

Universidad Nacional
ARTURO JAURETCHE

SISTEMAS DISTRIBUIDOS

LABORATORIO N° 2

Sockets, Threads, OpenMP

Autor
Emiliano SALVATORI

30 de octubre de 2020

Índice

1. Finalidad	2
2. Introducción	2
3. Precedente	2
4. Pthread	4
5. OpenMP	6
5.1. Ejemplo nº 1	6
5.2. Ejemplo nº 2	7
6. Sockets	8
6.1. Ejemplo	8
6.1.1. En en Nodo Maestro	10
6.1.2. En en Nodo Esclavo	11
6.2. Chat-Room con Pthread	11
6.2.1. Código ejecutado en el Maestro	12
6.2.2. Código ejecutado en el Nodo1	12
6.2.3. Código ejecutado en el Nodo2	12
6.2.4. En la sala de chat desde el Maestro	13
7. Chat-room con OpenMP	13
7.1. Código del Servidor y compilacion	13
7.2. Código del Cliente y compilacion	14
7.3. Conexión al Servidor e intercambio de mensajes entre Clientes	15
8. Conclusiones	17
Bibliografía	17

1. Finalidad

En el siguiente informe se detalla lo realizado como parte del laboratorio de la materia **Sistemas Distribuidos** para la **Comisión n° 1**.

La implementación realizada para este trabajo se basa en los siguientes objetivos:

- Familiarizarse con el uso de las librerías Pthread y OpenMP de entornos UNIX para procesamiento concurrente y paralelo para multiprocesamiento distribuido
- Familiarizarse con el uso de Sockets y diseño de clientes servidor con multi-hilo.
- Desarrollo de aplicaciones peer to peer.
- Técnicas de comunicación por TCP/IP como UDP. A nivel middleware.

Para este propósito se utilizan las máquinas virtuales modificadas en el Laboratorio n° 1. La cual contiene las siguientes características:

- Versión de linux Debian 10 sin entorno grafico.
- Usuario: linuxtest
Password: 123456
- Usuario: root
Password: 123456

2. Introducción

Como se hizo mención anteriormente, el objetivo del presente trabajo es familiarizar al estudiante con los conceptos del procesamiento concurrente y paralelo para la materia **Sistemas Distribuidos**.

Para este propósito se proveen de 4 archivos de ejemplo realizados en lenguaje C, los cuales utilizan las librerías de *Pthread* y *OpenMP*.

Por otro lado también se hace hincapié en el funcionamiento de sockets multihilo para simular una comunicación entre un Servidor y un Cliente.

3. Precedente

Antes de abordar de lleno el Laboratorio, se procede a visualizar que todo lo necesario para llevarlo a cabo, esté funcionando correctamente. Para ello se comprueba que las configuraciones de las máquinas virtuales del Laboratorio n° 1, funcionen y tengan una carpeta compartida para todo el cluster.

```

linuxtest@NodoMaestro:~$ pwd
/home/linuxtest
linuxtest@NodoMaestro:~$ ssh linuxtest@LinuxNodo1
Linux LinuxNodo1 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 26 20:44:41 2020 from 192.168.0.10
linuxtest@LinuxNodo1:~$
linuxtest@LinuxNodo1:~$
linuxtest@LinuxNodo1:~$ cerrar sesión
Connection to linuxnodo1 closed.
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$ ssh linuxtest@LinuxNodo2
Linux LinuxNodo2 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 26 20:44:47 2020 from 192.168.0.10
linuxtest@LinuxNodo2:~$
linuxtest@LinuxNodo2:~$ cerrar sesión
Connection to linuxnodo2 closed.
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$

```

```

linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$ ssh linuxtest@LinuxNodo1
Linux LinuxNodo1 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 26 20:49:14 2020 from 192.168.0.10
linuxtest@LinuxNodo1:~$
linuxtest@LinuxNodo1:~$
linuxtest@LinuxNodo1:~$ df -h
S.ficheros          Tamaño Usados  Disp Uso% Montado en
udev                480M         0   480M   0% /dev
tmpfs               99M          3,0M   96M   3% /run
/dev/sda1            6,9G        1,7G   4,8G  27% /
tmpfs               494M         0   494M   0% /dev/shm
tmpfs               5,0M         0    5,0M   0% /run/lock
tmpfs               494M         0   494M   0% /sys/fs/cgroup
NodoMaestro:/home/linuxtest/share 6,9G        1,8G   4,8G  27% /home/linuxtest/share
tmpfs               99M          0    99M   0% /run/user/1000
linuxtest@LinuxNodo1:~$

