CSE344
Systems Programming Course
Final Project Report

Furkan Özev
161044036

June 28^{th}, 2020

1 Main Idea

In this project, there are 2 programs. A threaded server and a client. The server will load a graph from a text file, and use a dynamic pool of threads to handle incoming connections. The clients will connect to the server and request a path between two arbitrary nodes, and the server will provide this service.

Also server process will be a daemon. So, It should not possible to start 2 instances of the server process. Server process has no controlling terminal. And its inherited open files will close.

After the server process loads the graph from the input file into memory,The server process will communicate with the client processes through stream/TCP socket based connections. 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 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. The pool of threads will be created and initialized at the daemon's startup, and each thread will execute in an endless loop, first waiting for a connection to be handed to them by the server, then handling it, and then waiting again, and so on.

Once a connection is established, the client will simply send two indexes i1 and i2, as two non negative integers, representing each one of the nodes of the graph, to the server, and wait for the server to reply with the path from i1 to i2. If the requested path from i1 to i2 has been already calculated during a past request, the thread handling this request should first check a cache, containing past calculations, and if the requested path is present in it, the thread should simply use it to respond to the client, instead of recalculating it. Otherwise, it

will use breadth-first search to find a path from i1 to i2. If it is not present in the cache then it must inevitably calculate it with breadth-first search, respond to the client, and add the newly calculated path into the data structure, so as to accelerate future requests. If no path is possible between the requested nodes, then the server will respond accordingly.

The cache will be common to all threads of the pool and can be thought of as a "database". It must not contain duplicate entries, and it must support fast access/search. This database includes multiple and serious synchronization. It used readers/writers paradigm, by prioritizing writers to provide synchronization. Readers are the threads attempting to search the data structure to find out whether a certain path is in it or not. Writers are the threads that have calculated a certain path and now want to add it into the data structure.

If the server receives the SIGINT signal it will wait for its ongoing threads to complete, return all allocated resources, and then shutdown gracefully.

Thread pool would be "dynamic". This means it will not have a fixed number of threads, instead the number of threads will increase depending on how busy the server is. Specifically, every time that the load of the server reaches 75%, i.e. 75% of the threads become busy, then the pool size will be enlarged by 25%.

The client is simple. Given the server's address and port number, it will simply request a path from node i1 to i2 and wait for a response, while printing its output on the terminal.

2 Server Code Flow

2.1 Main Thread Flow

To measures against double instantiation, When the program starts, it creates a temporary file. If a second server tries to open this file, it will get an error. Thus, double instantiation will be prevented.

```
...

int fd = open("/tmp/daemonLock", O_CREAT | O_EXCL);
if(fd == -1)
{
  fprintf(stderr, "\nIt should not be possible to start 2 instances
     of the server process\n\n");
  exit(EXIT_FAILURE);
}
```

After the parameters are read, the values are checked.Number of port mustn't be negative, number of threads can be at least 2, number of threads must not bigger than maximum allowed number of threads.

```
  ...
```

```
2
3  if(port < 0)
4  {
5    fprintf(stderr, "-p(number of port) mustn't be negative.\n");
6    unlink("/tmp/daemonLock");
7    exit(EXIT_FAILURE);
8  }
9
10 if(threadNumber < 2)
11 {
12   fprintf(stderr, "-s(number of threads) can be at least 2.\n");
13   unlink("/tmp/daemonLock");
14   exit(EXIT_FAILURE);
15 }
16
17 if(threadNumber > maxThread)
18 {
19   fprintf(stderr, "-s(number of threads) must not bigger than -x(
       maximum allowed number of threads)\n");
20   unlink("/tmp/daemonLock");
21   exit(EXIT_FAILURE);
22 }
23
24   ...
```

The becomeDaemon() function is called. Thus, process's terminal connection is canceled and inherited open files are closed. Then the input files are opened and the hands of the SIGINT signal are changed.

```
1  ...
2
3  becomeDaemon();
4  fd1 = fopen(filePath, "r");
5  fd2 = fopen(logPath, "w+");
6  signal(SIGINT, sigHandler);
7
8  ...
```

2.1.1 becomeDaemon

It was made by looking at the 770 th page of the book. Server process has no controlling terminal. And its inherited open files will close

```
1
2  ...
3  switch(fork())
4  {
5    case -1:
6      return -1;
7    case 0:
8      break;
9    default:
10     exit(EXIT_SUCCESS);
11 }
12
13 if(setsid() == -1)
```

```
  return -1;

switch(fork())
{
  case -1:
    return -1;
  case 0:
    break;
  default:
    exit(EXIT_SUCCESS);
}

umask(0);

maxfd = sysconf(_SC_OPEN_MAX);
if(maxfd == -1)
  maxfd = 8192;

// Close all of its inherited open files
for(fd = 0; fd < maxfd; fd++)
  close(fd);

...
```

It will initialize graph. The server process will load the graph to memory from its input file. Then, it will initialize catch. Then pool threads and dynamic coordinate thread are created.

```
...
initGraph(&graph1);
loadGraph(&graph1, fd1);
initCatch(&catch1, graph1.a, graph1.b, graph1.c);

// Create pool threads. Thread take number.
for(i = 0; i < threadNumber; i++)
    pthread_create(&(thread_id[i]), NULL, &thread_function,
    indexThread[i]);

// Create dynamic coordinate thread
pthread_create(&thread_pool, NULL, &thread_coord_pool, NULL);

...
```

Necessary socket operations are made to communicate between the server and clients.

```
...
struct sockaddr_in serverAddr;
int socket2;
int addrlen = sizeof(serverAddr);
int opt1 = 1;

// Create socket file descriptor
socket1 = socket(AF_INET, SOCK_STREAM, 0);

// Forcefully attaching socket to the port
setsockopt(socket1, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt1,
     sizeof(opt1));
```

```
12
13 serverAddr.sin_family = AF_INET;;
14 serverAddr.sin_addr.s_addr = INADDR_ANY;
15 serverAddr.sin_port = htons(port);
16
17 // Binds the socket to the address and port number
18 bind(socket1, (struct sockaddr *)&serverAddr, sizeof(serverAddr))
19
20 //  It waits for the client to approach the server to make a
       connection
21 listen(socket1, 4096)
22 ...
```

