Lab-hftp

Redes 2015

Nicolás Wolovick (2011) Carlos Bederián (2012, 2013)
Matias Molina (2015)

FaMAF, Universidad Nacional de Córdoba, Argentina

17 de Marzo de 2015

El plan

Explorar los distintos tipos de aplicaciones en una red.

Lab0 : Cliente
Proceso $\leftrightarrow$ 1 servidor

Lab1 : Servidor síncrono
1 cliente $\leftrightarrow$ Proceso

Lab3 : Servidor asíncrono
N clientes $\Leftrightarrow$ Proceso

Lab4 : Proxy
N clientes $\Leftrightarrow$ Proceso $\Leftrightarrow$ M servidores

No está en el lab : Peer-to-peer
Proceso $\Leftrightarrow$ N peers

Transporte, puerto y estilo conversacional

Transporte

TCP

Puerto

19500

Ver `/etc/services` y números de puerto de IANA.

Estilo conversacional

- cliente → servidor : pedidos
- servidor → cliente : respuestas

Pedidos

El cliente envía comandos terminados con \r\n:

- `get_file_listing`
- `get_metadata FILENAME`
- `get_slice FILENAME OFFSET SIZE`
- `quit`

Pedidos – Sintaxis

```
Eol ::= "\r\n"

Filename ::= (Alphanum | "." | "_" | "-")+

Integer ::= (Digit)+

Request ::= (Command Eol)+

Command ::= "get_file_listing"
  | "get_metadata" Space Filename
  | "get_slice" Space Filename Space Integer Space Integer
  | "quit"
```

Respuestas

El servidor responde con una línea que contiene un código, seguido de una cadena descriptiva del código:

0	La operación se realizó con éxito.
100	Se encontró un carácter \n fuera de un terminador de pedido \r\n.
101	Alguna malformación del pedido impidió procesarlo.
199	El servidor tuvo algún fallo interno al intentar procesar el pedido.
200	El comando no está en la lista de comandos aceptados.
201	La cantidad de argumentos no corresponde al comando, o los argumentos no tienen la forma correcta.
202	El pedido se refiere a un archivo inexistente.
203	El pedido se refiere a una posición inexistente en un archivo.

Si el código es 0, es seguido por el resultado del pedido.

Respuestas – Sintaxis

```
Reply ::= (Returncode (Result | ""))+

Returncode ::= Integer Space (Printable)+ Eol

Result ::= GFLresult | GMDresult | GSresult | Qresult
```

get_file_listing

Parámetros: Ninguno
Restricciones: Ninguna
Notas: Obtiene los nombres de los archivos disponibles en el directorio compartido, al momento de ejecutarse el comando. Notar las restricciones en el nombre de archivo que maneja para no tener problemas con los diferentes FS.
Respuestas posibles: 0, 199, 201
Resultado: `GFLresult ::= (Filename Eol)* Eol`
Ejemplos:

```
C: get_file_listing\r\n
S: 0 De pelos!\r\n
S: \r\n
```

```
C: get_file_listing\r\n
S: 0 OK\r\n
S: archivo1.txt\r\n
S: archivo2.jpg\r\n
S: \r\n
```

get_metadata filename

Parámetros: Nombre del archivo del que se quieren obtener los metadatos.
Restricciones: Ninguna
Respuestas posibles: 0, 100, 101, 199, 201, 202
Notas: El único metadato actual es el tamaño. Solo pueden obtenerse los metadatos de los archivos que serían devueltos por `get_file_listing`, y el tamaño reportado será el actual. (que puede no ser constante entre llamadas sucesivas)
Resultado: `GMDresult ::= Integer Eol`
Ejemplos:

```
C: get_metadata archivo1.txt\r\n
S: 0 OK\r\n
S: 3199\r\n
```

```
C: get_metadata noexiste.txt\r\n
S: 202 No existe ese archivo\r\n
```

get_slice filename offset size (1)

Parámetros: nombre del archivo, desplazamiento y tamaño de la porción.
Restricciones: ninguno
Respuestas posibles: 0, 101, 199, 201, 202, 203
Notas: mismas restricciones sobre el nombre de archivo. Se tiene que cumplir que $offset + size \leq file.size$. Se devuelve una secuencia de *fragmentos* $((size_i, seq_i))_{i=0}^{n}$, donde $0 \leq n$, $\sum_{i=0}^{n-1} size_i = size$, $(\forall : 0 \leq i < n : 0 < size_i = |seq_i|)$, $(size_n, seq_n) = (0, "")$ Además la concatenación de todas las secuencias debe ser la porción del archivo $\sum_{i=0}^{n-1} seq_i = file[offset, offset + size)$. Notar que puede haber substrings de `\r\n` dentro de la secuencia de bytes que no serán interpretados. Si hay un error la lectura del archivo, la suma de los tamaños será menor que el tamaño total.
Resultado: `GSresult ::= (Integer Space (Char)* Eol)+`

get_slice filename offset size (2)

