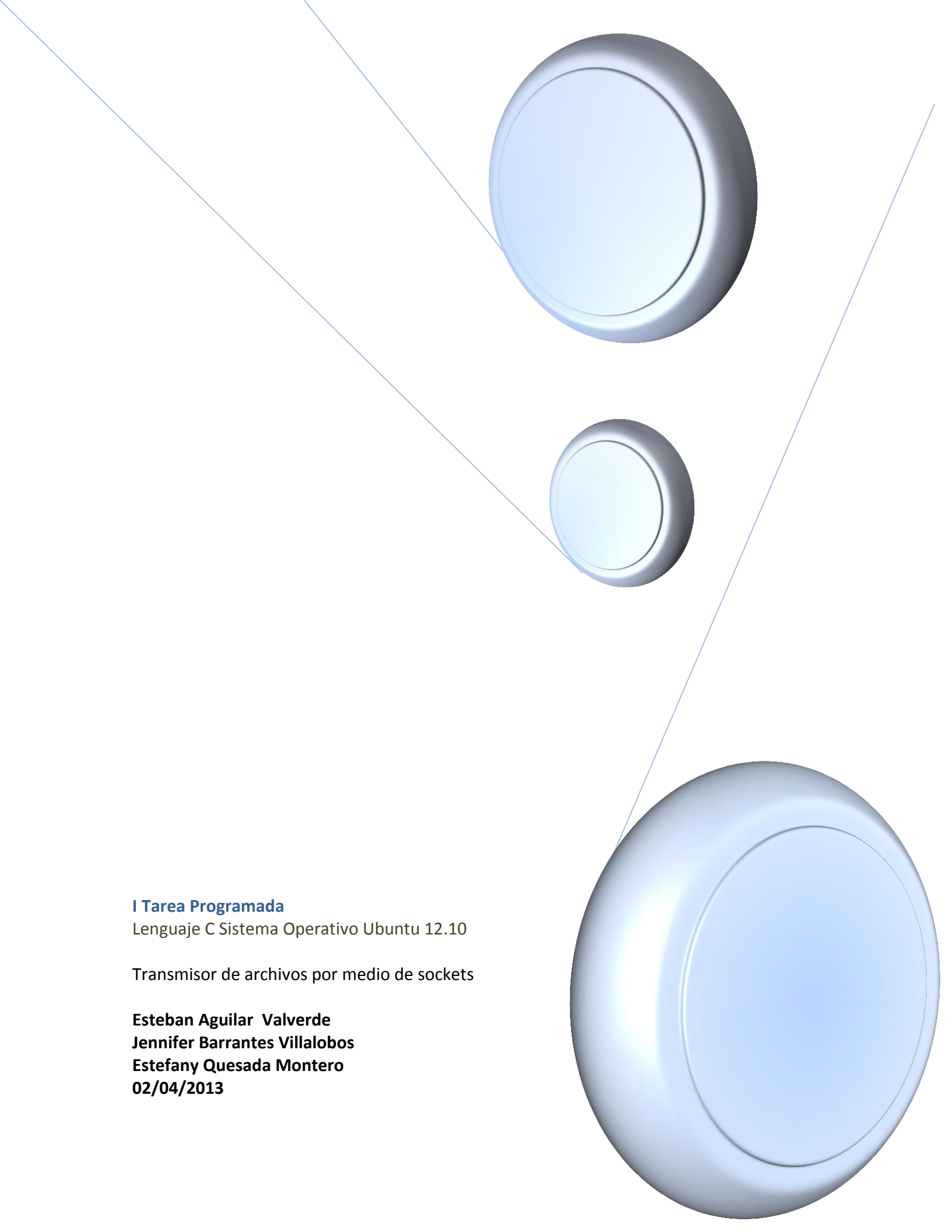


I Tarea Programada

Lenguaje C Sistema Operativo Ubuntu 12.10

Transmisor de archivos por medio de sockets

Esteban Aguilar Valverde
Jennifer Barrantes Villalobos
Estefany Quesada Montero
02/04/2013

Tabla de contenido

Descripción del Problema

Objetivo de la Tarea

El objetivo de esta tarea es familiarizarse con el desarrollo de programas en C, mediante la creación de un programa de intercambio de archivos entre computadoras. Para realizar la comunicación entre las computadoras y poder permitir el envío de archivos entre una computadora y otra se usarán sockets.

