of the simulation program. Some trains take more time to load, some less. After a train is loaded, it patiently awaits permission to cross the main track, subject to the requirements specified in Section 2.2. Most importantly, only one train can be on the main track at any given time. After a train finishes crossing, it magically disappears. You will use threads to simulate the trains approaching the main track from two different directions, and your program will schedule between them to meet the requirements in Section 2.2.

You will use C or C++ and the Linux workstation in ECS242 to implement and test your work.

### 2 Trains

Each train, which will be simulated by a thread, has the following attributes:

#### 1. Direction:

- If the direction of a train is Westward, it starts from the East station and travels to the West station.
- If the direction of a train is Eastward, it starts from the the West station and travels to the East station.

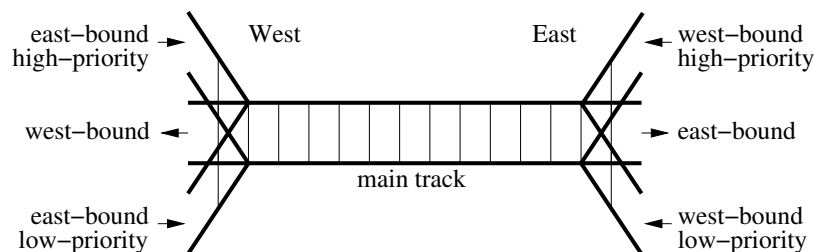


Figure 1: The railway system under consideration.

2. **Priority:** The priority of the station from which it departs.
3. **Loading Time:** The amount of time that it takes to load it (with goods) before it is ready to depart.
4. **Crossing Time:** The amount of time that the train takes to cross the main track.

Loading time and crossing time are measured in 10ths of a second. These durations will be simulated by having your threads, which represent trains, `usleep()` for the required amount of time.

## 2.1 Step 1: Reading the input file

Your program (`mts`) will accept two parameters on the command line:

1. The first parameter is the name of the input file containing the definitions of the trains.
2. The second parameter, an integer  $> 0$ , is the number of trains specified in the input file.

### 2.1.1 Input file format

The input files have a simple format. Each line contains the information about a single train, such that:

1. The first character specifies the **direction** of the train. It is one of the following four characters:

`e`, `E`, `w`, or `W`

`e` or `E` specify a train headed East (East-Bound): `e` represents an east-bound low-priority train, and `E` represents an east-bound high-priority train;  
`w` or `W` specify a train headed West (West-Bound): `w` presents a west-bound low-priority train, and `W` represents a west-bound high-priority train.

2. A colon(:) immediately follows the direction of the train.
3. Immediately following is an integer that indicates the **loading time** of the train.
4. A comma(,) immediately follows the previous number.
5. Immediately following is an integer that indicates the **crossing time** of the train.
6. A newline (`\n`) ends the line.

*You may assume* that the file always contains data for at least the number of trains specified in the second parameter. During our testing, the file specified on the command line will exist, and it will contain valid data.

### 2.1.2 An Example

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

```
e:10,6
W:6,7
E:3,10
```

It implies the following list of trains:

Train No.	Priority	Direction	Loading Time	Crossing Time
0	low	East	1.0s	0.6s
1	high	West	0.6s	0.7s
2	high	East	0.3s	1.0s

*Note:* Observe that Train 2 is actually the first to finish the loading process.

## 2.2 Step 2: Simulation Rules

The rules enforced by the automated control system are:

1. Only one train is on the main track at any given time.
2. Only loaded trains can cross the main track.
3. If there are multiple loaded trains, the one with the high priority crosses.
4. If two loaded trains have the same priority, then:
  - (a) If they are both traveling in the same direction, the train which finished loading last gets the clearance to cross first. If they finished loading at the same time, the one appeared last in the input file gets the clearance to cross first.
  - (b) If they are traveling in opposite directions, pick the train which will travel in the direction opposite of which the last train to cross the main track traveled.

## 2.3 Step 3: Output

For the example, shown in Section 2.1.2, the correct output is:

```
Train 2 is ready to go East
Train 2 is ON the main track going East
Train 1 is ready to go West
Train 0 is ready to go East
Train 2 is OFF the main track after going East
Train 1 is ON the main track going West
Train 1 is OFF the main track after going West
Train 0 is ON the main track going East
Train 0 is OFF the main track after going East
```

You must:

1. print the arrival of each train at its departure point (after loading) using the format string:

```
"Train %2d is ready to go %4s"
```

2. print the crossing of each train using the format string:

```
"Train %2d is ON the main track going %4s"
```

3. print the arrival of each train (at its destination) using the format string:

```
"Train %2d is OFF the main track after going %4s"
```

where:

- there are only two possible values for direction: "East" and "West"
- trains have identifying numbers in the range of [0, 99]. The ID number of a train is specified *implicitly* in the input file. The train specified in the first line of the input file has ID number 0.
- trains have loading and crossing times in the range of [1, 99].

