



Informe práctica 2

Sistemas Operativos

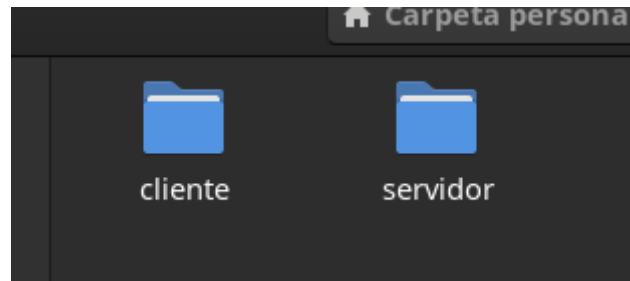
Diego Enrique Molina Sanchez, Andrés Fernando Rojas Pedroza

Ingeniería de sistemas

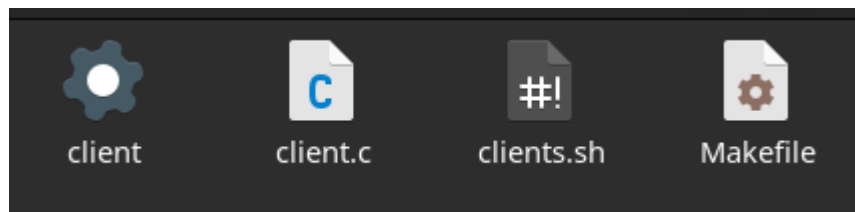
28 de Junio de 2022

1. Manual de uso

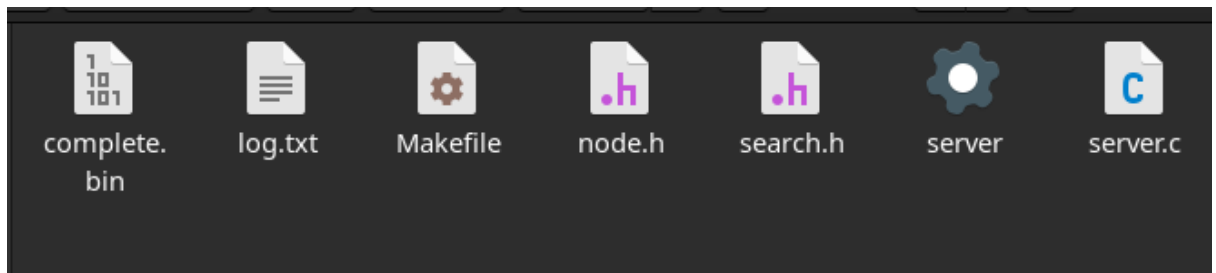
- Se tienen dos archivos: `client.c` y `server.c`
- Se deben poner dichos archivos en directorios diferentes para comprobar la comunicación usando sockets de red



- Además, se deben incluir en la carpeta destinada al proceso servidor dos archivos de cabecera para garantizar el correcto funcionamiento del programa: `node.h` y `search.h`. Además, se debe incluir un archivo binario precargado con la información correspondiente a los tiempos de viaje sobre los cuales se harán las consultas



carpeta cliente



carpeta servidor

- c. Se compilan los programas en terminales diferentes (primero el servidor) usando los comandos

```
gcc client.c -o client
./client

-----

gcc server.c -o server
./server
```

- d. En el caso de la terminal destinada al proceso cliente se mostrará la siguiente información

```
~ / Do / Si / Socket - C / cliente main 13 ?2 ./client
1. Ingresar origen
2. Ingresar destino
3. Ingresar hora
4. Buscar tiempo de viaje medio
5. Salir
Seleccione la opción: 
```

- e. Se empiezan a ingresar los datos de búsqueda para las opciones 1, 2 y 3 para generar la petición al servidor usando el comando 4