Enunciado

Para simplificar el desarrollo de la tarea, se recomienda empezar por una solución sencilla y luego irle añadiendo funcionalidad compleja.
La idea es que se puedan enviar y recibir archivos, desde y hacia otras computadoras. Inicialmente, deben lograr que un programa Emisor envíe un archivo a otro programa Receptor, y que el programa Receptor pueda responder al programa Emisor en caso de que haya funcionado correctamente la transferencia. Una vez que se haya terminado la transferencia, se deberán terminar los programas.
Al inicio, pueden ejecutar el Emisor y el Receptor en la misma computadora. Luego deberán ejecutar los programas en computadoras diferentes en la misma red. Posteriormente podrían ejecutar los programas de manera remota, por ejemplo, Emisor ejecutándose en las computadoras de los laboratorios y Receptor ejecutándose en la computadora de la casa.
En los programas descritos anteriormente, el Emisor sólo puede enviar archivos, y el Receptor sólo puede recibir archivos. La idea es que ambos puedan enviar archivos de forma simultánea, para lo que se deberán hacer algunas modificaciones para incorporar esa funcionalidad.
La idea es que una vez que Emisor y Receptor estén desarrollados, se unifiquen en un mismo programa Transmisor. Este programa creará dos sockets: uno para enviar archivos, y el otro sólo para recibir archivos. Después, mediante el uso de fork() el programa se bifurca en dos procesos, el Emisor (cliente) y el Receptor (servidor).
El proceso cliente atiende el socket de enviar, y cuando el usuario ingrese el nombre de un archivo, enviará el archivo (independientemente de que el socket servidor esté recibiendo un archivo). El proceso servidor atenderá el socket de recibir, y cada vez que reciba un archivo, lo descargará, lo guardará en el sistema de archivos, mostrará el nombre y el tamaño del archivo en pantalla y continuará esperando por otro archivo.
Los archivos pueden tener un tamaño máximo, el cual podrá ser definido por los estudiantes. Los archivos pueden ser especificados en tiempo de ejecución.
La idea es que el programa se ejecute de la siguiente forma:
./transmisor ipRemota puertoRemoto puertoLocal

Digite el nombre del archivo:/Users/juan/Docs/file.txt
Enviando archivo…
Servidor remoto: file.txt recibido correctamente
Servidor local: recibiendo archivo file2.txt
Servidor local: file2.txt recibido correctamente
Los argumentos de línea de comandos del programa son los siguientes:
• ipRemota: es el ip del transmisor remoto
• puertoRemoto: es el puerto del transmisor remoto
• puertoLocal: es el puerto en el cual se deberá abrir el socket a nivel local para recibir archivos
Los nombres de archivos no deben recibirse como argumentos de línea de comandos, sino que serán digitados por el usuario en tiempo de ejecución (en el caso anterior, sería el texto en azul).
Pueden usar colores para diferenciar los mensajes emitidos por el transmisor local, y los mensajes recibidos del transmisor remoto.
Una copia del transmisor se ejecutará en una computadora, y otra copia en otra. Para finalizar la ejecución, alguna de las copias escribirá el mensaje “Finalizar”, y se cerrarán los sockets. La otra copia detectará el mensaje “Finalizar”, y, después de desplegar el mensaje, cerrará sus sockets.
Deben usar ese mensaje para finalizar la ejecución del programa, ya que la idea es poder comunicar programas de diferentes grupos.

Aspectos Técnicos

El proyecto deberá estar escrito en el lenguaje de programación C (no C++), y deberá de funcionar en el sistema operativo Linux. En caso de requerir librerías adicionales para compilar y ejecutar el programa, deberán especificarlo en la documentación, ya que de lo contrario se descontarán puntos en la evaluación.

