

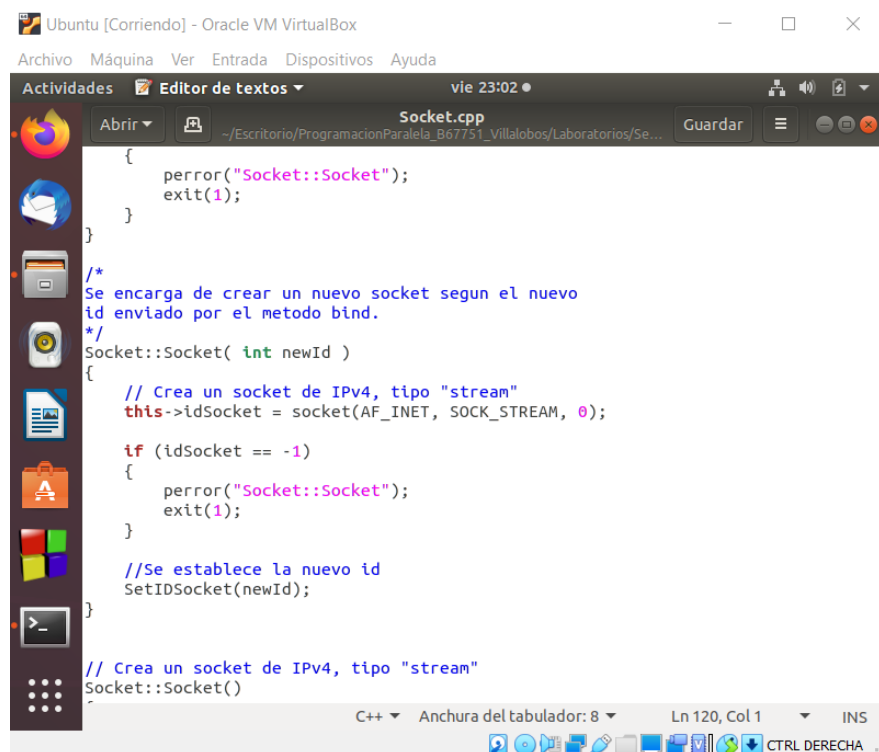
CI-0117 Laboratorio Socket (modalidad servidor) - Programación Paralela y Concurrente

Jennifer Villalobos Fernández | B67751

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

Para la clase del servidor se completó:

Socket(int), otro constructor que recibe un entero como parámetro: En este caso no son necesarios los condicionales para poder identificar el tipo de conexión, ya que esta vez la conexión será solo de ipv4, por lo tanto solo se utilizará *AF_INET* y el tipo de socket será establecido como stream (*SOCK_STREAM*). Dentro de este constructor se declara el id del socket mediante el método *SetIDSocket*.



```
Socket.cpp
{
    perror("Socket::Socket");
    exit(1);
}

/*
Se encarga de crear un nuevo socket segun el nuevo
id enviado por el metodo bind.
*/
Socket::Socket( int newId )
{
    // Crea un socket de IPv4, tipo "stream"
    this->idSocket = socket(AF_INET, SOCK_STREAM, 0);

    if (idSocket == -1)
    {
        perror("Socket::Socket");
        exit(1);
    }

    //Se establece la nuevo id
    SetIDSocket(newId);
}

// Crea un socket de IPv4, tipo "stream"
Socket::Socket()
{
    // ...
}
```

(a) Constructor del socket.

Socket(), otro constructor que no recibe parámetros: Este se implementó debido a que las pruebas lo utilizaban. Al igual que el anterior, se crea un socket con conexión ipv4 y de tipo stream.

```

// Crea un socket de IPv4, tipo "stream"
Socket::Socket()
{
    this->idSocket = socket(AF_INET, SOCK_STREAM, 0);

    if (idSocket == -1)
    {
        perror("Socket::Socket");
        exit(1);
    }
}

```

(b) Otro constructor del socket.

Listen(int): Este parámetro marca el socket como pasivo, es decir que va a recibir conexiones por medio del método Accept. El parámetro establece el tamaño de la cola. En caso de error, retorna -1.

```

int Socket::Listen( int queue )
{
    int spasivo = listen(this->idSocket, queue);

    if (spasivo == -1)
    {
        perror("Socket::Listen");
        exit(1);
    }
    return spasivo;
}

```

(c) Método Listen.

