

2012

Documentación Externa

Primera Tarea Programada

Estudiantes:

Yader Morales.

María Mercedes Escalante.

Frank Brenes.

Profesor:

Andrei Fuentes Leiva.



Índice

Descripción del problema.....	3
Diseño del programa	4
Librerías usadas.....	6
stdio.h	6
sys/types.h & sys/socket.h.....	6
netinet/in.h.....	6
unistd.h	6
stdlib.h	6
string.h	6
netdb.h.....	7
signal.h.....	7
Análisis de resultados.....	8
Manual de usuario (instrucciones uso, de ejecución y de compilación)	9
Conclusión personal	14
Bibliografía	15

Descripción del problema

El objetivo principal del desarrollo de la primera tarea programada fue la creación de un programa de mensajería instantánea utilizando el lenguaje de programación C en el sistema operativo Linux.

Para la realización de la tarea programada se presentan dos temas importantes por investigar, los cuales son de suma importancia para la correcta implementación del programa de mensajería.

El primer tema, se refiere a la comunicación de dos programas en una red, se refiere a la implementación de sockets TCP en el entorno Linux, estos permiten el intercambio de mensajes instantáneos entre dos computadoras.

Dentro del segundo tema investigado encontramos la división de procesos, para lo cual utilizamos `fork()`, el cual permite ejecutar los procesos de lectura y escritura al mismo tiempo.

Las características principales del problema son las siguientes:




- ✚ Tomar los dos puertos y el IP como parámetro.
- ✚ Identificación de mensajes de envío y recibido.
- ✚ Capacidad de enviar mensajes de forma continúa.
- ✚ Finalizar la sesión y cerrar los sockets con el envío o recepción del mensaje "Adios".

Diseño del programa

Al empezar el programa, este debe correr sobre un archivo y desde él se debe crear el servidor y el cliente, por lo que lo primero a utilizar dentro de la tarea programada es la función `fork()`, la cual explicaremos a continuación.

Esta función lo que realiza es bifurcar un proceso en dos idénticos (de tal manera que sea en una relación hijo, padre) con el cual se crea el cliente y el servidor. Dentro de una estructura de control `if` al inicio del mismo. La creación del Socket del servidor se encuentra en la función `servidor`, la cual recibe como parámetro el puerto que corresponde a la máquina que actuará como servidor.

Con respecto a las estructuras de los sockets, en la web encontramos tutoriales para la definición de los mismos, empezando por `sockaddr_in direccionServidor` que nos brinda la referencia al resto de los elementos del socket por parte del servidor, que comprenden:

```
 direccionServidor.sin_family = AF_INET; (AF_INET utiliza protocolo ARPA de internet)  
 direccionServidor.sin_addr.s_addr = INADDR_ANY; (no se utiliza dirección de internet)  
 direccionServidor.sin_port = htons(puertoServidor); (se asigna el puerto por medio de htons)
```

La función **htons** se encarga de convertir un valor de nodo a una variable corta de red, utilizada por `direccionServidor.sin_port`, cabe destacar que la aplicación de esta función se encuentra dentro de la función `servidor`. Esta implementación de `htons` se puede representar como la siguiente instrucción:

```
direccionServidor.sin_port = htons (puertoServidor);
```

A partir de esta estructura creada, se definen varias funciones como el `bind()` la cual se encarga de la unión del socket del servidor con el entrante y la función `connect()`. La función `fork()` se encargará de crear una copia idéntica de cliente-servidor, con atributos distintos a los de su padre.

En el cliente y en el servidor, existen estructuras `while (1)`, que van a crear un loop infinito para poder escuchar de los puertos, además de poder leer las cadenas de caracteres de los mismos. Una estructura de estas, almacena una función `connect()` la cual es la encargada de esperar conexión con el servidor para poder arrancar el servicio de mensajería.

El procedimiento de finalización de los sockets, también se encuentra en una estructura `while(1)` esperando la cadena de caracteres "Adiós", cuando esta sea capturada y la condición `strcmp(buffer, escape)== 0` se cumpla; tomando en cuenta de que `escape` se determina como una variable que contiene "Adios\n" para poder comparar y determinar cuándo se debe terminar el programa, se escribirá en el cliente-servidor la cadena con el mensaje, además se tomarán los ID de procesos creados por el `fork()` con el `getpid()`.

Estos identificadores se suman y se restan de acuerdo a si es servidor o cliente, por la razón de que Linux al crear los nuevos procesos los crea juntos, por lo tanto para poderles aplicar la función kill () se debe determinar el número de proceso para ser terminado.

Librerías usadas

stdio.h

Librería estándar de entrada/salida.

sys/types.h & sys/socket.h

Librería que contiene funciones que sirven para habilitar la comunicación entre computadoras, es decir, habilitar el socket.

La función socket () genera el proceso de comunicación entre usuarios.

La función bind () asocia el socket creado con el puerto de posee nuestra máquina.