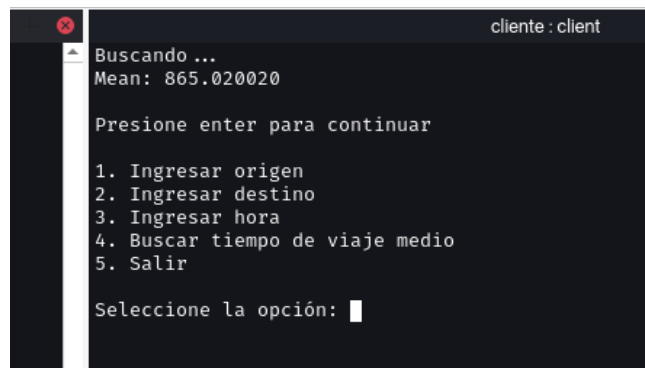
```
servidor
Nueva pestaña Dividir vista
servidor: server
base) (diego@kali)-[~/Escritorio/socket/Socket C/servidor]
$ ./server
```

```
Buscando ...
Mean: 865.020020

Presione enter para continuar

```

- f. Una vez procesada la solicitud, en la consola cliente se imprimirá la información solicitada y se preguntará si se desea continuar con la ejecución de una nueva petición



```
cliente : client
Buscando ...
Mean: 865.020020

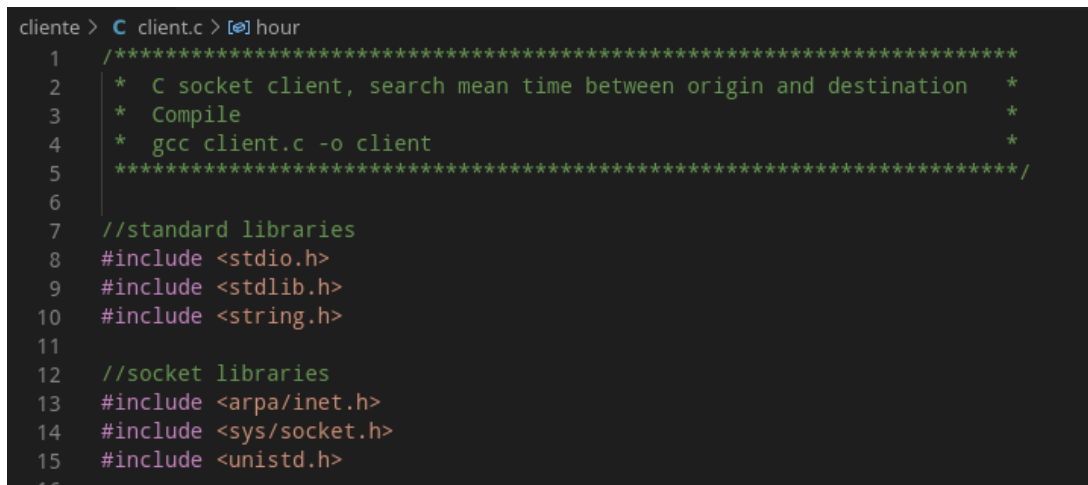
Presione enter para continuar

1. Ingresar origen
2. Ingresar destino
3. Ingresar hora
4. Buscar tiempo de viaje medio
5. Salir

Seleccione la opción: █
```