```

```
linuxtest@NodoMaestro:~$ ssh linuxtest@LinuxNodo2
Linux LinuxNodo2 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 26 20:46:17 2020 from 192.168.0.10
linuxtest@LinuxNodo2:~$
linuxtest@LinuxNodo2:~$ df -h
S.ficheros                Tamaño Usados  Disp Uso% Montado en
udev                      480M      0    480M   0% /dev
tmpfs                     99M      3,0M    96M   3% /run
/dev/sda1                 6,9G    1,7G    4,8G  27% /
tmpfs                     494M      0    494M   0% /dev/shm
tmpfs                     5,0M      0    5,0M   0% /run/lock
tmpfs                     494M      0    494M   0% /sys/fs/cgroup
NodoMaestro:/home/linuxtest/share 6,9G    1,8G    4,8G  27% /home/linuxtest/share
tmpfs                     99M      0    99M   0% /run/user/1000
linuxtest@LinuxNodo2:~$
linuxtest@LinuxNodo2:~$ cerrar sesión
Connection to linuxnodo2 closed.
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$ _
```

Una vez que se comprueba que todo está funcionando correctamente, se procede con el primer punto del Laboratorio nº 2.

4. Pthread

Como se hace mención en los enunciados, se provee de un ejemplo de paralelización el cual realiza una suma de un vector mediante la librería *Pthread*. Para que el código funcione, es necesario que los hilos estén sincronizados entre sí, para que ninguno incurra en la sección crítica del otro; para esto se emplean los denominados algoritmos de exclusión mutua (llamados comunmente *Mutex*).

Visualizamos que en la carpeta compartida, se encuentra el código de ejemplo:

```
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ ls -la  
total 12  
drwxr-xr-x 2 linuxtest linuxtest 4096 oct 26 20:56 .  
drwxr-xr-x 3 linuxtest linuxtest 4096 oct 26 20:55 ..  
-rwxr-xr-x 1 linuxtest linuxtest 2463 oct 26 20:56 pthread.c  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ ssh linuxtest@LinuxNodo1  
Linux LinuxNodo1 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Mon Oct 26 20:57:30 2020 from 192.168.0.10  
linuxtest@LinuxNodo1:~$  
linuxtest@LinuxNodo1:~$ ls -la share/  
ejemplo.c  host  lab2/  matrices  matrices.c  
linuxtest@LinuxNodo1:~$ ls -la share/lab2/  
total 12  
drwxr-xr-x 2 linuxtest linuxtest 4096 oct 26 20:56 .  
drwxr-xr-x 3 linuxtest linuxtest 4096 oct 26 20:55 ..  
-rwxr-xr-x 1 linuxtest linuxtest 2463 oct 26 20:56 pthread.c  
linuxtest@LinuxNodo1:~$ cerrar sesión  
Connection to linuxnodo1 closed.  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$
```

Compilamos utilizando el siguiente comando.

```
$ gcc pthread.c -o hilosPthread -lpthread
```

```
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ gcc pthread.c -o hilosPthread -lpthread  
linuxtest@NodoMaestro:~/share/lab2$
```

Una vez que compilamos el programa, se debe especificar el tamaño de la matriz y el número de hilos que ejecutarán el programa:

```
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ gcc pthread.c -o hilosPthread -lpthread  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ ./hilosPthread 4 4  
Primer bloque 3 ,3  
ultimo bloque 3 ,3  
Primer bloque 2 ,2  
ultimo bloque 2 ,2  
Primer bloque 1 ,1  
ultimo bloque 1 ,1  
Primer bloque 0 ,0  
ultimo bloque 0 ,0  
El resultado es: 10.  
2.002954483032227 milliseconds  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ _
```

Se puede observar el correcto funcionamiento del código, habiéndole pasado 2 parámetros a la hora de invocarlo.