La función connect () se utiliza para conectar a un puerto definido en una dirección IP.

La función listen () se usa esperar conexiones entrantes, es decir, esperar a que se quieran conectar a nuestra maquina.

La función accept () se utilizar para aceptar y conseguir la comunicación entrante.

netinet/in.h

Librería de familia del protocolo de Internet. Es una estructura que se utiliza para almacenar las direcciones de la familia de protocolos de Internet. Estos valores se deben convertir para poder utilizarlas con las interfaces de socket.

unistd.h

Biblioteca que contiene la función del fork(), proceso de bifurcación que se encarga de convertir el proceso en dos procesos idénticos llamados padre e hijo.

La función exit() se encarga de finalizar el proceso de bifurcación.

stdlib.h

La biblioteca que posee funciones resolver ciertos cálculos matemáticos y cadenas de manipulación.

La función atoi convierte una cadena de caracteres a un número entero. Se utiliza para convertir los números de puerto recompilados en consola a enteros y así manipularlos en el programa.

string.h

Librería que implementa las funciones bzero y bcopy.

La función bzero()se utiliza para llenar de ceros el array de caracteres y así limpiar la cadena de caracteres y desechar el mensaje anterior.

La función bcopy() hace una copia de bytes para capturar el mensaje del puerto asociado.

netdb.h

Biblioteca para el uso de base de datos en red. En esta librería se utilizan las funciones `hostent` que tiene información del estado del servidor y `gethostbyname` que obtiene las direcciones IP del servidor.

signal.h

Librería para el uso de la función `kill`, la cual es una señal de interrupción, la cual es necesaria para finalizar los procesos cliente y servidor.

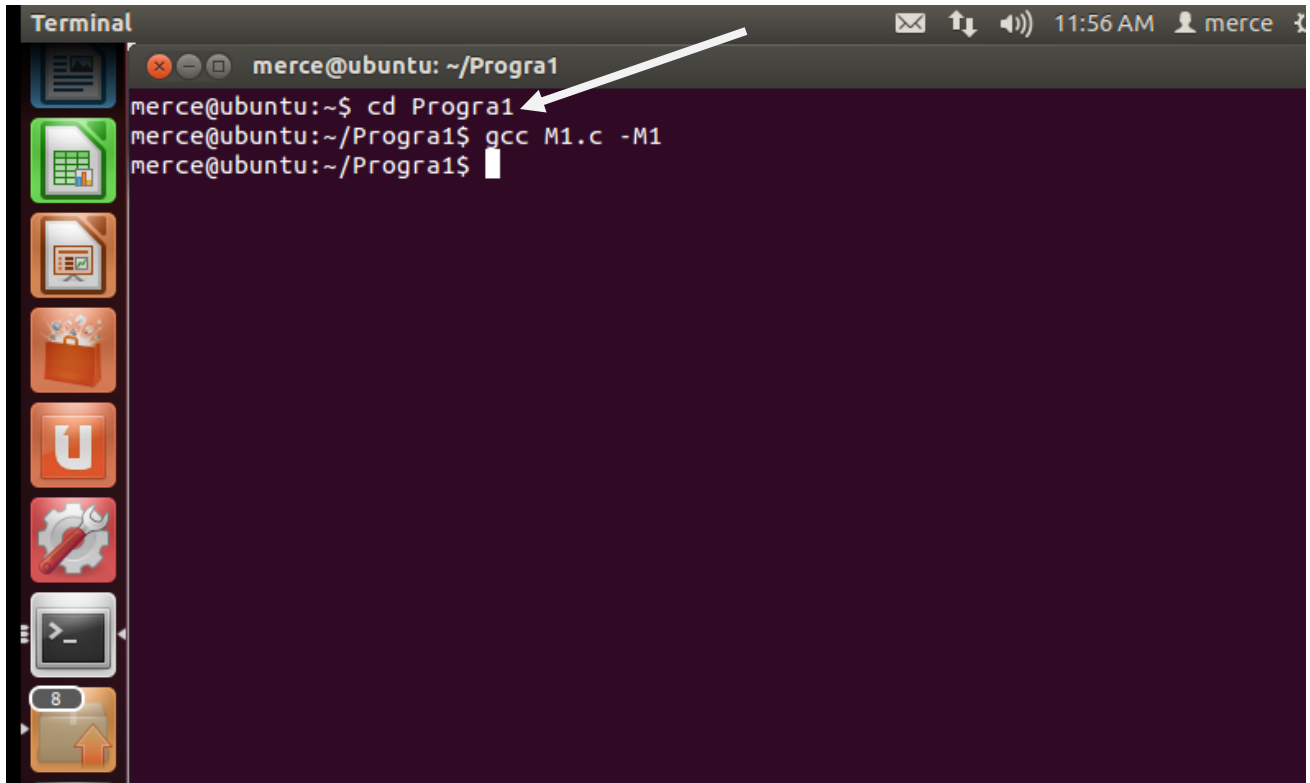
Análisis de resultados

Dentro de los objetivos que fueron alcanzados dentro de la tarea programada I tenemos:

- ✚ El programa es capaz de crear conexión en una computadora host, además de crear conexión en una red LAN sin enrutador con su debida configuración.
- ✚ El programa cierra los sockets de servidor y cliente en el momento que alguno de los usuarios manda la palabra clave “Adiós”.
- ✚ El usuario puede enviar mensajes los cuales cuentan con un color al desplegarse en la pantalla.
- ✚ Los puertos a utilizar por el programa a nivel de LOCAL HOST son 8080 – 8000 y a nivel de LAN serían 8080 8082.
- ✚ Los puertos a utilizar en el programa son los utilizados por otros sistemas de mensajería (8080), aunque si se desea, se pueden definir nuevos puertos, en la tabla de services de UNIX.

Manual de usuario (instrucciones uso, de ejecución y de compilación)

Primero accedemos a la carpeta en donde se encuentra el archivo cd "nombre de carpeta", como se muestra a continuación:

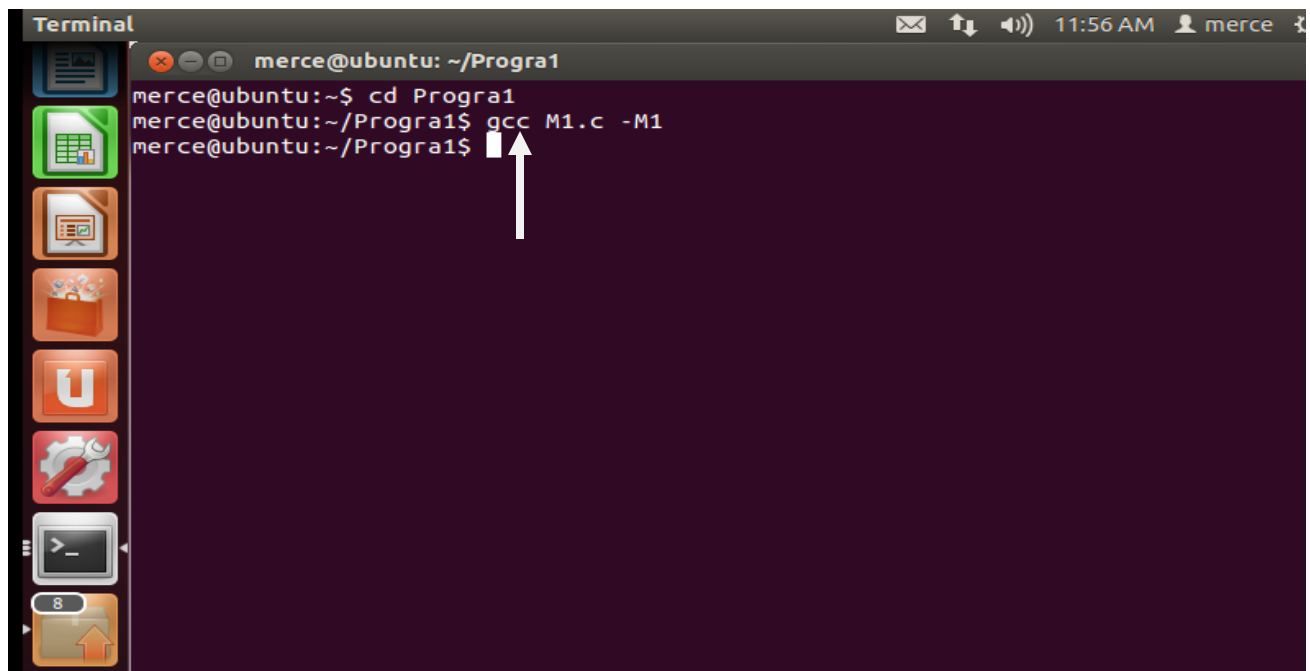


A terminal window titled "Terminal" with a dark purple background. The window shows the user "merce" at "ubuntu" in the directory "~/Progra1". The command prompt is "merce@ubuntu:~\$". The user enters "cd Progra1", and the prompt changes to "merce@ubuntu:~/Progra1\$". Then, the user enters "gcc M1.c -M1", and the prompt returns to "merce@ubuntu:~/Progra1\$". A white arrow points from the text "nombre de carpeta" in the preceding paragraph to the "Progra1" in the terminal command.

```
Terminal
merce@ubuntu: ~/Progra1
merce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ gcc M1.c -M1
merce@ubuntu:~/Progra1$
```

