

CSE344 System Programming Final Project Report

June 27, 2020

Student:Fatih Selim YAKAR

Student No:161044054

Instructor:Erchan APTOULA

1 Overview

In this assignment, I did synchronization and exchange between an indefinite number of client and server (and their threads) through sockets and pthread tools. The context was as follows: There was a file with the server's input. And this file contained a graph. The desired graph was to be read by the server and turned into a data structure, and then the path from the client, between 2 nodes, was to response the requests. There would be another system inside the server. This system is as follows: The server that will occur in the server main thread and calculator threads will transmit each incoming connection to the threads, then if there is a path in the cache structure, it will read it with the writers-readers paradigm, calculate it if not, then write it to the cache and send it to the client. On the other hand, I used an additional thread to make the server system dynamic. This additional thread had its own special condition variable and mutex. I created global arrays, so that all threads were able to reach. I created mutex for the number of threads for critical sections. For some cases, I solved the wait by creating a condition variable as many as the number of threads. For finishing in the Ctrl-c state and the nominal state, I used condition variables first. In any case, I freed all the resources while the program was over. You can find the details below.

2 Synchronization Problem Between Threads

In case of this homework, there are main thread also there are calculator threads. Threads have to work in parallel among themselves. On the other hand, in the critical section, it is necessary to add a connection fds to the queues of each thread with the main thread and to get a fd from the queues by the calculator threads.

2.1 Providing critical section

I used mutex to provide the critical section. If I tried to do this for all threads with a single mutex, not all threads would work at the same time. I created a separate mutex for each thread to ensure that all threads can work in parallel, and while adding a queue socket fd to the main thread, I locked the necessary thread's mutex and performed the necessary operation and unlocked it after doing the operation. Likewise, within calculator thread, each calculator thread made the process of path calculating with writers and readers paradigm with accessing cache.

2.2 Waiting with condition variables for queues

I created a fd queue for each calculator thread. When these queues are full, the main thread should not add socket fd to the queue and the florist should not receive socket fd from the queue when the queue is empty. So in these cases, it should wait for it to fill up or empty. I did not use condition variable to be full since I dynamically built queues. But since the calculator threads should wait if the queue is empty, I have defined the condition variable as many as the number of threads. I did pthread cond signal when adding socket fd to main thread queue. And in the calculator thread I made pthread cond wait if the size of the queue was zero so I solved the sync issue in the queues.

2.3 Readers-Writers paradigm for accessing cache

In reaching the cache, I used the writers readers paradigm to get writer priority or reader priority or not to give any priority (Bonus part). In order to implement the Writers-Readers paradigm, I used int variables named 4 active_readers, active_writers, waiting_readers and waiting_writers. In addition, I used 2 ok_to_read and ok_to_write condition variable and finally 1 mutex named rw_mutex. I implemented the paradigm as in the figure I showed below. (For the -r parameter, I edited it by doing a few if it was processed in the lesson.)

Writers-Readers Figure

Example: readers-writers

Reader

```
wait until no writers
access database
check out -- wake up waiting writer
```

Writer

```
wait until no readers or writers
access database
check out -- wake up waiting readers or writer
```

3 In Case Of SIGINT(ctrl-c)

Termination of the program (that is, SIGINT signal coming) was designed as follows: I have defined a sigint handler function with sigaction for the server program to exit. When I entered this function, I just printed an information that it was caught and then changed the global sigint_flag from 0 to 1. Then, since the main thread condition was while (sigint_flag == 0), I stopped receiving the main thread immediately. After that, I sent a cond signal to all threads to wake up the threads sleeping under the main thread. The threads receiving the signal entered and exit the finishing condition following the cond wait condition. After the expected threads with the pthread_join function in Main, I made a refund of the resources.

4 Cache Data Structure

For the cache data structure, I created a data structure similar to Hash table using the combination of 2 data structures. I have defined a CacheNode array with as many elements as Vertex in the structure. Then, every array element became a structure containing the next pointer, end node and path, like the linked list node. Thus, in the case of 0– > 20 searches, it went to the CacheNode at index 0 and looked at the equation of its endnode, and went through the linked list by making item = item->next, thus accessing and writing $O(n)$ [n = path number in that index].

5 Graph Data Structure

For the Graph data structure, I used a mix of adjacency list and adjacency matrix, structurally the same as the adjacency list, but in the rest of the arrays, I used a dynamic array, not a linked list. So I used a 2d matrix like an adjacency list.

