TECNOLÓGICO DE COSTA RICA
LICENCIATURA EN ADMINISTRACIÓN DE TECNOLOGÍAS DE INFORMACIÓN
LENGUAJES DE PROGRAMACIÓN

Mensajería Instantánea

Tarea Programada I

Evelyn Madriz Mata
Jose Daniel Chacón Bogarín

13 de Setiembre del 2012

Tabla de contenido

Descripción del Problema

El siguiente trabajo, basado en el desarrollo de una mensajería instantánea en lenguaje C, consiste en la implementación de un mecanismo de comunicación de procesos según una modalidad cliente-servidor orientada a una conexión de protocolo confiable o TCP mediante sockets. Esto para el curso de Lenguajes de Programación del Tecnológico de Costa Rica.

Dicha aplicación será capaz de atender, mediante un servidor, las solicitudes de de un cliente y desplegar en pantalla el mensaje recibido y enviado de ambos procesos (Cliente y Servidor respectivamente).

La interacción entre el cliente y servidor es síncrona por lo que se hace el uso de un método específico para la subdivisión o duplicación de procesos, esto permite un comportamiento aceptable de la aplicación.

Según el paradigma cliente-servidor, el primero es quien está activo y mantiene la iniciativa de iniciar el diálogo con el servidor enviando peticiones y obteniendo una respuesta. El servidor por otra parte, es quien está pasivo y espera las peticiones de los clientes para procesarlas y ofrecer una respuesta.

Tanto los mensajes enviados como los recibidos serán entremezclados en la Terminal, y serán identificados por quién los envíe con un respectivo color.

La palabra "Adios" será la que determine el fin de la comunicación y cierre de los sockets respectivamente.

Diseño del programa

Dentro de las decisiones de diseño de la aplicación, se implementó un solo archivo con extensión .c el cual contiene tanto las instrucciones del código cliente como las del servidor. Estos contienen todas las librerías y estructuras específicas para el uso de sockets en Unix.

El programa, el cual es ejecutado en consola, pide datos necesarios para inicializar los procesos, en este caso el IP y los puertos involucrados.

En el caso del cliente:

Las estructuras son usadas en la programación de sockets para almacenar información sobre direcciones, una de ellas es `struct sockaddr` la cual contiene información del socket, otra es `struct sockaddr_in` la cual nos ayuda hacer referencia a los elementos del socket y el `struct hostent` la cual nos permite

obtener por medio de la función `gethostbyname,` con la dirección IP como parámetro, la dirección de Internet para la conexión. Dichas estructuras son implementadas en la aplicación y son documentadas respectivamente.

Una de las funciones fundamentales para la creación y cierre de sockets, es la función `socket,` la cual crea un extremo de la comunicación y devuelve un int como el valor del conector. Dentro de sus parámetros recibe el dominio (usualmente AF_INET), tipo (especifica la semántica de la comunicación, en este caso SOCK_STREAM) y un valor 0 asignado a un protocolo.

Si la creación del socket es exitosa, se le asignarán valores a los atributos del socket como lo son el número de puerto, la dirección IP y la familia de direcciones en este caso AF_INET. Esto por medio de la estructura dada anteriormente: `sockaddr_in.`

Seguidamente, la función `connect` se usa para conectarse a un puerto definido en una dirección IP, en este caso según los datos apuntados por la estructura que se acaba de definir.

Si la función se realiza satisfactoriamente, se envía el mensaje específicamente.

Para ello entra a un ciclo en el que se captura el texto correspondiente mediante la función `gets` y se compara que si lo que se desea enviar es el arreglo de caracteres "Adios", para lo cual se cierra el socket. Y en caso contrario se enviaría el mensaje mediante la función `send.`

En el caso del servidor:

En el caso del servidor, se mantiene una estructura similar a la del cliente en cuanto a las estructuras. Se hace doble uso de `sockaddr_in` para cliente y servidor y no se hace el uso del `struct hostent.`

Se implementa la función `socket` y se asignan los valores correspondientes a la estructura del servidor.

Seguidamente se hace uso de la función `bind` la cual asocia el socket dado a la dirección local especificada, esto para que el socket quede asignado al puerto especificado en la misma.

Finalmente, y por medio de la función `listen` se especifica que el socket dado anteriormente desea aceptar conexiones y se continua con el proceso.