Main Thread: Endless loop, it just check sigint signal arrives. First it accept a connection. Then check system load capacity. If it has reached, it waits for the pool coordinate thread to create new threads. If it reaches the maximum number of threads, the threads are expected to finish. Then it give socket to any thread. Any thread is expected to receive the sent socket. The same process repeats. Mutexes are used in this section as there are critical regions.

```
1  ...
2  while(flagint == 0)
3  {
4      socket2 = accept(socket1, (struct sockaddr *)&serverAddr, (
       socklen_t*)&addrlen);
5      // Lock pool coordinate thread
6      pthread_mutex_lock(&mutex3);
7      // Lock pool threads
8    pthread_mutex_lock(&mutex1);
9
10   // Connection counter increased by 1
11   pcount2 += 1;
12
13   // Checks whether the capacity has reached 75%.
14   // If it has reached, it waits for the pool coordinate thread to
       create new threads.
15   while(capacity >= 75 && threadNumber != maxThread)
16   {
17       pthread_cond_broadcast(&pempty);
18     pthread_cond_wait(&pfull, &mutex3);
19   }
20   pthread_cond_broadcast(&pempty);
21   pthread_mutex_unlock(&mutex3);
22
23   // If it reaches the maximum number of threads, the threads are
       expected to finish.
24   while(pcount2 > maxThread || pcount3 >= threadNumber))
25       pthread_cond_wait(&cempty, &mutex1);
26
27   // Enqueue the provided socket to queue, And increase counters
28   enqueue(&ports, socket2);
29   pcount += 1;
30   pcount3 += 1;
31
32   pthread_cond_broadcast(&cfull);
33   pthread_mutex_unlock(&mutex1);
34
```

```
  // It is ensured that the connection provided by the main thread
    is forwarded to any thread.
  pthread_mutex_lock(&tmutex);
  while(tcount == 1)
    pthread_cond_wait(&tempty, &tmutex);
  tcount = 1;
  pthread_cond_broadcast(&tfull);
  pthread_mutex_unlock(&tmutex);
}
...
```

After sigint signal, it will join threads, close socket, close files, unlink daemon file, free all allocated memory.

```
...
// Join threads
for(i = 0; i < threadNumber; i++)
    pthread_join(thread_id[i], NULL);

pthread_join(thread_pool, NULL);

// Close files and sockets
close(socket1);
fclose(fd1);
fclose(fd2);
close(fd);

// Deallocate all memory
freeAll();

// unlink temprory file
unlink("/tmp/daemonLock");
...
```

2.2 Pool Threads Flow

Endless loop, it just check sigint signal arrives. It expects a connection to be provided by the main thread. When the main thread provides connection, any thread wakes up. It takes the socket and thus gets the nodes from the client. The message type of the client is "node1-node2". Parse this message. It informs the main thread that it has received the connection.

```
...
claf = 0;
while(flagint == 0)
{
  pthread_mutex_lock(&mutex1);
  // If any thread completes an loop
  if(cflag == 1)
  {
    // Decrase counters
    pcount3 -= 1;
    pcount2 -= 1;
    pthread_cond_signal(&cempty);
  }
  else
```

```
    cflag = 1;
  // Waits to be awakened by the main thread.
  while(flagint == 0 && pcount == 0)
    pthread_cond_wait(&cfull, &mutex1);

  // Take socket from queue
  temp = dequeue(&ports);
  pcount -= 1;

  // Indicates that the connection was received by any thread.
  pthread_mutex_lock(&tmutex);
  while(flagint == 0 && tcount == 0)
    pthread_cond_wait(&tfull, &tmutex);
  tcount = 0;
  pthread_cond_broadcast(&tempty);
  pthread_mutex_unlock(&tmutex);

  pthread_cond_broadcast(&cempty);
  pthread_mutex_unlock(&mutex1);

  // Parse message, then get nodes.
  for(i = 0; buffer[i] != '-'; i++);
  strncpy(buffer4, buffer, i);
  buffer4[i] = '\0';
  node1 = atoi(buffer4);
  strncpy(buffer4, &(buffer[i+1]), strlen(buffer)-i);
  buffer4[strlen(buffer)-i] = '\0';
  node2 = atoi(buffer4);
...
```

