

# CSE-344 SYSTEM PROGRAMMING

## FINAL PROJECT

GÖKHAN HAS - 161044067

June 28, 2020

# CSE-344 SYSTEM PROGRAMMING FINAL PROJECT

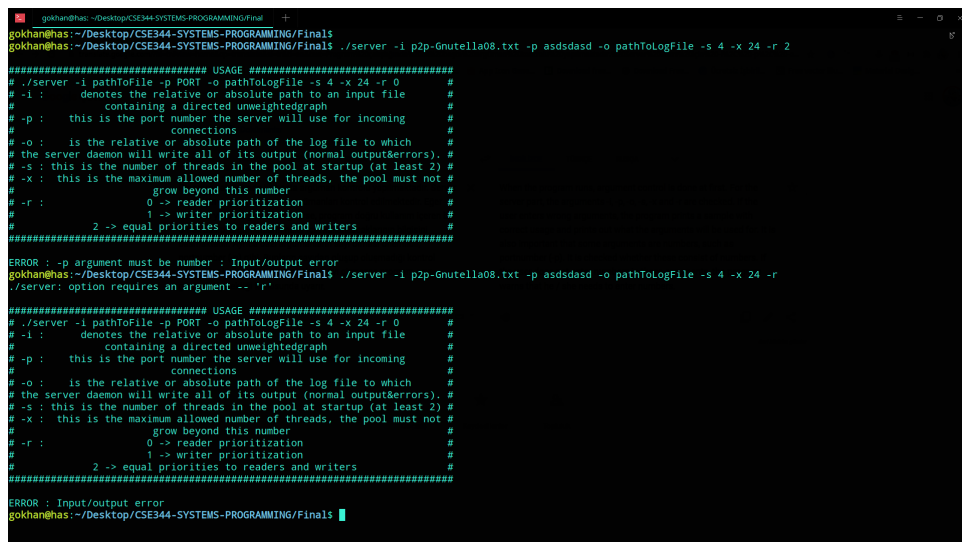
## 1. INTRODUCTION

The final project was a mixture of topics we learned throughout the semester. I also reinforced what I learned using everything I learned throughout the period. I solved the processes using daemon, using TCP sockets, making thread pool, transferring data between threads using pipe, and synchronization using mutexes and condition variables.

## 2. SERVER

The server part is located in "161044067\_final.c" file. As is known, it is the main part of the assignment. There are struct definitions for the thread pool at the beginning. Then, global variables used in the code are defined. The functions used are described here. The purpose of the functions is written in the code as a comment line. The purpose of the functions will be explained in the "used functions" section on the following pages.

When the program runs, argument control is done at first. For the server part, the arguments -i, -p, -o, -s, -x and -r are checked. If the user enters wrong arguments, the program prints a sample with correct usage and prints out what the arguments will be used for. It is also important that some arguments are numbers, such as portnumber (-p). It is checked whether these consist of numbers. If the user does not enter numbers in these arguments, the program warns that it needs to enter numbers.



```
gokhan@has: ~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./server -i p2p-Gnutella08.txt -p asdsdasd -o pathToLogFile -s 4 -x 24 -r 2
##### USAGE #####
./server -i pathToFile -p PORT -o pathToLogFile -s 4 -x 24 -r 0
# -i : denotes the relative or absolute path to an input file
#       containing a directed unweightedgraph
# -p : this is the port number the server will use for incoming
#       connections
# -o : is the relative or absolute path of the log file to which
#       the server daemon will write all of its output (normal output/errors).
# -s : this is the number of threads in the pool at startup (at least 2)
# -x : this is the maximum allowed number of threads, the pool must not
#       grow beyond this number
# -r : 0 -> reader prioritization
#       1 -> writer prioritization
#       2 -> equal priorities to readers and writers
#####
ERROR : -p argument must be number : Input/output error
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./server -i p2p-Gnutella08.txt -p asdsdasd -o pathToLogFile -s 4 -x 24 -r
./server: option requires an argument -- 'r'
##### USAGE #####
./server -i pathToFile -p PORT -o pathToLogFile -s 4 -x 24 -r 0
# -i : denotes the relative or absolute path to an input file
#       containing a directed unweightedgraph
# -p : this is the port number the server will use for incoming
#       connections
# -o : is the relative or absolute path of the log file to which
#       the server daemon will write all of its output (normal output/errors).
# -s : this is the number of threads in the pool at startup (at least 2)
# -x : this is the maximum allowed number of threads, the pool must not
#       grow beyond this number
# -r : 0 -> reader prioritization
#       1 -> writer prioritization
#       2 -> equal priorities to readers and writers
#####
ERROR : Input/output error
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$
```

Figure 1.1: Wrong Input Test

In the final project, SIGINT, ie CTRL C signal is requested to be caught. That's why I defined it at the beginning of the main using the sigaction structure and making the necessary assignments. It was necessary to define this structure. If it were done using the normal signal function, there would be a problem with the socket. The program would not close. So this structure was used and a signal\_handler function was written. This function assigns 1 to the variable SIGINT\_FLAG. The rest of the server has controls based on this flag.