Como resultado de una comunicación síncrona, se decide implementar el uso de la función `fork` la cual parte de un proceso original, en este caso el archivo .c.

En este caso el proceso original puede seguir atendiendo clientes que quieran conectarse a un socket, mientras que el proceso hijo atiende a un cliente que acaba de conectarse.

Si se atiende a un servidor específico, se entrará en un ciclo en el cual se recibirán los datos del cliente y si el texto ingresado como respuesta es diferente al arreglo de caracteres "Adios" se mantendrá la comunicación entre ambos procesos. De lo contrario, se saldrá del sistema.

Esta decisión de diseño implica que se duplique todo el espacio de memoria, sin repercutir a las variables que estén en uno u otro proceso.

Librerías usadas

- #include <sys/socket.h>
- #include <sys/types.h>
- #include <stdio.h>
- #include <stdlib.h>
- #include <netdb.h>
- #include <unistd.h>
- #include <sys/wait.h>
- #include <string.h>

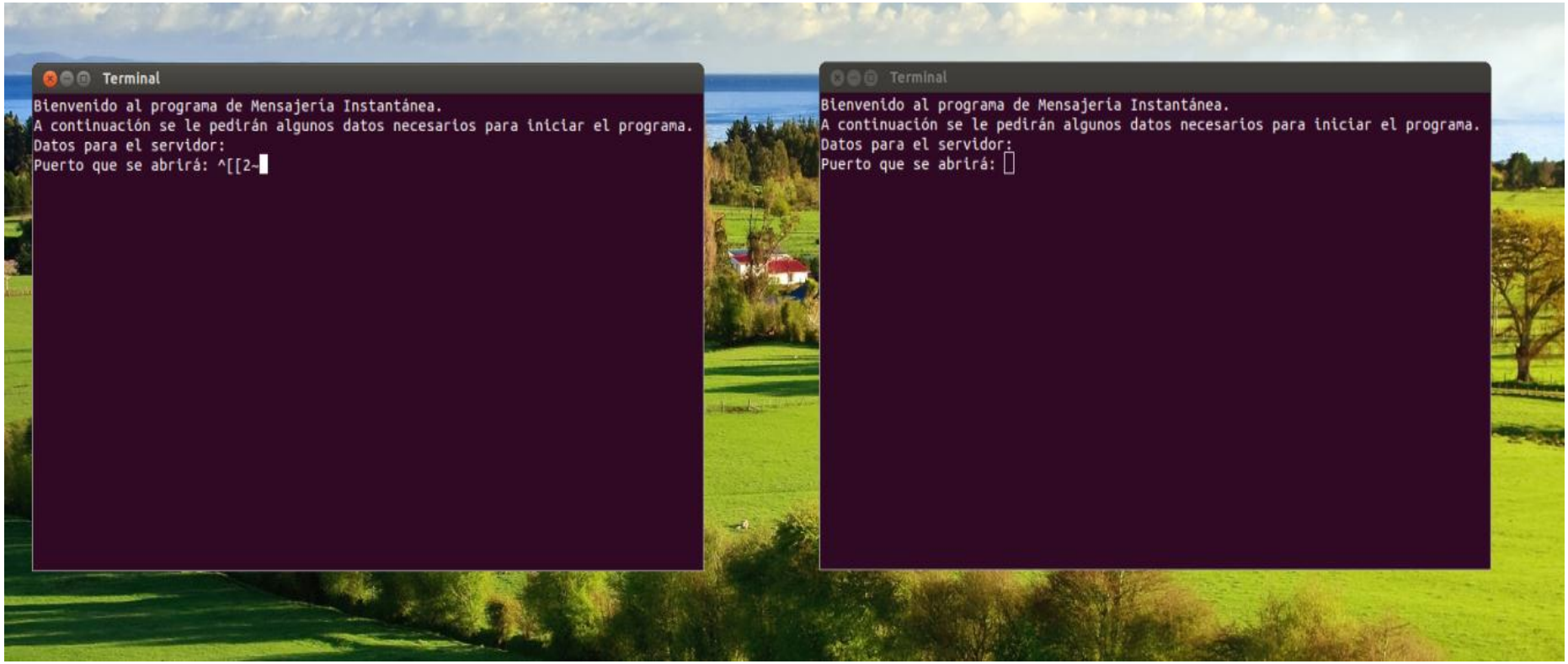
Análisis de resultados

Se alcanzaron los objetivos propuestos.