5. OpenMP

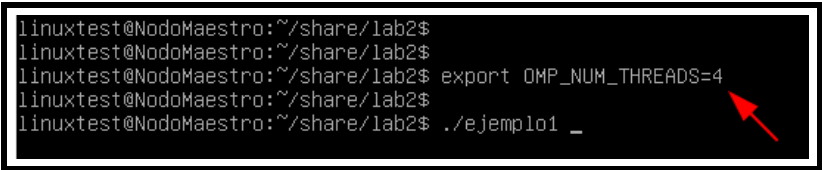
5.1. Ejemplo nº 1

Para poder comprobar el correcto funcionamiento de las librerías OpenMP, la cátedra de **Sistemas Operativos** proporciona dos ejemplos. El primero es un código que crea una cantidad determinada de hilos, esta cantidad debe estar predefinida antes de la ejecución del código, ya que no tiene previsto la posibilidad de asignación dinámica de memoria; se puede asimismo modificar la variable `OMP_NUM_THREADS` con el siguiente comando:

```
$ export OMP_NUM_THREADS=4
```

Una vez finalizado el bloque paralelizable, se imprime una línea que no se encuentra paralelizada indicando el número de hilos con el que se ejecutó.

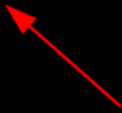
Como primer paso, realizamos la compilación del archivo de ejemplo de OpenMP habiendo modificado la cantidad de hilos previamente:

A terminal window with a black background and white text. The prompt is 'linuxtest@NodoMaestro:~/share/lab2\$'. The user enters 'export OMP_NUM_THREADS=4' and then './ejemplo1 _'. A red arrow points to the end of the second command line.

```
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ export OMP_NUM_THREADS=4  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ ./ejemplo1 _
```

Posteriormente visualizamos la salida de la ejecución:

```
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ export OMP_NUM_THREADS=4  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ ./ejemplo1  
Thread 0 de 4 en marcha  
El thread 0 ha terminado  
Thread 3 de 4 en marcha  
El thread 3 ha terminado  
Thread 2 de 4 en marcha  
El thread 2 ha terminado  
Thread 1 de 4 en marcha  
El thread 1 ha terminado  
A(0) = 10  
A(1) = 11  
A(2) = 12  
A(3) = 13  
A(4) = 0  
A(5) = 0  
A(6) = 0  
A(7) = 0  
A(8) = 0  
A(9) = 0  
A(10) = 0  
A(11) = 0  
A(12) = 0  
A(13) = 0  
A(14) = 0  
A(15) = 0  
A(16) = 0  
A(17) = 0  
A(18) = 0  
A(19) = 0  
A(20) = 0  
A(21) = 0  
A(22) = 0  
A(23) = 0  
linuxtest@NodoMaestro:~/share/lab2$ _
```



5.2. Ejemplo nº 2

Una vez finalizado el primer ejemplo, pasamos al segundo, también suministrado por la cátedra de la materia.

Para este ejemplo, nuevamente se utilizó el compilador GCC, con las siguientes directivas:

```
$ gcc ej2openmp.c -o ejemplo2 -fopenmp
```

A continuación visualizamos el resultado de la compilación:


```
linuxtest@NodoMaestro:~/share/lab2$ cd
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$ cp mnt/Lab2/ej2openmp.c share/lab2/
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$ cd share/lab2/
linuxtest@NodoMaestro:~/share/lab2$ ls -la
total 60
drwxr-xr-x 2 linuxtest linuxtest 4096 oct 26 21:10 .
drwxr-xr-x 3 linuxtest linuxtest 4096 oct 26 20:55 ..
-rwxr-xr-x 1 linuxtest linuxtest 902 oct 26 21:07 ej1openmp.c
-rwxr-xr-x 1 linuxtest linuxtest 900 oct 26 21:10 ej2openmp.c
-rwxr-xr-x 1 linuxtest linuxtest 17176 oct 26 21:07 ejemplo1
-rwxr-xr-x 1 linuxtest linuxtest 17400 oct 26 21:01 hilosPthread
-rwxr-xr-x 1 linuxtest linuxtest 2463 oct 26 20:56 pthread.c
linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$ gcc ej2openmp.c -o ejemplo2 -fopenmp
linuxtest@NodoMaestro:~/share/lab2$
```

