CSc 360: Operating Systems (Fall 2016)

Assignment 2: Multi-Flow Scheduler (MFS)

Code Due: 23:55 pm, Nov. 4, 2016

1 Introduction

In this assignment, you need to implement a task scheduler. 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 a router (in a broad sense, a router is a computer anyway) to schedule the transmission of multiple traffic flows. The scheduler is called Multi-Flow Scheduler (MFS).

As shown in Figure 1, n flows to the router should be transmitted via its output interface. Each flow is allocated to a dedicated queue. Assume that the router has only *one* output interface, which must be shared by the flows. To simplify your job, we assume flows are given before hand and their description is stored in a file (refer to Section 2.2).

You will use threads to simulate the flows arriving and waiting for transmission, and your program will schedule these flows to meet the requirements in Section 2.3.

2 Flows

2.1 Properties of Flows

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

1. **Arrival Time**: It indicates when the flow will arrive.
2. **Transmission Time**: It indicates the time required to transmit the flow (i.e., from the time when the flow is scheduled to transmit to the time when its transmission is over). We assume non-preemptive scheduling, meaning that when a flow is transmitting, other flows with a higher priority cannot interrupt its transmission.
3. **Priority**: It is an integer number between 1 (inclusive) and 10 (inclusive), indicating the priority of the flow. **A larger number means a lower priority.**

All times are measured in 10ths of a second. The times will be simulated by having your threads, which represent flows, to call `usleep()` for the required amount of time.

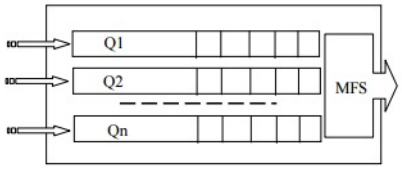

Figure 1: The multi-flow scheduling system

2.2 The Format of Input File

Your program (MFS) will accept *one* parameter on the command line:

```
MFS flow.txt
```

where `flow.txt` is the name of the input file.

2.2.1 File Format

The input file is a text file and has a simple format. The first line contains the number of flows that will be simulated. After that, each line contains the information about a single flow, such that:

1. The first character specifies the unique number of the flow.
2. A colon(`:`) immediately follows the unique number of the flow.
3. Immediately following is an integer that indicates the **arrival time** of the flow.
4. A comma(`,`) immediately follows the previous number.
5. Immediately following is an integer that indicates the **transmission time** of the flow.
6. A comma(`,`) immediately follows the previous number.
7. Immediately following is an integer that indicates the **priority** of the flow.
8. A newline (`\n`) ends a line.

To not kill yourself by checking the false input file, you *should* build a correct input file for your own test. We will **not** test your code with an input file that does not comply with the above format.

2.2.2 An Example

The following file specifies three flows.

```
3
1:3,60,3
2:6,70,1
3:3,50,3
```

The flow specifications are listed in the following table:

Flow No.	Arrival time	Transmission time	Priority
1	0.3s	6s	3
2	0.6s	7s	1
3	0.3s	5s	3

2.3 Simulation Rules

The rules enforced by the MFS are:

1. Only one flow is on transmission at any given time.
2. When a flow arrives, the arriving flow starts transmission if no other flow is on transmission; otherwise, the arriving flow must wait.
3. When multiple flows arrive at the same time and no other flow is on transmission, the arriving flow with the highest priority starts its transmission. If they have the same priority, the one that has the smallest transmission time starts its transmission. If there is still a tie, the one that appears first in the input file starts its transmission first.

4. When a flow finishes its transmission, all waiting flows compete for next transmission according to the following rules:
   (a) The one with the highest priority start its transmission first.
   (b) If there is a tie at the highest priority, the one whose arrival time is the earliest start its transmission first.
   (c) If there is still a tie, the one that has the smallest transmission time starts its transmission first.
   (d) If there is still a tie, the one that appears first in the input file starts its transmission first.

2.4 Output

Your simulation is required to output all events and state changes showing the internal behavior of MFS. The messages must include, but are not limited to:

1. A flow arrives
2. A flow waits for the completion of another transmitting flow.
3. A flow starts transmission
4. A flow finishes its transmission

You must:

1. print the arrival of each flow using the following format string to show the flow attributes:

   ```
   "Flow %2d arrives: arrival time (%.2f), transmission time (%.1f), priority (%2d). \n"
   ```

2. print the ID of the waiting flow and the ID of the waited flow using the format string:

   ```
   "Flow %2d waits for the finish of flow %2d. \n"
   ```

3. print the ID of the flow that starts its transmission and the time when it starts transmission using the format string:

   ```
   "Flow %2d starts its transmission at time %.2f. \n"
   ```

4. print the ID of the flow that finishes its transmission and the time when it finishes transmission using the format string:

   ```
   "Flow %2d finishes its transmission at time %d. \n"
   ```

Note that the output of times (including arrival time, the time when a flow starts transmission, the time when a flow finishes its transmission) is **relative machine time**, calculated by the machine time when the output event occurs minus the machine time when the simulation starts. Therefore, the output of the times may not exactly matches (but should be close to) the results with manual calculation.

3 Design before Coding

You should write a design document which answers the following questions. It is recommended that you think through the following questions *very carefully* before coding. Debugging will be harder after the basic design has been coded. Therefore, it is very important to ensure that the basic design is correct. So think about the following carefully and then write down the answers.

1. How many threads are you going to use? Specify the task that you intend each thread to perform.
2. Do the threads work independently? Or, is there an overall "controller" thread?
3. How many mutexes are you going to use? Specify the operation that each mutex will guard.