After the argument checks are done, the daemon process is created. After that, the log file opens, if no log file is available. If a file with the same name is found before, it is added to the end. Because it is tried to open in "a +" mode.

After the above processes are completed, the graph is initialized. How many nodes are there, one-dimensional array with so many elements is created. Thus, it is possible to access that node in  $O(1)$  time. When adding an edge, the next nodes of the nodes in that index are used. Structure is defined in "graph.h" file. Below is a shaped explanation of the graph structure I have set up. It usually takes a short time to read the input file and load the graph. Sometimes it takes a long time when working with Valgrind.

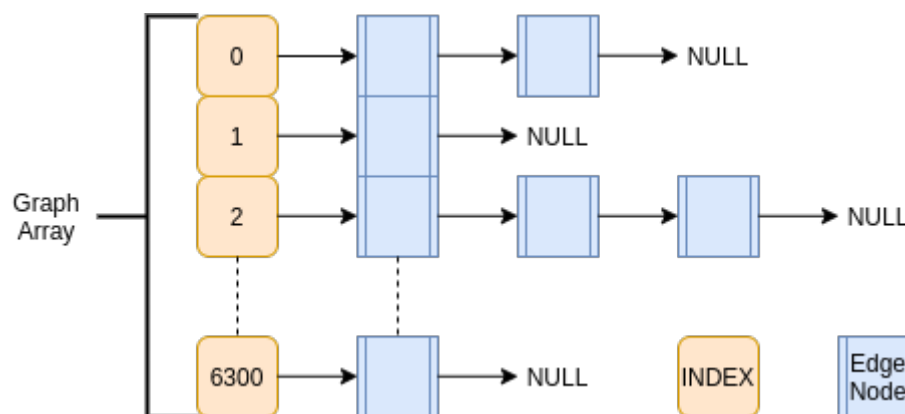


Figure 1.2: Graph Structure

Thread pool is created according to the min or maximum values entered by the user. A "ThreadPool\_t" structure contains first and last thread structures that point you to maxCapacity, ids array, conditionArr, mutexes, as well as two "OneThread" structures. The purpose is to make the add part in  $O(1)$  time in the function that performs the resize operation. This process is possible and effective since the last thread is held with "last". The "OneThread" structure contains the thread of id, isWorking, socket\_fd,

source, destination, pipe, pthread\_t and nextThread, which also holds the address of the next thread, "OneThread \*".

```
typedef struct _oneThread {
    int id;
    int isWorking;
    int socket_fd;
    int source;
    int destination;
    int pipe[2];
    pthread_t thread;
    struct _oneThread* nextThread;
} OneThread;

typedef struct _threadPool {
    int size;
    int maxCapacity;
    int* ids;
    void (*function) (void*);
    pthread_cond_t* conditionArr;
    pthread_mutex_t* mutexes;
    OneThread* firstThread;
    OneThread* lastThread;
} ThreadPool_t;
```

Figure 1.3: ThreadPool\_t Structure

At first the thread is created as many as the number -s argument. It is then done in a separate thread using the reinitializeThreadPool function. If Main says that new threads need to be loaded, this function works. Communication of these two functions was done using pipes. The function does not work until data is written to pipe. Since the last thread is kept in the thread pool, adding in  $O(1)$  time is done in this function (reinitializeThreadPool). Main continues to work.

Cache structure should have been very efficient. So I implemented it graph structure. I have a CacheEntry array. This actually holds the CacheEntry structure as much as the number of nodes. This includes structures of the cacheSize, first and CacheBlock \* data types. For example, the user wants to find the path from 1200 to 6200. First, by making cache [1200], I can reach which node from 1200 in  $O(1)$  time. Then, by doing cache [1200].nextBlock-last, I check if the value of 6200 is equal to this value. If it is equal, I return it by taking the int array in the CacheBlock structure. Thus, searching in cache is very fast. In the beginning, I assign values to -1. For example, you want to search from 1500 to 8000. But cache has not gone 1500 vertex anywhere. Since the value is -1, it is checked once.

```

typedef struct _cacheBlock {
    int last;
    int arraySize;
    int* path;
    struct _cacheBlock* next;
} CacheBlock;

typedef struct _cache {
    int cacheSize;
    int first;
    CacheBlock* nextBlock;
} CacheEntry;