Una vez compilado, procedemos a ejecutar el programa:

```
linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$ ./ejemplo2 2 2
Openmp
Openmp
El resultado es 4
linuxtest@NodoMaestro:~/share/lab2$ _
```

6. Sockets

Para poder ejemplificar el funcionamiento de sockets, la cátedra proveyó de un programa que consiste emular el comportamiento entre un cliente y un servidor, donde el cliente creará los hilos y estos son los encargados de realizar las solicitudes de conexión al servidor, el cual para esto crea un socket de conexión y envía la solicitud al servidor, con los datos del cliente sumándole el mensaje “hello”.

6.1. Ejemplo

Para que el programa se ejecute del mismo modo que un cliente-servidor en máquinas distintas, es necesario realizar una modificación en la dirección IP del servidor, debiéndose reemplazar la dirección 127.0.0.1 por la dirección IP de la máquina que hará de servidor, en este caso la máquina Maestro, cuya dirección es 192.169.0.10.

Visualizamos los archivos copiados a la carpeta compartida:

```

linuxtest@NodoMaestro:~$ cp mnt/Lab2/socket_
socket_client.c socket_server.c
linuxtest@NodoMaestro:~$ cp mnt/Lab2/socket_* share/lab2/
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$ ls share/lab2/
ej1openmp.c  ejemplo1  hilosPthread  socket_client.c
ej2openmp.c  ejemplo2  pthread.c    socket_server.c
linuxtest@NodoMaestro:~$
linuxtest@NodoMaestro:~$ _

```

A continuación se visualiza la modificación antes mencionada:

```

23  serverAddr.sin_family = AF_INET;
24  //Set port number, using htons function
25  serverAddr.sin_port = htons(8080);
26  //Set IP address to localhost
27  //serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
28  serverAddr.sin_addr.s_addr = inet_addr('192.168.1.10');
29  memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
30  //Connect the socket to the server using the address
31  addr_size = sizeof serverAddr;
32  connect(clientSocket, (struct sockaddr *) &serverAddr, addr_size);
33  strcpy(message, "Hello");
34  if( send(clientSocket, message, strlen(message), 0) < 0)
35  {
36      printf("Send failed\n");
37  }

```

Dentro del servidor, se genera un socket quien será el que atienda las peticiones para poder conectarse con el cliente, posteriormente crea los hilos de ejecución que se encargarán de enviar el mensaje de saludo al cliente.

Procedemos a compilar los archivos, tanto del cliente como del servidor con el siguiente comando:

```
$ gcc socket_client.c -o cliente -lnsl -pthread
```

Observamos que la compilación resultó exitosa:

```

linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$ gcc socket_server.c -o servidor -lnsl -pthread
linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$ gcc socket_client.c -o cliente -lnsl -pthread
linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$ _

```

Como se comentó anteriormente, en la máquina Maestro se ejecutará el código del Servidor, y en los nodos, el código Cliente. A continuación se visualiza los distintos códigos ejecutados en las distintas máquinas virtuales.

6.1.1. En en Nodo Maestro

```
Espera paquete
Llega al accept
Espera paquete
Llega al accept
Espera paquete
Llega al accept
Espera paquete
Llega al accept
Espera paquete
Llega al accept
Espera paquete
Entro en el thread
Recibió del cliete: Hello
Se envía : Hello Client : Hello

Entro en el thread
Entro en el thread
Entro en el thread
Entro en el thread
Exit socketThread
Recibió del cliete: Hello
Se envía : Hello Client : Hello

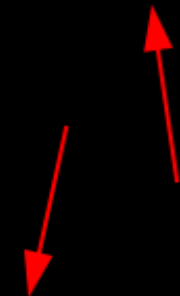
Recibió del cliete: Hello
Se envía : Hello Client : Hello

Recibió del cliete: Hello
Se envía : Hello Client : Hello

Exit socketThread
Exit socketThread
Exit socketThread
Exit socketThread
```