Diseño del programa

Investigaciones

Uso de los sockets de red en Linux.

Con C en Linux/Unix tenemos una serie de funciones que nos permiten enviar y recibir datos de otros programas, en C o en otros lenguajes de programación, que estén corriendo en otros ordenadores de la misma red.

En algunas máquinas Unix es necesario indicarle al compilador con qué librerías debe linkear para resolver las referencias externas producidas por la utilización de funciones de manejo de sockets.
Ejemplo: $gcc clienteUDP.c -o clienteUDP -lsocket -lnsl
En Linux no suele ser necesario la utilización de -lsocket –lnsl

En Ubuntu 12.10 basta con compilarlo.

Procesos mediante el uso de fork.

Se suelen crear procesos con fork() cuando el servidor no es capaz de atender todas las peticiones de los clientes a suficiente velocidad. Al tener un proceso dedicado a cada cliente, se puede atender a varios "simultaneamente". Se suele usar select() cuando podemos atender a los clientes lo suficientemente rápido como para hacerlo de uno en uno, sin hacer esperar demasiado a nadie.

La creación de nuevos procesos en Unix se realiza por la vía de duplicar un proceso existente invocando al comando fork(). Al proceso original se le llama "padre" y al nuevo proceso "hijo". El PPID de un proceso es el PID de su proceso padre.
El mecanismo de creación de nuevos procesos en Unix con el comando fork() se ve con más detalle en el apartado "Ciclo de vida de un proceso".

Decisiones de diseño

- ✓ Debido a la complejidad del de la tarea se decidió manipular los archivos en binario, esto para poder resolver los inconvenientes de que fueran de tipos diferentes los archivos, enviados y recibidos.

- ✓ También se usó un tamaño máximo para el envió de archivos ya que estos deben de enviarse y guardar un espacio en memoria para recibirlos.

- ✓ Se manejan funciones de enviados y recibidos, por aparte, esto para un control en los sockets y mayor orden.

- ✓ En cuanto a cliente y servidor, en nuestro caso se manejan en un mismo archivo el cual solo recibe de Entradas:
    1- Puerto de escucha del servidor interno
    2- Puerto de escucha del otro computador
    3- Ip del otro Computador

- ✓ La función de leer archivo solo recibe como parámetro el nombre del archivo, y el tamaño del mismo, esta se debe encargar de llamar a escribir archivo para ser enviada por medio del socket, a la computadora receptora, la cual escribe el archivo, y lo guarda para dar por termina la funcionalidad.

- ✓ La variable del nombre es guardada para poder llamar al nuevo archivo (el archivo enviado), de la misma forma.

- ✓ Todos los archivos enviados son verificados por su tamaño, para dar por concluido el envió, en caso de que sea diferente se da un error de envió.

- ✓ Para la programación de esta, se decidió utilizar “geany”, ya que este no es complejo de utilizar, y no es un generador de código, lo cual es una ventaja a la hora de optimizar el trabajo.

- ✓ El sistema Operativo elegido fue el de Ubuntu 12.10, por simpleza de uso, además de ser uno de los más utilizados por los integrantes del grupo.

Algoritmos usados

Sockets

```
//Validaciones de 3 argumentos(4 contando la invocacion)
if (argc != 4) error("Ingrese los 3 argumentos necesarios");
pid_t pID=fork();//Inicia el fork y guarda el identificador del proceso

if (pID == 0) {
        int socket_cliente;//ID del socket de conexion
        struct sockaddr_in direc_servidor;//Estructura para la direccion del otro


        char buffer[256];//Variable para almacenar las escrituras en socket
        socket_cliente = socket(AF_INET, SOCK_STREAM, 0);//Pide el socket tcp/ip
```

