**LABORATORIO 8**

**CLEMENCIA GIRALDO**

**SILVIA VEGA**

**PABLO HENAO**

**UNIVERSIDAD DE ANTIOQUIA**

**2011**

**LABORATORIO 8**

* **Ejercicio 1.**

**ENUNCIADO:**

Implemente un servidor y un cliente de intercambio de ficheros entre máquinas mediante el protocolo UDP. Recuerde que este protocolo no garantiza la distribución de los mensajes ni la conservación de su secuencia original. Esto obligará a diseñar un protocolo a nivel de aplicación que garantice la fiabilidad de la comunicación. No olvide que si se pierden los mensajes el cliente debe solicitar al servidor que retransmita los mensajes perdidos. Para descargar un fichero se usara el comando:

**$ descargar “ruta absoluta del archivo en el servidor”**

NO OLVIDE QUE VARIOS ARCHIVOS PUEDEN DESCARGARSE A LA VEZ!

**Implementación**

El código consta de dos archivos fuente: el servidor (server.cpp) y el cliente (user.cpp).

**Programa servidor:**

En un principio se estableció el puerto, la ruta y el tamaño de buffer que se utilizaría a lo largo de todo el programa; esto con el objetivo de fijar características de la conexión y tamaños deseados. Seguidamente, se comenzó declarando las variables que almacenan los retornos de las funciones que más adelante pasarán a ser los descriptores de las mismas, y las estructuras que contienen la información del servidor y del cliente. Luego, como es debido, se estableció el *socket* para trabajar con red, el cual permite la apertura del canal y necesita de parámetros como AF\_INET para emplear el protocolo de internet UDP y SOCK\_DGRAM que emplea protocolos de tipo datagrama. Después, se llevó a cabo la validación respectiva en caso de no poderse ejecutar con éxito la apertura de la conexión. Más adelante, se configuraron los miembros de la estructura *sockaddr* la cual requería de especificaciones como la familia de direcciones (AF\_INET para este caso), la herramienta de conversión *network byte order* y la dirección IP. A continuación, se establece la conexión a través de la implementación del *Bind* cuya tarea es atar el socket a un puerto y a una dirección. De igual forma, se valida el fallo de esta conexión.

Posteriormente, se comienzan a elaborar las funciones de transferencia de datos y para esto es conveniente establecer una variable que almacene la ruta que el cliente ingresará. Pronto, se comienza un ciclo infinito con el fin de recibir peticiones y luego se hace uso de la función *recvfrom()* la cual permite, junto con *sendto*(), la implementación de datagramas. *Recvfrom* retorna el número de bytes recibidos.

En seguida, se instancia un objeto de tipo *fstream* en modo binario y con el permiso de lectura, junto con la ruta que indica la ubicación del archivo que debe abrirse. A continuación, se declara la estructura encargada de obtener el estado del fichero y su respectiva información. Más adelante, se crea el hilo para la ejecución por medio de un *fork()* y en el momento en el cual el proceso hijo se crea, se procede a ejecutar la apertura del socket para enviar datos (*socket\_envio*) y luego se implementa la función *sendto()* para enviar el fichero requerido, la cual necesita de parámetros como el número de bytes del fichero (bytes\_fichero) y el puerto de destino (user). Consecutivamente, se leen los datos y se ubica al puntero al inicio del fichero.

Inmediatamente se procede a recorrer el fichero y conforme se lee, se almacena la información en el buffer para luego enviársela al cliente. Además, se recibe la información desde el cliente de los datos que han sido transferidos y finalmente, se realizan las respectivas validaciones de los posibles errores que puedan surgir a lo largo del proceso.

**Programa cliente:**

De forma similar al programa servidor, se estableció el puerto y el tamaño del buffer que se utilizarían a lo largo del programa. Luego, se comenzó la construcción del *main* sabiendo que esta función recibiría los parámetros: IP del servidor a quien hace la petición y la ruta del fichero que será transferido.