6 Dynamic thread pool

I used 1 thread 1 mutex and 1 condition variable to increase the dynamic thread pool, that is, to increase the capacity of the system's threads by 75% and above 25% capacity. As soon as the pool I used was working in thread and main, I made mutex lock and unlock to provide critical region as normal. I sent `pthread_cond_signal` so that the main thread can work with the pool thread in every loop. I did `pthread_cond_wait` on the pool thread with the condition of While (busy state is less than 75% or thread count has reached the limit). Then I created new threads with `pthread_create`, increasing 25% if it exits this `pthread_cond_wait` loop.

7 Functions Used And Their Explanations

7.1 server.c

- **static void create_daemon():**Transforms the current process to daemon.
- **void get_request(int sockfd,int thread_index):**Function that takes the path request from the client and calculates the path according to the request. Applies the Writers-readers thread structure.
- **void open_socket(int port):**Opens a new socket.
- **void *calculator_threads (void *arg):**Thread function that processes connections (accepted fds) from the main thread and returns to the client.
- **void *pool_thread(void *arg):**Thread function that increases the number of threads by 25% if it looks at the thread size and operates at over 75%.
- **void sigint_handler(int signum):**SIGINT handler function.
- **int main(int argc,char *argv[]):**The files are opened, the daemon process is created, then the sockets are opened and after all threads are started, main thread runs until the sigint signal comes.

7.2 client.c

- **void send_request(int sockfd,int source,int destination):**Sends request to get path.
- **int main(int argc,char *argv[]):**Opens sockets and sends path request according to command line arguments.

7.3 cache.c/h

- **struct Cache* create_cache(int capacity):**Creates the cache stucture and returns.
- **int find_end_node_in_path(char *path):**Finds the end node of char* path by seperating tokens.
- **int find_first_node_in_path(char *path):**Finds the first node of char* path by seperating tokens.
- **void add_path_in_cache(char* path,struct Cache* cache,int output_fd):**Add new path in the linkedlist array.
- **char* find_path_in_cache(int start_node,int end_node,struct Cache* cache):**Finds the path in the cache.
- **void free_cache(struct Cache* cache):**Frees the cache data structure.

7.4 graph.c/h

- **int find_maximum_index(char* file_name,int output_fd)**: Finds and returns the maximum indexed node to determine the number of nodes.
- **struct Graph* initialize_graph(char* file_name,int output_fd)**: Initializes the graph reading line by line.
- **int add_node(int first_node,int second_node,struct Graph* graph)**: Adds node in graph.
- **int is_there_node(int first_node,int second_node,struct Graph* graph)**: Controls if there is a node in graph.
- **int BFS(struct Graph* graph, int s, int d, int* parent_array)**: BFS that stores the parents.
- **int find_path_in_graph(struct Graph* graph, int s, int d,char *return_path)**: Cover function of BFS.
- **void free_graph(struct Graph* graph)**: Frees the graph structure.

7.5 input_output.c/h

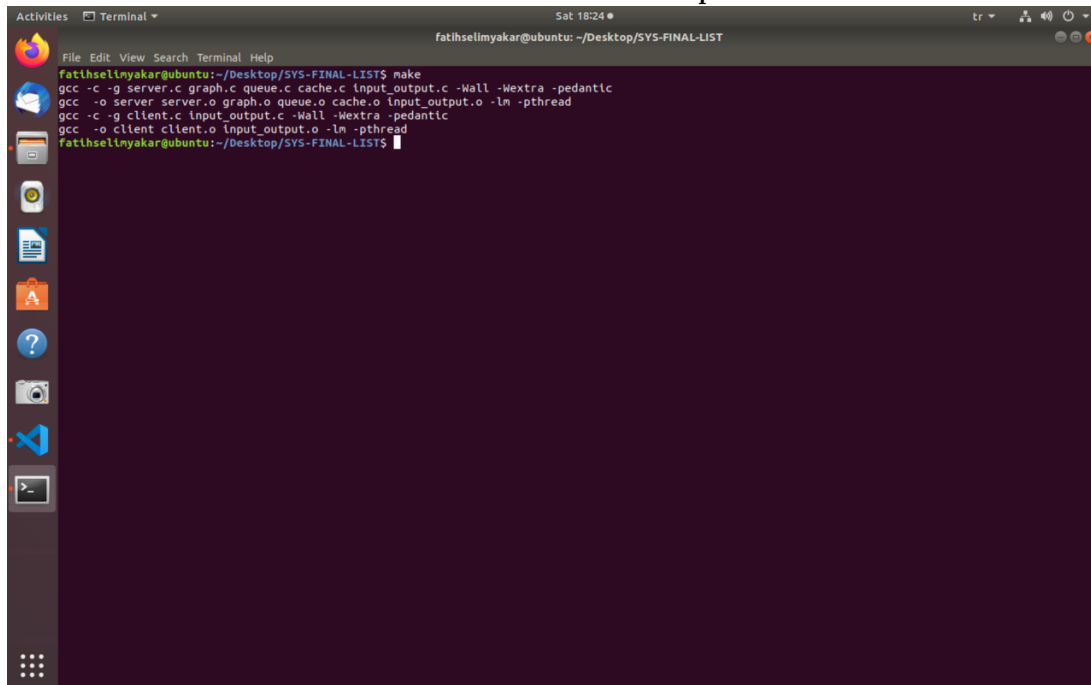
- **unsigned long get_time_microseconds()**: Gets time according to microseconds.
- **void print_error(char error_message[])**: Prints error in the STDERR.
- **void print_string_output_fd(char string[],int output_fd)**: Prints string in the output_fd.
- **void print_string(char string[])**: Prints string in the output_fd.