```

Figure 1.4: Cache Structure

## 2.1. Used Function In Server Side

- void errorExit(char\* error) : Print error message and exit gracefully.
- void printUsage() : Print program usage if the user entered wrong input.
- int controlIfArgumentDigit(char\* argumentValue) : Same arguments must be digit, fx portnumber, control these arguments.
- int controlDuplicateServer() : The server should not run twice at the same time.
- size\_t getMaksimumNodeNumber(char\* fileName, int\* edgeNumber) : Returns the maximum number of nodes in the graph.
- int readFileAndInitializeGraph(char \*fileName, Graph\* graph, int nodeNumber, int edgeNumber) : The graph is initialized by reading the file line by line.
- void printStartMessageToLogFile(char\* iVal, char\* oVal) : When the server runs for the first time in the log file, the messages are printed. Only timestamp is placed in the last of these messages.
- int BFSalgorithm(Graph\* graph, int source, int destination, int\* visitedArr, int\* path, int\* pathIndex, int\* count, int maxEdge) : It is the function that calculates the classical BFS algorithm. With the help of the queue, bfs is calculated. At the level level, it is not always possible to go to the next level before the upper level ends. If there is an edge between nodes, the shortest is always found.

- `void getPath(Queue * queue, int* path, int destination, int maxNode, int kIndex, int count)` : According to the BFS algorithm, the path between the nodes must be found and saved in the cache or sent to the clients.
- `void* reinitializeThreadPool(void* argument)` : It is the thread function that changes the reputation of the thread pool. Adding to threadpool is done in this function.
- `void* response_threads(void* argument)` : It is the main function where threads will work. Incoming requests are calculated here, if any, search or add operation in cache.
- `void initializeThreadPool()` : This is the function that initializes threadpool initially.
- `void freeThreadPool()` : Resources used for threadpool are free.
- `OneThread* returnEmptyThread()` : Returns the thread structure not working.
- `int getRunningThreadCount()` : Returns the number of threads running.
- `void initializeBFSarrays(int* path, int* visitedArr)` : Prepares the variables required for BFS.
- `void printPathFromDataBase(CacheBlock* block)` : If path cache is found, this function is used to write to the log file.
- `void signal_handler(int sigNo)` : It is the signal handler function.
- `void create_server()` : Creating a socket in C programming language is done in this function.
- `void waitAllThreads()` : When CTRL C (SIGINT) arrives, threads running at that time must be finished.
- `void printConnectionMessage(int id)` : Print log file messages
- `void printSearchDatabase(int id, int src, int dest)`
- `void printNoDatabase(int id, int src, int dest)`
- `void printPathCalculated(int id, Queue* path_queue)`
- `void printRespondingMsg(int id)`

- void printPathNoPossible(int id, int src, int dest)
- void printPathFoundDatabase(int id, CacheBlock\* searched)

### 3. CLIENT

The client side is quite simple. Argument control is done first. It is checked whether the necessary arguments for the program are of the same type. If the user has entered the arguments incorrectly, the program will print the screen and display the correct use of the error. Then socket operations are made, sent to source and destination server side. And response is expected. The reply will be in the form of int array. And after the element at the end, -1 is assigned. Control has been done accordingly. If there is no path, -1 is returned directly. Then, the information message is printed on the client side and the program is terminated.

```

gokhan@has: ~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./client -a 127.0.0.1 -p 7070 -s 4 -d abc
##### USAGE #####
# ./client -a 127.0.0.1 -p PORT -s 768 -d 979 #
# -a : IP address of the machine running the server #
# -p : port number at which the server waits for connections #
# -s : source node of the requested path #
# -d : destination node of the requested path #
#####
ERROR : -d argument must be number : Input/output error
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./client -a 127.0.0.1 -p 7070 -s
./client: option requires an argument -- 's'
##### USAGE #####
# ./client -a 127.0.0.1 -p PORT -s 768 -d 979 #
# -a : IP address of the machine running the server #
# -p : port number at which the server waits for connections #
# -s : source node of the requested path #
# -d : destination node of the requested path #
#####
ERROR : Input/output error
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./client -a 127.0.0.1 -p 7070 -s 5 -d 5
Sat Jun 27 15:34:11 2020 : Client (14785) connecting to 127.0.0.1:7070
ERROR ! connection problem in client : Connection refused
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ SERVER IS NOT RUNNING NOW[]

```

Figure 1.5: Wrong Input Test In Client

#### 3.1 Used Functions In Client

- int controlIfArgumentDigit(char\* argumentValue) : Same arguments must be digit, fx portnumber, control these arguments.
- void errorExit(char\* error) : Print error message and exit gracefully.
- void printUsage() : Print program usage if the user entered wrong input.