Archivos

```
//Variables de manejo de archivos
        char *nombre_archivo = (char*) malloc(FILENAME_MAX *
sizeof(nombre_archivo));



// Método de Lectura y Escritura

void leer_archivo (char *nombre_archivo){
                FILE *archivo;
    char caracter;
    printf("\n %s", nombre_archivo);
    archivo = fopen(nombre_archivo,"rb");

    if (archivo == NULL){

        printf("\nError de apertura del archivo. \n\n");
    }else{


      printf("\nEl contenido del archivo de prueba es \n\n");

      //while (feof(archivo) == 0)
      //{
       //   caracter = fgetc(archivo);
        //  printf("%c",caracter);
      //}
    }
    fclose(archivo);
      printf("\nEl archivo ha sido enviado \n\n");
}
```

Cerrar procesos

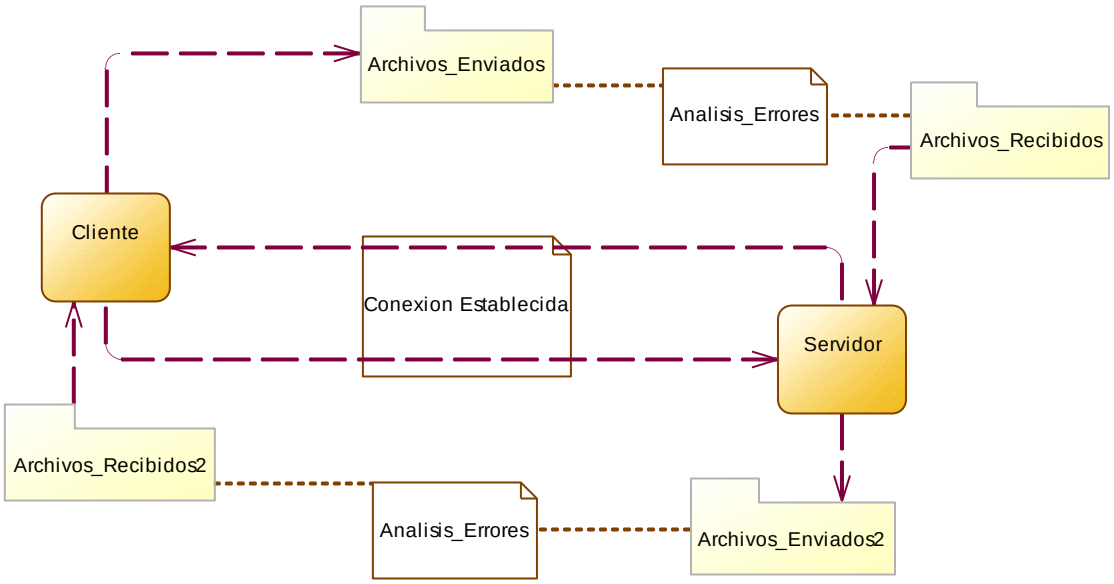
```
fclose(filer); //Cerrar Archivo

free(nombre_archivo);//libera el espacio en memoria
```

proceso padre para cerrar sockets

```
                                free(nombre_archivo);
                                close(socket_cliente);
                                int muerto;
                                muerto = kill(otro_proceso,9);
                                usleep(10000);
                                ciclo = 0;
                                char *borre_killed = "";
                                if(muerto==0) borre_killed ="\033[A\033[2K";
                                printf( "\033[2K\r"
                                "\033[01;32m%s"
                                "Conexion Finalizada"
                                "\nPresione Enter para salir"
                                "\033[01;37m",borre_killed);
                                }
                        }
```

Diagramas lógicos



Librerías Usadas

Para el desarrollo de dicho programa se utilizaron diferentes librerías las cuales son:

<*stdio.h*> Se utiliza para manejar las entradas y salidas de consola.
<*stdlib.h*> Se utiliza la función atoi() para pasar caracteres a enteros.
<*unistd.h*> Se utiliza en el fork()
<*string.h*> Comparación de grupos de caracteres
<*sys/types.h*> , <*sys/socket.h*> Socket
<*netinet/in.h*> , <*netdb.h*> Manejo de direcciones de internet

Análisis de Resultados

La tarea programada no se logró completar al 100%, ya que se conectan las 2 computadoras pero no se envían los archivos, desde un mismo programa.
Más aún, en funciones individuales, toma un archivo lo lee y envía dentro de la misma computadora.