In cache access, the reader-writer paradigm is used. Reader paradigm: First search in cache, if path does not exist in cache, it will breadth-first search in graph. Writer paradigm: If it finds path with bfs, it will send path to client and write this path to cache. If path exists in cache, it will send path to client. If path is not found, it reports this to the client.

```
...
/ Readers are the threads attempting to search the data structure
    to find out whether a certain path is in it or not
pthread_mutex_lock(&mutex2);
while((AW + WW) > 0)
{
  WR++;
  pthread_cond_wait(&okToRead, &mutex2);
  WR--;
}
AR++;
pthread_mutex_unlock(&mutex2);

sprintf(logbuf, "Thread #%d: searching database for a path from
    node %d to node %d", x, node1, node2);
printLog(logbuf);

res = 0;
// Determine indexs, It is checked whether the path is in catch or
    not.
if(node1 >= catch1.sizea || node2 >= catch1.sizeb)
```

```
  index = -2;
else
{
  index = -1;
  nodetemp = catch1.index[node1]->front;
  while(nodetemp != NULL)
  {
    if(nodetemp->item == node2)
    {
      index = nodetemp->index;
      break;
    }
    nodetemp = nodetemp->next;
  }
}

// If index is negative, path does not exist in cache
// If not, it is the index where the path is located.
if(index != -1 && index != -2)
{
  // Converts path to string
  strcpy(buffer2, " ");
  strcpy(buffer3, " ");
  nodetemp = catch1.paths[index]->front;
  i = 0;
  while(nodetemp != NULL)
  {
    sprintf(buffer3, "%d", nodetemp->item);
    if(i == 0)
      strcpy(buffer2, buffer3);
    else if(nodetemp->next != NULL)
    {
      strcat(buffer2, "->");
      strcat(buffer2, buffer3);
    }
    else
    {
      strcat(buffer2, "->");
      strcat(buffer2, buffer3);
    }

    nodetemp = nodetemp->next;
    i += 1;
  }

  // The path is sent to the client.
  pthread_mutex_lock(&sendmutex);
  sprintf(logbuf, "Thread #%d: path found in database: %s", x,
    buffer2);
  printLog(logbuf);
  send(temp, buffer2, strlen(buffer2), 0);
  close(temp);
  pthread_mutex_unlock(&sendmutex);
}
else
{
  sprintf(logbuf, "Thread #%d: no path in database, calculating %d
```

```
    ->%d ", x, node1, node2);
  printLog(logbuf);
}

// Reading process completed.
// It awakens the waiting threads.
pthread_mutex_lock(&mutex2);
AR--;
if(AR == 0 && WW > 0)
  pthread_cond_signal(&okToWrite);
pthread_mutex_unlock(&mutex2);

// Search path with bfs algroithm
if(index == -1)
{
  // If bfs return -1, there is no path
  // If not, there is a path and it will fill resArr array
  res = bfs(&graph1, node1, node2, resArr);

  // Writers are the threads that have calculated a certain path
    and now want to add it into the data structure
  pthread_mutex_lock(&mutex2);
  while((AW + AR) > 0)
  {
    WW++;
    pthread_cond_wait(&okToWrite, &mutex2);
    WW--;
  }
  AW++;
  pthread_mutex_unlock(&mutex2);

  if(res != -1)
  {
    // Converts path to string
    strcpy(buffer2, " ");
    strcpy(buffer3, " ");
    for(i = 0; i <= res; i++)
    {
      sprintf(buffer3, "%d", resArr[i]);
      if(i == 0)
        strcpy(buffer2, buffer3);
      else if(i != res)
      {
        strcat(buffer2, "->");
        strcat(buffer2, buffer3);
      }
      else
      {
        strcat(buffer2, "->");
        strcat(buffer2, buffer3);
      }
    }

    // The path is sent to the client.
    pthread_mutex_lock(&sendmutex);
    sprintf(logbuf, "Thread #%d: path calculated: %s", x, buffer2);
    printLog(logbuf);
```

```
    send(temp, buffer2, strlen(buffer2), 0);
    close(temp);
    pthread_mutex_unlock(&sendmutex);

    // Path found in BFS is added to cache
    enqueue2(catch1.index[node1], node2, catch1.count);
    catch1.count += 1;
    if(catch1.count == 1)
      catch1.paths = (Queue**) calloc(catch1.count, sizeof(Queue*))
    ;
    else
      catch1.paths = (Queue**) realloc(catch1.paths, catch1.count *
     sizeof(Queue*));

    catch1.paths[catch1.count - 1] = (Queue*) malloc(sizeof(Queue))
    ;
    catch1.paths[catch1.count - 1]->front = NULL;
    catch1.paths[catch1.count - 1]->rear = NULL;

    for(i = 0; i <= res; i++)
      enqueue(catch1.paths[catch1.count - 1], resArr[i]);
  }

  // Writting process completed.
  // It awakens the waiting threads.
  pthread_mutex_lock(&mutex2);
  AW--;
  if(WW > 0)
    pthread_cond_signal(&okToWrite);
  else if (WR > 0)
    pthread_cond_broadcast(&okToRead);
  pthread_mutex_unlock(&mutex2);
}

// Path not found with bfs or in cache
if(res == -1 || index == -2)
{
  pthread_mutex_lock(&sendmutex);
  sprintf(logbuf, "Thread #%d: path not possible from node %d to %d
     ", x, node1, node2);
  printLog(logbuf);
  send(temp, "N", 1, 0);
  close(temp);
  pthread_mutex_unlock(&sendmutex);
}

...
```

2.3 Pool Coordinate Thread Flow

This thread coordinate thread pool. When systeam load is reached 75%, it will expand pool with creating new threads. Checks whether the capacity has reached 75%. If it has reached, it will start for the extend pool, If not, it will wait.
The pool size will be enlarged by 25%, so it will calculate new size.

