Barbe Victor 403715
Arturo Arellano Coyotl 168824
Ruben Jaramillo Gomez 168406
Pierre Louis Gaucher 403783

Programación concurrente – Usando socket

1 – Client side

In this activity, we will try to send data using sockets from the client to the server. We will run two codes simultaneously, to create the server and the client. Both could be on different machines, but here we will be using the localhost address 127.0.0.1, which means the data will be sent to the same computer.

We will use the first code from class that allows to send an integer using sockets and modify it with the activity from last week to send 4 char values and an integer in big Endian and then print it on the server side in little Endian.

In the client side, we first define the variables we want to send. There will be 4 chars with values 'A', 'B', 'C', 'D', and an int of value 16.

```
//declaration of variables that will be sent to the server
char c1='A';
char c2='B';
char c3='C';
char c4='D';
int i = 16;
```

Then in the main, we will create a socket for each information we want to send. To send the char value, we just create a socket, call the function send_integer to send the variable we want using the socket we created, and then close the socket. Here is the code showing how the two first chars are sent.

```
//sending the 1st char using a socket, then closing it
int sock = connection();
send_integer(sock,c1);
close(sock);

//sending the 2nd char with an other socket
sock = connection();
send_integer(sock,c2);
close(sock);
```

The function "send_integer" takes as a parameter a socket and the 'integer' variable, to then send the variable using the socket.

Then, we will send the integer in Big Endian format. To do so, we will use a pointer on the value of I called "pointer". Since we want to send the data in big Endian format, we will first send the last byte

used in the representation of ‘i’, using pointer[3]. Then we will send pointer[2], ect. This means the 4 bytes of the variable ‘i’ have been sent one by one in a different socket, as we can see here:

```
//sending the 4th byte of the int
sock = connection();
send_integer(sock,pointeur[3]);
close(sock);

//sending the 3rd byte of the int
sock = connection();
send_integer(sock,pointeur[2]);
close(sock);

//sending the 2nd byte of the int
sock = connection();
send_integer(sock,pointeur[1]);
close(sock);

//sending the 1st byte of the int
sock = connection();
send_integer(sock,pointeur[0]);
close(sock);
```

After that, we can send the two other char variables:

```
//sending the 3rd char
sock = connection();
send_integer(sock,c3);
close(sock);

//sending the 4th char
sock = connection();
send_integer(sock,c4);
close(sock);
```

Now we will look at how the data is retrieved on the sever side, and the integer reassembled in little endian.

2 – Sever side

In the server program, we will first define the 5 variables with different values than the one in the client script. The goal will be to assign to these variables the values sent in the socket, to verify if the connection works properly, and if the integer is correctly reassembled.

```
//variables iniciated to a value, to check if the program worked properly
char c1='U';
char c2='U';
char c3='U';
char c4='U';
int i=-1;
//pointer for the int
char *pointeur = (char *) &i;
```

We will also again create a pointer on the integer variable that will be sent in big endian format. Then in the main we will initialize the socket. We will make an infinite loop, that will allow the server to receive for as long as it isn't shut down.

Then in the main we create a loop with a counter, that will allow to retrieve the data in the correct order. At each iteration of the loop the counter is incremented by one, which allows us to get the data in the correct order in the correct variables (first the two chars, then the int in big endian, then the two other chars).

Here we get the first two chars.

```
if(counter==0)
{c1=integer;
}
//getting 2nd char
if(counter==1)
{c2=integer;
}
```

Then we will get the int using the pointer we declared above. Since the data was sent in big endian, we retrieve it in this order, byte by byte:

```
if(counter==2)
{
pointeur[3] = integer;
}
    if(counter==3)
{
pointeur[2] = integer;
}
    if(counter==4)
{
pointeur[1] = integer;
}
    if(counter==5)
{
pointeur[0] = integer;
}
```

Now the data will again be stored in little endian format, and we will get 16 as an output when we will print the value of 'i'. After that, we can retrieve the rest of the chars:

```
//getting 3rd char
if(counter==6)
{c3=integer;
}
//getting 4th char
if(counter==7)
{c4=integer;
}
```

At the end of the iteration of the loop we will call the display method in order to see if the variables have been modified properly.

```c
//function to display data
void display()
{
    printf("c1 : %c\n",c1);
    printf("c2 : %c\n",c2);
    printf("i : %d\n",i);
    printf("c3 : %c\n",c3);
    printf("c4 : %c\n",c4);
}
```

The recv_integer function is used to retrieve data from the client socket. It is called at the beginning of each iteration of the loop to get the data.

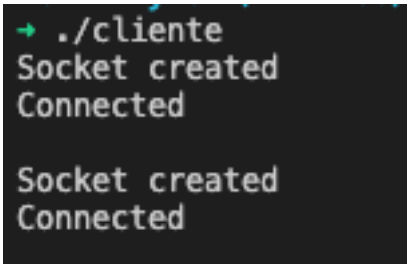
```c
//function to recieve data using the socket
int recv_integer(int client_sock)
{
    int read_size, integer;

    // receive a message from client
    read_size = recv(client_sock, &integer, sizeof(integer), 0);

    if(read_size == 0) {
        puts("Client disconnected");
        fflush(stdout);
    }
    else if(read_size == -1) {
        perror("recv failed");
    }

    return integer;
}
```

```c
//infinite loop
while(1) {
//declaring variables used to get the data
int client_sock = connection(socket_desc);
int socket_desc = initialization();
int integer = recv_integer(client_sock);
```

3 – Running the code

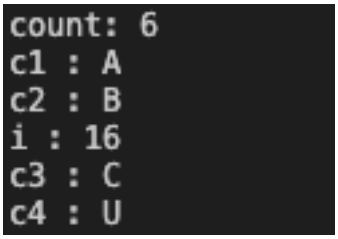
To compile the code, we first start the server side using ‘cc server.c utils.c -o server’ and then ./server Then we can start the client side using ‘cc cliente.c utils.c -o cliente’ and then ./cliente

We will start the server script first, so it listens for upcoming data using ./server. After that, we can execute cliente and we will get a socket created message for each socket.



If we go back to the terminal, were we started server we will get the following output:

We can see our data has been modified properly, we can even see each iteration and how each variable was modified at each iteration. Here on the former iteration, C4 didn't got its value yet.



We can also see that the values of 'i' were different since it got reassembled byte by byte.

In the end, we managed to send the data using sockets from a client to a server, with an integer represented in big endian.