May 26th, 2021

CSE344 – System Programming - Final – v2
Sockets and threads

Reminder: according to the syllabus you must have a non-zero midterm grade to participate at the final. If you didn't submit the midterm, don't bother with the final.

Context

Database servers are the backbone of our information society. Everytime you login to social media, make/like a post, check your bank balance, shop online, make a phonecall, a database server somewhere receives and processes one or more queries. There are many database management systems (DBMS): Oracle, IBM DB2, postgre, mssql, access, mysql just to name a few. Although they are produced and maintained by different companies and have different software structures, all DBMS organize their data in terms of classic **tables** containing rows and named columns, and understand the universal Structured Query Language (SQL) for reading and writing to the underlying data. This project is about implementing a very rudimentary database server/client.

Objective

You are expected to write 2 programs: a client and a server. The server will run as a single instance, will open up a stream socket and wait for clients to connect and conduct SQL queries. The client can/will run as multiple instances (i.e. there will be more than one client processes running concurrently), read its SQL queries from a file, send them to the server, and print the received responses. Once all their queries are done, the clients will exit.

--How (Server) --

The server process will be a daemon, this means that you should:

- take measures against double instantiation (it should not be possible to start 2 instances of the server process)
- make sure the server process has no controlling terminal
- close all of its inherited open files

```
./server -p PORT -o pathToLogFile –l poolSize –d datasetPath
```

PORT: this is the port number the server will use for incoming connections.

pathToLogFile: is the relative or absolute path of the log file to which the server daemon will write all of its output (normal output & errors).

poolSize: the number of threads in the pool (>= 2)

datasetPath: is the relative or absolute path of a csv file containing a single table, where the first row contains the column names. It will be selected from the URL: https://www.stats.govt.nz/large-datasets/csv-files-for-download/

The server's main thread: The server will start by loading the dataset into memory. How you store/represent the dataset is up to you; but make sure you **justify your data structure selection**.

The server will possess a pool of POSIX threads, and in the form of an endless loop, as soon as a new connection arrives, it will forward that new connection to an available (i.e. not busy) thread of the pool, and immediately continue waiting for a new connection. If no thread is available, then it will wait in a blocked status until a thread becomes available. Plan your synchronization carefully. The server process must exit gracefully when it receives SIGINT.

The server's pool threads: all of them will execute the same function. Each of them will operate in an endless loop waiting for a connection to be delegated to them from the main thread. If there is no job to do, they will remain blocked. Once the connection is handled they will wait again for a new connection.

Server: Handling a connection
The connected client will send a single SQL query through the socket, the pool thread will parse it, execute it and return its result to the client. The query is assumed to have correct syntax. The thread will additionally sleep for 0.5 seconds to simulate intensive database execution.

Only 2 commands will be supported: select and update.

The *select* command only reads the dataset, whereas the *update* command modifies the dataset. Implement the readers-writers paradigm, and attribute priority to the writers. This means that read-only/select queries will be executed concurrently by all threads, while only a single writer/update query can execute at any given time by any thread. If the client has more than one queries, it will continue to send them to the server using the same connection (meaning that the thread might change roles from reader to writer and vice versa). Don't use file locks.

The various steps taken by the servers' threads will be printed to the log file, as shown at the sample output.

Output (Server) (every row will start with a timestamp)

```
Executing with parameters:
-p 34567
-o /home/erhan/sysprog/logfile
-l 8
-d /home/erhan/sysprog/dataset.csv
Loading dataset...
Dataset loaded in 0.25 seconds with 217831 records.
A pool of 8 threads has been created
Thread #5: waiting for connection
Thread #4: waiting for connection
Thread #6: waiting for connection
Thread #7: waiting for connection
Thread #3: waiting for connection
Thread #0: waiting for connection
Thread #2: waiting for connection
Thread #1: waiting for connection
A connection has been delegated to thread id #5
Thread #5: received query 'SELECT * FROM columnName1;'
Thread #5: query completed, 12 records have been returned.
…
No thread is available! Waiting…
…
Termination signal received, waiting for ongoing threads to complete.
All threads have terminated, server shutting down.
```