6.1.2. En en Nodo Esclavo

```
linuxtest@LinuxNodo1:~/share/lab2$  
linuxtest@LinuxNodo1:~/share/lab2$  
linuxtest@LinuxNodo1:~/share/lab2$ ./cliente  
In thread  
Creo el socket  
In thread  
Creo el socket  
In thread  
Creo el socket  
In thread  
Creo el socket  
In thread  
Creo el socket  
Data received: Hello Client : Hello  
^C  
linuxtest@LinuxNodo1:~/share/lab2$
```



6.2. Chat-Room con Pthread

Al igual que en el ejercicio anterior, se debe modificar el código del cliente en la parte que respecta a la dirección IP, modificando la dirección 127.0.0.1 por la IP de la máquina servidor, la cual será en este caso la IP del nodo Maestro 192.168.1.10:

```
79 // Socket information  
80 struct sockaddr_in server_info, client_info;  
81 int s_addrlen = sizeof(server_info);  
82 int c_addrlen = sizeof(client_info);  
83 memset(&server_info, 0, s_addrlen);  
84 memset(&client_info, 0, c_addrlen);  
85 server_info.sin_family = PF_INET;  
86 //server_info.sin_addr.s_addr = inet_addr("127.0.0.1");  
87 server_info.sin_addr.s_addr = inet_addr("192.168.0.10");  
88 server_info.sin_port = htons(8888);  
89  
90 // Connect to Server  
91 int err = connect(sockfd, (struct sockaddr *)&server_info, s_addrlen);  
92 if (err == -1) {  
93     printf("Connection to Server error!\n");  
94     exit(EXIT_FAILURE);  
95 }
```

Una vez realizada esta modificación, se procede a compilar el programa utilizando el comando “make”, el cual permite la generación automática de los códigos compilados:

```

linuxtest@NodoMaestro:~/share/lab2$ make
gcc -O3 -Wall -c src/server.c
In file included from src/server.c:12:
src/server.h: In function 'newNode':
src/server.h:17:5: warning: 'strncpy' specified bound 16 equals destination size [-Wstringop-truncation]
    strncpy(np->ip, ip, 16);
    ^~~~~~
In function 'newNode',
    inlined from 'main' at src/server.c:135:25:
src/server.h:17:5: warning: 'strncpy' specified bound 16 equals destination size [-Wstringop-truncation]
    strncpy(np->ip, ip, 16);
    ^~~~~~
gcc -O3 -Wall -pthread -o server.out server.o
gcc -O3 -Wall -c src/client.c
gcc -O3 -Wall -c src/string.c
gcc -O3 -Wall -pthread -o client.out client.o string.o
linuxtest@NodoMaestro:~/share/lab2$

```

Una vez compilado todo, se debe ejecutar el código del Servidor en el Maestro, y en cada esclavo, se ejecuta el código del cliente.

6.2.1. Código ejecutado en el Maestro

```

linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$
linuxtest@NodoMaestro:~/share/lab2$ ./server.out
Start Server on: 0.0.0.0:8888

```

6.2.2. Código ejecutado en el Nodo1

```

linuxtest@LinuxNodo1:~/share/lab2$ ./client.out
Please enter your name: Jose
Connect to Server: 192.168.0.10:8888
You are: 192.168.0.11:49472
>

```

6.2.3. Código ejecutado en el Nodo2

```

linuxtest@LinuxNodo2:~/share/lab2$ ./client.out
Please enter your name: Emiliano
Connect to Server: 192.168.0.10:8888
You are: 192.168.0.12:49006
>

```

Una vez ejecutado el código del cliente, se solicitará un nombre de sesión para el chat. El servidor, no solo podrá ver los usuarios que se conecten, sino que también podrá visualizar los mensajes que se envíen entre los clientes.

