CI-0117 Laboratorio Socket - Programación Paralela y Concurrente
Jennifer Villalobos Fernández | B67751

---

1. Completar la clase "Socket" para poder intercambiar mensajes entre procesos que no comparten memoria.

Para la clase del cliente se implementó:

Socket, el constructor (socket): Se establecen condicionales para poder identificar el tipo de conexión, así como el tipo de socket. Si el booleano "ipv6" es verdadero, se inicia el socket con AF_INET6, en caso contrario, se emplea AF_INET. El tipo de socket será definido por la variable char entrante, en caso de que esta sea una "d" el socket se definirá como datagram ($SOCK_DGRAM$), si resulta ser una "s" será establecido como stream ($SOCK_STREAM$).

```
Socket::Socket( char tipo, bool ipv6 )
{
    //Si el bool es true
    if(ipv6)
    {
        if(tipo == 'd')
        {
            this->idSocket = socket(AF_INET6, SOCK_DGRAM, 0);
        }
        else
        {
            this->idSocket = socket(AF_INET6, SOCK_STREAM, 0);
        }
    }
    else     //conexion ipv4
    {
        if(tipo == 'd')
        {
            this->idSocket = socket(AF_INET, SOCK_DGRAM, 0);
        }
        else
        {
            this->idSocket = socket(AF_INET, SOCK_STREAM, 0);
        }
    }
    //En caso de error
    if (idSocket == -1)
    {
        perror("Socket::Socket");
        exit(1);
    }
}
```

(a) Constructor del socket.

Close, para destruir el Socket (close): Este método "apaga" el socket con el identificador que se le envíe como parámetro, en este caso, el mismo construido. El destructor llama a este método para terminar la conexión.

```
Socket::~Socket()
{
    Close();
}


void Socket::Close()
{
    int apagado = close(this->idSocket);
    //En caso de error
    if (apagado != 0)
    {
        perror("Socket::Close");
        exit(1);
    }

}
```

(b) Método Close y destructor del socket.

Connect, para conectarse (connect): para el IPv4 (conectar con host y puerto): se crea una estructura *sockaddr_in* (que ayuda a hacer referencia a los elementos del socket). En esta se esfecifica la familia de la dirección y el puerto a utilizar para la conexión. El htons() se encarga de convertir el host a una variable corta de red. Finalmente, se establece la conexión con el método connect, de parámetros se utilizan el id del socket, la dirección y el tamaño de esta.

```
int Socket::Connect( char * hostip, int port )
{
    //Para definir los elementos del socket
    struct sockaddr_in direccion;
    direccion.sin_family = AF_INET;
    direccion.sin_port = htons(port); //Se convierte a variable de red

    inet_pton(AF_INET, hostip, & direccion.sin_addr);

    int conexion = connect(this->idSocket, (struct sockaddr *) & direccion, sizeof(direccion));
    return conexion;

}
```

(c) Método Connect(host, puerto).

Connect, para conectarse (connect): para el IPv6. Se utiliza la variable hints (para posteriormente usarla en otro comando), encargada de seleccionar ciertos criterios de la dirección del socket. Se especifica la familia de la dirección (utilizando una que soporta tanto IPv4 como IPv6), el tipo de socket, la bandera y protocolo (cualquiera). Se implementa la herramienta getaddrinfo, la cual por medio del host y el servicio, transmite la dirección a internet a la variable "resultado". Esta se utiliza para invocar el connect y luego se libera por medio de la herramienta freeaddrinfo.

```
int Socket::Connect( char *host, char *service )
{
    int conexion;
    struct addrinfo hints, *resultado, *rp;

    memset(&hints, 0, sizeof(struct addrinfo));

    //Criterios de la direccion del socket
    hints.ai_family = AF_UNSPEC;  //Permite IPv4 o IPv6.
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_flags = 0;
    hints.ai_protocol = 0; //Cualquier protocolo

    //Resultado guarda la direccion a internet
    conexion = getaddrinfo(host, service, &hints, &resultado);

    for(rp = resultado; rp; rp = rp->ai_next)
    {
        conexion = connect(idSocket, rp->ai_addr, rp->ai_addrlen);
        if(conexion == 0)
        {
            break;
        }

    }

    //Se libera la direccion guardada
    freeaddrinfo(resultado);
    return conexion;
}
```

(d) Método Connect(host, servidor).

Read, para leer información por el Socket (read): Se utiliza el comando read. Como parámetros se utilizan el id del socket, el texto por leer y el tamaño de este. En caso de error, retorna -1.

```
int Socket::Read( char *text, int len )
{
    int lectura = read(this->idSocket, (void *)text, len);
    //En caso de error
    if (lectura == -1)
    {
        perror("Socket::Read");
        exit(1);
    }
    return lectura;
}
```

(e) Método Read.