--How (Client) --
The client's job is easy. The query file will contain an arbitrary number of queries for each client process. For example, 5 queries for client 1, 10 queries for client 2, in random order.

```
./client –i id -a 127.0.0.1 -p PORT -o pathToQueryFile
```

`-a:` IPv4 address of the machine running the server

`-p:` port number at which the server waits for connections (>1000, the first 1000 are reserved to kernel processes)
`-o:` relative or absolute path of the file containing an arbitrary number of queries
`-i:` integer id of the client (>=1)

The query file will contain one query per line. The line will start with the client id, have a single space, and the SQL query will then follow; such as (check online resources for what they do):

```
1 SELECT * FROM TABLE;
1 SELECT columnName1, columnName2, columnName3 FROM TABLE;
2 UPDATE TABLE SET columnName1='value1', columnName2='value2' WHERE columnName='valueX'
1 SELECT DISTINCT columnName1,columnName2 FROM TABLE;
```

The possible commands will be:
SELECT (single or multiple columns),
SELECT DISTINCT,
and UPDATE (single equality condition).

The client process will read the file, and for every query with its ID, it will send it to the server, and print the result that it will receive. Once all its queries are completed, it will terminate.

Output (Client) (every row will start with a timestamp)

```
Client-1 connecting to 127.0.0.1:45647
Client-1 connected and sending query 'SELECT * FROM TABLE;'
Server's response to Client-1 is 5 records, and arrived in 0.3 seconds.
Customer Name            Contact        Address            City      PostalCode Country
1        Alfreds         Maria Anders   Obere Str. 57      Berlin    12209      Germany
2        Ana Trujillo    Ana Trujillo   Avda. de la Const. México    05021      Mexico
3        Antonio Moreno  Antonio Moreno Mataderos 2312     México    05023      Mexico
4        Around the Horn Thomas Hardy   120 Hanover Sq.    London    WA1 1DP    UK
5        Berglunds       Christina      Berguvsvägen 8     Luleå     S-958 22   Sweden
A total of 3 queries were executed, client is terminating.
```

Evaluation
You program will be evaluated with an arbitrary dataset file and arbitrary parameters.

Requirements:
- Solve all synchronization issues with condition variables and mutexes.
- Don't modify the input file in any way

Rules:
1) Compilation error: grade set to 1; if the error is resolved during the demo, then evaluation continues.
2) Compilation warning (with respect to the **-Wall** flag); -1 for every warning until 10. -20 points if there are more than 10 warnings; no chance of correction at demo.
3) No makefile: -20
4) No pdf (other formats are inadmissible) report submitted (or submitted but insufficient, e.g. 3 lines of text or no design explanation, etc): -20.
5) A report prepared via latex (pdf and tex source submitted together): +5
6) If the required command line arguments are missing/invalid, your program must print usage information and exit. Otherwise: -10
8) The program crashes/freezes and/or doesn't produce expected output with normal input: -80
9) Presence of memory leak (regardless of amount – checked with valgrind) -30
10) Zombie process (regardless of number) -30
11) Deadlock of any kind due to poor synchronization -50

12) Use of poor synchronization: busy waiting/trylock/trywait/timedwait/sleep: -80
13) Late submissions will not be accepted
14) In case of an arbitrary error, exit by printing to stderr a nicely formatted informative message. Otherwise: -10

Is my homework submission valid?
If at least one query is executed successfully, then yes.

Submission rules:

- Your source files, your makefile and a report; place them all in a directory with your student number as its name, and zip the directory.
- Your report must contain: how you solved this problem, your design decisions, which requirements you achieved and which you have failed.
- The report **must be in English**.
- Your makefile must only compile the program, not run it!
- Do not submit any binary executable files. The TAs will compile them on their own boxes.
- Any deviation from the submission rules will results in automatic (without content evaluation) failure of the homework.

Good luck.