System Programming Final Project

Zafer ALTAY 161044063

June 2021

1 Problems and Solutions

First, let me explain the structs I used in the assignment.I kept every element in the csv file as an Element struct.Inside this structure, I keep the row and column numbers next to the string of the csv element.In my second struct, there are the pointers and socket numbers that the server threads will reach.That pointer is the pointer holding csv files.

Now let's go back to main thread.First I saved the command line values with getopt.I kept the csv file I read with one dimensional Element struct pointer.First, I reserved 1 element with malloc and then added it each new value with realloc I read. While doing this, I recorded the number of rows and columns by counting at the same time.I read each element I read letter by letter with read().I added the character I read to the temp string. If it is not in quotation marks, I add commas and newline characters until I read them.If I read one of these, I have completely taken the element in the file. I am adding this element with realloc.If it is a newline character, the number of lines has increased.

After parsing and saving all the data in this way, I created the threads for the thread pool.I have stored the threads' ids in the tid array. The argument I send when the thread is created is a struct.I mentioned the struct that stores the pointers to which the threads will reach, I am using it exactly here. Because we have to send the elements to the thread while creating it. But because we created a thread pool, threads should not die, here I can't send new members, so I changed this struct so that the thread can access these changes.So instead of sending a new element to a thread, I am changing the struct. Thread, on the other hand, reaches the change by using the contents of the struct it receives. So what's changed?When I change the connection number that the socket accepts with accept in the struct, I am communicating with the new client with the new connection number.By the way, what I sent to the thread is not just 1 struct, it is the starting point of the struct array so that no matter which thread wakes up, it will reach the head of the array and get the number and run.To explain in more detail, when a new link comes, I write it to a struct, ok, but which thread's struct should I write it to? How do I know which one will wake up? At this point, I keep head and tail and add each new number to tail and increase

it by 1.he thread that wakes up reaches the head index of the struct array and uses that struct.Thus, no matter which thread wakes up, it reaches the waiting accept number.I created the socket structure. Then I started an infinite loop in the main. As the client came, I saved its number and increased the total number of clients I kept by 1. If the total number of clients in the thread pool time, I put myself on hold for the main thread with a condition variable.If there is a thread that has finished, it reduces this number and sends a signal, so that the main thread wakes up and waits for requests because there is space.

Now let's go over what 1 server thread does.I make a thread that occurs for the first time sleep with a condition variable as it will receive a signal when the element comes.When all threads are created, they come to this barrier and wait. When it receives a signal, it waits to wake up.Any thread that receives a signal and wakes up takes the number indicated by the head, increases the head and unlocks the mutex. It uses this number as the socket number. This number will be used when communicating with the client.Then it checks whether the query sent by the client is update or select. While doing this, I read the entire query and tokenized it word by word.Since the token in the first row is the id, I looked at the token in the second row and understood this.If this is token select I'm reading, this client is a reader. If I'm reading token update, it will update. This is a writer.This is important because if an author has arrived, he or she will only be able to access the database alone at that time.Because while making an update, reading the others at the same time may cause erroneous readings. By the way, I provided the synchronization with the reader-writer problem solution.Then I started the query parse process and sending data to the client. According to the words I tokenized, I applied the desired operations on the elements indicated by the number of rows and columns I kept.Using keywords like *, FROM, WHERE, I did the right thing. For example, if * comes after select, or * comes after select distinct, I printed the entire table.If distinct does not come after select, it is a colon that comes in, and everything that comes up to the FROM keyword will always be a colon.I would keep a list of it, of course, as the number of elements can be unlimited so I keep it as a pointer. Then I look for matching columns and find the column number and print the elements that have that column number.I also do this for the columns that come after the Distinct, but here I save every element I write and if the new record was written before, I skip it.I follow a similar process in the update token and find the right columns. I take the column and the value after the WHERE word and look for the matches.I send the results to the client after making the changes in the row with the matches.After the process is finished, I send the special word 'OK' to the client. After the client receives it, if there is a new query, it sends it if there is no new query, it sends the 'exit' keyword to the server. If the server does not exit, it works for a new query. If it get exit keyword, it makes the necessary adjustments and starts to wait.This way it works in loop.

Things are easier for the client. It sends a query to the server according to the information it reads with getopt, and waits for the results.After receiving the result, according to the arrow keyword sent by the server, if there is another

query ,client send it, if not it sends the exit keyword.And it closes.

2 Pass File

In general, I cite the expectations and shortcomings for convenience:

I created a thread pool.
All queries are working.
I read the CSV file and saved it to memory.
I designed it in accordance with the Reader-Writer problem.
I enabled Server-Client synchronization.
I did all the synchronizations as requested.
I checked with Wall. (I have 3 warnings).
I checked for leaks with Valgrind. (No definetly memory leak).