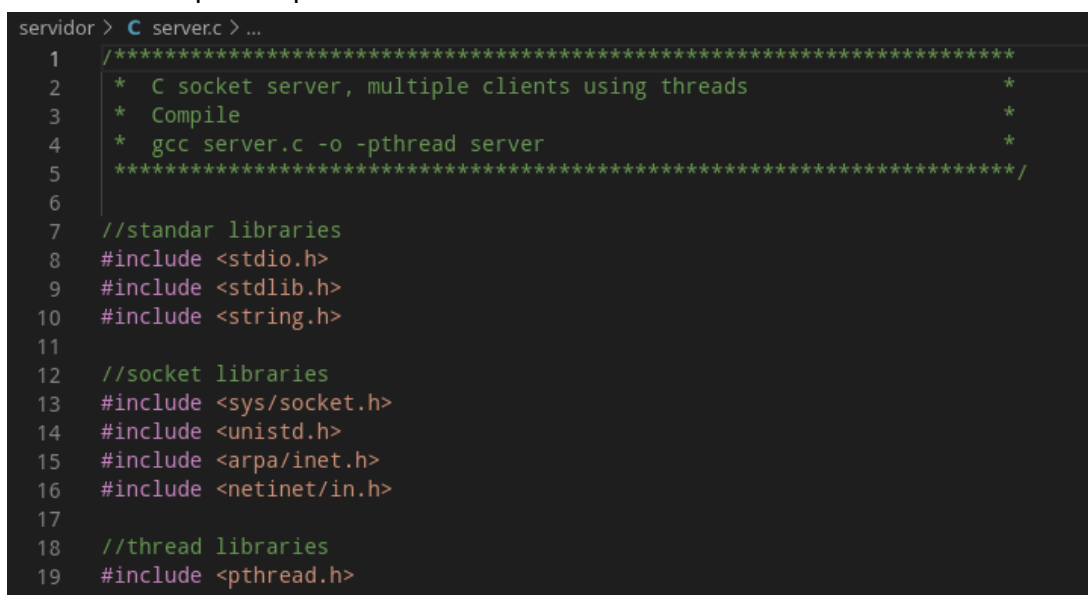
2. Informe de elaboración

- a. Para esta práctica se hizo uso de las siguientes librerías
- i. para el proceso cliente



```
cliente > C client.c > [F5] hour
1  /*****
2  * C socket client, search mean time between origin and destination *
3  * Compile *
4  * gcc client.c -o client *
5  *****/
6
7  //standard libraries
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11
12 //socket libraries
13 #include <arpa/inet.h>
14 #include <sys/socket.h>
15 #include <unistd.h>
16
```

- ii. para el proceso servidor



```
servidor > C server.c > ...
1  /*****
2  * C socket server, multiple clients using threads *
3  * Compile *
4  * gcc server.c -o -pthread server *
5  *****/
6
7  //standar libraries
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11
12 //socket libraries
13 #include <sys/socket.h>
14 #include <unistd.h>
15 #include <arpa/inet.h>
16 #include <netinet/in.h>
17
18 //thread libraries
19 #include <pthread.h>
20
```

- b. Se establecieron los parámetros necesarios para la comunicación por interfaz de red

- i. para el proceso cliente

```
17 //server parameters
18 #define SERVER_ADDRESS "127.0.0.1" /* server host address
19 #define PORT 8080 /* port client will connect to
```

- ii. para el proceso servidor

```
27
28 //server parameters
29 #define PORT 8080
30 #define SERVER_ADDRESS "127.0.0.1"
31 #define BUF_SIZE 100
32
```

- c. Se inicializa la comunicación por interfaz de red a través de la creación de los sockets y la asignación de las direcciones, puertos y protocolos necesarios

- i. para el proceso cliente

```
42 /*
43 int sock = 0, valread, client_fd;
44 struct sockaddr_in serv_addr;
45
46 //socket creation
47 if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
48     printf("\n Socket creation error \n");
49     return -1;
50 }
51
52 //assign ip, port, protocol
53 serv_addr.sin_family = AF_INET;
54 serv_addr.sin_port = htons(PORT);
55
56 //convert IPv4 and IPv6 addresses from text to binary form
57 if (inet_pton(AF_INET, SERVER_ADDRESS, &serv_addr.sin_addr)<= 0) {
58     printf("\nInvalid address/ Address not supported \n");
59     return -1;
60 }
```

- ii. para el proceso servidor

```
50 //listening socket and connection socket file descriptors
51 int server_fd, new_socket, valread;
52 struct sockaddr_in address;
53 int opt = 1;
54 int addrlen = sizeof(address);
55
56 //creating socket file descriptor
57 if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
58     perror("socket failed");
59     exit(EXIT_FAILURE);
60 }
61
62 //forcefully attaching socket to the port 8080
63 if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
64     perror("setsockopt");
65     exit(EXIT_FAILURE);
66 }
```

```

68      /*
69      * Put the server information into the server structure.
70      * The port must be put into network byte order.
71      */
72      address.sin_family = AF_INET;
73      address.sin_addr.s_addr = INADDR_ANY;
74      address.sin_port = htons(PORT);
75
76      //forcefully attaching socket to the port 8080
77      if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
78          perror("bind failed");
79          exit(EXIT_FAILURE);
80      }
81
82      //set the socket for listening (queue backlog of 3)
83      if (listen(server_fd, 3) < 0) {
84          perror("listen");
85          exit(EXIT_FAILURE);
86      }

