GIT Department of Computer Engineering
CSE 344 - Spring 2020
Final Project Report

Esra EMIRLI
151044069

June 28, 2020

Structs

Server.c :

threadParam

- **id :** Specifies thread number
- ***connection :** It is a connection sequence used to deliver clients successfully connecting to threads.
- ***cache :** Struct to be used as a database. It is available dynamically to keep every database record.
- ***cindex :** It keeps the number of records in cache.

Cache

- ***path :** It is a dynamic integer array. Keeps the path between source and destination included.
- **src :** Holds the source node.
- **dest :** Holds the destination node.
- **pathSize :** Holds the number of nodes between the source and destination path.

Graph.c :

AdjList : Holds the AdjListNode struct in it. Using nodes to start.

AdjListNode : Specifies the nodes. Creates the linkedlist structure. It contains a data and a pointer of its kind.

Graph : It exists to hold nodes read from the file. It holds the number of vertex and the vertex array.

Result : It has an array to hold the path between the source and destination, and the variable that holds the size of that array. Holds the path between two nodes.

Functions

Server.c :

freeGarbages() : It frees dynamically received memories in the program.

timeStamp() : It creates the necessary format for timestamp in the log file and writes to the output file.

make_daemon() : It performs the fork and session operations required to make process a deamon process.
In addition, *"grep MorningStar / var / log / syslog"* command writes its pid to the logs so that its pid can be seen from the logs.

findSlot(int* arr, int size, int order) : It takes an array, array size, and variable as parameters.
If the value of the variable is -1, it holds a random number on the array and if the element of the array in this number is -1, the index returns.
If the value is not -1, it holds the new random number and repeats until it finds -1.
If a number other than Order -1 returns the first non -1 index.

intHandler() : It captures the SIGINT signal. Required for the termination of the program. It waits for the running threads to end.
When the working threads are finished, they return the files opened and the memories taken from the system and ensure that the program ends safely.

func(void*) : It is the function where client requests captured in the mail are processed. This function runs threads.
It meets the request from the client and waits for the new client to come after interacting with the client and completing its service.

main() : It is the starter function of the program. It processes the parameters from the terminal. Reads the graph according to these parameters, opens the files for logs. It makes the TCP settings so that the clients can connect. The client waits and accepts the requests according to the opened port and sends them to the threads. It continues to operate until the **SIGINT** signal arrives.

Graph.c :

createGraph(int) : Takes the number of vertex as a parameter. Creates and returns graphs according to Vertex number.

addEdge(struct Graph*, int, int) : It takes graph and two nodes as parameters. It creates a connection with the two nodes it takes and adds the grapha that comes as a parameter.

newAdjListNode(int) : Takes the destination node as a parameter. Used to add neighbors to the nodes. Adds destination information to its data and returns.

findPath(struct Graph *, int, int) : It is the function called to find the path of the source and destionation node from the client.
First, it calls the bfs function to find the neighbors of the source.

If the size of the returned array is zero or one of the neighbors of the source node is not a destination, the struct containing the array with the size 0 is returned to indicate that there is no path.

Otherwise the nerden function is called in a loop until the return value equals the source node, and each returned node value is stored in an array. The struct containing the array where the inverse of this array is saved is returned.

bfs(struct Graph *, int) : The source node entered by the client is sent as a parameter and returns the struct containing the array and size of all the neighbors of that node.

nerden(struct Graph *, int, int) : Source and destination nodes are given as parameters and the last node that accesses the destination is returned.

Some helps

I did graph related operations (create, addEdge) from the `https://www.geeksforgeeks.org/graph-and-its-representations/` link.

I made the bfs function of Breadth First Search functions by getting help from the `https://www.geeksforgeeks.org/print-paths-given-source-destination-using-b` link.

Client.c :

func(int) : It sends two node information to the server using the connection socket to the server and then presses the server's response to the terminal.

main() : It provides connection to the server with the parameters coming from the argument. The port read from the parameters sends connection request to the server with the IP address information. If this request is successful after sending the connection request, it transmits the connection socket to the relevant method.

timeStamp() : It is used to print the timestamp information required for the information to be displayed in the terminal in a format.

Problem Solution Approach

Server

When the program is run, the parameters from the argument are read and the process continues to run as a deamon process.

As soon as the server is opened, a graph is created. Files are created for logging and the file description is kept open for use in other functions.

In order to use threads as common, threadParam structure is created and prepared as a parameter for threads. It then creates the minimum number of threads specified. All of the threads that are created take a number for themselves.

It controls the sequence they can control connection with the server. If there is no connection, it waits depending on the condition variable.

All of them are in a pending state since there is no connection when they first formed. The server expects a request from the client in an infinite loop.

When the request comes to the server, it holds the socket for connection. If there is a thread that controls 75% of the minimum number of threads working, increase the number of thread pool and create new thread. If the connection is successful, it writes this connection information to a suitable place in the connection pool. It awakens a random thread.

Controls the awakened thread connection pool. Increases the number of threads running. It gets the connection information and goes into reading mode. It receives 2 node information from the client over the socket. It controls the start and end nodes from the database. If it exists in the database, it writes the path between the two nodes from the database to the log file and send it to the client. If it is not in the database, it searches the start and end nodes in the graph in the cache structure. It adds a record to the database whether or not it finds it in Graph. Writes the response to the socket to send to the client. Reduces the number of threads running. The socket communicated with the client is closed and waited for the new connection.

When requests to the server exceed the size of the thread pool, the server receives the connection and adds it to the connection pool. Sends a signal to the signal condition variable to evoke a random thread. Since all threads are running at that moment, no thread wakes up. The connection stops at the connection pool. When the threads are finished, they check the connection pool and if there is a connection, it serves the client of that connection as mentioned above.

Some operations occur when the **SIGINT** signal comes. There are small details for these processes to take place. Threads are not in infinite loop. There is a variable whose status changes after the signal is formed. As long as the signal indicating the signal status is negative, the threads run in a continuous loop. When the signal occurs, the function that controls the signal broadcasts to wake up all of the threads in standby state. Then it waits with pthread_join to wait for all threads to end. All threads in standby mode wake up and exit the function without any action. When Broadcast arrives, running threads exit the function without logging again when they are done. Thus, when all threads are terminated, the program ends by releasing the resources it receives while the program is running.

Client

Connection to the server is provided with the parameters read as arguments. Two integers are sent to the server for the start and end nodes via the socket opened for connection. After sending these two information, if there is a way between the two nodes from the server, this information is received. If there is no connection between the two nodes, the corresponding answer is transmitted. The client shows the response it receives for each case in the terminal.

Sync problems encountered

Each time a thread is connected, it waits in the wait state depending on the condition variable. A dynamic array was used because they used the same connection pool. The structure opened for the database also used a dynamic array. When the connection comes, the server wakes up a random thread and sends a signal to the condition variable.

The number of thread threads that are awakened, performs a secure process alone through a mutex during the search and writing operations in database.

Making Deamon :
Parent process fork.
parent terminates himself.
child process get session.
ignore SIGCHLD and SIGHUP.
fork again and the parent is destroyed.
new authorization for child.
open file descriptions close.
pid written to log file.

grep MorningStar / var / log / syslog command to go to the log where the server's pid is written
The server is terminated by sending SIGINT signal to the server with kill -2 pid.

Memory Leak

When I checked memory leak with valgrind, I encountered some leaks. These were problems with thread creation and completion.
It is written that when I search for these leaks on google, I cannot solve them with free, it is a normal situation.
`https://stackoverflow.com/questions/17642433/why-pthread-causes-a-memory-leak`

Warning ! The log file cannot be completed before the server is terminated.