7.6 queue.c/h

- **struct Queue* create_queue(int capacity)**: Creates the queue and returns it.
- **int is_empty(struct Queue* queue)**: Controls if it is empty.
- **int is_full(struct Queue* queue)**: Controls if it is full.
- **int add_item(struct Queue* queue,int item)**: Adds item to rear.
- **int get_item(struct Queue* queue)**: Gets item in the front.
- **int get_item_and_delete(struct Queue* queue)**: Gets item in the front and deletes it (pop).
- **void free_queue(struct Queue* queue)**: Frees the queue structure.

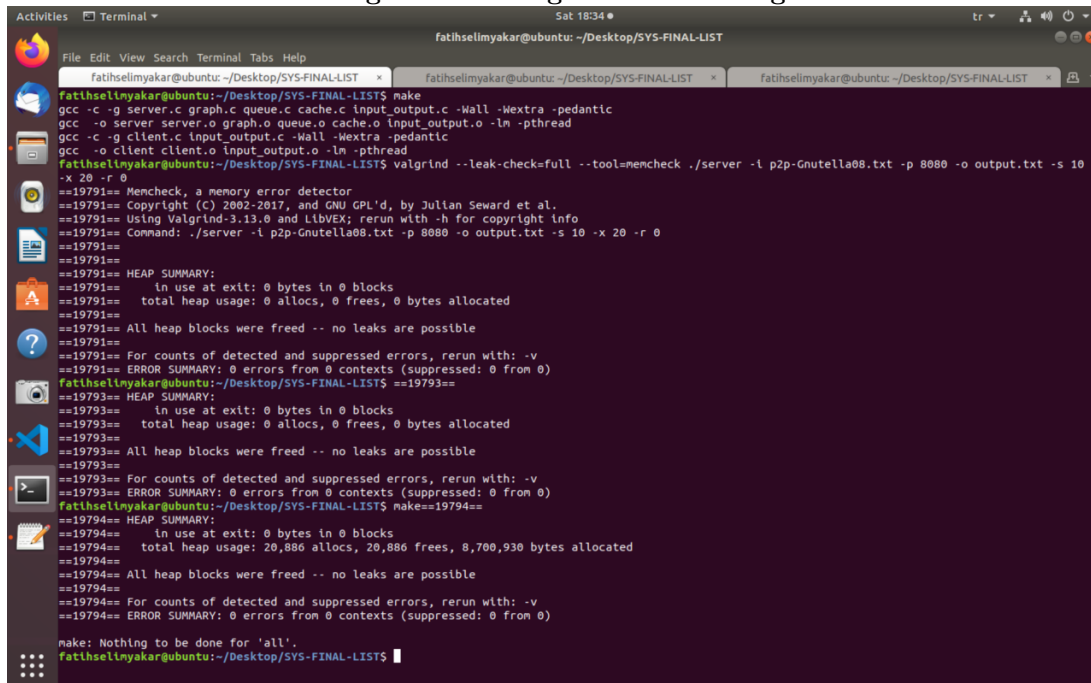
8 Sample Running Screenshots

Makefile with -Wall -Wextra -pedantic



```
fatihselimyakar@ubuntu: ~/Desktop/SYS-FINAL-LIST$ make
gcc -c -g server.c graph.c queue.c cache.c input_output.c -Wall -Wextra -pedantic
gcc -o server server.o graph.o queue.o cache.o input_output.o -lm -pthread
gcc -c -g client.c input_output.c -Wall -Wextra -pedantic
gcc -o client client.o input_output.o -lm -pthread
fatihselimyakar@ubuntu:~/Desktop/SYS-FINAL-LIST$
```

