Arquitectura y Tecnología de Computadores

Práctica 1

Programación con el interfaz de sockets.

1. Servidor `echo` con concurrencia aparente

Se programará un servidor para el servicio `echo` (un servicio que simplemente vuelve a escribir en el socket de datos todo lo que ha recibido por él), con capacidad para atender hasta 5 clientes simultáneos mediante el uso de la función `select()`. El número de puerto en que escucha el servidor se ha de poder especificar desde la línea de comandos. No es necesario desarrollar clientes, ya que se puede utilizar `telnet` para hacer las pruebas.

2. Cliente POP3

El ejercicio consiste en construir un cliente para el protocolo POP3 (Post Office Protocol Versión 3). El programa tiene que ser capaz de interaccionar correctamente con cualquier servidor de POP3.

La especificación del protocolo POP3 se puede encontrar en la página web de la Internet Engineering Task Force:[1] `http://www.ietf.org/rfc/rfc1939.txt`. A la versión pdf del mismo documento se puede acceder desde este enlace: `http://www.faqs.org/rfcs/rfc1939.html`.

El objetivo del cliente es que un usuario pueda, desde línea de comandos, conectarse con su servidor POP3 para averiguar cuántos mensajes tiene en el buzón, el asunto y tamaño de cada uno de ellos, y tenga la posibilidad de descargárselos a un archivo de su disco local, o de borrarlos directamente del servidor sin descargarlos. Para ello, el cliente tendrá un interfaz de comandos desde el cual el usuario pueda solicitar ciertas acciones y observar en pantalla la respuesta.

Lo primero que debe hacer el cliente es solicitar nombre y clave[2] del usuario y enviárselas al servidor, tras lo cual, si la autenticación tiene éxito, el cliente queda en un bucle en el que puede recibir comandos del usuario y proporcionarle las respuestas apropiadas. Por ejemplo:

```
$ ./pop3cli maquinaservidora puerto
Introduzca el nombre de usuario: asfkag
Introduzca la clave: ******
Usuario identificado con éxito. Puede escribir HELP para ayuda.
POP3> _
```

En la invocación del cliente, el número de puerto será opcional, tomando como valor por defecto el especificado en el estándar. Los comandos que el cliente debe implementar son:

[1] La IETF se dedica a proponer y establecer los estándares que rigen la Internet. El proceso de adopción de un estándar se puede consultar en la propia página de la IETF pero básicamente consiste en la propuesta de un protocolo y posteriormente, cuando está asentado y probado, en su elevación a la categoría de estándar. En el caso de POP3, la "propuesta para comentar" (*Request for Comment* RFC) se hizo en Mayo de 1996 y ocupaba la posición 1939 de todas las realizadas hasta la fecha. Posteriormente, el protocolo se convirtió en el estándar de Internet número 53

[2] Para pedir la clave al usuario sin que esta sea visible en pantalla puede usarse la fiunción `getpass()`

STAT Muestra un mensaje indicando cuántos mensajes hay en el buzón y el tamaño total ocupado por los mismos (en kilobytes).

Ejemplo

```
POP3> STAT
Tu buzón tiene 3 mensajes que ocupan 28Kb.
```

HEADS Muestra una lista con tantos elementos como mensajes haya en el servidor. Para cada uno de ellos mostrará:

- El número de mensaje (el que suministraría el servidor ante una petición LIST)
- El tamaño en kilobytes del mensaje
- El remitente (campo `From:` de la cabecera)
- El asunto del mensaje (campo `Subject:` de la cabecera)

Ejemplo

```
POP3> HEADS
1: 23Kb
   From: qqwioe@asdfg.com
   Subject: Hello
2: 1Kb
   From: nadie@pruebas.com
   Subject: Esto es una prueba
```

DELE seguido de un número de mensaje, enviará al servidor la petición de borrado de ese mensaje

RSET enviará al servidor la petición de que anule los borrados previos

READ seguido de un número de mensaje, pedirá dicho mensaje al servidor y lo almacenará en un archivo local, cuyo nombre será igual al número del mensaje y la extensión `.txt` El fichero contendrá el mensaje completo, incluyendo la cabecera.

TOP seguido de un número de mensaje y de un contador de líneas, mostrará por pantalla los campos de remite (`From:`) y asunto (`Subject:`) de ese mensaje, junto con las primeras líneas del mensaje especificadas en el contador. El resto de la cabecera del mensaje no debe ser mostrada.

Ejemplo

```
POP3> TOP 2 2
   ------------------------------
   From: nadie@pruebas.com
   Subject: Esto es una prueba
   ------------------------------
   Hola,
   Este mensaje es solo una prueba para saber si
```

QUIT enviará al servidor la petición de desconexión y finalizará el cliente

Naturalmente todos estos comandos deben detectar apropiadamente si el servidor produce una respuesta positiva o negativa, y en el segundo caso imprimir información del problema.

Ejemplo

```
POP3> DELE 100
Error del servidor: No such message
```

Observar que algunos de estos comandos están directamente soportados por el protocolo POP3, mientras que otros (como HEAD) no lo están, y requerirán de la combinación de varias órdenes POP3 y un procesamiento de la respuesta en el lado del cliente.