Ejemplos:

```
C: get_slice uno.txt 10 30\r\n
S: 0 Todo ok\r\n
S: 10 r de La Ma\r\n
S: 20 ncha de cuyo nombre \r\n
S: 0 \r\n
```

```
C: get_slice uno.txt 10 30\r\n
S: 0 Okisssssss\r\n
S: 1 r\r\n
S: 1  \r\n
S: 10 de La Manc\r\n
S: 18 ha de cuyo nombre \r\n
S: 0 \r\n
```

```
C: get_slice uno.txt 16 0\r\n
S: 0 5/5\r\n
S: 0 \r\n
```

```
C: get_slice uno.txt 131072 1\r\n
S: 203 Fuera de archivo\r\n
```

quit

Parámetros: Ninguno
Restricciones: Ninguna
Respuestas posibles: 0, 100, 199
Notas: Corta la conexión con el servidor
Resultado: `Qresult ::= ""`
Ejemplos:

```
C: quit\r\n
S: 0 OK\r\n
```

```
C: quit\r\n
S: 199 Falla general en el server\r\n
```

La vida de un servidor TCP

La vida de un servidor TCP

1. s = socket()
2. s.bind()
3. s.listen()
4. c = s.accept()*
   - 4.1 (c.recv()$^+$ $\rightarrow$ c.send()*)*
   - 4.2 c.close()
5. s.close()

Lecturas cortas o largas

Podía suceder en el `read` o `write` de SistOp.

En Redes está acentuado para sus `recv`, `send`.

Lecturas cortas o largas

Podía suceder en el `read` o `write` de SistOp.
En Redes está acentuado para sus `recv`, `send`.

Con probabilidad cercana a 0

```
recv(7, "get_file_listing\r\n", 4096, 0) = 18
recv(7, "get_metadata uno.txt\r\n", 4096, 0) = 22
```

Lecturas cortas o largas

Podía suceder en el `read` o `write` de SistOp.
En Redes está acentuado para sus `recv`, `send`.

Con probabilidad cercana a 0

```
recv(7, "get_file_listing\r\n", 4096, 0) = 18
recv(7, "get_metadata uno.txt\r\n", 4096, 0) = 22
```

Lo que siempre pasa

```
recv(7, "get_", 4096, 0) = 4
recv(7, "file_listing\r\nget_", 4096, 0) = 16
recv(7, "metadata", 4096, 0) = 6
```

Lecturas cortas o largas

Podía suceder en el `read` o `write` de SistOp.
En Redes está acentuado para sus `recv`, `send`.

Con probabilidad cercana a 0

```
recv(7, "get_file_listing\r\n", 4096, 0) = 18
recv(7, "get_metadata uno.txt\r\n", 4096, 0) = 22
```

Lo que siempre pasa

```
recv(7, "get_", 4096, 0) = 4
recv(7, "file_listing\r\nget_", 4096, 0) = 16
recv(7, "metadata", 4096, 0) = 6
```

Implementar mecanismos de buffering interno hasta completar mensajes enteros.

Desconexión

¿Y si el cliente se va sin hacer `quit`?

Desconexión

¿Y si el cliente se va sin hacer `quit`?

- `send` lanza una excepción
- `recv` devuelve 0 bytes o lanza una excepción

Objetivos del Lab

- Realizar un programa servidor de archivos secuencial.
- Dado un protocolo definido de manera rigurosa, estudiarlo e implementarlo.
- Aprender a usar las primitivas de sockets para servidores.
- Entender que las aplicaciones de red son agnósticas respecto a la arquitectura, el sistema operativo y el lenguaje de programación utilizados.
- Comprender e implementar algunas cualidades de un servidor:
  - funcionamiento permanente,
  - robustez,
  - tolerancia a fallas,
  - seguridad,
  - uso apropiado de los recursos.

Fin

Plazo de entrega: Lunes 30 de Marzo, 23:59

¿Preguntas?