Running and finishing server with valgrind



```
fatihselimyakar@ubuntu:~/Desktop/SYS-FINAL-LIST$ make
gcc -c -g server.c graph.c queue.c cache.c input_output.c -Wall -Wextra -pedantic
gcc -o server server.o graph.o queue.o cache.o input_output.o -lm -pthread
gcc -c -g client.c input_output.c -Wall -Wextra -pedantic
gcc -o client client.o input_output.o -lm -pthread
fatihselimyakar@ubuntu:~/Desktop/SYS-FINAL-LIST$ valgrind --leak-check=full --tool=memcheck ./server -l p2p-Gnutella08.txt -p 8080 -o output.txt -s 10 -x 20 -r 0
==19791== Memcheck, a memory error detector
==19791== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==19791== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==19791== Command: ./server -l p2p-Gnutella08.txt -p 8080 -o output.txt -s 10 -x 20 -r 0
==19791==
==19791== HEAP SUMMARY:
==19791==    in use at exit: 0 bytes in 0 blocks
==19791==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==19791==
==19791== All heap blocks were freed -- no leaks are possible
==19791==
==19791== For counts of detected and suppressed errors, rerun with: -v
==19791== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
fatihselimyakar@ubuntu:~/Desktop/SYS-FINAL-LIST$ ==19793==
==19793== HEAP SUMMARY:
==19793==    in use at exit: 0 bytes in 0 blocks
==19793==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==19793==
==19793== All heap blocks were freed -- no leaks are possible
==19793==
==19793== For counts of detected and suppressed errors, rerun with: -v
==19793== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
fatihselimyakar@ubuntu:~/Desktop/SYS-FINAL-LIST$ make==19794==
==19794== HEAP SUMMARY:
==19794==    in use at exit: 0 bytes in 0 blocks
==19794==   total heap usage: 20,886 allocs, 20,886 frees, 8,700,930 bytes allocated
==19794==
==19794== All heap blocks were freed -- no leaks are possible
==19794==
==19794== For counts of detected and suppressed errors, rerun with: -v
==19794== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

make: Nothing to be done for 'all'.
fatihselimyakar@ubuntu:~/Desktop/SYS-FINAL-LIST$
```

Running client with valgrind

```
Activities Terminal Sat 18:35 fatihselimyakar@ubuntu: ~/Desktop/SYS-FINAL-LIST
File Edit View Search Terminal Tabs Help
fatihselimyakar@ubuntu: ~/Desktop/SYS-FINAL-LIST x fatihselimyakar@ubuntu: ~/Desktop/SYS-FINAL-LIST x fatihselimyakar@ubuntu: ~/Desktop/SYS-FINAL-LIST x
==20171==
==20171== All heap blocks were freed -- no leaks are possible
==20171==
==20171== For counts of detected and suppressed errors, rerun with: -v
==20171== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
fatihselimyakar@ubuntu: ~/Desktop/SYS-FINAL-LIST$ valgrind --leak-check=full --tool=memcheck ./client -a 127.0.0.1 -p 8080 -s 0 -d 2098
[1] 20173
==20174== Memcheck, a memory error detector
==20174== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20174== Using Valgrind-3.13.0 and LlbVEX; rerun with -h for copyright info
==20174== Command: ./client -a 127.0.0.1 -p 8080 -s 0 -d 2098
==20174==
==20173== Memcheck, a memory error detector
==20173== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20173== Using Valgrind-3.13.0 and LlbVEX; rerun with -h for copyright info
==20173== Command: ./client -a 127.0.0.1 -p 8080 -s 4 -d 4
==20173==
[1593272122692684]Client(20174) connecting to 127.0.0.1:8080
[1593272122704720]Client(20174) connected and requesting a path from node 0 to 2098
[1593272122713095]Server's response to (20174): 0->7->145->2098, arrived in 0.006357 seconds.
==20174==
==20174== HEAP SUMMARY:
==20174==   in use at exit: 0 bytes in 0 blocks
==20174==   total heap usage: 1 allocs, 1 frees, 1,200,000 bytes allocated
==20174==
==20174== All heap blocks were freed -- no leaks are possible
==20174==
==20174== For counts of detected and suppressed errors, rerun with: -v
==20174== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
fatihselimyakar@ubuntu: ~/Desktop/SYS-FINAL-LIST$ [159327212277009]Client(20173) connecting to 127.0.0.1:8080
[1593272122787382]Client(20173) connected and requesting a path from node 4 to 4
[1593272122796966]Server's response to (20173): 4->144->367->4, arrived in 0.008174 seconds.
==20173==
==20173== HEAP SUMMARY:
==20173==   in use at exit: 0 bytes in 0 blocks
==20173==   total heap usage: 1 allocs, 1 frees, 1,200,000 bytes allocated
==20173==
==20173== All heap blocks were freed -- no leaks are possible
==20173==
==20173== For counts of detected and suppressed errors, rerun with: -v
==20173== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Output log file after the ctrl-c

