Sistemas Operativos y Redes (E0224)
Año 2021

Trabajo Práctico N°5

Grupo N°4:

Ignacio Hamann - 68410/3
Juan Pablo Elisei - 68380/5
Tomás Tavella - 68371/4

Resumen

En este informe se detalla la implementación de un servidor concurrente mediante el protocolo TCP, el cual acepta conexiones de clientes para almacenar archivos.

Facultad de Ingeniería
Universidad Nacional de La Plata

Índice

1. Enunciado

Queremos un programa ***servidor concurrente*** que permita a un cliente que se conecta utilizando el protocolo TCP enviar un archivo para que el servidor lo almacene.

- Cuando un cliente se conecta, el servidor enviará el mensaje: "listo" y esperará recibir la palabra "archivo".
- A continuación el servidor espera un espacio y luego el nombre con que se quiere guardar el archivo (solo letras, números y el carácter ".") finalizado con un espacio.
- Luego se quedará esperando un número codificado en `ascii` que indica el tamaño en bytes del archivo, también finalizado con un espacio.
- A continuación comenzará la recepción de los datos, que serán almacenados en un archivo cuyo nombre es el recibido en primer término, hasta completar la cantidad de bytes correspondiente.
- El servidor debe permitir la recepción de archivos binarios.
- Una vez recibida la totalidad de los datos, el servidor contestará con el siguiente mensaje:
  - "Archivo xx completo, tamaño declarado yy bytes, tamaño real zz bytes."
    - xx: nombre del archivo.
    - yy: tamaño enviado en el mensaje del cliente.
    - zz: total de bytes recibidos por el servidor.
- Luego el servidor cerrará la conexión con el cliente.
- El servidor debe llevar un archivo de registro de todas las conexiones entrantes, con fecha y hora de inicio y de finalización, tamaño del archivo recibido, cantidad de bytes enviados y cantidad de bytes recibidos en formato `csv`.
- Defina explícitamente cómo será el manejo de errores.
- Para probar el programa puede utilizar el cliente TCP que se vio como ejemplo, o el programa `nc`.
- Recuerde verificar que no queden procesos *zombie* cuando finalizan los hijos generados.

2. Interpretación del problema

Se debe crear el programa de un servidor concurrente que utilice el protocolo TCP, para que los clientes que se conecten puedan jugar una partida de la "Escoba de 15". El procedimiento a seguir para el desarrollo del juego es el siguiente:

1. Una vez que un cliente establezca la conexión con el servidor, si es el primero, se le pregunta la cantidad de jugadores.
2. Luego, al primer cliente y a los demás que se conecten se les pide el nombre desde un proceso hijo, y estos envían un mensaje al padre indicando que el jugador está listo, quedando a la espera de la partida.
3. Cuando todos los jugadores están listos, el padre comienza a repartir las cartas, y envía los mensajes de sincronización a cada hijo para indicar que acción debe realizar.
4. Estas acciones pueden ser anunciar de quien es el turno, pedir la jugada, anunciar que jugada se realizó, o finalizar la partida y mostrar puntajes.
5. Luego de cada ronda el padre reparte nuevamente y vuelve a enviar los mensajes.
6. Finalizada la partida, se muestran los puntajes a los jugadores, y el servidor queda a la espera de una conexión nueva para comenzar un nuevo juego.

Adicionalmente, si el proceso padre recibe la señal *SIGHUP*, espera a que termine la partida actual (o si no se está jugando, la siguiente partida), libera la memoria compartida y termina.

Para el manejo de errores se presentan varios casos de errores distintos:

- Cantidad de jugadores inválida: En estos casos simplemente se pide nuevamente que se ingrese un valor válido.
- Jugada inválida: Ídem al caso anterior, se pide que ingrese un valor válido.
- Si la suma de las cartas es mayor a 15, o no alcanzan las cartas de la mesa, se pide que se repita la jugada.

3. Resolución

Se utilizaron las siguientes bibliotecas de C para poder llevar a cabo la resolución del problema planteado:

- `<sys/types.h>`: biblioteca de *System V* para la definicion de tipos de variables.
- `<sys/stat.h>`: biblioteca de *System V* para la comunicación entre procesos.
- `<sys/socket.h>`: biblioteca de *System V* para utilizar sockets.
- `<netinet/in.h>`: biblioteca que contiene varias definiciones útiles de macros y tipos de datos para manejo de sockets de red.
- `<arpa/inet.h>`: biblioteca que contiene definiciones de varias funciones útiles para el manejo de direcciones IP.
- `<unistd.h>`:
- `<sys/time.h>`: biblioteca para obtener el tiempo de la *timestamp*.

Para la simplificación del código se crearon 2 macros, `SEND()` y `SEND_RECV()` las cuales son utilizadas para enviar un mensaje al cliente y para enviar y recibir un mensaje del cliente respectivamente.

Al ser un servidor concurrente, un proceso hijo dentro del servidor toma a cada cliente y dentro de este se hace la copia del archivo.

Los buffers de transmisión y recepción son de 1500 bytes, es por eso que se envia esta cantidad de bytes desde el archivo que se quiere copiar hasta llegar al *end of file* del mismo.