Los usuarios primero que todo deberán compilar el programa en consola o en cualquier otro compilador, por ejemplo:

gcc M1.c – M1

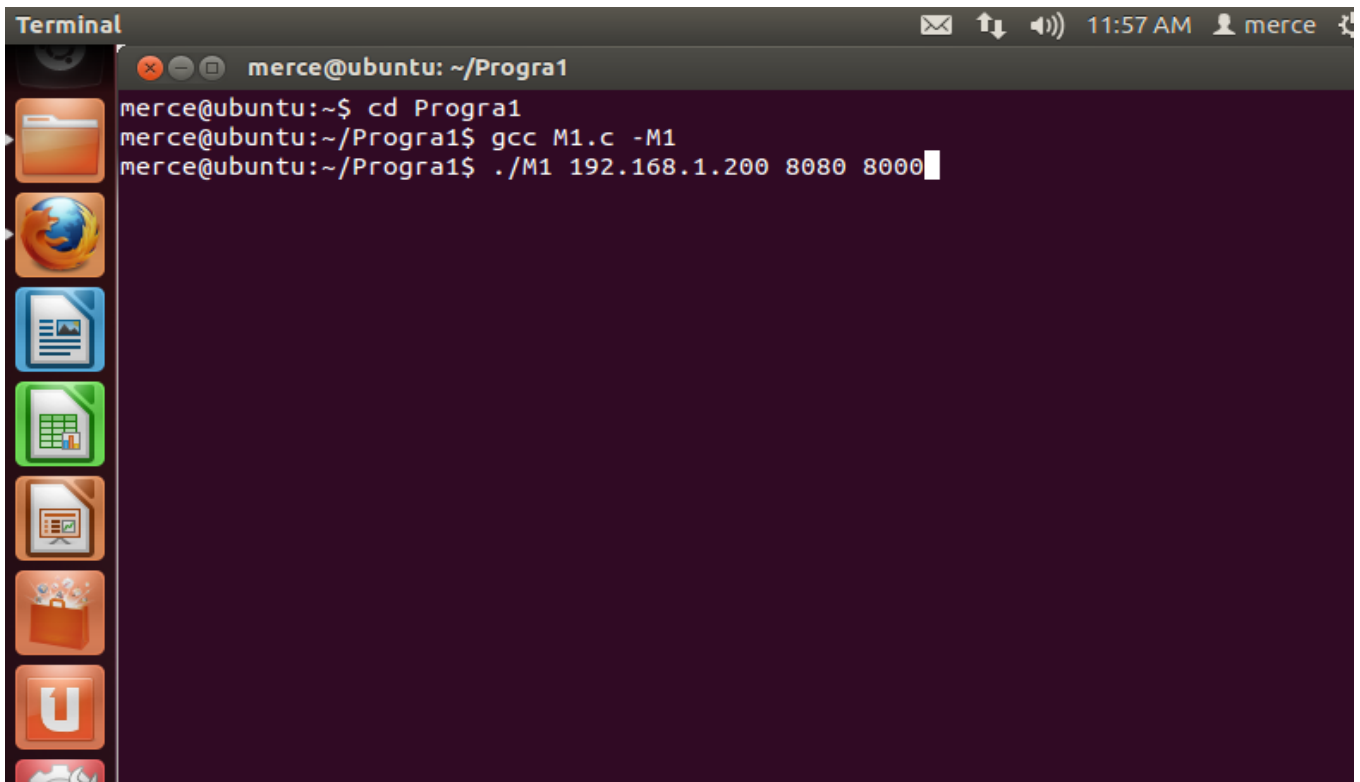


A terminal window titled "Terminal" with a dark purple background. The window shows the user "merce" at "ubuntu" in the directory "~/Progra1". The command prompt is "merce@ubuntu:~\$". The user enters "cd Progra1", and the prompt changes to "merce@ubuntu:~/Progra1\$". Then, the user enters "gcc M1.c -M1", and the prompt returns to "merce@ubuntu:~/Progra1\$". A white arrow points from the text "por ejemplo:" in the preceding paragraph to the "gcc" command in the terminal.

```
Terminal
merce@ubuntu: ~/Progra1
merce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ gcc M1.c -M1
merce@ubuntu:~/Progra1$
```

El compilador le generará un archivo a.out el cual por default lo genera con el nombre a.out si gusta cambiarle el nombre deberá de escribir lo siguiente:

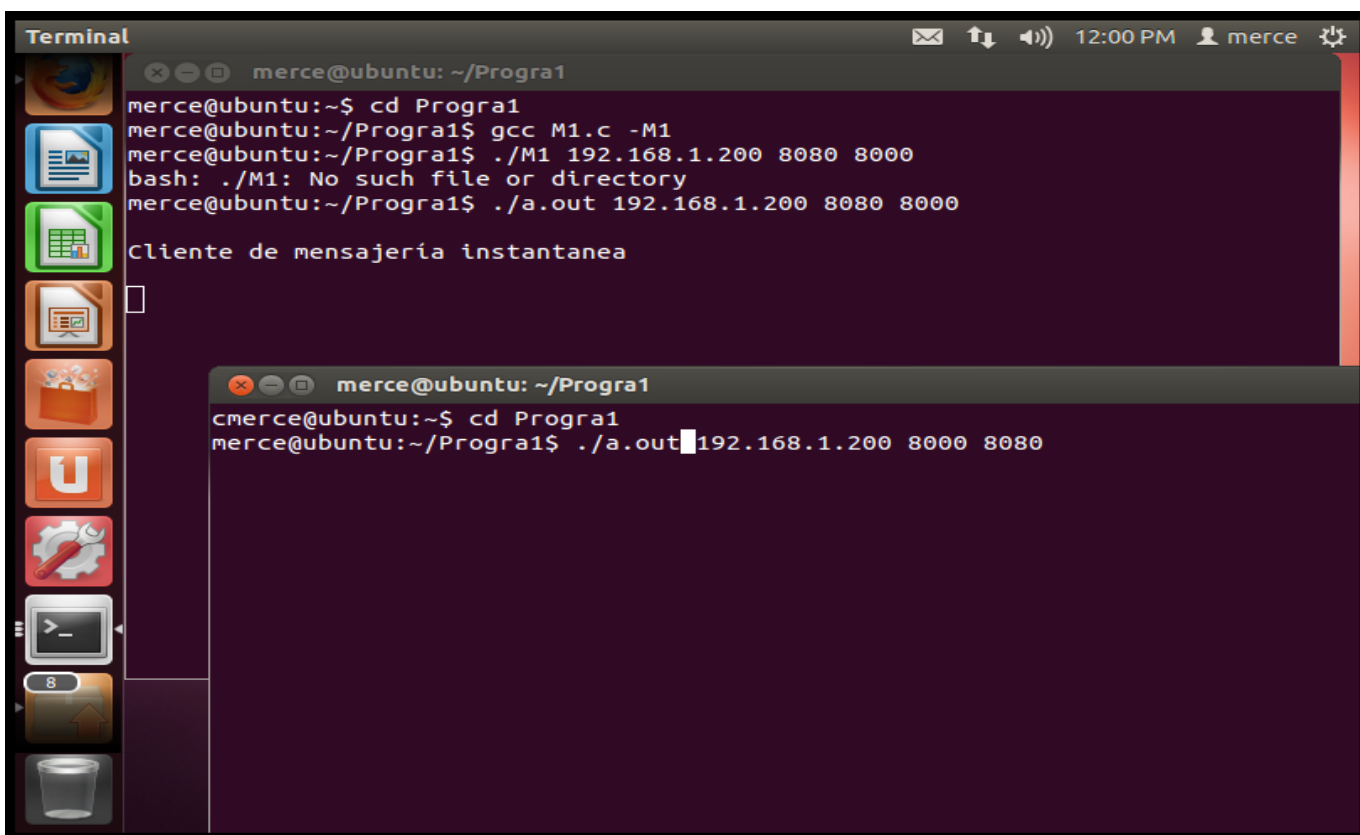
gcc TP1.c -o M1.out



A terminal window titled "Terminal" with a dark purple background. The window shows the following commands and output:

```
merce@ubuntu: ~/Progra1
merce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ gcc M1.c -M1
merce@ubuntu:~/Progra1$ ./M1 192.168.1.200 8080 8000
```

Así el nombre que le pondrá al archivo ya compilado va a ser M1.out.