Opcionalmente se pueden implementar comandos para que el usuario pueda dar al servidor directamente mandatos del protocolo POP3, como USER, PASS, NOOP, LIST. En realidad, se recomienda este enfoque para la realización de pruebas durante la fase de desarrollo.

3. Servidor POP3

Implementar un servidor de POP3 siguiendo las directrices de la RFC 1939. En la especificación del protocolo POP3 existen una serie de mandatos que son obligatorios para toda implementación del mismo. Son los siguientes: `STAT, LIST, RETR, DELE, NOOP, RSET, QUIT`. Aparte de esos mandatos, existe un conjunto de órdenes opcionales. Para esta práctica deben implementarse todos los obligatorios, y de los opcionales, se implementarán únicamente `USER, PASS` y `TOP`.

Debe ser posible especificar desde línea de comandos el número de puerto en el que escuchará el servidor, por ejemplo:

```
$ ./servidor 48765
```

Se sugiere usar como número de puerto las cinco últimas cifras del DNI del programador, para reducir las probabilidades de que dos alumnos intenten el mismo número de puerto.

No se pide que el servidor soporte concurrencia de varios clientes, por tanto basta un bucle en el que espera a que un cliente se conecte, y le da servicio hasta que éste se desconecte (ya sea con el comando QUIT o bien cerrando el socket sin más ceremonia, aunque en este segundo caso, según especifica el estándar, no se realiza la fase de "*update*"), tras lo cual vuelve a la espera por otro cliente.

El servidor mantendrá mensajes para un único usuario, cuyo nombre ha de ser "prueba" y cuya clave ha de ser el número de puerto por el que escucha el servidor. Los mensajes de este usuario estarán almacenados en un buzón (o *maildrop*). En la RFC no se describe cómo tiene que ser este *maildrop*. En nuestro caso vamos a optar por el siguiente formato:

- Los mensajes se almacenaran en un directorio denominado `mail` que colgará del directorio desde el que se lance la ejecución del servidor pop3.
- Dentro del directorio `mail`, cada mensaje se encontrará en un fichero. El nombre del fichero no tiene interés alguno. Puede ser cualquier nombre.
- Cada fichero, en formato ASCII, contiene un mensaje. Los mensajes siguen el siguiente formato:
  1. El fichero comienza con un número variable de líneas de cabecera. Cada línea de cabecera comienza con una palabra (*tag*) que indica la información que viene a continuación, seguida de dos puntos (`:`), un espacio, y la información asociada al tag. La línea termina con un carácter retorno de carro y otro de línea nueva. Ejemplo:
     `Date: Sun, 27 Dec 1998 18:58:37 +1300`
  2. Tras la cabecera, aparece una línea en blanco (contiene únicamente los caracteres `CRLF`), que actúa como separador entre la cabecera y el cuerpo del mensaje.
  3. Las restantes líneas del fichero constituyen el cuerpo del mensaje, hasta alcanzar el fin de fichero.

A modo de ejemplo, en la página web de la asignatura se suministra un archivo con varios mensajes (`http://www.atc.uniovi.es/inf_superior/4atc/DISTRIBUIDOS/practicas.html`). El alumno puede descomprimir este archivo en la carpeta `mail` antes mencionada para tener un *maildrop* con el que hacer pruebas.

Quizás la parte más compleja del servidor sea el manejo de este buzón. Se suguiere la siguiente estrategia. Una vez el cliente ha superado la fase de autenticación, antes de entrar en la fase de transacción, se construye en memoria una estructura de datos con información sobre el buzón. Esto constará de los siguientes pasos:

- Creación de un fichero "cerrojo" (será un indicador de que el buzón está siendo utilizado). Se creará en la carpeta desde la que corre el servidor y puede llamarse, por ejemplo "mail.lock". Si no fuera posible crear el fichero se retornaría un error al cliente.

- Obtención de acceso exclusivo sobre el fichero anterior (usando la función `flock()`[3]). Si el acceso exclusivo se obtiene con éxito, se entiende que ningún otro proceso está accediendo al buzón, y podemos proseguir. Si otro proceso hubiera obtenido antes este acceso exclusivo, el servidor quedará bloqueado hasta que el otro proceso libere el cerrojo. Se puede evitar este bloqueo añadiendo la opción `LOCK_NB` al llamar a `flock()`. En este caso `flock()` pondrá en la variable `errno` el valor `EWOULDBLOCK`, y el servidor entenderá que el cerrojo está ocupado y por tanto no debe acceder al *maildrop*, por lo que retornará un error al cliente.

- Obtención de información sobre los mensajes existentes en el buzón, y almacenamiento de esta información en una estructura de datos en el servidor. Será esta estructura la que se manipulará mediante los mandatos LIST, STAT, DELE y RSET, y no los mensajes del buzón.