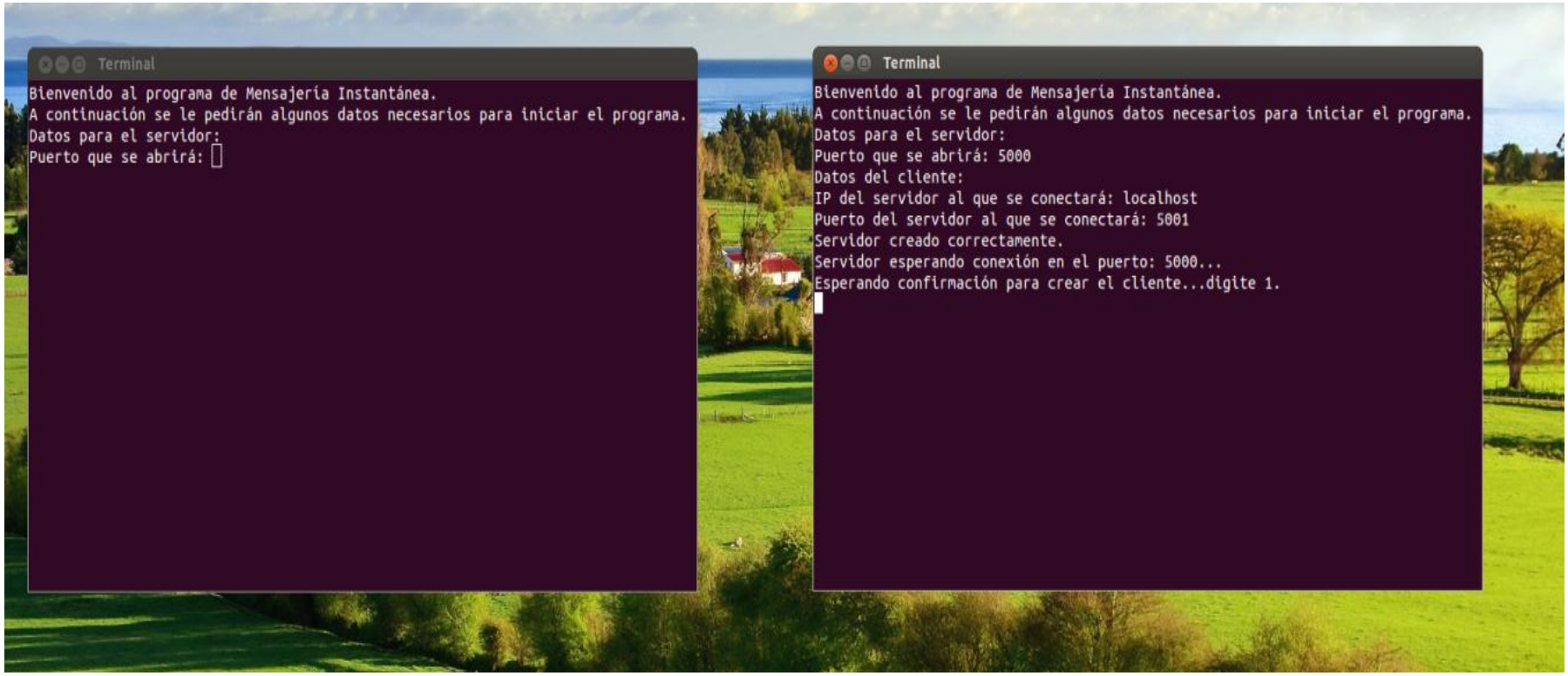
Manual de usuario

Al ejecutar la aplicación, la terminal pedirá los datos necesarios para la apertura de comunicación entre sockets.