## 4. OTHER USED FUNCTIONS IN HEADER FILES

### 4.1 graph.h

- `Graph* initializeGraph(int max)` : Initializes graph by node number.
- `void addEdge(Graph* graph, int source, int destination)` : This function works when you want to add an edge to the graph.
- `Graph* reinitializeGraph(Graph* oldGraph, int maxNumberNode)` : It is written to re-initialize the graph when necessary.
- `void freeGraph(Graph* _graph)` : The blocks separated by malloc for the graph structure are returned.
- `void printGraph(Graph* graph)` : It is written to understand whether the graph is properly read or not. It is not used in main.

### 4.2 queue.h

- `void initializeQueue(Queue* queue)` : Queue is initialized according to the initial values.
- `void push(Queue* queue, int element)` : Adding an element to the end of the queue. Since the last node is held, it is added in  $O(1)$  time.
- `int pop(Queue* queue)` : Extracts an element from the queue. And it returns that element. Since the first node is held, the element is removed in  $O(1)$  time.
- `void freeQueue(Queue* queue)` : Resources used for the queue are free.
- `void printPathQueue(Queue* queue, FILE* fileptr)` : Queue elements are printed in sequence.



### 4.3 cache.h

- void initializeCache(CacheEntry\* cacheEntryArr, int size) : Assigns initial values to all elements of the cache array.
- void addLast(CacheEntry\* cacheEntryArr, Queue\* pathx) : This function works when an element is added to the cache. Necessary information is taken from Queue \* as a parameter.
- void freeCache(CacheEntry\* cacheEntryArr) : All resources used by cache are free.
- CacheBlock\* searchCache(CacheEntry\* cacheEntryArr, int source, int destination) : It is used in the case of searching for a path in cache. Returns the data structure of type CacheBlock \*. Path is printed and send to from this structure.
- void printCacheBlock(CacheBlock\* block) : In the cache, it is written for control purposes to understand whether the elements are added properly or not.

### 5. IMPORTANT NOTES

- If you get a warning like the picture below while performing the valgrind test, the error will go away when you run the place I marked in the red box by adding it to the valgrind argument. Not all kinds of memory leak. Only errors are given.

```
./server -i p2p-Gnutella08.txt -p 7070 -o pathToLogFile -s 100 -x 150 -r 1
==2513== Memcheck, a memory error detector
==2513== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2513== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==2513== Command: ./server -i p2p-Gnutella08.txt -p 7070 -o pathToLogFile -s 100 -x 150 -r 1
==2513==
==2513== HEAP SUMMARY:
==2513==   in use at exit: 0 bytes in 0 blocks
==2513==   total heap usage: 1 allocs, 1 frees, 952 bytes allocated
==2513==
==2513== All heap blocks were freed -- no leaks are possible
==2513==
==2513== For counts of detected and suppressed errors, rerun with: -v
==2513== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
gwen@has:~/Desktop/CE344-SYSTEMS-PROGRAMMING/Final9_==12314== Warning: client switching stacks?  SP change: 0x5e4ef
0 --> 0x5b399f0
==2514== Thread 1
==2514==   to suppress, use: --max-stackframe=3200256 or greater
==2514== Invalid write of size 8
==2514==   at 0x403CA: response_threads (161044067.final.c:739)
==2514==   by 0x4068FA2: start_thread (pthread_create.c:486)
==2514==   by 0x49784CE: clone (clone.S:95)
==2514== Address 0x5e4e6e8 is on thread 3's stack
==2514==   in frame #0, created by response_threads (161044067.final.c:732)
==2514==
==2514== Invalid read of size 8
==2514==   at 0x403E80: __tz_convert (tzset.c:587)
==2514==   by 0x40372C: time (time.c:27)
==2514==   by 0x403CDF: response_threads (161044067.final.c:740)
==2514==   by 0x4068FA2: start_thread (pthread_create.c:486)
==2514==   by 0x49784CE: clone (clone.S:95)
==2514== Address 0x5e4e6e8 is on thread 3's stack
==2514==   in frame #2, created by response_threads (161044067.final.c:732)
==2514==
==2514== Invalid read of size 8
==2514==   at 0x4036D43: __offtime (offtime.c:34)
==2514==   by 0x403E7F: __tz_convert (tzset.c:610)
==2514==   by 0x40372C: time (time.c:27)
==2514==   by 0x403CDF: response_threads (161044067.final.c:740)
==2514==   by 0x4068FA2: start_thread (pthread_create.c:486)
==2514==   by 0x49784CE: clone (clone.S:95)
==2514== Address 0x5e4e6e8 is on thread 3's stack
==2514==   in frame #3, created by response_threads (161044067.final.c:732)
==2514==
==2514== Warning: client switching stacks?  SP change: 0x6449ef0 --> 0x6b3b9f0
==2514==   to suppress, use: --max-stackframe=3200256 or greater
==2514== Warning: client switching stacks?  SP change: 0x6447ef0 --> 0x33a9f0
==2514==   to suppress, use: --max-stackframe=3200256 or greater
==2514== further instances of this message will not be shown.
clear
```