4. Will the main thread be idle? If not, what will it be doing?
5. How are you going to represent flows? what type of data structure will you use?
6. How are you going to ensure that data structures in your program will not be modified concurrently?
7. How many convars are you going to use? For each convar:
   (a) Describe the condition that the convar will represent.
   (b) Which mutex is associated with the convar? Why?
   (c) What operation should be performed once `pthread_cond_wait()` has been unblocked *and* re-acquired the mutex?
8. In 25 lines or less, briefly sketch the overall algorithm you will use. You may use sentences such as:

   If a flow finishes transmission, release trans mutex.

4 Submission Requirements (Due: 23.55 pm, Nov. 4, 2016

1. The name of the submission file must be `p2.tar.gz`, which contains all your files, including Makefile, source code, your input file, readme, design document.
2. Your document must be in pdf file format. Other file format or handwriting will **not** be accepted.

Note: We may test your code with our own input file following the format described in Section 2.2,

5 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 may submit your code to an automated plagiarism detection service.

6 Miscellaneous

6.1 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 at least familiar with the following functions:

1. File access functions:
   (a) `atoi`
   (b) `fopen`
   (c) `feof`
   (d) `fgetc` and `fgets`
   (e) `fclose`
2. Thread creation functions:
   (a) `pthread_create`
   (b) `pthread_exit`

(c) `pthread_join`

3. Mutex manipulation functions:

   (a) `pthread_mutex_init`

   (b) `pthread_mutex_lock`

   (c) `pthread_mutex_unlock`

4. Condition variable manipulation functions:

   (a) `pthread_cond_init`

   (b) `pthread_cond_wait`

   (c) `pthread_cond_broadcast`

   (d) `pthread_cond_signal`

It is absolutely critical that you read the `man` pages, and attend the tutorials.
Your best source of information, as always, is the `man` pages.
For help with POSIX threads:

`http://www.opengroup.org/onlinepubs/007908799/xsh/pthread.h.html`

A good overview of `pthread` can be found at: `https://computing.llnl.gov/tutorials/pthreads/`

6.2 Important Notes

We want to (re-)emphasize the following points:

1. **You are required to type in your design document. Hand writing will not be accepted.**

2. We will give a **time quota of** 2 **minutes** for your program to run on a given input. This time quota is given so that non-terminating programs can be killed. So make sure your input file does not include too many long flows (e.g., arrive too late or transmit too long). Since your program simulates flows in 10ths of a second, this should not be an issue, at all.

3. It is required that you use **relative machine time**. This is to avoid cheating with an implementation that does not really simulate the flows but instead performs an offline analysis to obtain scheduling results. The markers will read your C code to ensure that the `pthread` library is used as required. Offline analysis means that your program does not simulate mutual exclusion and thread synchronization but obtains the output based on algorithm analysis. **You will get** 0 **marks if you are caught using offline analysis.**

4. As you progress through your degree, the projects and assignments will continue to become more complicated and difficult. It is impossible to describe in detail every possible case/feature in the assignment specification. Instead you are expected to apply the techniques you have learned so far in your degree, use common sense, and ask questions to lecture instructor or TAs when you "feel" something is unclear. We will announce further clarification, if necessary, on `Connex`. Complaining the specification is unclear at the last minute is not acceptable.

5. You are required to use C. Any other language is not acceptable.

6. You are required to strictly follow the input file format. Failing to do so will result in the deduction of scores.

7. Your work will be tested on `linux.csc.uvic.ca`.

8. Programming with semaphore is permitted but not recommended. You have been warned that debugging with semaphore is much harder.

7 Marking

We will mark your design document and your code submission.

7.1 Design Document (10% of this assignment)

You are required to answer all questions listed in Section 3. Your design will be marked based on the clear and correct logic and the correctness of your algorithm.

7.2 Code Submission(90% of this assignment)

7.2.1 Functionality

1. Your `MFS` must correctly schedule the flow transmissions, with our own test files. We will not disclose all test files before the final submission. This is very common in software test.
2. You are required to catch return errors of important function calls, especially when a return error may result in the logic error or malfunctioning of your program.
3. Your program must output at least the information described in Section 2.4.

7.2.2 Code Quality

We cannot specify completely the coding style that we would like to see but it includes the following:

1. Proper decomposition of a program into subroutines (and multiple source code files when necessary)—A 1000 line C program as a single routine would fail this criterion.
2. Comment—judiciously, but not profusely. Comments also serve to help a marker, in addition to yourself. To further elaborate:
   (a) Your favorite quote from Star Wars or Douglas Adams' Hitch-hiker's Guide to the Galaxy does not count as comments. In fact, they simply count as anti-comments, and will result in a loss of marks.
   (b) Comment your code in English. It is the official language of this university.
3. Proper variable names—`leia` is not a good variable name, it never was and never will be.
4. Small number of global variables, if any. Most programs need a very small number of global variables, if any. (If you have a global variable named `temp`, think again.)
5. **The return values from all system calls and function calls listed in the assignment specification should be checked and all values should be dealt with appropriately.**

7.3 Detailed Test Plan

The detailed test plan for the code submission is as follows.

Components	Weight
Make file	1
Input file	1
Normal cases	4
Priority tie	2
Arrival time tie	2
Transmission time tie	2
All tie	2
Special cases (illegal values)	1
Output format	1
Catch system call return values	1
Comments (functional decomposition)	2
Code style	1
Critical sections	2
Readme	1
Total Weight	23

Note that a score of 23/23 means that you obtain 90% (Section 7.2) of this assignment. If that is the case, congratulations!