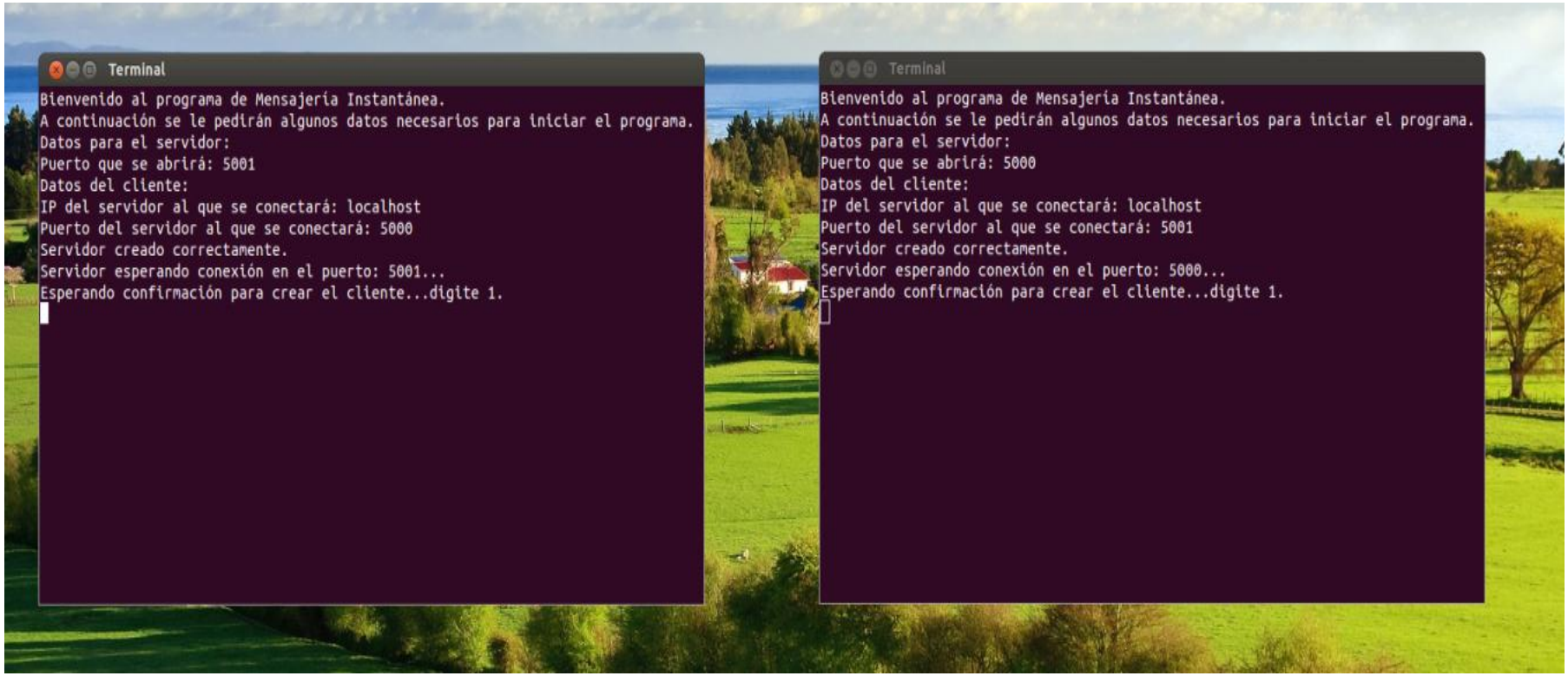
A continuación se muestra el panorama general de la aplicación:



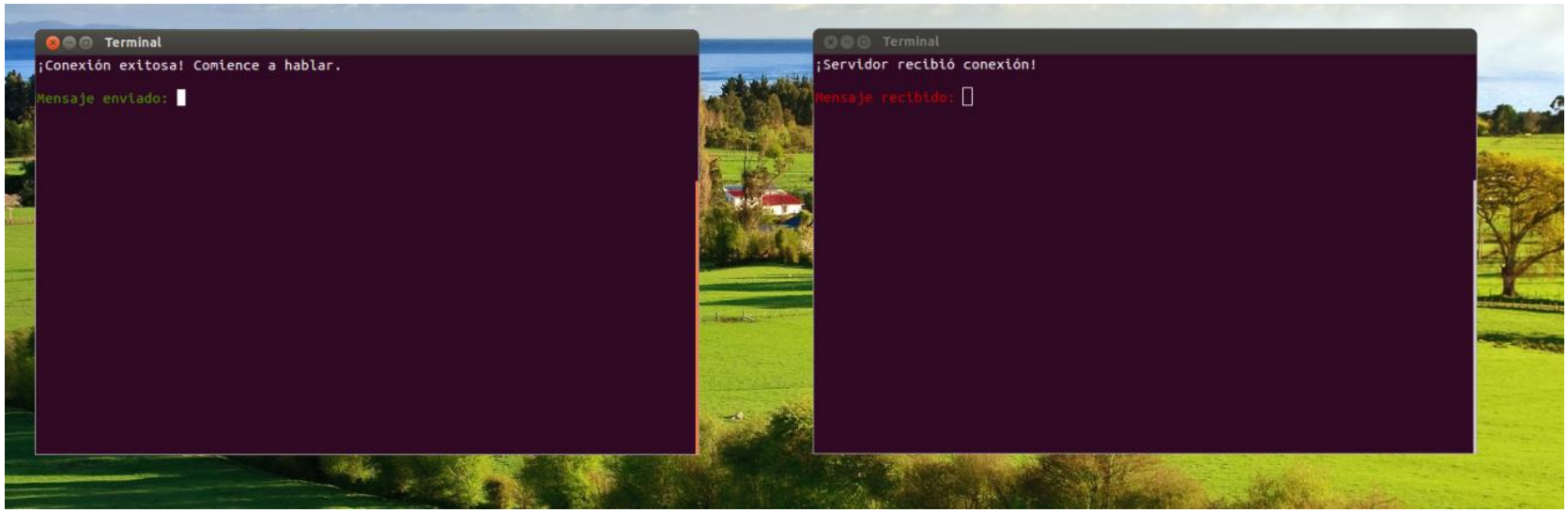
Al ingresar los datos necesarios en la primera conexión, aparecerá el mensaje “Esperando conexión para cerrar el cliente…digite 1”, esto porque espera a que el otro extremo de la comunicación esté habilitado.

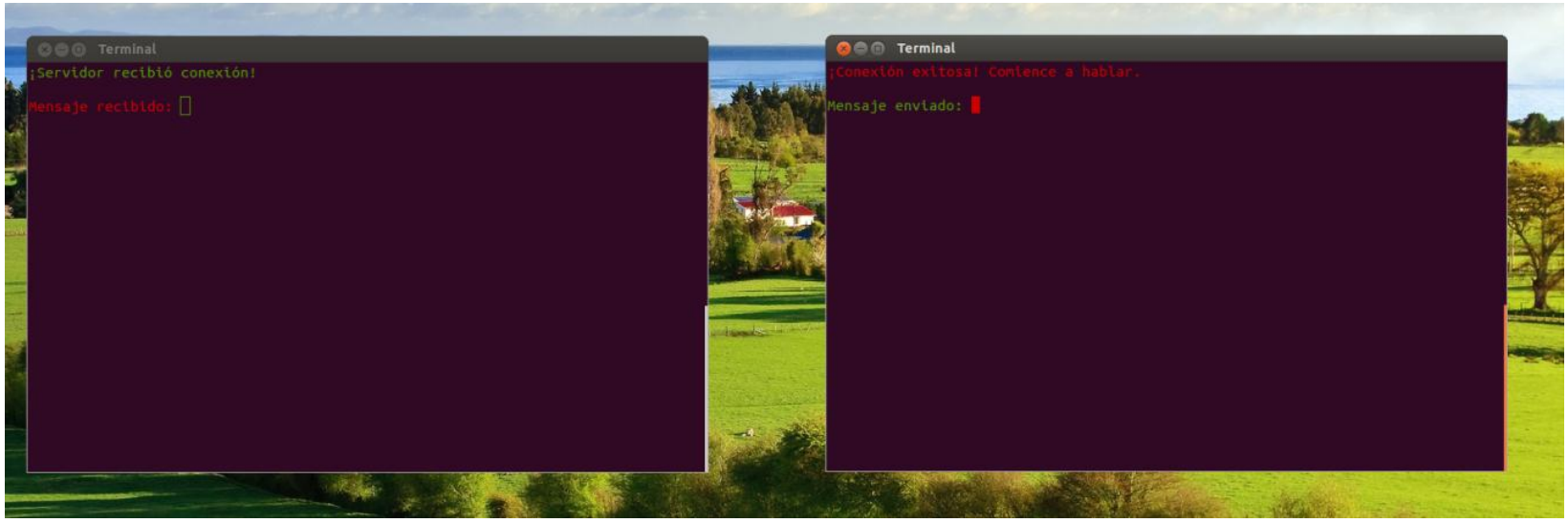


Para ingresar el 1 ambas terminales deben indicar el mismo mensaje.

