ARIZONA STATE UNIVERSITY

CSE 430, SLN 14814 — Operating Systems — Spring 2015

Instructor: Dr. Violet R. Syrotiuk

Project #2

Available 02/25/2015; milestone due Wednesday 03/18/2015; full project due Monday 03/30/2015

In shared memory multi-core architectures, threads can be used to implement parallelism. A standardized C language threads programming interface has been specified by the IEEE POSIX 1003.1c standard. Implementations that adhere to this standard are referred to as *POSIX threads*, or `pthreads`. The purpose of this project is to gain some experience using `pthreads` to create and synchronize threads using semaphores.

1 An Online Social Networking Service: Tweeter

In this project you will simulate a system of n *users*, each of which is connected to an online social networking service *Tweeter* via a *streamer* using a total of $n + 2$ threads: one thread for each user, and one each for Tweeter and the streamer.

Each thread simulating a *user* i, $0 \leq i \leq n - 1$, and executes a sequence of commands found in the text file `useri.txt`. Valid commands include:

- `Handle @name`, where `name` is a character string of maximum length 20 alphabetic characters representing the user's handle.
- `Start #tag`, where `tag` is a character string of maximum length 20 alphabetic characters representing the topic of the tweet. The text that follows this command until the `End` make up the contents of the tweet from user $@h_i$ with hashtag `#tag`. The tweet may be up to 140 characters in length, and may span multiple lines; it begins on the line following the `Start` command. The tweet is to be streamed to Tweeter via the streamer. If the tweet spans multiple lines, then each line must be streamed separately.
- `End #tag`, indicates the end of the tweet with tag `#tag`; the `Start` and `End` commands will come in pairs with matching tags.
- `Follow #tag`, causes Tweeter to transfer *complete* tweets that have been streamed to it matching the `tag` to the streamer for display at the user making the request. If there is no match at Tweeter to the tag, a message to that effect should be returned.
- `Read`, the user pauses for a (short) random amount of time to read tweets.
- `Exit`, indicates that there are no more commands for this user to execute, i.e., user i logs off Tweeter. Thus the thread simulating the user should terminate *gracefully*.

For example, if user i finds the following command sequence in `useri.txt` it would set it's handle to `@happy`, stream a tweet on the topic of `#haiku` to Tweeter, then retrieve any tweets Tweeter has on the topic of `#bostonweather`, pause briefly to read them, then stream another tweet on the topic of `#haiku`, and exit.

```
Handle @happy
Start #haiku
A file that big?
It might be very useful.
But now it is gone.
End #haiku
Follow #bostonweather
Read
```

```
Start #haiku
The website you seek
cannot be located but
endless others exist.
End #haiku
Exit
```

The Tweeter thread stores tweets categorized by `#hashtag`. It builds this repository by accepting tweets issued from users with `#hashtag` streamed to it via the streamer thread. It also accepts requests to follow a `#hashtag` relayed by the streamer thread, responding to the user with any tweets matching the `#hashtag`.

The streamer has limited buffer space for streaming tweets from users to Tweeter, and tweets users wish to follow. In particular, you may assume the streamer has only enough buffer space to hold *one* complete tweet from each of the n users. To simplify the streamer, you may assume that there is a buffer dedicated to holding tweets from user i to Tweeter, and another for tweets being followed by i from Tweeter, $0 \leq i \leq n-1$, i.e., there are two buffers associated with each user. Only when a complete tweet has been streamed by the user to the streamer may it stream the full tweet to Tweeter.

There is only one Tweeter thread to which the users stream tweets and Tweeter is connected to the streamer. It is possible that by the time user i wishes to send another tweet to Tweeter, its previous tweet may not have been received at Tweeter from the streamer. Since user i always uses buffer i it is clear that user i cannot stream the next tweet until the previous one has been streamed to Tweeter, otherwise the integrity of the tweet would be compromised. In this case, user i must wait to be signalled by the streamer before it may start transferring the strings making up the next tweet. Once the streamer indicates that user i can go ahead, it will overwrite buffer i by the contents of the next tweet fed to it by the user.