6.2.4. En la sala de chat desde el Maestro

```
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$  
linuxtest@NodoMaestro:~/share/lab2$ ./server.out  
Start Server on: 0.0.0.0:8888  
Client 192.168.0.12:49006 come in.  
Emiliano(192.168.0.12)(4) join the chatroom.  
Client 192.168.0.11:49472 come in.  
Jose(192.168.0.11)(5) join the chatroom.  
Send to sockfd 4: "Jose(192.168.0.11) join the chatroom."
```

Podemos ver el chat entre los clientes también:

```
linuxtest@NodoMaestro:~/share/lab2$ ./server.out  
Start Server on: 0.0.0.0:8888  
Client 192.168.0.12:49006 come in.  
Emiliano(192.168.0.12)(4) join the chatroom.  
Client 192.168.0.11:49472 come in.  
Jose(192.168.0.11)(5) join the chatroom.  
Send to sockfd 4: "Jose(192.168.0.11) join the chatroom."  
Send to sockfd 4: "Jose♦ Hola from 192.168.0.11"  
Send to sockfd 5: "Emiliano♦ Hola Jose from 192.168.0.12"  
Send to sockfd 4: "Jose♦ como andas? from 192.168.0.11"  
Send to sockfd 5: "Emiliano♦ todo bien che, ¿vos? from 192.168.0.12"  
Send to sockfd 4: "Jose♦ todo genial, haciendo lo de sistemas distribuidos from 192.168.0.11"  
-
```

El código del servidor lo que realiza es una conexión TCP para los puertos 8888 de cada máquina que solicite la conexión. Del lado del cliente además de solicitar una conexión, realiza el envío y recepción de mensajes por lo que se crean los hilos para que lleven a cabo esta función.

7. Chat-room con OpenMP

Una vez que se realizaron las pruebas pertinentes y aceptando que todo funciona correctamente, se pasa a realizar la modificación propuesta en el último punto para las librerías de OpenMP, para permitir una paralelización entre mensajes.

Lo que se hizo fue inhabilitar las líneas que hacían uso de los pthread, paralelizando las funciones de envío y recepción de los mensajes del lado del cliente. De esta forma, se definió que un hilo por vez podrá ejecutar el código provisto para el cliente.

Una aclaración que vale la pena, es que hubo problemas en la utilización del archivo *Makefile*, ya que al pasarle los flags, de *fopenmp*, se generaba un error que sólo pudo ser solventado compilando los archivos mediante la línea de comando.¹

7.1. Código del Servidor y compilacion

Se muestra el código modificado y comentado:

¹Se intentó realizar la siguiente solución al problema planteado en el siguiente link:
<https://stackoverflow.com/questions/733926/openmp-coding-warning-ignoring-pragma-omp-parallel>

```

133 // La cantidad de hilos y de conexiones que soportara el servidor
134 int OMP_NUM_THREADS=3;
135
136 // Esto es la parte paralelizable -----
137 #pragma omp parallel num_threads(OMP_NUM_THREADS)
138 {
139
140     // Acepta la solicitud de conexion del cliente
141     client_sockfd = accept(server_sockfd, (struct sockaddr*)&client_info, (socklen_t*)&c_addrln);
142
143     // Se imprime la conexion del cliente
144     getpeername(client_sockfd, (struct sockaddr*)&client_info, (socklen_t*)&c_addrln);
145     printf("Client %s:%d come in.\n", inet_ntoa(client_info.sin_addr), ntohs(client_info.sin_port));
146
147     // Se genera una lista para los clientes entrantes
148     // Genera un lista de todas las solicitudes de conexion
149     ClientList *c = newNode(client_sockfd, inet_ntoa(client_info.sin_addr));
150     printf("la lista es ");
151     c->prev = now;
152     now->link = c;
153     now = c;
154
155     // Funcion que se encarga de enviar los mensajes a cada nodo.
156     client_handler( (void *)c);
157 }
158 //-----

```

A continuación se visualiza el código compilado mediante la línea de comando:

```

linuxtest@NodoMaestro:~/share/lab2/src_openMP$
linuxtest@NodoMaestro:~/share/lab2/src_openMP$ gcc server.c -o server.out -fopenmp
linuxtest@NodoMaestro:~/share/lab2/src_openMP$
linuxtest@NodoMaestro:~/share/lab2/src_openMP$ _