If 25% excess of the current thread is still equal to it and 1 excess is possible, 1 is increased.
If 25% excess of the current thread is not equal to it and pool will enlarged with new thread number.
Otherwise, Thread has reached the maximum number.
It will create new thread for enlarge pool and it will print new system load and thread number in pool

```
...
//Endless loop, it just check sigint signal arrives.
while(flagint == 0)
{
    // Critical section between this thread and main thread
  pthread_mutex_lock(&mutex3);
  while(flagint == 0 && capacity < 75 && threadNumber != maxThread)
    pthread_cond_wait(&pempty, &mutex3);
  if(flagint != 0)
  {
    pthread_cond_broadcast(&pfull);
    pthread_mutex_unlock(&mutex3);
    break;
  }

  temp2 = threadNumber;
  capacity = ((double)pcount2) / ((double)threadNumber) * 100;

  // the pool size will be enlarged by 25%
  temp = threadNumber;
  newamount = round(threadNumber * 1.25);

  // If 25% excess of the current thread is still equal to it and 1
     excess is possible, 1 is increased.
  if(threadNumber == newamount && (newamount + 1) <= maxThread)
    threadNumber = newamount + 1;
  // If 25% excess of the current thread is not equal to it and
    pool will enlarged with new thread number.
  else if(threadNumber != newamount && newamount <= maxThread)
    threadNumber = newamount;
  // Otherwise, Thread has reached the maximum number.
  else
    threadNumber = maxThread;

  // Create new thread for enlarge pool.
  if(temp != threadNumber)
  {
    thread_id = (pthread_t*) realloc(thread_id, threadNumber *
    sizeof(pthread_t));
    indexThread = (int**) realloc(indexThread, threadNumber *
    sizeof(int*));

    for( ; temp < threadNumber; temp++)
    {
      indexThread[temp] = malloc(sizeof(int));
      *(indexThread[temp]) = temp;
      if(pthread_create(&(thread_id[temp]), NULL, &thread_function,
     indexThread[temp]) != 0)
```

```
      {
        sprintf(logbuf, "errno = %d: %s pthread_create\n", errno,
    strerror(errno));
        printLog(logbuf);
        unlink("/tmp/daemonLock");
        exit(EXIT_FAILURE);
      }
    }
  }

  // Print new system load and thread number in pool
  if(temp2 != threadNumber)
  {
    sprintf(logbuf, "System load %lf%%, pool extended to %d threads
    ", capacity, threadNumber);
    printLog(logbuf);
  }

  pthread_cond_broadcast(&pfull);
  pthread_mutex_unlock(&mutex3);
}
return NULL;
...
```

2.4 Graph

For graph, Adjacency list (queue) is kept in queue structure. The Load function also keeps the maximum number of nodes while reading the file. Therefore, these numbers are given to the graph at the beginning. When creating the Graph list, it determines its size based on this number amounts. This saves memory as much as possible.

```
graph->size = graph->a;
graph->adjList = (Queue**) calloc(graph->size, sizeof(Queue*));
for(i = 0; i < graph->size; i++)
{
  graph->adjList[i] = (Queue*) malloc(sizeof(Queue));
  graph->adjList[i]->front = NULL;
  graph->adjList[i]->rear = NULL;
}
graph->nodelist = (int *) calloc(graph->b, sizeof(int));
```

Then, the elements are added to the graph in order. While doing this, it checks whether the edge has been added before.

```
void addEdge(Graph* graph, int x, int y)
{
  if(existInList(graph->adjList[x], y) == -1)
  {
    enqueue(graph->adjList[x], y);
    graph->edges += 1;
  }

  graph->nodelist[x] = 1;
  graph->nodelist[y] = 1;
}
```

2.5 Cache

The Load function also keeps the maximum number of nodes while reading the file. Therefore, these numbers are given to the cache at the beginning. When creating the cache list, it determines its size based on this number amounts. This saves memory as much as possible.

```
void initCatch(Catch* catch, int a, int b, int c)
{
  int i;
  catch->sizea = a;
  catch->sizeb = b;
  catch->sizec = c;
  catch->count = 0;

  catch1.index = (Queue**) calloc(catch->sizea, sizeof(Queue*));
  for(i = 0; i < catch->sizea; i++)
  {
    catch->index[i] = (Queue*) malloc(sizeof(Queue));
    catch->index[i]->front = NULL;
    catch->index[i]->rear = NULL;
  }
  catch->paths = NULL;
}
```

The previously calculated paths are kept in cache. Paths are kept in a queue. The indexes of the paths are kept in another 2-dimensional queue. When a path is queried, first the index is found from the 2D queue and then the pathe is accessed directly.

```
// Search index in 2D queue
// For example 25->150
// node1 = 25, node2 = 150
index = -1;
nodetemp = catch1.index[node1]->front;
while(nodetemp != NULL)
{
  if(nodetemp->item == node2)
  {
    index = nodetemp->index;
    break;
  }
  nodetemp = nodetemp->next;
}

// If index is negative, path does not exist in cache
// If not, it is the index where the path is located.
if(index != -1)
{
    // Paths in nodetemp. So it can be directly accessed.
    nodetemp = catch1.paths[index]->front;
}
```

2.6 Signal Handler and Exit

Replaces the value of the global variable (flagint) with 1. Therefore, the threads will understand that a sigint signal has arrived and will end after it have finished it job. It must wait for its ongoing threads to complete. So it will wakeup ongoing threads. Thus deadlock is prevented.

```
if(sig == SIGINT)
{
  sprintf(logbuf, "Termination signal received, waiting for ongoing
     threads to complete.");
  printLog(logbuf);
  // Replaces the value of the global variable with 1.
  flagint = 1;
  if(socket1 != -1)
  {
    shutdown(socket1, SHUT_RD);
    close(socket1);
  }
  socket1 = -2;

  // It will wait for its ongoing threads to complete. So it will
    wakeup ongoing threads.
  pthread_cond_broadcast(&cfull);
  pthread_cond_broadcast(&pfull);
  pthread_cond_broadcast(&tfull);
    pthread_cond_broadcast(&cempty);
  pthread_cond_broadcast(&pempty);
  pthread_cond_broadcast(&tempty);
}
```

2.7 Logging

Threads directly call this function for printing. To prevent conflict in the file, it is locked before printing, after the process is completed, mutex unlock is done. Thus, 1 printing will be done at the same time. Necessary actions are taken for timestamp. The message is printed on the file.

```
void printLog(char* str)
{
  pthread_mutex_lock(&printmutex);
  time_t ltime;
  struct tm lresult;
  char stime[32];
  size_t n;

  ltime = time(NULL);
  localtime_r(&ltime, &lresult);
  asctime_r(&lresult, stime);
  n = strlen(stime);
  if(n && stime[n-1] == '\n') stime[n-1] = '\0';
  fprintf(fd2, "[%s] %s\n", stime, str);
  pthread_mutex_unlock(&printmutex);
}

```

```
\* Like that:
[Sun Jun 28 15:28:37 2020] Executing with parameters:
[Sun Jun 28 15:28:37 2020] -i p2p-Gnutella08.txt
[Sun Jun 28 15:28:37 2020] -p 45
*/
```

