SEGUNDA PRACTICA SISTEMAS OPERATIVOS

Eduard Joel Ostos Castro — eostos@unal.edu.co
Tania Julieth Araque Dueñas — taraqued@unal.edu.co
Andres Felipe Sanchez Ladino — asanchezla@unal.edu.co

1. ELABORACIÓN DEL SERVIDOR

Para la elaboración del servidor, o más bien, puesta en funcionamiento de este, primero crearemos un archivo server.c veasé el archivo contenido en el repositorio de github, ahora bien, debemos tener en cuenta las librerias que debemos usar para poder acceder a la red y a los sockets:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
```

Entonces, las librerias más importantes para este programa son socket.h e in.h puesto que nos proveen las funciones necesarias para crear sockets y conectarnos a internet.
Ahora vamos con el código, el esquema que se usa para crear este tipo de aplicaciones es el siguiente.
1. Crear un socket.
2. Hacer bind al socket para darle una dirección IP.
3. Luego recibir una conexión, con accept
4. Mandar esa conexión a ejecutar lo que se desea.
5. Devolver la conexión.

Ahora ahondaremos más en detalles, entonces, para (1), usaremos la función dada por socket:

```
socketfd = socket(AF_INET, SOCK_STREAM, 0);
```

socket[?] tiene 3 parametros:

```
socket(int domain, int type, int protocol);
```

Ahora, como dicta el manual de referencia de la librería de C[?], la creación de un socket no lo asigna de manera automatica a una dirección, esta asignación debe hacerse de manera manual, con una estructura llamada sockaddr que en nuestro programa es llamado server_direccion, ahora la conexión se hizo así, por medio de bind[?]:

```
bind(socketfd, (struct sockaddr *) &
    server_direccion, sizeof(server_direccion))
```

Entonces, en esa linea de codigo hacemos bind, entonces tiene 3 parametros, el primero, el socket al que le asignaremos la dirección, segundo la estructura sockaddr, que si vemos, hicimos un casting, porque así estaba la variable server_direccion antes:

```
struct sockaddr_in server_direccion;
```

A la cual antes de hacer el bind le cambiamos unos parametros, así:

```
server_direccion.sin_port = htons(numero_puerto);
server_direccion.sin_family = AF_INET;
server_direccion.sin_addr.s_addr = INADDR_ANY;
```

Viendo sin_port lo que hace es asignar a la estructura un puerto, el cual se da al iniciar el programa, entonces, ese número de puerto es asignado a la estructura. En la segunda línea modificamos sin_family para darle el valor de AF_INET, coonsultando de nuevo en el manual de C podemos darnos cuenta que AF_INET[?] es la palabra reservada para el protocolo de internet IPv4; Ahora la tercera línea, sin_add.s_addr la modificamos para darle una dirección a nuestra estructura, INADDR_ANY indica cualquiera, sin embargo, esta dirección es escogida por el equipo donde se ejecute el programa.
Retomando el bind, una vez hecho el bind a nuestro socket ahora podemos decir que nuestro socket está completo y lo que nos sigue es hacer listen en el socket, así:

```
listen(socketfd, 5);
```

Donde 5 es el máximo número de conexiones que recibiremos. Ahora bien, para manejar toda esta cantidad de conexiones sin que luego de una respuesta del servidor se muera lo que haremos es usar fork()[?], una función de C que permite crear multiples procesos y ejecutarlos al mismo tiempo, por lo tanto no habrá ningún fallo en nuestro programa. Una vez logrado esto, si la conexión es satisfactoría llameremos a la función busqueda, que recibirá un socket, el cual es un socket que aceptamos, así:

```
socket_cliente = accept(socketfd, (struct sockaddr
    *) &cliente_direccion, &tamano_cliente);
```

Con esta línea de codigo hacemos lo anteriormente mencionado, aceptamos[?] un socket cliente, entonces, una vez aceptado podremos acceder a los mismos metodos de un socket normal. En nuestra función busqueda, una vez recibido nuestro socket, abriremos el archivo en el que queremos hacer la busqueda, luego creamos un buffer que

será de tipo de dato char, puesto que por medio de este será posible la lectura y escritura de los datos necesarios, ya lo veremos más adelante.