```

7.2. Código del Cliente y compilacion

Como se explicó en la sección , al tener que compilarse el código de manera manual (sin hacer uso del archivo *Makefile* y por lo tanto tampoco del archivo *string.c*) se descomentan el código perteneciente a las funciones sobre el tratamiento de las cadenas de caracteres:

```

24 _
25 //Funciones para el tratamiento de las cadenas. Se descomentan porque no se
26 //utiliza el archivo string.c
27 void str_trim_lf (char* arr, int length) {
28     int i;
29     for (i = 0; i < length; i++) { // trim \n
30         if (arr[i] == '\n') {
31             arr[i] = '\0';
32             break;
33         }
34     }
35 }
36
37 //Se descomenta porque no se utiliza el archivo string.c
38 void str_overwrite_stdout() {
39     printf("\n%s", "> ");
40     fflush(stdout);
41 }
42

```

Se visualiza la parte modificada del código del Cliente:

```

123
124 // Definicion de la cantidad de hilos para el envío de mensajes
125 int OMP_NUM_THREADS=3;
126
127 //Se define el codigo que sera paralelizable, y se le pasa como
128 //argumento la cantidad de hilos a ejecutar
129 #pragma omp parallel num_threads(OMP_NUM_THREADS)
130 {
131
132     #pragma omp sections
133     {
134         // Definicion de seccion un solo hilo se encargara de
135         // ejecutar el codigo
136         #pragma omp section
137         {
138             send_msg_handler();// Se envia el mensaje
139             exit(EXIT_FAILURE);
140         }
141
142         // Un sólo hilo se encarga de ejecutar esta seccion
143         #pragma omp section
144         {
145             recv_msg_handler();// Recepcion de mensajes
146             exit(EXIT_FAILURE);
147         }
148     }
149 }
150

```

A continuación se visualiza el código compilado mediante la línea de comando:

```

linuxtest@NodoMaestro:~/share/lab2/src_openMP$
linuxtest@NodoMaestro:~/share/lab2/src_openMP$
linuxtest@NodoMaestro:~/share/lab2/src_openMP$ gcc client.c -o client.out -fopenmp
linuxtest@NodoMaestro:~/share/lab2/src_openMP$
linuxtest@NodoMaestro:~/share/lab2/src_openMP$
linuxtest@NodoMaestro:~/share/lab2/src_openMP$ _

```

7.3. Conexión al Servidor e intercambio de mensajes entre Clientes

A continuación podemos observar cómo es que cada uno de los clientes al conectarse al servidor, puede efectivamente intercambiar mensajes entre si. Desde el lado del Servidor es posible observar que se van imprimiendo en pantalla tanto las conexiones que se van gestionando como también los mensajes intercambiados entre cada cliente.

Se pone a correr el código del Servidor en el Nodo Maestro:

```

linuxtest@NodoMaestro:~/share/lab2/src_openMP$
linuxtest@NodoMaestro:~/share/lab2/src_openMP$
linuxtest@NodoMaestro:~/share/lab2/src_openMP$ ./server.out
Start Server on: 0.0.0.0:8888

```

En el Nodo Esclavo nº 1 se ejecuta el código del Cliente

```

linuxtest@LinuxNodo1: ~/share/lab2/src_openMP$
linuxtest@LinuxNodo1:~/share/lab2/src_openMP$
linuxtest@LinuxNodo1:~/share/lab2/src_openMP$
linuxtest@LinuxNodo1:~/share/lab2/src_openMP$ ./client.out
Please enter your name: JOSE
Connect to Server: 192.168.0.10:8888
You are: 192.168.0.11:59894
>

```


En el Nodo Esclavo nº 2 se ejecuta el código del Cliente

```
linuxtest@LinuxNodo2: ~/share/lab2/src_openMP$  
linuxtest@LinuxNodo2:~/share/lab2/src_openMP$  
linuxtest@LinuxNodo2:~/share/lab2/src_openMP$ ./client.out  
Please enter your name: EMILIANO  
Connect to Server: 192.168.0.10:8888  
You are: 192.168.0.12:38018  
>
```

Podemos observar cómo desde el lado del Servidor, se visualizan las conexiones:

```
linuxtest@NodoMaestro:~/share/lab2/src_openMP$  
linuxtest@NodoMaestro:~/share/lab2/src_openMP$ ./server.out  
Start Server on: 0.0.0.0:8888  
Client 192.168.0.12:38018 come in.  
la lista es EMILIANO(192.168.0.12)(4) join the chatroom.  
Client 192.168.0.11:59894 come in.  
la lista es JOSE(192.168.0.11)(5) join the chatroom.  
Send to sockfd 4: "JOSE(192.168.0.11) join the chatroom."
```