```
Activities Text Editor Sat 18:37 output.txt
Open ~ Desktop/SYS-FINAL-LIST Save
[1593272091103846]Executing with parameters:
[1593272091116857]-t p2p-Gnutella08.txt
[1593272091119313]-p 8080
[1593272091120214]-o output.txt
[1593272091120661]-s 10
[1593272091124376]-x 20
[1593272091124853]-r 0
[1593272091132769>Loading graph...
[1593272091435831]Graph loaded in 0.298457 seconds with 6301 nodes and 20777 edges.
[1593272091456998]A pool of 10 threads has been created
[1593272091835473]Thread #0: Waiting for the connection.
[1593272091840034]Thread #2: Waiting for the connection.
[1593272091840110]Thread #7: Waiting for the connection.
[1593272091840136]Thread #1: Waiting for the connection.
[1593272091840158]Thread #5: Waiting for the connection.
[1593272091840178]Thread #6: Waiting for the connection.
[1593272091840197]Thread #8: Waiting for the connection.
[1593272091840217]Thread #4: Waiting for the connection.
[1593272091840236]Thread #3: Waiting for the connection.
[1593272091840256]Thread #9: Waiting for the connection.
[1593272096879430]A connection has been delegated to thread id #0, system load 0.000000%
[1593272096889742]Thread #0: searching database for a path from node 4 to node 4
[1593272096892160]Thread #0: no path in database, calculating 4->4
[1593272096901117]Thread #0: path calculated: 4->144->367->4
[1593272096902425]Thread #0: responding to client and adding path to database
[1593272122691558]A connection has been delegated to thread id #1, system load 0.000000%
[1593272122707459]Thread #1: searching database for a path from node 0 to node 2098
[1593272122707574]Thread #1: no path in database, calculating 0->2098
[1593272122708945]Thread #1: path calculated: 0->7->145->2098
[1593272122709016]Thread #1: responding to client and adding path to database
[1593272122770674]A connection has been delegated to thread id #2, system load 0.000000%
[1593272122789043]Thread #2: searching database for a path from node 4 to node 4
[1593272122789499]Thread #2: path found in database: 4->144->367->4
[1593272168881313]Termination signal(2) received, waiting for ongoing threads to complete.
[1593272168882645]Thread 3 is exiting.
[1593272168886778]Thread 9 is exiting.
[1593272168886856]Thread 1 is exiting.
[1593272168886898]Thread 2 is exiting.
[1593272168886962]Thread 6 is exiting.
[1593272168887003]Thread 0 is exiting.
[1593272168887283]Thread 5 is exiting.
[1593272168889475]Thread 7 is exiting.
[1593272168890681]Thread 4 is exiting.
[1593272168890823]Thread 8 is exiting.
[1593272168902364]Pool thread is exiting.
```

9 Notes

- The program started with 2 and 3 threads due to the extension rule up to 25% will not extend due to $2/4$ and $3/4$ being less than 1.
- There are limits to 999999 characters for the path when writing and reading socket.
- It creates an empty file with the name "running" to prevent a 2nd program from running. It prevents restarting because it cannot be deleted in an unexpected error (seg. Fault etc.). It is necessary to delete "running" to try again
- It has been tried as 5-25 threads with wiki-Talk.txt and soc-LiveJournal1.txt graphs on the given website.
- Nodes are accepted from 0 to node number-1. So node must have 0,1,2,3,4 nodes for size = 5.
- Due to the permissions on the computer, it may be necessary to run with sudo.
- After the sigint signal is thrown with "kill -2 pid", it should be waited for a while to run again. In this case, although the sockets are closed sometimes, the socket may generate creation error.
- A bonus part of 30 points has been implemented.