It is important that the streamer is not blocked by a user reading tweets. That is, the streamer should not be blocked by a reading user and be able to continue streaming tweets from other users. That is, if the user i is reading, the streamer should move on to process a command from user $(i+1) \bmod n$. One suggestion is for the streamer to loop in a round robin fashion from user to user, processing users' commands.

When Tweeter receives a tweet from the streamer it should store the user's handle and the tweet according to the `#hashtag`.

When a user follows a `#hashtag`, Tweeter must transfer all tweets matching the `#hashtag`, along with the `@handle` of the user issuing the tweet. Clearly it must not intermix the lines of tweets from different users!

The entire system should terminate *gracefully*. One way to do this is for the main program to wait for each user to terminate. Once all users have terminated, the streamer could inform Tweeter to terminate (after all tweets have been fed to it), and can then terminate itself.

Your system should be able to handle at most $n = 10$ users.

1.1 Program Requirements for Project #2

You are required to write a C/C++ program using the `pthreads` library to implement Tweeter, an online social networking service, synchronizing the threads using semaphores.

- The number of users, n, is to be read as a command line argument.
- Spawn n user threads.
- Spawn one thread to implement the streamer, responsible for streaming tweets to/from Tweeter. The streamer thread synchronizes with the Tweeter thread using semaphores
- Finally, spawn a thread to implement Tweeter, maintaining a repository of tweets that it can stream to followers.
- Each user i reads and processes commands from a text file `useri.txt`, $0 \leq i \leq n-1$. It synchronizes with the streamer thread using semaphores to implement the commands; see the description in §1.

The output of your program should include:

- A trace of each thread in sufficient detail that it is possible to understand its state.
- The output of a `Follow #tag` command is all tweets that match the hashtag `#tag`, labelled by the user's handle.

Pseudo-code synchronizing the threads, as well as a "shell" of the program is the milestone for this project; see §2.1 for details. The milestone submission deadline is before noon on Wednesday, 03/18/2015; see §2.

A full implementation of the Tweeter service, filling in the details in the "shell" constitute the full project. Your program must run on `general.asu.edu` as it will be tested there. The full project deadline is before noon on Monday, 03/30/2015; see §2.

Sample input files will be provided on Blackboard; use them to test your program.

2 Submission Instructions

This project has two submission deadlines.

1. The milestone deadline is before noon on Wednesday, 03/18/2015.
2. The full project deadline is before noon on Monday, 03/30/2015.

2.1 Requirements for Milestone Deadline

Submit electronically, before class time (noon) on Wednesday, 03/18/2015 using the submission link on Blackboard for Project #2 Milestone, a zip[1] file named `yourFirstName-yourLastName.zip` containing the following items:

Design and Analysis (50%): Write pseudo-code using semaphores to implement the Tweeter online social networking service. Specifically, give pseudo-code for the user, streamer, and Tweeter threads meeting the requirements described in §1.

1. Indicate the purpose of each semaphore you introduce.
2. To what must each semaphore be initialized and why?
3. Discuss how you attempt to maximize concurrency in your design.

Correctness (50%): Implement a "shell" of the user, streamer, and Tweeter threads in C or C++. That is, the only concern of the milestone is synchronization among the threads not any other required functionality. This program must compile and run on `general.asu.edu` and illustrate that the desired synchronization among threads is achieved.

The milestone is worth 30% of the total final project.

2.2 Requirements for Full Deadline

Submit electronically, before class time (noon) on Monday, 03/30/2015 using the submission link on Blackboard for Project #2, a zip file named `yourFirstName-yourLastName.zip` containing the following items:

Design and Analysis (30%): Provide a description of the methodology you followed to implement your pseudo-code. Specifically,

[1] **Do not** use any other archiving program except `zip`.

- describe any errors that arose (e.g., race conditions, synchronization problems, deadlocks) and how you solved them,
- describe the data structure you used to maintain the repository of tweets in the Tweeter threads, and
- any other

Implementation (50%): Provide your documented C/C++ source code for your Tweeter online social networking service. In addition to correct synchronization among threads, this program must fully implement each command.

You may write your code only in C or C++. You must not alter the requirements of this project in any way.

Correctness (20%) Your programs **must** run on `general.asu.edu` using the sample input files. Our TA will test your program there, on these and on additional input files.