Bind(int): Este método relaciona el socket con el proceso. El parámetro determina el puerto a utilizar. En caso de error, retorna -1.

```

}
return spasivo;
}

int Socket::Bind( int port )
{
    //Se detallan las cualidades del socket
    struct sockaddr_in direccion;
    direccion.sin_family = AF_INET;
    direccion.sin_addr.s_addr = htonl(INADDR_ANY);
    direccion.sin_port = htons(port);

    //Se genera la conexion
    int identificacion = bind(this->idSocket, (const sockaddr*)&direccion,
sizeof(direccion));
    if(identificacion == -1)
    {
        perror("Socket::Bind");
        exit(1);
    }
    return identificacion;
}

Socket * Socket::Accept()

```

(d) Método Bind.

Socket * Accept: Este método establece conexiones con el servidor. Crea una identidad nueva para un nuevo socket. Para esto, se genera un nuevo socket tipo stream con el constructor que recibe un entero como parámetro, de esta manera se le envía la nueva id. Dentro de este constructor se declara la nueva identidad mediante el método SetIDSocket.

```

Socket * Socket::Accept()
{
    struct sockaddr_in direccion;
    socklen_t direccion_len = sizeof(direccion);

    //Se crea una nueva direccion
    int newId = accept(this->idSocket, (struct sockaddr*) &direccion,
&direccion_len);

    if(newId == -1)
    {
        perror("Socket::Accept");
        exit(1);
    }

    //Se crea un nuevo Socket con la direccion obtenida
    Socket * newSocket = new Socket(newId);

    return newSocket;
}

```

(e) Método Accept.

Shutdown(int): Se encarga de cerrar parcialmente el Socket de acuerdo con el parámetro.

```

int Socket::Shutdown( int mode )
{
    int apagado = shutdown(this->idSocket, mode);

    if(apagado == -1)
    {
        perror("Socket::Shutdown");
        exit(1);
    }
    return apagado;
}

```

(f) Método Shutdown.

Write(char*): Se declaró este método ya que las pruebas lo solicitaban. El tamaño del texto (que es el faltante como parámetro) se obtuvo mediante el comando strlen.

```

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

```

(g) Método Write (segundo).

2. Los ejemplos "MirrorServerTest.cc" y "MirrorClientTest.cc" deben funcionar correctamente.

El programa ejecuta correctamente la prueba del servidor, el cual espera a que una conexión se establezca con él. Con la prueba client se realizaron cambios en el main, al modificarle la dirección del connect por la propia. Luego se ejecutan ambas pruebas en dos terminales distintas. La conexión resulta satisfactoria y los programas se ejecutan correctamente

The screenshot shows a terminal window titled "Ubuntu [Corriendo] - Oracle VM VirtualBox". The terminal content is as follows:

```
jennifer@jennifer-VirtualBox: ~/Escritorio/ProgramacionParalela_B67751_Villalobos/Labora...
Archivo Editar Ver Buscar Terminal Ayuda
jennifer@jennifer-VirtualBox:~$ cd ~/Escritorio/ProgramacionParalela_B67751_Villalobos/Laboratorios/Semana4/Socket
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos/Laboratorios/Semana4/Socket$ ./MirrorServerTest
```

(h) Ejecución prueba servidor.

The screenshot shows a terminal window titled "Ubuntu [Corriendo] - Oracle VM VirtualBox". The terminal content is as follows:

```
jennifer@jennifer-VirtualBox: ~/Escritorio/ProgramacionParalela_B67751_Villalobos/Labora...
Archivo Editar Ver Buscar Terminal Ayuda
jennifer@jennifer-VirtualBox:~$ cd ~/Escritorio/ProgramacionParalela_B67751_Villalobos/Laboratorios/Semana4/Socket
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos/Laboratorios/Semana4/Socket$ ./MirrorClientTest Paralela
Paralela
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos/Laboratorios/Semana4/Socket$
```

(i) Ejecución prueba cliente.