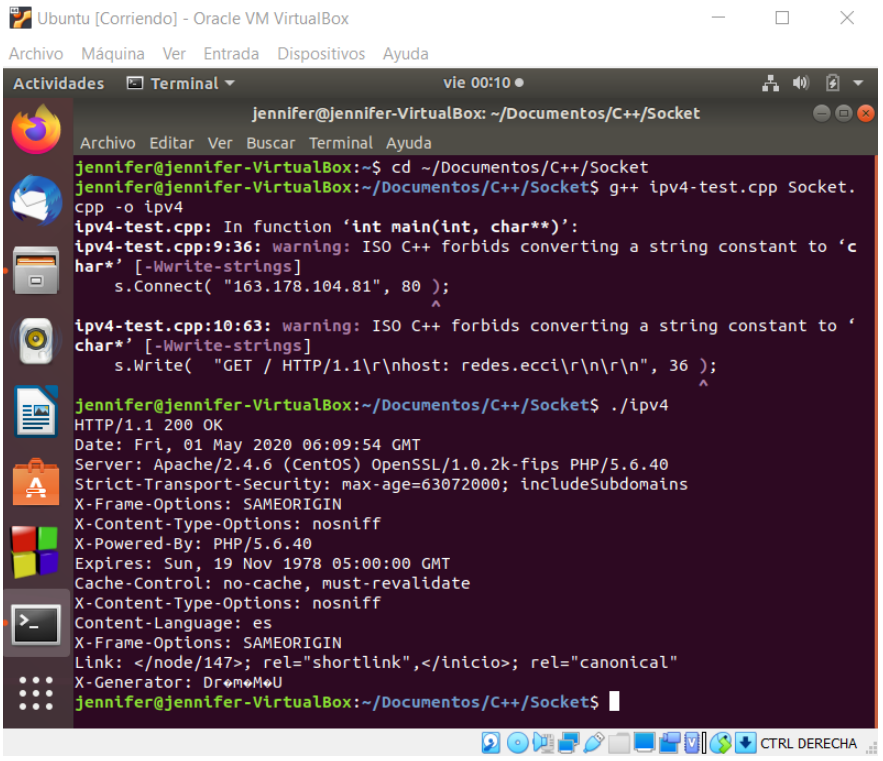
Write, para escribir por el Socket (write): Se utiliza el comando write. Como parámetros se utilizan el id del socket, el texto que será enviado y el tamaño de este. En caso de error, retorna -1.

```
int Socket::Write( char *text, int len)
{
    int escritura = write(this->idSocket, (void *)text, len);
    if (escritura == -1)
    {
        perror("Socket::Write");
        exit(1);
    }
    return escritura;
}
```

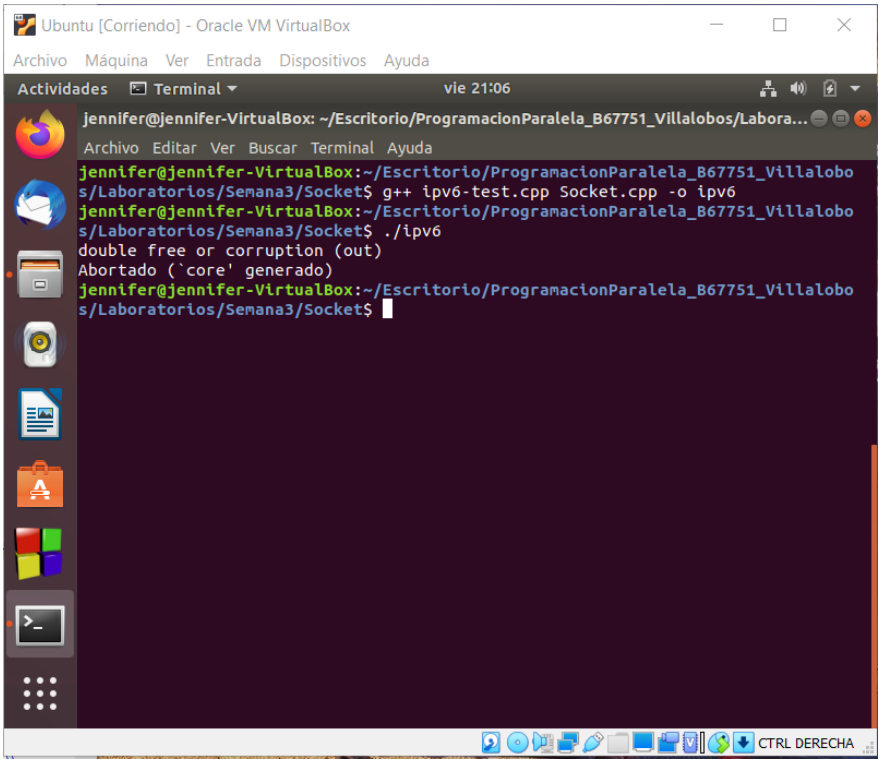
(f) Método Write.

2. Los ejemplos "ipv4-test.cc" y "ipv6-test.cc" deben funcionar correctamente.

El programa ejecura correctamente la prueba IPv4. Sin embargo, con la prueba IPv6 persistieron problemas de conexión.



(g) Compilación y ejecución prueba IPv4.



(h) Compilación y ejecución prueba IPv6.