```

- d. Se establecen en el proceso servidor las funciones necesarias para el manejo de hilos cada vez que se hace una petición desde una terminal diferente que ejecuta el programa cliente

```

88      //array for several threads
89      pthread_t thread_id[40];
90      int i = 0;
91
92      while(1){
93
94          //accept socket connection from one new client
95          if ((new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
96              perror("accept");
97              exit(EXIT_FAILURE);
98          }
99
100         //create one new thread for each new client
101         int * newsock = malloc(sizeof(int));
102         * newsock = new_socket;
103         pthread_create(&thread_id[i++], NULL, connection_handler, newsock);
104         pthread_detach(thread_id[i++]);
105
106         //condition only 32 clients at the same time
107         if(i >= 32){
108             i = 0;
109             while(i < 32){
110                 pthread_join(thread_id[i++], NULL);
111             }
112             i = 0;
113         }
114     }
115 }

```

- e. Enseguida se establece la lógica de comunicación para cada uno de los procesos: en el caso del cliente se definen las variables que se enviarán al servidor para que éste sea quien las reciba y haga la búsqueda en el archivo binario
- La función de búsqueda se encuentra en el archivo de cabecera "search.h" importado inicialmente

```

164 //Getting ip from client socket
165 struct sockaddr_in * pV4Addr = (struct sockaddr_in*)&new_socket;
166 struct in_addr ipAddr = pV4Addr->sin_addr;
167 char strip[INET_ADDRSTRLEN];
168 inet_ntop(AF_INET, &ipAddr, strip,INET_ADDRSTRLEN);
169
170 //search in binary file source/dst/hour
171 int ret = snprintf(answer, sizeof answer, "%f", searchMean(source,dst,timeHour));
172 send(new_socket, answer, strlen(answer), 0);
173
174 if(read_size == 0){
175
176 }

```

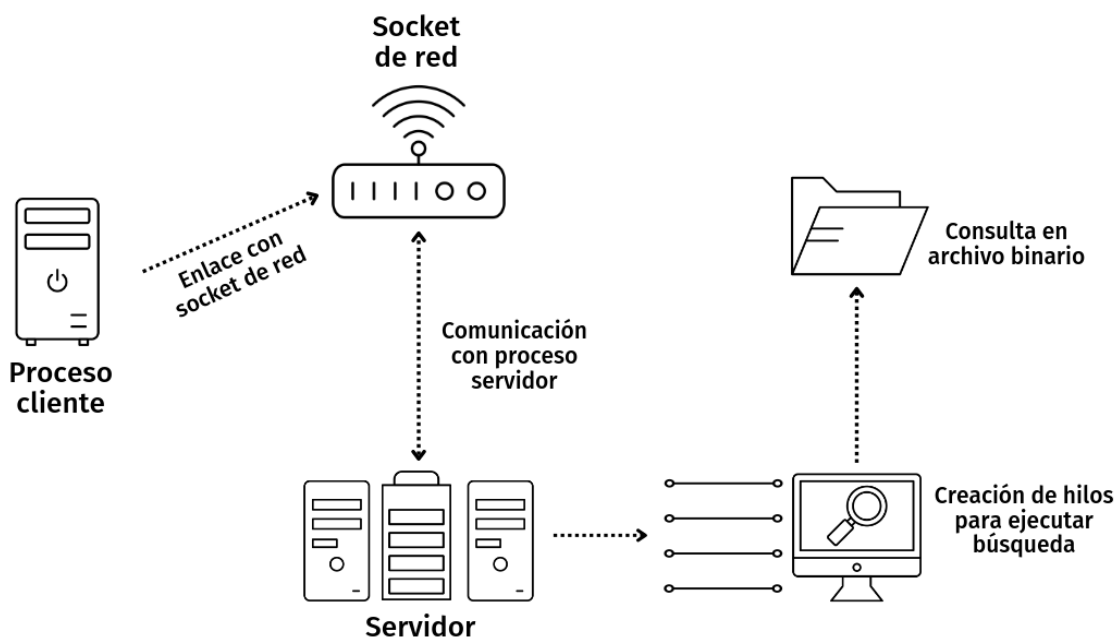
- f. Finalmente, el programa servidor se encarga de llevar un registro de las peticiones en un archivo *log*

```

//write to server log file
FILE * file = fopen("log.txt","a");
fprintf(file,"[Fecha %s] Cliente [%s] [búsqueda - %d - %d - %d]\n",tim,strip,source,dst,timeHour);
fclose(file);

```

3. Diagrama de comunicaciones



4. Diagrama de bloques

