Servidor - Implementación del servidor de un pseudo-FTP

Código de Apoyo

El código de apoyo puede descargarse de la aquí: servidor.2015a.tgz

Debe llevarlo a su HOME de la máquina triqui.

También se encuentra disponible en triqui en la ruta:
`/home2/datsi/ssoo/pub/servidor.2015a.tgz`

```
[usuario@triqui ??] cd
[usuario@triqui ~] cp ~ssoo/pub/servidor.2015a.tgz .
```

Este código de apoyo, al descomprimirlo, debe crear la ruta `DATSI/so/servidor.2015a`. Compruebe previamente que en su cuenta este directorio no existe o está vacío. Para descomprimirlo, realice los siguientes mandatos.

```
[usuario@triqui ??] cd
[usuario@triqui ~] tar xzf servidor.2015a.tgz
[usuario@triqui ~] cd DATSI/so/servidor.2015a
[usuario@triqui ~/DATSI/so/servidor.2015a]
```

Compilación

El código de apoyo incluye un *Makefile* que controla los detalles de compilación todos los fuentes de apoyo.

```
[usuario@maquina ~/DATSI/so/servidor.2015a] make
...
```

Corrección

La puntuación que se obtendrá en este ejercicio será la alcanzada por medio de un corrector. El corrector se pasará sobre las prácticas entregadas.

Entrega

En cualquier momento se podrá entregar el código desarrollado, en el estado en el que se encuentre. La única entrega válida será la última. La entrega se podrá realizar directamente mediante el mandato:

```
[usuario@triqui ~/DATSI/so/servidor.2015a] make entrega
...
entrega.ssoo Versión 2004.3, Recolector de fuentes de practicas
entrega.ssoo Copyright (c) 1995..2004 Francisco Rosales (frosal@fi.upm.es)

entrega.ssoo
   El alumno "XXXXX"
   desea entregar la practica "servidor.2015a".

entrega.ssoo
   Departamento: Arquitectura y Tecnología de Sistemas Informáticos
```

```
   Asignatura: Sistemas Operativos
   Encargado: Francisco Rosales <frosal@fi.upm.es>
   Titulo: Disenyo: Servidor
   Directorio origen: ~/DATSI/so/servidor.2015a
   Ficheros: autores.txt bitacora.txt ...
....
```

Objetivos

El alumno debe desarrollar una implementación de un sencillo servidor de un protocolo simplificado de tipo FTP, para transferencia de ficheros por medio de *sockets*.

Descripción

Uso de la aplicación

El programa `servidor` se invocará desde el intérprete de mandatos (*shell*), este programa no recibe ningún parámetro y, al ponerlo a ejecutar, se debe quedar a la espera de peticiones por parte de uno o varios clientes, de forma secuencial.

Sintaxis de la línea de mandato del programa

La estructura general de una llamada a `servidor` se encuentra recogida en la siguiente estructura:
`servidor`

Como ya se ha dicho, este programa no recibe ningún parámetro.

Apartados

Funcionamiento básico

El servidor se encuentra inicialmente escuchando mensajes por un puerto UDP. Los mensajes representan solicitudes de transferencia de ficheros. Las transferencias, en esta versión simplificada del servidor FTP siempre se harán del servidor hacia el cliente. El cliente transmite un mensaje en el cual indica la operación a realizar y el nombre del fichero original (en el servidor) y el nombre local (en el cliente).

El servidor responde con la misma estructura de mensaje (vía UDP también) indicando si se puede realizar o no la operación (dependiendo de que el fichero original exista o no) y, en el caso de ser posible la transferencia, el puerto TCP que se va a usar.

La transmisión del fichero se realiza a través de conexiones *stream* por medio de TCP.

Descripción general del protocolo

Una transferencia tipo entre cliente y servidor consta de los siguientes pasos:

- El servidor arranca, reserva un puerto UDP y se queda a la escucha en él.
- El cliente accede a dicho puerto de servicio UDP y transmite un mensaje de tipo `REQUEST` indicando un nombre de fichero en el servidor y el nombre local que se usará cuando se transfiera a

local.
- El servidor comprueba que el fichero existe. Si no existe responde con un mensaje de tipo `ERROR`.
- Si el fichero existe, el servidor abre un puerto TCP y responde al cliente, usando el anterior puerto UDP con un mensaje de tipo `OK`, en el que incluye el número de puerto TCP que ha reservado (el cliente no lo conoce a priori).
- El servidor se queda esperando en el nuevo puerto TCP y el cliente abre dicha conexión.
- Por la conexión TCP establecida se transmite el contenido completo del fichero.
- El servidor cierra la conexión con el cliente y se queda, de nuevo, esperando nuevos mensajes en el puerto UDP.

Configuración y puertos

El servidor, que el alumno debe implementar, debe reservar dos puertos que estén libres, primero uno UDP y luego otro de tipo TCP. Para realizar esta reserva y asegurarse que se busca un puerto libre, se solicitará al sistema (en la llamada `bind`) el puerto 0. Dicho número de puerto indica al sistema que debe buscar un puerto libre cualquiera.

No se debe ir probando puertos de forma secuencial hasta que uno esté libre. Se debe solicitar uno libre por medio de este número de puerto 0, y luego se debe consultar cuál es el número de puerto asignado (ver llamada `getsockname`).

La forma mediante la cual se notifica al cliente el puerto utilizado es diferente para cada tipo (UDP y TCP, en este programa):

- Puerto UDP: En un caso real estos puertos son puertos estándar, pero, para permitir que varios usuarios ejecuten servidores en la misma máquina, se asigna buscando un puerto UDP cualquiera que esté libre y el servidor escribe en un fichero el número de puerto usado. El cliente lee este fichero y así sabe qué puerto es el del servidor. Hay que tener en cuenta que este mecanismo es un poco artificial, pero permite que se ejecuten prácticas independientes en la misma máquina. El código de apoyo incluye la función para escribir este fichero y que lo pueda leer el cliente (función `Escribir_Puerto`).