3 Client Code Flow

After the parameters are read, the values are checked.Number of port mustn't be negative, node1 and node2 mustn't negative.

```
  ...

// Check values
if(port < 0)
{
  printf("-p(number of port) must be positive.\n");
  exit(EXIT_FAILURE);
}

if(node1 < 0)
{
  printf("-s(source node of the requested path) must not negative.\
    n");
  exit(EXIT_FAILURE);
}
if(node2 < 0)
{
  printf("-d(destination node of the requested path) must not
    negative.\n");
  exit(EXIT_FAILURE);
}

  ...
```

Necessary socket operations are made to communicate between the server and clients.

```
...
// Create socket file descriptor
if((socket1 = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
  fprintf(stderr, "\nerrno = %d: %s Socket creation error\n\n",
    errno, strerror(errno));
  exit(EXIT_FAILURE);
}

serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port);

// Convert IPv4 and IPv6 addresses from text to binary form
if(inet_pton(AF_INET, ip, &serverAddr.sin_addr)<=0)
{
  fprintf(stderr, "\nerrno = %d: %s Invalid address/ Address not
    supported\n\n", errno, strerror(errno));
  exit(EXIT_FAILURE);
```

```
}

if(connect(socket1, (struct sockaddr *)&serverAddr, sizeof(
    serverAddr)) < 0)
{
  fprintf(stderr, "\nerrno = %d: %s Connection Failed \n\n", errno,
     strerror(errno));
  exit(EXIT_FAILURE);
}

printf("Client (%d) connected and requesting a path from node %d to
    %d\n", pid, node1, node2);
// Send nodes to server
send(socket1, bufwrite, strlen(bufwrite), 0);

// Calculate response time
gettimeofday(&start, NULL);
// Take result from server
size = read(socket1, bufread, SIZE);
gettimeofday(&end, NULL);
long seconds = (end.tv_sec - start.tv_sec);
long micros = ((seconds * 1000000) + end.tv_usec) - (start.tv_usec)
    ;
double diff_t = (double) micros / 1000000;

// Server response
if(bufread[0] == 'N' || size == 0)
  printf("Server's response to (%d) : NO PATH, arrived in %
    lfseconds, shutting down\n", pid, diff_t);
else
  printf("Server's response to (%d) : %s, arrived in %lfseconds.\n"
    , pid, bufread, diff_t);

close(socket1);
...
```

4 Compile & Run

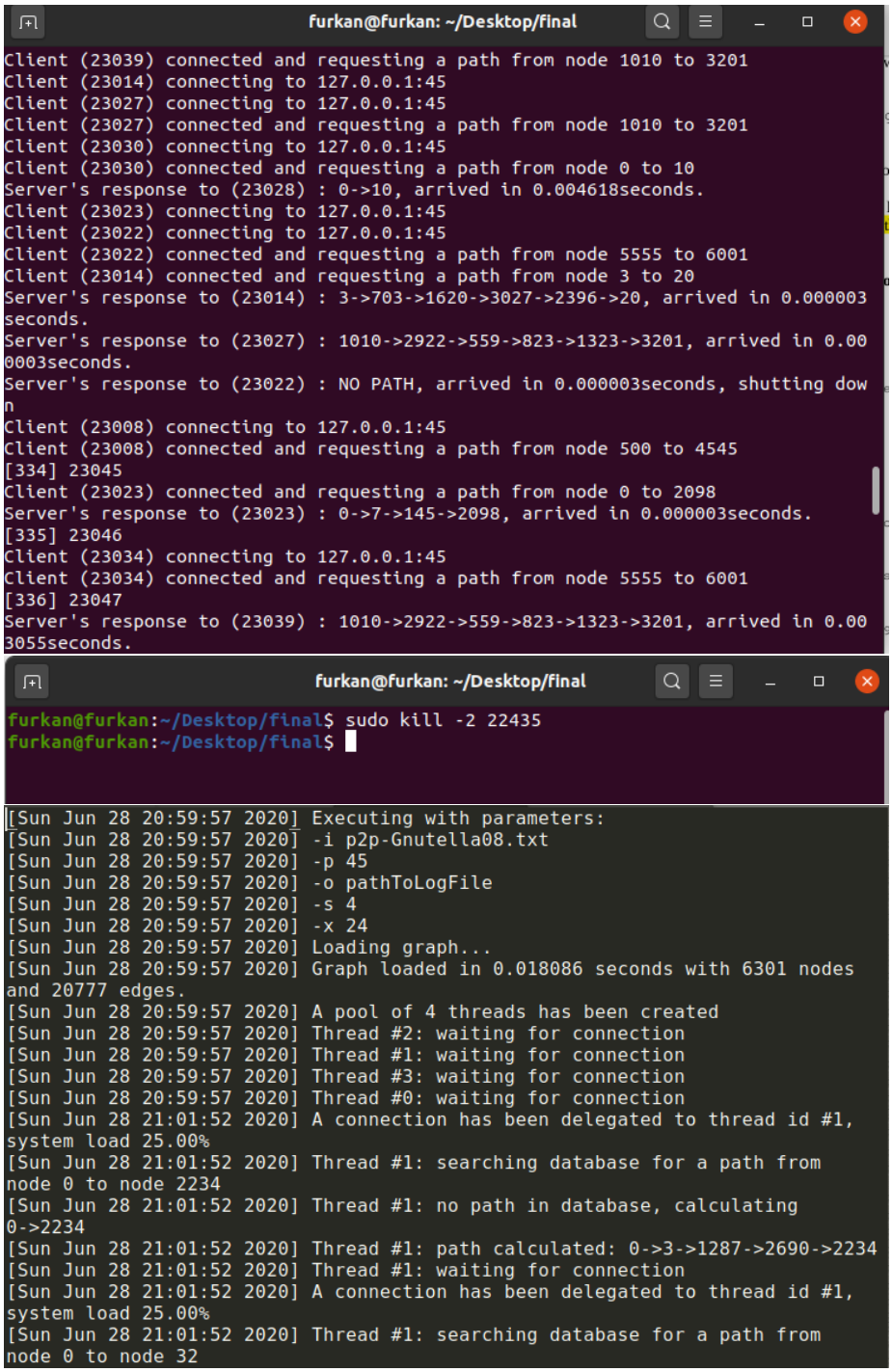
```
furkan@furkan:~/Desktop/final$ make
gcc -Wall -o server server.c -pthread -lm
gcc -Wall -o client client.c -pthread -lm
furkan@furkan:~/Desktop/final$ sudo ./server -i p2p-Gnutella08.txt -p 45 -o path
ToLogFile -s 4 -x 24
furkan@furkan:~/Desktop/final$ sudo ./server -i p2p-Gnutella08.txt -p 45 -o path
ToLogFile -s 4 -x 24

It should not be possible to start 2 instances of the server process

furkan@furkan:~/Desktop/final$
```

```
furkan     22311    1905  0 20:59 ?        00:00:00 /usr/libexec/tracker-store
root       22435    1905  0 20:59 ?        00:00:00 ./server -i p2p-Gnutella08.t
root       22477       1  2 21:00 ?        00:00:00 /lib/systemd/systemd-hostnam
furkan     22503    1905  4 21:00 ?        00:00:00 /usr/libexec/tracker-extract
root       22522     352  0 21:00 ?        00:00:00 /lib/systemd/systemd-udevd
root       22524     352  0 21:00 ?        00:00:00 /lib/systemd/systemd-udevd
root       22526       2  0 21:00 ?        00:00:00 [kworker/3:2-events]
furkan     22527    7247  0 21:00 pts/1    00:00:00 ps -ef
furkan@furkan:~/Desktop/final$
```