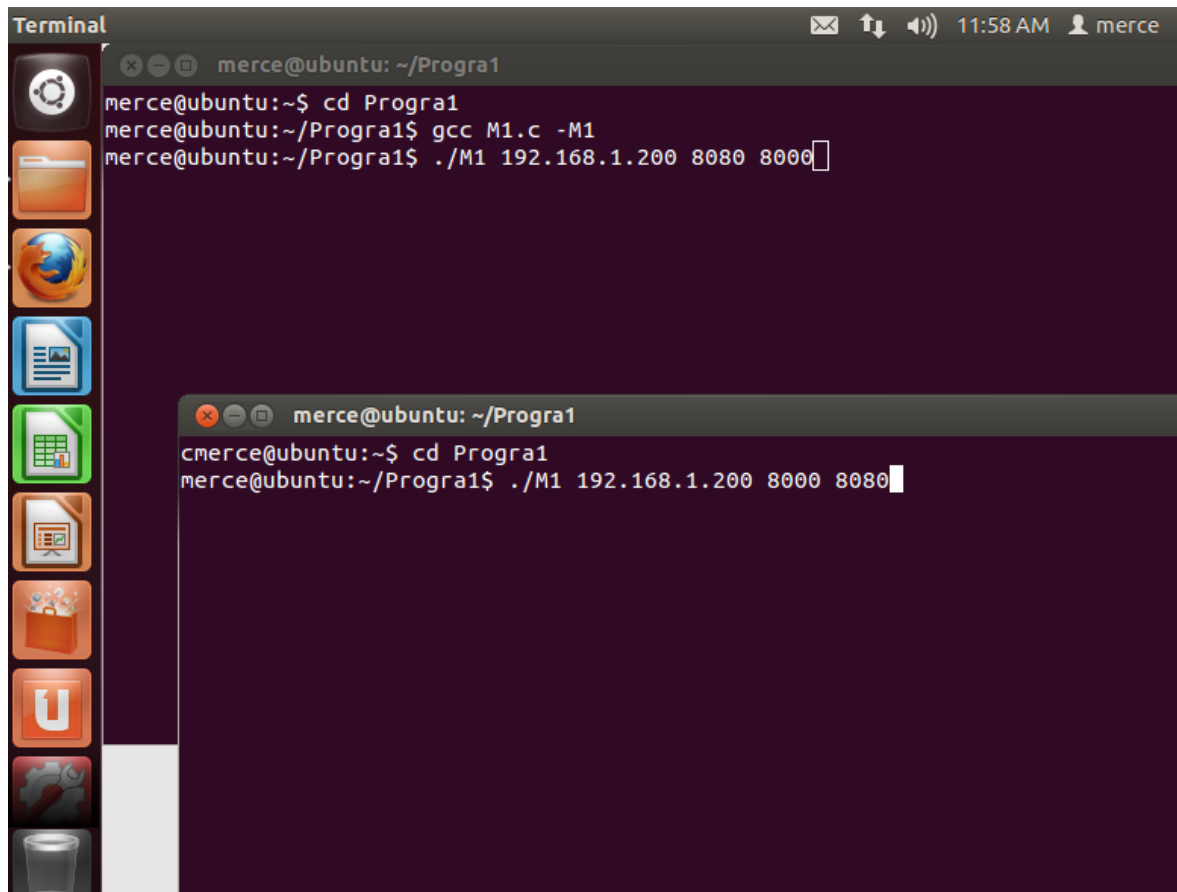
A terminal window titled "Terminal" with a dark purple background. The window shows the following commands and output:

```
merce@ubuntu: ~/Progra1
merce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ gcc M1.c -M1
merce@ubuntu:~/Progra1$ ./M1 192.168.1.200 8080 8000
bash: ./M1: No such file or directory
merce@ubuntu:~/Progra1$ ./a.out 192.168.1.200 8080 8000
Cliente de mensajería instantanea

```

Below the main terminal window, there is a smaller terminal window showing the same commands and output as the main window.

Ya generado el archivo lo podemos ejecutar de la siguiente manera. Para ejecutarlo correctamente el usuario debe digitar la dirección IP a la cual se va a conectar, el puerto y el puerto de salida al cual se conectará.



The image shows a terminal window titled "Terminal" with a dark purple background. The window has a title bar with standard Ubuntu window controls and system status icons (mail, network, volume, time 11:58 AM, user merce). On the left side, there is a vertical dock with icons for Dash, Home, Firefox, LibreOffice Writer, LibreOffice Calc, LibreOffice Impress, a folder, the Ubuntu logo, and a glass. The terminal content shows the following commands and output:

```
merce@ubuntu: ~/Progra1
merce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ gcc M1.c -M1
merce@ubuntu:~/Progra1$ ./M1 192.168.1.200 8080 8080
```

Below the main terminal window, there is a smaller, overlapping terminal window with the same title bar and dock. It shows the same commands and output, but with a slight delay in the execution command:

```
merce@ubuntu: ~/Progra1
merce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ ./M1 192.168.1.200 8080 8080
```

Una vez que tanto el servidor como el cliente ponen las direcciones IP y los puertos de entrada y salida se establece la conexión entre ambos, por lo que se pueden enviar mensajes desde el servidor o desde el cliente.

```
Terminal
merce@ubuntu: ~/Progra1
merce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ gcc M1.c -M1
merce@ubuntu:~/Progra1$ ./M1 192.168.1.200 8080 8000
bash: ./M1: No such file or directory
merce@ubuntu:~/Progra1$ ./a.out 192.168.1.200 8080 8000

Cliente de mensajería instantanea

Se ha establecido conexion

```

```
merce@ubuntu: ~/Progra1
cmerce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ ./a.out 192.168.1.200 8000 8080

Cliente de mensajería instantanea

Se ha establecido conexion

```

De esta manera una vez establecida la conexión ambas partes pueden hablar hasta que se escriba la palabra de escape.

```
Terminal
merce@ubuntu: ~/Progra1
merce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ gcc M1.c -M1
merce@ubuntu:~/Progra1$ ./M1 192.168.1.200 8080 8000
bash: ./M1: No such file or directory
merce@ubuntu:~/Progra1$ ./a.out 192.168.1.200 8080 8000

Cliente de mensajería instantanea

Se ha establecido conexion
Mensaje Recibido: Hola!
Mensaje Recibido: como estas??