## 2.4 Manual Pages

Be sure to study the [man](#) pages for the various functions to be used in the assignment. For example, the [man](#) page for [pthread\\_create](#) can be found by typing the command:

```
$ man pthread_create
```

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

1. File access functions:

(a) [atoi](#)

- 103 (b) `fopen`
- 104 (c) `feof`
- 105 (d) `fgetc` and `fgets`
- 106 (e) `fclose`

107 2. Thread creation functions:

- 108 (a) `pthread_create`
- 109 (b) `pthread_exit`
- 110 (c) `pthread_join`

111 3. Mutex manipulation functions:

- 112 (a) `pthread_mutex_init`
- 113 (b) `pthread_mutex_lock`
- 114 (c) `pthread_mutex_unlock`

115 4. Condition variable manipulation functions:

- 116 (a) `pthread_cond_init`
- 117 (b) `pthread_cond_wait`
- 118 (c) `pthread_cond_broadcast`
- 119 (d) `pthread_cond_signal`

120 It is absolutely critical that you read the `man` pages, and attend the tutorials.  
121 Your best source of information, as always, is the `man` pages.  
122 For help with the POSIX interface (in general):

123 <http://www.opengroup.org/onlinepubs/007908799/>

124 For help with POSIX threads:

125 <http://www.opengroup.org/onlinepubs/007908799/xsh/pthread.h.html>

126 A good overview of `pthread` can be found at: <http://www.llnl.gov/computing/tutorials/pthreads/>

### 127 3 Submission: Deliverable 1 (Design Due: October 27, 2010)

128 You will write a design document which answers the following questions. It is recommended that you think through  
129 the following questions *very carefully* before answering them.

130 Unlike P0 and P1, no amount of debugging will help after the basic design has been coded. Therefore, it is very  
131 important to ensure that the basic design is correct. Answering the following questions haphazardly will basically  
132 ensure that Deliverable 2 does not work.

133 So think about the following for a few days and then write down the answers.

- 134 1. How many threads are you going to use? Specify the work that you intend each thread to perform.
- 135 2. Do the threads work independently? Or, is there an overall “controller” thread?
- 136 3. How many mutexes are you going to use? Specify the operation that each mutex will guard.
- 137 4. Will the main thread be idle? If not, what will it be doing?
- 138 5. How are you going to represent stations (which are collections of loaded trains ready to depart)? That is, what  
139 type of data structure will you use?
- 140 6. How are you going to ensure that data structures in your program will not be modified concurrently?
- 141 7. How many convars are you going to use? For each convvar:

- (a) Describe the condition that the `conv` will represent.
- (b) Which mutex is associated with the `conv`? Why?
- (c) What operation should be performed once `pthread_cond_wait()` has been unblocked *and* re-acquired the mutex?

8. In 15 lines or less, briefly sketch the overall algorithm you will use. You may use sentences such as:

If train is loaded, get station mutex, put into queue, release station mutex.

The marker will not read beyond 15 lines.

**Note:** Please submit answers to the above on 8.5"×11" paper in 10pt font, single spaced with 1" margins left, right, top, and bottom. 2 pages maximum (cover page excluded), on October 27 at the lecture.

## 4 Bonus Features

Only a simple control system simulator with limited scheduling and synchronization features is required in this assignment. However, students have the option to propose bonus features to include more scheduling and synchronization functions (e.g., to address scheduling fairness issues).

If you want to design and implement a bonus feature, you should contact the course instructor for permission before the due date of Deliverable 1, and clearly indicate the feature in the submission of Deliverable 2. The credit for the correctly implemented bonus feature will not exceed 25% of the full marks for this assignment.

## 5 Submission: Deliverable 2 (Code Due: November 12, 2010)

The code is submitted through [connex](#). The tutorial instructor will give the detailed instruction in the tutorial.

### 5.1 Submission Requirements

Your submission will be marked by an automated script. The script (which isn't very smart) makes certain assumptions about how you have packaged your assignment submission. We list these assumptions so that your submission can be marked thus, in a timely, convenient, and hassle-free manner.

1. The name of the submission file must be `p2.tar.gz`
2. `p2.tar.gz` must contain all your files in a directory named `p2`
3. Inside the directory `p2`, there must be a `Makefile`.
4. Invoking `make` on it must result in an executable named `mts` being built, *without user intervention*.
5. You may *not* submit the assignment with a compiled executable and/or object (`.o`) files; the script will delete them before invoking `make`.

#### Note:

1. The script will give a time quota of 1 minute for your program to run on a given input. This time quota is given so that non-terminating programs can be killed.  
Since your program simulates train crossing delays in 10ths of a second, this should not be an issue, at all.
2. Follow the output rules specified in the assignment specification, so that the script can tally the output produced by your program against text files containing the correct answer.
3. The markers will read your C code after the script has run to ensure that the `pthread` library is used as required.

## 6 Plagiarism

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.