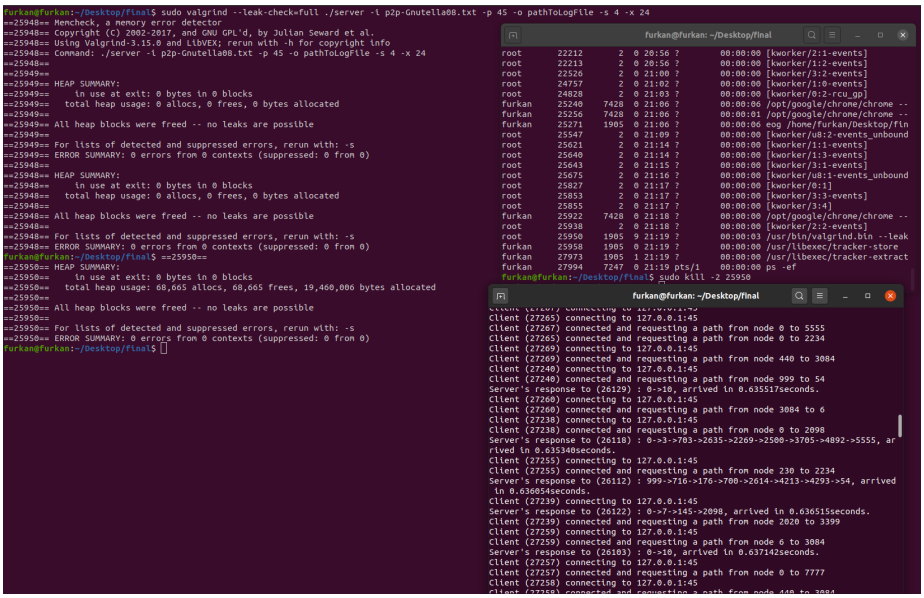
```
furkan@furkan:~/Desktop/final$ ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./clie
nt -a 127.0.0.1 -p 45 -s 0 -d 32 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./
client -a 127.0.0.1 -p 45 -s 32 -d 6032 & ./client -a 127.0.0.1 -p 45 -s 0 -d 22
34 & ./client -a 127.0.0.1 -p 45 -s 230 -d 2234 & ./client -a 127.0.0.1 -p 45 -s
 0 -d 5555 & ./client -a 127.0.0.1 -p 45 -s 0 -d 7777 & ./client -a 127.0.0.1 -p
 45 -s 440 -d 3084 & ./client -a 127.0.0.1 -p 45 -s 6 -d 3084 & ./client -a 127.
0.0.1 -p 45 -s 3084 -d 6 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./client -
a 127.0.0.1 -p 45 -s 0 -d 32 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./clie
nt -a 127.0.0.1 -p 45 -s 32 -d 6032 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 &
 ./client -a 127.0.0.1 -p 45 -s 230 -d 2234 & ./client -a 127.0.0.1 -p 45 -s 0 -
d 5555 & ./client -a 127.0.0.1 -p 45 -s 0 -d 7777 & ./client -a 127.0.0.1 -p 45
-s 440 -d 3084 & ./client -a 127.0.0.1 -p 45 -s 6 -d 3084 & ./client -a 127.0.0.
1 -p 45 -s 3084 -d 6 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./client -a 12
7.0.0.1 -p 45 -s 0 -d 32 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./client -
a 127.0.0.1 -p 45 -s 32 -d 6032 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./c
lient -a 127.0.0.1 -p 45 -s 230 -d 2234 & ./client -a 127.0.0.1 -p 45 -s 0 -d 55
55 & ./client -a 127.0.0.1 -p 45 -s 0 -d 7777 & ./client -a 127.0.0.1 -p 45 -s 4
40 -d 3084 & ./client -a 127.0.0.1 -p 45 -s 6 -d 3084 & ./client -a 127.0.0.1 -p
 45 -s 3084 -d 6 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./client -a 127.0.
0.1 -p 45 -s 0 -d 32 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./client -a 12
7.0.0.1 -p 45 -s 32 -d 6032 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./clien
t -a 127.0.0.1 -p 45 -s 230 -d 2234 & ./client -a 127.0.0.1 -p 45 -s 0 -d 5555 &
 ./client -a 127.0.0.1 -p 45 -s 0 -d 7777 & ./client -a 127.0.0.1 -p 45 -s 440 -
d 3084 & ./client -a 127.0.0.1 -p 45 -s 6 -d 3084 & ./client -a 127.0.0.1 -p 45
-s 3084 -d 6 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./client -a 127.0.0.1
-p 45 -s 0 -d 32 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./client -a 127.0.
0.1 -p 45 -s 32 -d 6032 & ./client -a 127.0.0.1 -p 45 -s 0 -d 2234 & ./client -a
 127.0.0.1 -p 45 -s 230 -d 2234 & ./client -a 127.0.0.1 -p 45 -s 0 -d 5555 & ./c
```