```

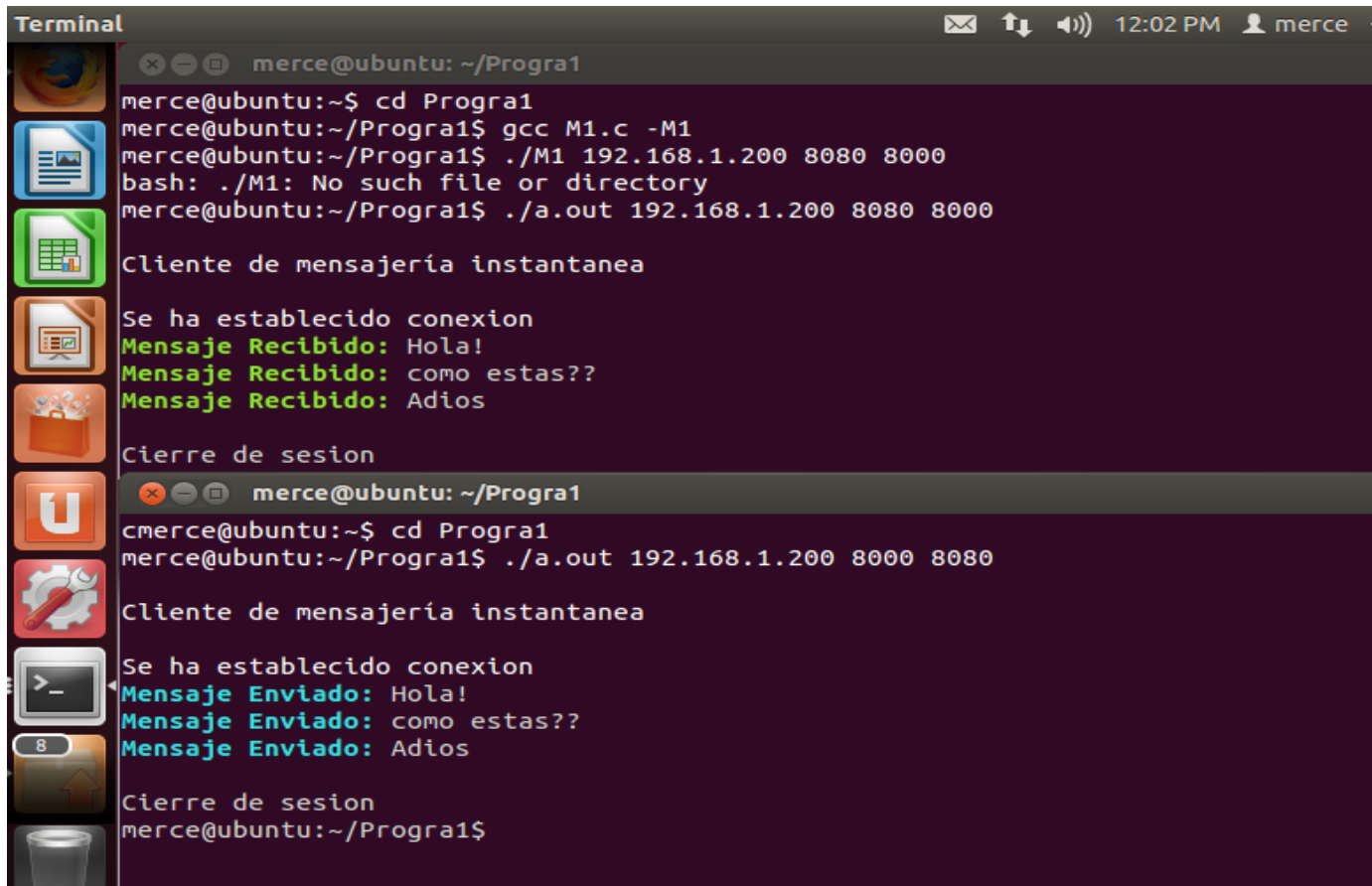
```
merce@ubuntu: ~/Progra1
cmerce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ ./a.out 192.168.1.200 8000 8080

Cliente de mensajería instantanea

Se ha establecido conexion
Mensaje Enviado: Hola!
Mensaje Enviado: como estas??

```

Una vez que alguno de los usuarios digita la palabra de escape, la cual corresponde a la palabra “Adios”, el programa manda la palabra y cierra ambas conexiones, tanto la del cliente como la del servidor, además, en cada una de las pantallas aparece un mensaje de comprobación de que ambas conexiones se han cerrado con éxito.



```
Terminal
merce@ubuntu: ~/Progra1
merce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ gcc M1.c -M1
merce@ubuntu:~/Progra1$ ./M1 192.168.1.200 8080 8000
bash: ./M1: No such file or directory
merce@ubuntu:~/Progra1$ ./a.out 192.168.1.200 8080 8000

Cliente de mensajería instantanea

Se ha establecido conexion
Mensaje Recibido: Hola!
Mensaje Recibido: como estas??
Mensaje Recibido: Adios

Cierre de sesion

merce@ubuntu: ~/Progra1
cmerce@ubuntu:~$ cd Progra1
merce@ubuntu:~/Progra1$ ./a.out 192.168.1.200 8000 8080

Cliente de mensajería instantanea

Se ha establecido conexion
Mensaje Enviado: Hola!
Mensaje Enviado: como estas??
Mensaje Enviado: Adios

Cierre de sesion
merce@ubuntu:~/Progra1$
```

Conclusión personal

La I Tarea Programada fue una herramienta para aprender y aplicar nuestros conocimientos de programación en un nuevo paradigma de programación, lo que nos permitió asociarnos a una plataforma desconocida de desarrollo, descubriendo prácticamente los beneficios y déficits del paradigma funcional y lenguaje de programación C.