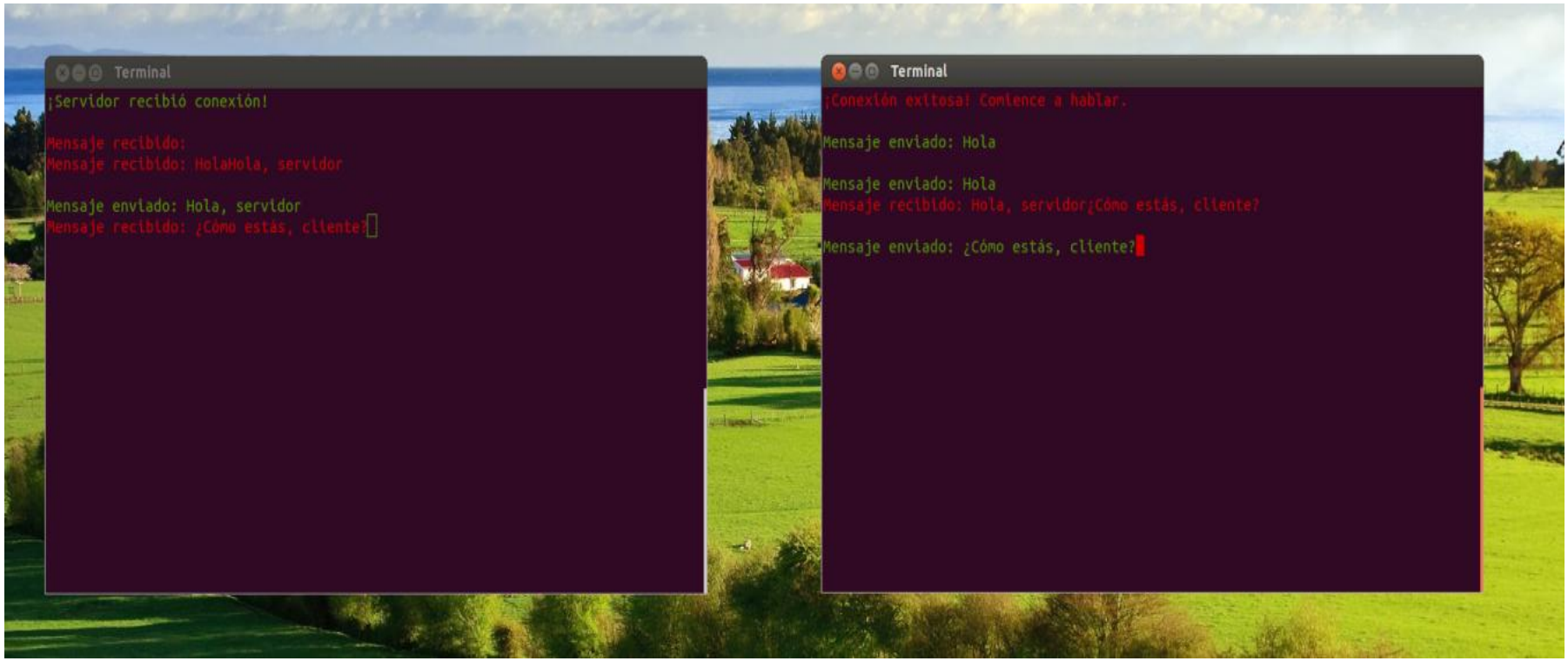


Se deberá digitar el 1 para ambos procesos, para lo cual se mostrará el siguiente resultado:





E iniciar la conversación correspondiente,



Las validaciones para terminar la conversación, están tomadas en cuenta.

Conclusión personal

A lo largo del desarrollo de la tarea, se lograron los objetivos planeados.

La interacción con el lenguaje C fue fácil, sin embargo resultó ser más compleja que algunos paradigmas y lenguajes antes vistos, esto por la instalación de múltiples componentes para el desarrollo e implementación y por el acceso a características de bajo nivel del lenguaje.

La búsqueda de recursos en la web contribuyó al desarrollo de la aplicación sin embargo, el contenido de material físico en la biblioteca de la universidad resultó escaso.

Referencias

http://www.gta.ufrj.br/ensino/eel878/sockets/sockaddr_inman.html

http://www.linuxhowtos.org/manpages/2/socket.htm

http://www.beej.us/guide/bgnet/output/html/multipage/sockaddr_inman.html

http://www.beej.us/guide/bgnet/output/html/multipage/bindman.html

en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

http://shoe.bocks.com/net/