  Esta estructura puede ser por ejemplo un array o una lista enlazada, en la que cada elemento representa un mensaje. Para cada mensaje almacenará la siguiente información:

  - El número del mensaje (el que se reportará con LIST y al que el cliente hace referencia en los comandos DELE, TOP, RETR, etc.)
  - El nombre del fichero en el que ese mensaje está almacenado (lo necesitará para abrir ese fichero y enviárselo al cliente cuando éste lo solicite con RETR o TOP)
  - Un indicador de si ese mensaje debe borrarse o no. Inicialmente el mensaje se marca para no borrar, pero cambiará de estado con los mandatos DELE y RSET. Este indicador se examinará en la fase "*update*" para actualizar el buzón. Observar que este indicador debe consultarse también para determinar si el mensaje debe listarse bajo LIST o si se permiten otras operaciones sobre el mensaje (como RETR, TOP, etc.)
  - El tamaño del mensaje en bytes. Resulta conveniente tenerlo pre-almacenado para reportarlo rápidamente ante el mandato LIST o para calcular el tamaño total ocupado por los mensajes (excepto los marcados para borrar) ante el mandato STAT.

  Para obtener toda la información anterior será necesario utilizar las funciones `opendir()`, `readdir()` y `closedir()`[4], que permiten obtener los nombres de los ficheros que hay en una carpeta[5]. Para averiguar el tamaño de un mensaje, la forma más rápida puede ser abrir el fichero correspondiente (`fopen()`), colocar el puntero de lectura al final del fichero (`fseek()`) y averiguar el valor del puntero de lectura (`ftell()`).

Cuando el servidor pase a la fase de "*update*", deberán realizarse los siguientes pasos:

- Actualizar el buzón. Es decir, recorrer la estructura anterior y borrar los mensajes marcados para borrado. Para borrar un fichero debe usarse la función `unlink()`.
- Liberar el espacio ocupado por la estructura de datos.
- Liberar el acceso exclusivo sobre el cerrojo (de nuevo usando la función `flock()`).
- Eliminar el fichero que actuó como cerrojo.

[3] El uso de la función `flock()` y del resto de funciones mencionadas en este documento se puede leer en las páginas de `man`

[4] Consultar el manual

[5] Cuidado, los pseudo-ficheros llamados "." y ".." también son reportados por estas funciones, pero evidentemente no son mensajes del buzón

4. Entrega de ejercicios

La práctica se entregará en las máquinas `sirio` y `orion` en un subdirectorio del directorio raíz de cada alumno denominado `distribuidos/p1`. En ese directorio tiene que haber un fichero comprimido denominado `p1.zip` o `p1.tar.gz` que al ser descomprimido tiene que generar todos los ficheros necesarios para que los distintos ejercicios se puedan compilar y enlazar en la máquina en que se encuentre. Dentro del archivo comprimido se debe incluir algún tipo de mecanismo de compilación automatizada de los ejercicios de la práctica (un fichero de mandatos del *shell* o bien un `Makefile`). Tiene que ser posible descomprimir y ejecutar los programas en cualquier entorno y por cualquier usuario.

Los ejecutables generados por la compilación se deben llamar **`echoserv`** (servidor concurrente de `echo`), **`pop3cli`** (cliente POP3) y **`pop3serv`** (servidor POP3). En la carpeta donde esté el ejecutable **`pop3serv`** debe haber también una carpeta llamada **`mail`** que contenga unos mensajes para probar el servidor (puede tratarse de los mensajes de prueba suministrados por los profesores).

5. Corrección de los ejercicios

El servidor `echo` se probará conectando a él varios clientes vía `telnet`.

En cuanto a la implementación del protocolo POP3, al tratarse de un protocolo estándar de Internet, la corrección de los ejercicios consistirá en la ejecución de los mismos contra implementaciones que ya estén en uso. Así por ejemplo, el servidor de POP3 se probará con el programa `popclient` existente en `sirio`, o alguno similar[6], mientras que el cliente se probará con algún servidor de POP3 disponible públicamente.

Se recomienda a los alumnos que se utilicen esos programas para comprobar la correcta realización de la práctica.

Anexo: Funciones útiles

Probablemente sea necesario utilizar varias de las siguientes funciones en diferentes puntos de las prácticas. Se deja al alumno que averigue la sintaxis y el uso de estas funciones en el manual de Unix (`man`):

Proceso de cadenas: `strdup()`, `strcmp()`, `strtok()`, `strlen()`, `ischar()`, `isspace()`, `iscntrl()`, `toupper()`, `tolower()`.

Conversión desde/hacia cadenas: `sprintf()`, `sscanf()`, `atoi()`.

Lectura de directorios: `opendir()`, `readdir()`, `rewinddir()`, `closedir()`.

Manejo de ficheros: `fopen()`, `fclose()`, `fseek()`, `ftell()`, `fscanf()`, `fprintf()`, `open()`, `lockf()`, `close()`, `unlink()`.

Entrada/Salida: `printf()`, `perror()`, `scanf()`, `fgets()`, `getpass()`, `getchar()`, `setvbuf()`.

Gestión de memoria: `malloc()`, `free()`.

[6] Cualquier cliente de correo con soporte POP3, como Outlook, Mozilla o Thunderbird también podría servir para probar el servidor.