Figure 1.6: You can see screenshot in directory

```
gokhan@has: ~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ make
gcc -g 161044067_final.c graph.c queue.c cache.c graph.h queue.h cache.h -o server -lpthread -Wall
gcc -g client.c -o client -Wall
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all --track-origins=yes --max-stackframe=3200256 ./server -i p2p-Gnutella08.txt -p 7070 -o pathTolLogFile -s 100 -x 150 -r 1
==17090== Memcheck, a memory error detector
==17090== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==17090== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==17090== Command: ./server -i p2p-Gnutella08.txt -p 7070 -o pathTolLogFile -s 100 -x 150 -r 1
==17090==
==17090== HEAP SUMMARY:
==17090==   in use at exit: 0 bytes in 0 blocks
==17090==   total heap usage: 1 allocs, 1 frees, 552 bytes allocated
==17090==
==17090== All heap blocks were freed -- no leaks are possible
==17090==
==17090== For counts of detected and suppressed errors, rerun with: -v
==17090== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ==17091==
==17091== HEAP SUMMARY:
==17091==   in use at exit: 0 bytes in 0 blocks
==17091==   total heap usage: 43,479 allocs, 43,479 frees, 2,199,099 bytes allocated
==17091==
==17091== All heap blocks were freed -- no leaks are possible
==17091==
==17091== For counts of detected and suppressed errors, rerun with: -v
==17091== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$
```

Figure 1.7: Memory Leak Test In Server

```
gokhan@has: ~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./client -a 127.0.0.1 -p 7070 -s 0 -d 10
Sat Jun 27 16:47:36 2020 : Client (18267) connecting to 127.0.0.1:7070
Sat Jun 27 16:47:36 2020 : Client (18267) connected and requesting a path from node 0 to 10
Sat Jun 27 16:47:36 2020 : Server's response to (18267): 0->10, arrived in 0.00 seconds
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./client -a 127.0.0.1 -p 7070 -s 0 -d 150
Sat Jun 27 16:47:43 2020 : Client (18270) connecting to 127.0.0.1:7070
Sat Jun 27 16:47:43 2020 : Client (18270) connected and requesting a path from node 0 to 150
Sat Jun 27 16:47:43 2020 : Server's response to (18270): 0->3->1287->331->335->141->150, arrived in 0.00 seconds
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./client -a 127.0.0.1 -p 7070 -s 0 -d 241
Sat Jun 27 16:47:55 2020 : Client (18276) connecting to 127.0.0.1:7070
Sat Jun 27 16:47:55 2020 : Client (18276) connected and requesting a path from node 0 to 241
Sat Jun 27 16:47:55 2020 : Server's response to (18276): 0->5->127->175->3066->926->241, arrived in 0.00 seconds
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./client -a 127.0.0.1 -p 7070 -s -150 -d 241

##### USAGE #####
# ./client -a 127.0.0.1 -p PORT -s 768 -d 879 #
# -a : IP address of the machine running the server #
# -p : port number at which the server waits for connections #
# -s : source node of the requested path #
# -d : destination node of the requested path #
#####
ERROR : -s argument must be number : Input/output error
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$ ./client -a 127.0.0.1 -p 7070 -s 145 -d 387
Sat Jun 27 16:50:06 2020 : Client (18376) connecting to 127.0.0.1:7070
Sat Jun 27 16:50:06 2020 : Client (18376) connected and requesting a path from node 145 to 387
Sat Jun 27 16:50:06 2020 : Server's response to (18376): 145->149->250->1389->1390->387, arrived in 0.00 seconds
gokhan@has:~/Desktop/CSE344-SYSTEMS-PROGRAMMING/Final$
```

Figure 1.8: Sample Client Examples