El fallo, que evitó que se concluyera con el trabajo, fue el trabajar todas las funciones individualmente para unirlas al final, porque al final, no se contó con el tiempo necesario para terminar de manera exitosa el mismo.

¿Cómo se puede solucionar el problema para que el trabajo sea completado?
Trabajando en el proyecto, a pesar de que su hora de entrega ya pasó. Para lograr conectar las funciones que se tienen por aparte, en un sólo programa.

Manual de Usuario

Instrucciones de uso, ejecución y compilación

Pasos:

1. Para poder ejecutar el programa es necesario que el programa este en ambas computadoras.
2. Ambas deben ejecutarse al mismo tiempo.
3. Los Usuarios primero que todo deberán compilar el programa en consola o en

cualquier otro compilador, por ejemplo:
gcc -o transmisor transmisor.c

4. Ya generado el archivo lo podemos ejecutar de la siguiente manera.
   ./transmisor puerto1 puerto2 IP

5. Donde “puerto1” significa el puerto que su programa va a utilizar, “puerto2” el puerto que el programa de la otra persona utiliza y “IP” es el IP de la otra persona.

   Por ejemplo:
   ./transmisor 3550 8886 192.168.2.2
   ./transmisor 8886 3550 192.168.2.4

6. En caso que quiera utilizar dos consolas para realizar pruebas se debe utilizar la dirección de IP: “127.0.0.1” en los dos programas y los puertos deberán ser distintos e intercambiados de posición.
7. Una vez establecida la conexión, esta le pide a los usuarios que inserten el nombre del archivo que desean enviar.

8. Se introduce la dirección donde se encuentre el archivo, y este se envía al otro computador(no implementado)

Conclusión Personal

Esto nos ayuda a la hora que ocupamos tener un archivo que se va a trabajar en diferentes computadoras, para que este archivo pueda fácilmente ser enviado. Así nos evitamos estar trasportando el documento por todas las computadoras y con esto se disminuye la posibilidad de documentos dañados, que al final, se pierda tiempo en buscar una manera de enviar el archivo, por medio de un dispositivo.

Como se mencionó en el enunciado, muchos de los programas actuales permiten esta funcionalidad pero sólo si está conectado a Internet, lo que nos puede limitar un poco. Pero si se usan sockets esto se puede evitar, el único inconveniente, es que se necesita que se ejecute el programa en las computadoras al mismo tiempo.

Aunque esta no se pudo concluir, durante el tiempo estimado, la investigación que se llevó a cabo fue de mucho aprendizaje, a tal punto, que si se tuviera un poco más de tiempo esta se llevaría a concluir al 100%.

Referencias

- http://www.chuidiang.com (2007). ***Programación de sockets en C de Unix/Linux***. [ONLINE] Available at: http://www.chuidiang.com/clinux/sockets/sockets_simp.php#sockets. [Last Accessed 21 de marzo].

- J. Llorente (27 de setiembre). ***Cómo utilizar los sockets en Linux***. [ONLINE] Available at: http://www.macnux.com/portal/secciones-mac-linux-redes/sololinux/195-cutilizar-los-sockets-en-linux. [Last Accessed 21 de marzo].

- profesores.elo.utfsm.cl (e.g. 2011). ***Control de Procesos***. [ONLINE] Available at: http://profesores.elo.utfsm.cl/~agv/elo330/2s09/lectures/ControlProcesos.html. [Last Accessed 22 de marzo].

- nereida.deioc.ull.e (e.g. 2011). ***La Función fork***. [ONLINE] Available at: http://nereida.deioc.ull.es/~pp2/perlexamples/node66.html. [Last Accessed 22 de marzo].