Durante el desarrollo de la tarea programada fomentamos habilidades claves para ser programadores exitosos, tales como la investigación de funciones, las cuales se basaban en facilitar la comunicación entre computadores (sockets, direcciones IP, puertos), bifurcación de procesos (fork) y manipulación de diferentes tipos de datos.

En esta tarea realizamos muchas acciones que nos ayudaron a poder mejorar nuestros conocimientos relacionados al lenguaje C, además de que el proyecto nos llamó mucho la atención por su cantidad de aplicaciones hoy en día. De la misma manera, con base en el desarrollo de la tarea desarrollamos mejores prácticas de programación.

Es importante recalcar, que con el desarrollo de esta tarea no solo aprendimos sobre distintos puntos de vista para resolver un mismo problema; si no también reforzamos el trabajo en equipo, el desarrollo de algoritmos en conjunto, conocimientos de cada uno de los integrantes y lo más importante nos enseñó a motivarnos entre nosotros para lograr concluir una tarea en común.

Bibliografía

Multitarea en C usando Fork para OS tipo Unix (2012, 08 de Marzo). Recuperado el 4 de Setiembre del 2012, de <http://www.lastdragon.net/?p=180>

Fgets (2012, 08 de Marzo). Recuperado el 1 de Setiembre del 2012, de <http://pubs.opengroup.org/onlinepubs/009695399/functions/fgets.html>

Funciones Importantes (s. f.). Recuperado el 1 de Setiembre del 2012, de <http://www.ibiblio.org/pub/linux/docs/LuCaS/Tutoriales/PROG-SOCKETS/prog-sockets/x142.html>

Manual de C tutorial de c (2012, 08 de Marzo). Recuperado el 1 de Setiembre del 2012, de <http://www.programatium.com/manuales/c/21.htm>

Manual de C tutorial de c (2012, 08 de Marzo). Recuperado el 3 de Setiembre, de <http://www.programatium.com/manuales/c/21.htm#SECTION00211000000000000000>

Procesos (2012, 08 de Marzo). Recuperado el 3 de Setiembre, de <http://www.slideshare.net/jinsi2035/procesos-1352629>

Procesos en C. Función kill (2012, 09 de Marzo). Recuperado el 3 de Setiembre del 2012, de <http://www.mailxmail.com/curso-informatica-sincronizacion/procesos-c-funcion-kill>

Programación Básica de Sockets en Unix para Novatos (s. f.). Recuperado el 4 de Setiembre del 2012, de <http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets.html>

Programación Básica de Sockets en Unix para Novatos, Diferentes tipos de sockets en Internet (s. f.). Recuperado el 1 de Setiembre del 2012, de <http://www.ibiblio.org/pub/linux/docs/LuCaS/Tutoriales/PROG-SOCKETS/prog-sockets/x31.html>

Programación de redes en Linux: Sockets en C (s. f.). Recuperado el 4 de Setiembre del 2012, de <http://www.reloco.com.ar/linux/prog/redes.html>

Socket de Internet - Wikipedia, la enciclopedia libre (s. f.). Recuperado el 5 de Setiembre del 2012, de http://es.wikipedia.org/wiki/Socket_de_Internet

Sockets en C de Unix/Linux (2012, 08 de Marzo). Recuperado el 10 de Setiembre del 2012, de http://www.chuidiang.com/clinix/sockets/sockets_simp.php

Sockets Tutorial (s. f.). Recuperado el 6 de Setiembre del 2012, de http://www.linuxhowtos.org/C_C++/socket.htm

Función Signal C (s. f.). Recuperado el 4 de Setiembre del 2012, de <http://docs.mis-algoritmos.com/c.funcion.signal.html>

Protocolos de Comunicación (s. f.). Recuperado el 4 de Setiembre del 2012, de <http://www.forest.ula.ve/~mana/cursos/redes/protocolos.html>

Cambiar color de texto en consola (s. f.). Recuperado el 10 de Setiembre del 2012, de <http://mclinows.wordpress.com/2009/10/02/cambiar-color-de-texto-en-consola-cc/>

COMO PUEDO CAMBIAR EL COLOR DE TEXTO EN C/C++ (s. f.). Recuperado el 11 de Setiembre del 2012, de http://www.lawebdelprogramador.com/foros/C_Visual_C/12583-COMO_PUEDO_CAMBIAR_EL_COLOR_DE_TEXTO_EN_C_C++.html