CSci 4061: Assignment 3

(Multi-threading and Thread Synchronization)

Released: Nov 2, 2015 Due: Nov 18, 2015 11:59 PM

Total Points: 90

Introduction

The purpose of this homework is to get started with thread programming. For this purpose, we are going to implement a multi-threaded server- *twitterTrend*.

Finding what is trending on Twitter is becoming more popular than ever. Here is your chance to help finding top 3 trending keywords in different cities. *twitterTrend* is a simple program that takes city names and gives top 3 trending keywords in each of those cities. As you can imagine, there could be multiple users (clients) sending requests to twitterTrend. Forking new processes is expensive and therefore, twitterTrend should be a multi-threaded program. In this assignment, we are going to implement a simulation for a client-server program. Real client-server programs usually communicate over networks. We do NOT consider network programming here. However, this assignment asks you to implement the server and simulate client requests. We are going to use a file to simulate client requests.

Rules

- You may work on this assignment individually or in pairs. You are strongly advised to work in pairs.
- Please go through the complete document to understand the assignment. There is a detailed example discussed in section 3.
- Section 5 gives some of the testcases that will be used for grading.
- If you have any questions, please post those on moodle.

1. twitterTrend

a. Setup

None

b. Your Tasks

Your task is to implement a twitterTrend program that is called from the terminal as follows:

```
./twitterTrend intput_file_path num_threads
```

The executable file MUST be named twitterTrend. In addition, it must take two command line arguments.

input_file_path

The input file (clients.in) contains one client per line. Each client (client1.txt, client2.txt, etc.) is represented by a file path. Each client file contains a city name. twitterTrend needs to find trending keywords for each of these cities.

num_threads

The second parameter is the maximum number of clients, the server should be able to handle concurrently. twitterTrend should create num_threads threads to handle concurrency.

The main program should insert the clients from the intput file (clients.in) in a shared queue. Each thread should access the shared queue to get a client and process it. All clients included in input file must be processed. If the number of clients is larger than the maximum number of threads passed to the program, one or more threads need to handle more than one clients i.e. after handling a client, a thread should get another client to handle and so on until the queue is empty. The twitterTrend main program must wait for all child threads to finish processing. Note that the shared queue size is unbounded (unless you implement the extra credit), so all clients can be inserted in this queue once, regardless of the number of children threads. Obviously, you need to use **synchronization** mechanisms **to manage the access to this shared queue** of clients.

NOTE: It is required that all the threads are created first and then the shared queue be populated.

Clients should be served concurrently. Each child thread should be assigned an integer id that ranges from 1 to num_threads. Each client should be identified by the client file it is processing (e.g. client1.txt, client2.txt, etc.). When a thread starts handling a client, the message "Thread *tid* is handling client *client1.txt*" should be printed to stdout, where tid is the thread id and client1.txt is the client file. For each client, twitterTrend should produce a result file containing the result in the form <cityname : keyword1,keyword2,keyword3> **appended by a newline**. The name of the result file should be client file name appended by ".result" i.e. if the client file name is client1.txt, the result file will be named "client1.txt.result". The result file must be created in the same directory as that of client file. After saving the output file, the thread should print to stdout a message "Thread *tid* finished handling client *client1.txt*" with the same notations used above.

Dataset

The Twitter dataset file is a simple comma separated text file named "TwitterDB.txt", which contains a city name followed by 3 keywords per line. You need to look up this dataset by city name to find corresponding trending keywords. Since, reading through a file multiple times is not efficient, you need to store the contents of TwitterDB.txt into an in-memory data structure of your choice, so that the look up is efficient.

c. Extra Credit

To get extra credit, twitterTrend should be able to work with a limited shared queue size. To limit the queue size, you must treat the num_threads (2nd argurment to the program) option as the queue size. If the number of input clients is larger than the size of shared queue, then the main program must initially fill the queue with num_threads clients. Then, the main program will wait for an empty space in the queue and insert each remaining client into the queue as slots become free. Whenever the main program must wait to add clients due to a full queue, it should print a message to stdout: "Waiting to add clients to the full queue"

d. Assumptions and Hints

1. Dataset filename can be hardcoded to "TwitterDB.txt".
2. Every line in TwitterDB.txt is less than 100 characters.
3. Citynames are less than 15 characters.
4. All client*.txt files have only 1 city name. Note: client*.in file can have more than 1 client*.txt files.
5. You do not need a lock to access the data structure containing TwitterDB dataset, because it is a READ_ONLY access and the dataset is not going to be modified during the execution of your program.