3.0.1. Pseudocódigo

```
Servidor:
INICIO
    Declarar y asignar variables, macros y estructuras;
    Se crea el descriptor del socket;
    Mientras(No se termine de leer el archivo){
        Acepta la conexion de un cliente (en el caso de no poder conectarlo imprime error);
        Toma el tiempo inicial;
        Imprime los datos del cliente conectado;
        Si(Estoy en el hijo){
            Declarar y asignar variables;
            Imprimen los datos del hijo;
            Cierra el socket que no es usado;
            Envia el mensaje "Listo" (en el caso de no poder enviarlo imprime error);
            Aumenta la variable de bytes enviados;
            Recibe el mensaje "Archivo" (en el caso de no poder recibirlo imprime error);
            Aumenta la variable de bytes recibidos;
            Si(Se recibe el mensaje "Archivo"){
                Imprime que se recibio la palabra archivo;
                Recibe el nombre del archivo y los bytes que contiene (en el caso de no
poder recibirlo imprime error);
                Aumenta la variable de bytes recibidos;
                Imprime el nombre del archivo;
                Crea el archivo para escritura en binario;
            }
            Sino{
                Imprime error;
                Envia el mensaje de error (en el caso de no poder enviarlo imprime error);
```

```
                    Aumenta la variable de bytes enviados;
                    Toma el tiempo final;
                    Escribe en el archivo de registro;
                    Cierra la conexion con el cliente;
                    Termina el programa;
                }
                Hacer{
                    Recibe un buffer con la informacion contenida en el archivo y lo escribe en el nuevo archivo (en el caso de no poder recibirlo imprime error);
                    Decrementa la variable que contiene los bytes del archivo;
                    Aumenta la variable de bytes recibidos;
                    Imprime la cantidad de bytes recibidos del cliente;
                    Si(La variable que contiene los bytes del archivo es menor a 0){
                        Imprime que llegaron bytes de mas;
                        Envia el mensaje de error (en el caso de no poder enviarlo imprime error);
                        Aumenta la variable de bytes enviados;
                        Toma el tiempo final;
                        Escribe en el archivo de registro;
                        Cierra la conexion con el cliente;
                        Termina el programa;
                    }
                }Mientras(No se llegue al final del archivo)
                Cierra el archivo;
                Imprime que la recepcion finalizo sin errores
                Envia el mensaje al cliente;
                Aumenta la variable de bytes enviados;
                Cierra la conexion con el cliente;
                Toma el tiempo final;
                Escribe en el archivo de registro;
                Termina el programa;
            }
            Cierra la conexion con el cliente;
        }
        Espera que el hijo termine se ejecucion;
        Cierra la conexion con el cliente;
        Termina el programa;
    FIN

    Cliente:
    INICIO
    Declarar y asignar variables, macros y estructuras;
    Si el segundo argumento no del formato www.xxx.yyy.zzz imprime error;
    Se crea el descriptor del socket;
    Se conecta con el servidor (en caso de no poder conectarse imprime error);
    Imprime la informacion del servidor;
    Recibe el mensaje "Listo" (en el caso de no poder recibirlo imprime error);
    Aumenta la variable de bytes recibidos;
    Si(Se recibe el mensaje "Listo"){
        Imprime que se recibio la palabra listo;
        Envia el mensaje "Archivo" (en el caso de no poder enviarlo imprime error);
    }
    Sino{
        Imprime error;
        Envia el mensaje de error (en el caso de no poder enviarlo imprime error);
        Termina el programa;
    }
    Espera a que se ingrese el nombre del archivo;
    Imprime el nombre del archivo y lo abre;
    Si(No puede abrirse el archivo){
        Imprime error;
        Envia el mensaje de error (en el caso de no poder enviarlo imprime error);
        Termina el programa;
    }
    Sino{
        Obtiene el tamano del archivo;
        Envia el nombre del archivo y el tamano (en el caso de no poder enviarlo imprime error);
    }
    Espera a que se presione enter para enviar los datos;
    Mientras(No se llegue al end of file){
        Se lee parte del archivo y se envian esos datos;
    }
    Se cierra el archivo;
    Se recibe que se termino la recepcion;
    Se cierra al cliente;
    Termina el programa
}
```

3.1. Compilación y ejecución de los programas

Con el código completo, para compilar los programas a un archivo binario ejecutable se llama al comando `gcc` (*GNU C Compiler*):

```
$ gcc servidorTCP.c -o servidorTCP
 $ gcc clienteTCP.c -o clienteTCP
```

Con lo que se obtienen dos archivos binarios ejecutables, los cuales, estando situados en la carpeta en la que se encuentran, se ejecutan desde la terminal de la siguiente manera:

```
$ ./servidor
$ ./cliente 127.0.0.1
```

Es importante ejecutar el servidor antes que el cliente.

4. Conclusiones

El uso de un servidor concurrente TCP es esencial para los casos en que es necesario atender a mas de un cliente en simultáneo, debido a que los procesos hijos se encargan de toda la comunicación con cada cliente, y el proceso padre esta libre para atender nuevas conexiones y derivarlas.