furkan@furkan: ~/Desktop/final
Client (23039) connected and requesting a path from node 1010 to 3201
Client (23014) connecting to 127.0.0.1:45
Client (23027) connecting to 127.0.0.1:45
Client (23027) connected and requesting a path from node 1010 to 3201
Client (23030) connecting to 127.0.0.1:45
Client (23030) connected and requesting a path from node 0 to 10
Server's response to (23028) : 0->10, arrived in 0.004618seconds.
Client (23023) connecting to 127.0.0.1:45
Client (23022) connecting to 127.0.0.1:45
Client (23022) connected and requesting a path from node 5555 to 6001
Client (23014) connected and requesting a path from node 3 to 20
Server's response to (23014) : 3->703->1620->3027->2396->20, arrived in 0.000003 seconds.
Server's response to (23027) : 1010->2922->559->823->1323->3201, arrived in 0.00 0003seconds.
Server's response to (23022) : NO PATH, arrived in 0.000003seconds, shutting dow n
Client (23008) connecting to 127.0.0.1:45
Client (23008) connected and requesting a path from node 500 to 4545
[334] 23045
Client (23023) connected and requesting a path from node 0 to 2098
Server's response to (23023) : 0->7->145->2098, arrived in 0.000003seconds.
[335] 23046
Client (23034) connecting to 127.0.0.1:45
Client (23034) connected and requesting a path from node 5555 to 6001
[336] 23047
Server's response to (23039) : 1010->2922->559->823->1323->3201, arrived in 0.00 3055seconds.
furkan@furkan: ~/Desktop/final
furkan@furkan:~/Desktop/final$ sudo kill -2 22435
furkan@furkan:~/Desktop/final$
[Sun Jun 28 20:59:57 2020] Executing with parameters:
[Sun Jun 28 20:59:57 2020] -i p2p-Gnutella08.txt
[Sun Jun 28 20:59:57 2020] -p 45
[Sun Jun 28 20:59:57 2020] -o pathToLogFile
[Sun Jun 28 20:59:57 2020] -s 4
[Sun Jun 28 20:59:57 2020] -x 24
[Sun Jun 28 20:59:57 2020] Loading graph...
[Sun Jun 28 20:59:57 2020] Graph loaded in 0.018086 seconds with 6301 nodes and 20777 edges.
[Sun Jun 28 20:59:57 2020] A pool of 4 threads has been created
[Sun Jun 28 20:59:57 2020] Thread #2: waiting for connection
[Sun Jun 28 20:59:57 2020] Thread #1: waiting for connection
[Sun Jun 28 20:59:57 2020] Thread #3: waiting for connection
[Sun Jun 28 20:59:57 2020] Thread #0: waiting for connection
[Sun Jun 28 21:01:52 2020] A connection has been delegated to thread id #1, system load 25.00%
[Sun Jun 28 21:01:52 2020] Thread #1: searching database for a path from node 0 to node 2234
[Sun Jun 28 21:01:52 2020] Thread #1: no path in database, calculating 0->2234
[Sun Jun 28 21:01:52 2020] Thread #1: path calculated: 0->3->1287->2690->2234
[Sun Jun 28 21:01:52 2020] Thread #1: waiting for connection
[Sun Jun 28 21:01:52 2020] A connection has been delegated to thread id #1, system load 25.00%
[Sun Jun 28 21:01:52 2020] Thread #1: searching database for a path from node 0 to node 32

```
[Sun Jun 28 21:01:52 2020] Thread #5: waiting for connection
[Sun Jun 28 21:01:52 2020] A connection has been delegated to thread id #2,
system load 25.00%
[Sun Jun 28 21:01:52 2020] Thread #2: searching database for a path from
node 0 to node 2234
[Sun Jun 28 21:01:52 2020] Thread #7: waiting for connection
[Sun Jun 28 21:01:52 2020] Thread #2: path found in database:
0->3->1287->2690->2234
[Sun Jun 28 21:01:52 2020] A connection has been delegated to thread id #5,
system load 25.00%
[Sun Jun 28 21:01:52 2020] Thread #5: searching database for a path from
node 0 to node 2234
[Sun Jun 28 21:01:52 2020] Thread #5: path found in database:
0->3->1287->2690->2234
[Sun Jun 28 21:01:52 2020] Thread #5: waiting for connection
[Sun Jun 28 21:01:52 2020] A connection has been delegated to thread id #5,
system load 25.00%
[Sun Jun 28 21:01:52 2020] Thread #5: searching database for a path from
node 0 to node 32
[Sun Jun 28 21:01:52 2020] Thread #5: path found in database:
0->4->144->146->3337->32
[Sun Jun 28 21:01:52 2020] Thread #5: waiting for connection
[Sun Jun 28 21:01:52 2020] Thread #2: waiting for connection
[Sun Jun 28 21:01:52 2020] A connection has been delegated to thread id #1,
system load 12.50%
[Sun Jun 28 21:01:52 2020] Thread #1: searching database for a path from
node 0 to node 32
```

```
[Sun Jun 28 21:01:53 2020] Thread #9: no path in database, calculating
5555->6001
[Sun Jun 28 21:01:53 2020] Thread #9: path not possible from node 5555 to
6001
[Sun Jun 28 21:01:53 2020] Thread #9: waiting for connection
[Sun Jun 28 21:02:54 2020] Termination signal received, waiting for ongoing
threads to complete.
[Sun Jun 28 21:02:54 2020] All threads have terminated, server shutting
down.
```