2. Submission Instructions and Deliverables

a. All of your code - *.c and *.h files (if any). Your code should compile and run on CSELabs Linux machines.

b. A Makefile that works. Your code should compile, just by us typing **make**. If we can't compile your code by typing "make", then you're definitely going to lose points.

c. You should create a binary named- **twitterTrend**, because we are going to run all the programs as ./twitterTrend arg1 arg2

d. A file named **exactly** README (**not README.txt or any other name**) with the following info. If it does not, you are not following directions and you will lose 5 points for not following directions. Change the commented code below to include your personal information (and that of your partner if you have one) in the same exact format:

```
/* CSci4061 Assignment 3
* name: full name1, full name2 (for partner)
* id: x500_1, x500_2*/
```

3. An Example

Contents of file client1.in:

```
client1.txt
client2.txt
client3.txt
client4.txt
client5.txt
```

Contents of client1.txt

```
London
```

Contents of client2.txt

```
Paris
```

Contents of client3.txt

```
Minneapolis
```

Contents of client4.txt

```
Newyork
```

Contents of client5.txt

```
Berlin
```

<u>Command:</u> ./twitterTrend client1.in 2

For non-extra credit execution:

1. 2 threads will be created
2. All the 5 clients will be added to the shared queue:

client1.txt	client2.txt	Client3.txt	Client4.txt	Client5.txt	…

3. Each thread processes one client at a time until all clients are processed

Sample prints on stdout:

```
Thread 1 is handling client client1.txt
Thread 2 is handling client client2.txt
Thread 1 finished handling client client1.txt
Thread 1 is handling client client3.txt
Thread 2 finished handling client client2.txt
Thread 1 finished handling client client3.txt
Thread 2 is handling client client4.txt
Thread 1 is handling client client5.txt
Thread 2 finished handling client client4.txt
Thread 1 finished handling client client5.txt
```

Output result files:

Contents of client1.txt.result

```
London : MU,Soccer,LondonEye
```

Contents of client2.txt.result

```
Paris : EiffelTower,Cheese,Fashion
```

Contents of client3.txt.result

```
Minneapolis : UMN,Lakes,Snow
```

Contents of client4.txt.result

```
Newyork : Patriots,DeAndre,CharlesNelson
```

Contents of client5.txt.result

```
Berlin : TVOG,nx15,Umfragen
```

For extra credit execution:

1. 2 threads will be created
2. First only 2 clients will be added to the shared queue:

client1.txt	client2.txt

3. Each thread processes one client at a time until all clients are processed
4. Add next two clients to the shared queue:

Client3.txt	Client4.txt

5. Process these 2 clients.
6. Add remaining client to the queue

Client5.txt

7. Process the client.

Sample prints on stdout:

```
Thread 1 is handling client client1.txt
Thread 2 is handling client client2.txt
Waiting to add clients to the full queue
Thread 1 finished handling client client1.txt
Thread 1 is handling client client3.txt
Thread 2 finished handling client client2.txt
Thread 1 finished handling client client3.txt
Thread 2 is handling client client4.txt
Waiting to add clients to the full queue
Thread 1 is handling client client5.txt
Thread 2 finished handling client client4.txt
Thread 1 finished handling client client5.txt
```

The output result files will be same as above.

4. Rubric:

Testcases: 60
Extra Credit: 10
Code Documentation: 25
README: 5
Total: 90

5. Sample test cases:

Here are some of the testcases we will be using for grading. These files are given in the released folder.

Input files:

```
==> client1.in <==
client1.txt

==> client2.in <==
client2.txt
client3.txt

==> client3.in <==
client4.txt
client5.txt
client6.txt
client7.txt

==> client4.in <==
client13.txt
client14.txt
client15.txt
client16.txt
client17.txt
client18.txt
client19.txt
client20.txt
client21.txt
client22.txt
```

Client files:

```
<client file name:city name>
client1.txt:Minneapolis
client10.txt:Sydney
client11.txt:Montreal
client12.txt:Mumbai
client13.txt:Frankfurt
client14.txt:Rio
client15.txt:Calgary
client16.txt:Chicago
client17.txt:Bordeaux
client18.txt:Toronto
client19.txt:Munich
client2.txt:Paris
client20.txt:Newyork
client21.txt:Canberra
```

```
client22.txt:Minsk
client23.txt:Vancouver
client24.txt:Melbourne
client25.txt:Perth
client3.txt:London
client4.txt:Vienna
client5.txt:Berlin
client6.txt:Austin
client7.txt:Brasilia
client8.txt:Boston
client9.txt:Delhi
```

Sample execution:

```
./twitterTrend client1.in 1
./twitterTrend client2.in 1
./twitterTrend client2.in 2
./twitterTrend client2.in 3
./twitterTrend client3.in 2
```

For extra credit:

```
./twitterTrend client4.in 3
```