Observamos el chat realizado por el Nodo Esclavo nº 1 y su posterior salida:

```
linuxtest@LinuxNodo1:~/share/lab2/src_openMP$ ./client.out  
Please enter your name: JOSE  
Connect to Server: 192.168.0.10:8888  
You are: 192.168.0.11:59894  
EMILIANO♦ Hola Jose from 192.168.0.12  
> Hola Emiliano ¿como estas?  
EMILIANO♦ Todo bien Jose ¿vos? from 192.168.0.12  
> bien, gracias por preguntar  
> Me despido. Abrazo  
> exit  
linuxtest@LinuxNodo1:~/share/lab2/src_openMP$ _
```

Observamos el chat realizado por el Nodo Esclavo nº 2 y su posterior salida:

```
linuxtest@LinuxNodo2:~/share/lab2/src_openMP$  
linuxtest@LinuxNodo2:~/share/lab2/src_openMP$ ./client.out  
Please enter your name: EMILIANO  
Connect to Server: 192.168.0.10:8888  
You are: 192.168.0.12:38018  
JOSE(192.168.0.11) join the chatroom.  
> Hola Jose  
JOSE♦ Hola Emiliano ¿como estas? from 192.168.0.11  
> Todo bien Jose ¿vos?  
JOSE♦ bien, gracias por preguntar from 192.168.0.11  
JOSE♦ Me despido. Abrazo from 192.168.0.11  
JOSE♦ exit from 192.168.0.11  
JOSE(192.168.0.11) leave the chatroom.  
> Saludos  
> exit  
linuxtest@LinuxNodo2:~/share/lab2/src_openMP$ _
```

Vemos desde el lado del Servidor cómo todo el chat ingresado por los Nodos Esclavos, se ve impreso por pantalla:

```

linuxtest@NodoMaestro: ~/share/lab2/src_openMP$ ./server.out
Start Server on: 0.0.0.0:8888
Client 192.168.0.12:38018 come in.
la lista es EMILIANO(192.168.0.12)(4) join the chatroom.
Client 192.168.0.11:59894 come in.
la lista es JOSE(192.168.0.11)(5) join the chatroom.
Send to sockfd 4: "JOSE(192.168.0.11) join the chatroom."
Send to sockfd 5: "EMILIANO# Hola Jose from 192.168.0.12"
Send to sockfd 4: "JOSE# Hola Emiliano ¿como estas? from 192.168.0.11"
Send to sockfd 5: "EMILIANO# Todo bien Jose ¿vos? from 192.168.0.12"
Send to sockfd 4: "JOSE# bien, gracias por preguntar from 192.168.0.11"
Send to sockfd 4: "JOSE# Me despido. Abrazo from 192.168.0.11"
Send to sockfd 4: "JOSE# exit from 192.168.0.11"
JOSE(192.168.0.11)(5) leave the chatroom.
Send to sockfd 4: "JOSE(192.168.0.11) leave the chatroom."
EMILIANO(192.168.0.12)(4) leave the chatroom.

```

8. Conclusiones

Luego de haber realizado las pruebas pertinentes para corroborar el buen funcionamiento de la gestión de hilos, se puede concluir que si bien tanto *Pthreads* como *OpenMP* representan dos paradigmas de multiprocesamiento totalmente distintos, la primera permite tener a mano una API de bajo nivel para trabajar con subprocesos, por lo que se tiene un control muy detallado sobre la administración de estos procesos, siendo la segunda librería de alcances más generales, pensada para aportar mayor portabilidad, sin la necesidad de limitarse al uso del lenguaje C. Esta última hace más fácil su escalabilidad, permitiendo dividir la ejecución en varios subprocesos con relativa facilidad.

Aún así, en el bajo nivel, *OpenMP* se termina implementando como *pthread*s, siendo la librería *Pthreads* más sencillas de ejecutar para las secciones paralelas.

Bibliografía

- [1] Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw Hill, 2003.