- Puerto TCP: Por su parte, el puerto TCP se notifica como respuesta válida del servidor a una petición de transferencia. Cuando se comprueba que se puede hacer la transferencia se transmite el número de puerto TCP que se va a usar.

Estructura de los mensajes UDP

Todos los mensajes intercambiados entre cliente y servidor por medio de UDP se basan en la estructura `UDP_Msg`. Dicha estructura, descrita en el fichero `message.h`, contiene los siguientes campos:

```
typedef struct UDP_Msg_st {
  int    op;          /* Operación */
  char   local[128];  /* Fichero local */
  char   remoto[128]; /* Fichero remoto */
  int    puerto;      /* Puerto del servidor para la transmisión del fichero vía TCP */
} UDP_Msg;
```

El campo `op` puede tomar los siguientes valores:

- `REQUEST`: Petición del cliente de una transferencia.

- OK: Respuesta afirmativa del servidor, el fichero existe y se puede hacer la transferencia.
- ERROR: Respuesta negativa del servidor, el fichero no existe y no se puede hacer la transferencia.
- QUIT: Petición del cliente para que el servidor finalice su ejecución.

Los campos `local` y `remoto` sólo tienen sentido en el mensaje `REQUEST` del cliente al servidor. Indican el nombre del fichero local (para el cliente) y remoto (del servidor). La transmisión siempre es del servidor hacia el cliente.

El campo `puerto` es el que rellena el servidor en la respuesta OK y en dicho campo se indica el puerto TCP reservado.

Traza de ejecución esperada en el servidor

Para comprobar que el servidor está funcionando correctamente, y detectar eventuales errores, se exige una secuencia de trazas (impresas por la salida estándar del programa `servidor`) estricta.

Dicha secuencia es la siguiente:

Ejemplo de trazas

```
[usuario@maquina ~]  servidor
SERVIDOR: Creacion del socket UDP: OK
SERVIDOR: Asignacion del puerto servidor: OK
SERVIDOR: Puerto guardado en fichero ./puerto_servidor: OK
SERVIDOR: Creacion del socket TCP: OK
SERVIDOR: Asignacion del puerto servidor: OK
SERVIDOR: Aceptacion de peticiones: OK
SERVIDOR: Puerto TCP reservado: OK
SERVIDOR: Esperando mensaje.
SERVIDOR: Mensaje del cliente: OK
SERVIDOR: REQUEST(prueba.txt.local,prueba.txt)
SERVIDOR: Enviando del resultado [OK]: OK
SERVIDOR: Llegada de un mensaje: OK
```

Cada mensaje se producirá tras la ejecución de una de las operaciones principales del servidor, y se indicará si ha funcionado o no.

- `SERVIDOR: Creacion del socket` *`tipo`*: Tras la creación del *socket* del tipo correspondiente (UDP o TCP).
- `SERVIDOR: Asignacion del puerto servidor`: Tras finalizar la reserva del puerto (ver `bind`). Debe haber dos trazas de este tipo en el servidor una por tipo de puerto (TCP o UDP) usado.
- `SERVIDOR: Aceptacion de peticiones`: Después de haber invocado `listen`.
- `SERVIDOR: Puerto TCP reservado`: Una vez finalizada la creación y reserva del *socket* TCP.
- `SERVIDOR: Esperando mensaje`: Antes de quedarse bloqueado en una recepción de mensaje UDP.
- `SERVIDOR: Mensaje del cliente`: Después de recibir un mensaje UDP.
- `SERVIDOR: Enviando del resultado [`*`mensaje`*`]`: Envio del mensaje de respuesta correspondiente (OK o ERROR) al cliente.
- `SERVIDOR: Llegada de un mensaje`: Antes de quedarse bloqueado en la espera de una conexión TCP (ver `accept`).

Finalización del servidor

Para finalizar la ejecución del servidor, el cliente puede mandar un mensaje de tipo `QUIT` en lugar de una petición `REQUEST` de un fichero. Tras la recepción de este mensaje el servidor debe terminar su ejecución.

Detalles sobre el código de apoyo

En este programa, el código de apoyo proporcionado contiene algún elemento más que lo habitual en los otros programas. El código de apoyo incluye:

- `cliente.c`: En el subdirectorio `Cliente` se incluye una implementación de un cliente para el protocolo propuesto. Esta implementación ***no se debe modificar***, pero conviene consultarla para ver el tipo de interacción que espera la otra parte del protocolo.
- `message.h`: Define el prototipo de estructura intercambiada por medio de los mensajes UDP, así como varias constantes usadas, tanto por cliente como por servidor.
- `lanzador.sh`: Un sencillo *script* de apoyo que permite arrancar una instancia del servidor y toda una batería de clientes que soliciten transferencia de ficheros. Este *script* es especialmente útil para automatizar ciertas pruebas.

Páginas de Manual a Consultar

A título de ayuda se recomienda la consulta de las siguientes funciones o llamadas en las páginas del manual. Dichas páginas se pueden consultar vía el mandato `man`:

- `socket` (2)
- `bind` (2)
- `accept` (2)
- `listen` (2)
- `connect` (2)
- `getsockname` (2)
- `sendto` (2)
- `recvfrom` (2)
- `send` (2)
- `recv` (2)
- `exit` (2)

docencia/asignaturas/so4/prv/practicas/disenyo/servidor.2015a.txt · Última modificación: 2015/01/30 10:22 por frosaladm