```
FILE *file = fopen("prueba.bin", "rb");
char buffer[256];
```

Ahí asignamos las variables, como explicamos anteriormente.
Entonces, ahora, para tomar los datos que el cliente desea buscar, los tomaremos de los que están escritos en el socket del cliente, cuando se explique el archivo cliente.c se entenderá porque reciden estos datos en el socket. Ahora bien, para el manejo de estos datos lo que haremos es pasar estos datos al buffer que creamos antes, así:

```
int n = read(socket, buffer, 256);
```

La función read[?] toma 3 parametros, el primero es el socket del que leeremos, el segundo es el destino de los datos leídos y el tercero es la cantidad de datos de que se leerán.
Una vez que los datos ya están en el buffer, debemos asignarlos en a unas variables, los datos recibidos son 3, que son sourceid, dstid y hod, que son los que se requieren para la busqueda, entonces para la asignación usaremos sscanf[?], así:

```
sscanf(buffer, "%d,%d,%d", &sourceid,&dstid,&hod);
```

Lo que hacemos ahí es "escanear" los datos del buffer, darles un formato y enviarlos a una dirección de memoria, aquí está la declaración de las variables antes mencionadas:

```
int hod,dstid,sourceid;
```

Ahora que tenemos los datos a buscar en las variables deseadas, empezaremos la busqueda por medio de un while que tenga como parametro fread[?], que como sabemos, retorna 0 para cuando no hay más datos o un número diferente de 0 para cuando quedan más, nos sirve para que el while se rompa una vez que el archivo ya ha sido recorrido por completo y además nos sirve para guardar los datos en data, que es la instancia de la estructura TravelData, este es el while: Primero vemos la estructura TravelData, en una instancia de esta estructura grabaremos los datos obtenidos con fread

```
typedef struct {
  int sourceid;
  int dstid;
  int hod;
  float mean_travel_time;
} TravelData;
```

```
while (fread(&data, sizeof(data), 1, file)){
  if(data.hod == hod && data.dstid == dstid &&
      data.sourceid == sourceid ){
    mean_travel_time = data.mean_travel_time;
    break;
  }else{
    mean_travel_time = -1;
  }
}
```

Lo que hace el while es asignar cada linea del archivo de busqueda y la coloca en una estructura, entonces luego en el if se hace la comparación de cada atributo de la instancia data con las variables otorgó el cliente por medio del socket si se encuentra entonces la variable mean_travel_time se modifica para que sea igual al atributo mean_travel_time de la instancia que cumplió con las condiciones, si no se encuentra entonces mean_travel_time toma el valor de -1.
Ahora vamos con otro paso, una vez que tenemos mean_travel_time queremos enviarla de vuelta al socket del cliente, entonces haremos lo inverso a lo que hicimos arriba, vamos a escribir el valor de la variable en el socket, no sin antes colocar todo el socket en 0, para evitar que los valores que traía dañen nuestra información, usamos bzero() para esto, luego lo que hacemos es usar sprintf()[?] y por ultimo usamos write para escribir desde el buffer hacia el socket, así:

```
bzero(buffer, 256);
sprintf(buffer, "%f", mean_travel_time);
n = write(socket, buffer, strlen(buffer));
```

bzero[?] toma 2 parametros, el lugar donde vamos a escribir los 0 y luego la cantidad de 0's que queremos crear, sprintf toma 3 parametros, primero el lugar donde vamos a colocar los datos, luego el formato y por ultimo la variable; write toma 3 que son los mismos que se vieron en la función read, explicada más arriba.
Y listo, con esto habremos terminado nuestra tarea en el servidor, ahora vamos con el cliente.

2. ELABORACIÓN DEL CLIENTE

Primero haremos las peticiones al cliente, le preguntaremos que desea buscar, la explicación de esto no es fundamental para nuestros objetivos, por lo tanto no se hará.
El esquema que usaremos ahora será el siguiente:
1. Crearemos un socket.
2. Crearemos una estructura de tipo hostent y otra sockadrr_in
3. Enviamos la petición de conectar al servidor.

Estas son las variables que usaremos:

```
int socketfd;
struct sockaddr_in server_direccion;
struct hostent *server;
char buffer[256];
```

Vamos con lo primero, la creación del socket es igual que en el servidor, luego la parte importante es conectarnos al servidor, esto lo lograremos con dos funciones, gethostbyname y connect, entonces, gethostbyname tomará como parametro la dirección que el

usuario le pasará como argumento al iniciar el programa, buscará esa dirección y de existir se podrá acceder a sus atributos, puesto que es una estructura, entiendase:

```
server = gethostbyname(argv[1]);
```

Una vez que tenemos esto, podremos acceder a la información de server y la copiaremos en server_direccion, lo importante acá es copiar la dirección, de esta forma:

```
bcopy((char *) server->h_addr, (char *) &
    server_direccion.sin_addr.s_addr, server->
    h_length);
```

bcopy toma 3 parametros, el primero es lo que va a copiar y lo segundo es a donde lo copiará, entonces así se entiende que toma la dirección de server, que es server->h_addr y la envia a sin_addr.s_addr, es importante tomar en cuenta que primero casteamos convertir en punteros estas direcciones de memoria y así modificar sus valores. El esquema de server_direccion se ve así:

```
server_direccion.sin_family = AF_INET;

bcopy((char *) server->h_addr, (char *) &
    server_direccion.sin_addr.s_addr, server->
    h_length);

server_direccion.sin_port = htons(numero_puerto);
```

Luego haremos la mejor parte, la conexión con el servidor, esto gracias a la función connect, así:

```
connect(socketfd, (struct sockaddr *)&
    server_direccion, sizeof(server_direccion)
```

Esto sigue los mismos parametros que la función accept que vimos en la explicación del servidor.
Y ahora de la misma manera que en el servidor usaremos el buffer que creamos arriba para escribir los datos que queremos y luego recibirlos, así:

```
sprintf(buffer, "%d,%d,%d", origin, destination,
    hod);
int n = write(socketfd, buffer, strlen(buffer));
```

Ya vimos el funcionamiento de estas 2 funciones anteriormente, entonces, en esas líneas de codigo enviamos los datos, luego cuando ya se hizo la busqueda en el servidor hacemos esto para extraer la información:

```
bzero(buffer, 256);
n = read(socketfd, buffer, 255);
float result;
sscanf(buffer, "%f", &result);
```

Entonces, dejamos en 0, luego leemos del socket y escribimos al buffer, luego escaneamos del buffer y pasamos el resultado a la variable result, ya luego podremos impimir en pantalla.
Y con esto terminamos la explicación de nuestro programa.

3. DIAGRAMA DE FLUJO

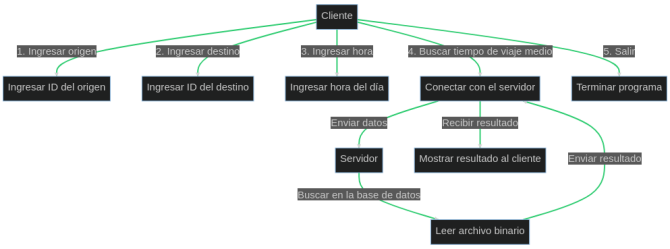

Fig. 1. Diagrama de flujo.

4. DIAGRAMA DE BLOQUES

Fig. 2. Diagrama de bloques.

References:

5. REFERENCES

[1] accept(2) linux programmer's manual. https://man7.org/linux/man-pages/man2/accept.2.html. Accessed: 2023-05-17.

[2] bind(2) linux programmer's manual. https://man7.org/linux/man-pages/man2/bind.2.html. Accessed: 2023-05-17.

[3] bzero(3) linux programmer's manual. https://man7.org/linux/man-pages/man2/bzero.3.html. Accessed: 2023-05-17.

[4] fork(2) linux programmer's manual. https://man7.org/linux/man-pages/man2/fork.2.html. Accessed: 2023-05-17.

[5] fread(3) linux programmer's manual. https://man7.org/linux/man-pages/man2/fread.3.html. Accessed: 2023-05-17.

[6] The gnu c library reference manual. https://www.gnu.org/software/libc/manual/pdf/libc.pdf. Accessed: 2023-05-17.

[7] ip(7) linux programmer's manual. https://man7.org/linux/man-pages/man7/ip.7.html. Accessed: 2023-05-17.

[8] read(2) linux programmer's manual. https://man7.org/linux/man-pages/man2/read.2.html. Accessed: 2023-05-17.

[9] Socket(2) linux programmer's manual. https://man7.org/linux/man-pages/man2/socket.2.html. Accessed: 2023-05-17.

[10] sprintf(3p) linux programmer's manual. https://man7.org/linux/man-pages/man3/sprintf.3p.html. Accessed: 2023-05-17.

[11] sscanf(3p) linux programmer's manual. https://man7.org/linux/man-pages/man3/sscanf.3p.html. Accessed: 2023-05-17.