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, el cual aceptará conexiones de clientes que envían archivos para almacenar en el servidor. El procedimiento a seguir para la esta transferencia es el siguiente:

1. Una vez que un cliente establezca la conexión con el servidor, este ultimo le enviará el mensaje "listo" al cliente, quedando en espera del mensaje de confirmación "archivo" por parte del cliente.
2. Luego el servidor debe recibir un caracter *espacio*, seguido del nombre para almacenar el archivo (solo caracteres alfanuméricos y un punto ".") y un caracter *espacio* al final.
3. Una vez recibido un nombre valido, el servidor espera la recepción de un numero codificado en *ascii* que indique el tamaño del archivo en bytes, seguido de un caracter *espacio*.
4. Ahora, el servidor comienza la recepción de datos, los que almacena en un archivo binario con el nombre obtenido en el paso 2, hasta llegar a la cantidad de bytes obtenidos en el paso 3.
5. Como forma de confirmación, una vez terminada la recepción el servidor le enviará al cliente el siguiente mensaje:
   "Archivo «*Nombre de archivo*», tamaño declarado «*Cantidad de bytes declarada*» bytes, tamaño real «*Cantidad de bytes recibidos*» bytes.".

6. Finalmente, el servidor cierra la conexión con el cliente.

Adicionalmente, el servidor debe mantener un archivo en formato `csv` que contenga registro de:

- Las conexiones entrantes.
- Fecha y hora de comienzo de la conexiones.
- Fecha y hora de finalización de la conexiones.
- Tamaño de los archivos recibidos.
- Cantidad de bytes enviados.
- Cantidad de bytes recibidos.

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

- Error en el nombre de archivo y en el mensaje de confirmación "archivo": En estos casos, como no se llega a abrir el archivo, simplemente se envía un mensaje de error al cliente y se cierra la conexión.
- Si el archivo enviado es de tamaño superior al declarado, el archivo se trunca al llegar a la cantidad declarada de bytes y se cierra con un código de error, se envía un mensaje de error al cliente y se cierra la conexión.
- Si el archivo enviado es de tamaño inferior al declarado, debe existir un tiempo de *timeout*. Una vez excedido este tiempo, si se recibió una cantidad de bytes menor a la declarada, se cierra el archivo con un código de error, se envía un mensaje de error al cliente y se cierra la conexión.
- Si se cierra la conexión con el cliente prematuramente, se cierra el archivo con un código de error.

Adicionalmente, en todos estos casos el código de error correspondiente se almacena en el registro que lleva el servidor.

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;
```

```
86          Envia el mensaje de error (en el caso de no poder enviarlo imprime error);
87          Termina el programa;
88      }
89      Sino{
90          Obtiene el tamano del archivo;
91          Envia el nombre del archivo y el tamano (en el caso de no poder enviarlo imprime error)
        ;
92      }
93      Espera a que se presione enter para enviar los datos;
94      Mientras(No se llegue al end of file){
95          Se lee parte del archivo y se envian esos datos;
96      }
97      Se cierra el archivo;
98      Se recibe que se termino la recepcion;
99      Se cierra al cliente;
100     Termina el programa
101 }
```

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 protocolo TCP intercambia paquetes llamados segmentos entre aplicaciones e IP, es utilizado en aplicaciones 1 a 1 (como en este caso cliente a servidor), es confiable y útil para LAN y WAN.