5 Valgrind


furkan@furkan:~/Desktop/final$ sudo valgrind --leak-check=full ./server -i p2p-Gnutella08.txt -p 45 -o pathToLogFile -s 4 -x 24
==25948== Memcheck, a memory error detector
==25948== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==25948== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==25948== Command: ./server -i p2p-Gnutella08.txt -p 45 -o pathToLogFile -s 4 -x 24
==25948==
==25949==
==25949== HEAP SUMMARY:
==25949==   in use at exit: 0 bytes in 0 blocks
==25949==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==25949==
==25949== All heap blocks were freed -- no leaks are possible
==25949==
==25949== For lists of detected and suppressed errors, rerun with: -s
==25949== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==25948==
==25948== HEAP SUMMARY:
==25948==   in use at exit: 0 bytes in 0 blocks
==25948==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==25948==
==25948== All heap blocks were freed -- no leaks are possible
==25948==
==25948== For lists of detected and suppressed errors, rerun with: -s
==25948== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
furkan@furkan:~/Desktop/final$ ==25950==
==25950== HEAP SUMMARY:
==25950==   in use at exit: 0 bytes in 0 blocks
==25950==   total heap usage: 68,665 allocs, 68,665 frees, 19,460,006 bytes allocated
==25950==
==25950== All heap blocks were freed -- no leaks are possible
==25950==
==25950== For lists of detected and suppressed errors, rerun with: -s
==25950== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
furkan@furkan:~/Desktop/final$
furkan@furkan: ~/Desktop/final
root 22212 2 0 20:56 ? 00:00:00 [kworker/2:1-events]
root 22213 2 0 20:56 ? 00:00:00 [kworker/1:2-events]
root 22526 2 0 21:00 ? 00:00:00 [kworker/3:2-events]
root 24757 2 0 21:02 ? 00:00:00 [kworker/1:0-events]
root 24828 2 0 21:03 ? 00:00:00 [kworker/0:2-rcu_gp]
furkan 25240 7428 0 21:06 ? 00:00:06 /opt/google/chrome/chrome --
furkan 25256 7428 0 21:06 ? 00:00:01 /opt/google/chrome/chrome --
furkan 25271 1905 0 21:06 ? 00:00:06 eog /home/furkan/Desktop/fin
root 25547 2 0 21:09 ? 00:00:00 [kworker/u8:2-events_unbound
root 25621 2 0 21:14 ? 00:00:00 [kworker/1:1-events]
root 25640 2 0 21:14 ? 00:00:00 [kworker/1:3-events]
root 25643 2 0 21:15 ? 00:00:00 [kworker/3:1-events]
root 25675 2 0 21:16 ? 00:00:00 [kworker/u8:1-events_unbound
root 25827 2 0 21:17 ? 00:00:00 [kworker/0:1]
root 25853 2 0 21:17 ? 00:00:00 [kworker/3:3-events]
root 25855 2 0 21:17 ? 00:00:00 [kworker/3:4]
furkan 25922 7428 0 21:18 ? 00:00:00 /opt/google/chrome/chrome --
root 25938 2 0 21:18 ? 00:00:00 [kworker/2:2-events]
root 25950 1905 9 21:19 ? 00:00:03 /usr/bin/valgrind.bin --leak
furkan 25958 1905 0 21:19 ? 00:00:00 /usr/libexec/tracker-store
furkan 27973 1905 1 21:19 ? 00:00:00 /usr/libexec/tracker-extract
furkan 27994 7247 0 21:19 pts/1 00:00:00 ps -ef
furkan@furkan:~/Desktop/final$ sudo kill -2 25950
furkan@furkan: ~/Desktop/final
Client (27265) connecting to 127.0.0.1:45
Client (27267) connected and requesting a path from node 0 to 5555
Client (27265) connected and requesting a path from node 0 to 2234
Client (27269) connecting to 127.0.0.1:45
Client (27269) connected and requesting a path from node 440 to 3084
Client (27240) connecting to 127.0.0.1:45
Client (27240) connected and requesting a path from node 999 to 54
Server's response to (26129) : 0->10, arrived in 0.635517seconds.
Client (27260) connecting to 127.0.0.1:45
Client (27260) connected and requesting a path from node 3084 to 6
Client (27238) connecting to 127.0.0.1:45
Client (27238) connected and requesting a path from node 0 to 2098
Server's response to (26118) : 0->3->703->2635->2269->2508->3705->4892->5555, ar
rived in 0.635340seconds.
Client (27255) connecting to 127.0.0.1:45
Client (27255) connected and requesting a path from node 230 to 2234
Server's response to (26112) : 999->716->176->700->2614->4213->4293->54, arrived
in 0.636054seconds.
Client (27239) connecting to 127.0.0.1:45
Server's response to (26122) : 0->7->145->2098, arrived in 0.636515seconds.
Client (27239) connected and requesting a path from node 2020 to 3399
Client (27259) connecting to 127.0.0.1:45
Client (27259) connected and requesting a path from node 6 to 3084
Server's response to (26103) : 0->10, arrived in 0.637142seconds.
Client (27257) connecting to 127.0.0.1:45
Client (27257) connected and requesting a path from node 0 to 7777
Client (27258) connecting to 127.0.0.1:45