Ulteriormente, se establecen las variables que almacenan el retorno de las funciones del *socket* y del *sendto* y, además, la estructura que almacena información del servidor tal como la IP, el host local, el puerto y su tamaño. A continuación, se procede a la obtención del host mediante la función *gethostbyname* y luego, configuramos los valores de la estructura que contiene los datos del servidor llevándose a cabo un proceso similar al ejecutado en el programa servidor, salvo la dirección del servidor que proviene de uno de los argumentos del main digitados por el usuario. Igualmente, se realiza la apertura del socket para trabajar con red.

Posteriormente, se implementa la función para comenzar el envío de datos, la cual en un principio, debe enviar la ruta que recibirá el servidor y después, se debe configurar el tiempo de espera para que el servidor responda. Pronto, se monitorea el conjunto de sockets por medio de la función *select(),* la cual requiere de parámetros tales como el tiempo y descriptores de socket.

Más adelante, se comienzan a implementar funciones para recibir los datos. De este modo, la función *recvfrom()* retorna la cantidad de bytes del archivo que han llegado y verifica si efectivamente se está transfiriendo un fichero. En seguida, se instancia un objeto *fstream* con el fin de crear un archivo para guardar los datos. La ruta de este fichero se obtiene desde el segundo argumento del main y mediante la función *strrchr()* se captura únicamente el nombre del fichero. Así pues, en el buffer se almacenarán los paquetes que llegan.

Finalmente, se recorre el fichero mientras que los bytes acumulados no superen a los bytes totales del archivo que se está transfiriendo y simultáneamente, se comienza a escribir en el archivo aquello que proviene del buffer y al culminar este proceso se envía una confirmación por haber recibido el archivo completamente, se cierra el fichero y se cierra el socket.

* **Ejercicio 2.**

**ENUNCIADO:**

Utilice la familia de conectores AF\_UNIX para hacer un juego de mesa de dos jugadores, al contrario de un juego normal, los jugadores en este caso son los procesos padre e hijo, se deben generar meros aleatorios del 1 al 6 por turnos, el primero en llegar hasta 30 gana.

Consideraciones:

– NO DEBE SUPERARSE EL NUMERO 30.

– SI UN PROCESO LLEGARA A UNA “CASILLA” DONDE SE ENCUENTRE EL OTRO,

SE CONSIDERA QUE EL PROCESO QUE ACABA DE LLEGAR A LA “CASILLA” SE HA

“COMIDO” AL OTRO, ENVIANDOLO A LA CASILLA CERO.

– EL PROGRAMA DEBE ESPERAR UNA TECLA ANTES DE HACER LOS

LANZAMIENTOS Y REPORTAR CADA MOVIMIENTO Y RESULTADO.

**Juego:**

Inicialmente, se declaran acumuladores y variables que desempeñarán el rol de contadores. A continuación, se crea un arreglo de sockets. Luego se implementa la función *socketpair(),* la cual permite crear un par de conectores enlazados y necesita de parámetros tales como la familia de conectores ( AF\_UNIX, para procesos en una misma máquina ).

Seguidamente, se implementa la función *fork()* y de acuerdo con su retorno se puede crear un proceso padre o un proceso hijo, que corresponden al jugador 1 y al jugador 2 respectivamente.

Cuando ocurre la creación del proceso padre, se cierra el socket del hijo y se inicia el juego. Así, se lleva a cabo el primer lanzamiento del dado, del cual se obtiene un número aleatorio de 1 a 6 por medio la función *rand().* Después, se ingresa a un ciclo infinito y a través de la función *send()* se informa acerca de la posición del padre al hijo y luego, con la función *recv()* se recibe el avance y la posición final del hijo. A continuación, se comienzan a evaluar una serie de condiciones de acuerdo con el resultado obtenido y las adiciones pertinentes.

Por otra parte, cuando suceda la creación del proceso hijo, se cierra el socket del padre ya que es innecesario. En seguida, se ingresa a un ciclo infinito e inmediatamente, esperamos el resultado del padre con la implementación de la función *recv().* Igualmente se evalúan las condiciones y los resultados obtenidos en cada lanzamiento y de acuerdo con ellos se establece el ganador o la continuación del juego.