.UBAfiuba
FACULTAD DE INGENIERÍA

Guía de Ejercicios 4: Sockets en Rust

75.42 / 95.08 Taller de Programación I - 1er C 2021
Ing. Pablo Deymonnaz

Ejercicio 1 - Introducción

Ítem A

Escribir un programa tal que el hilo main crea un thread hijo que actuará como cliente, mientras el padre actúa como servidor. La comunicación se establece para enviar y recibir un saludo, por ejemplo: "Hola hijo" y "Buen día Papá".

Ítem B

Modificar el programa del ejercicio anterior para que el servidor pueda gestionar más de un cliente.

Ejercicio 2 - Mini Chat

Implementar un programa para armar una sala de chat.

El programa inicia y le pide un *nickname* al usuario. Luego, abre un socket servidor ligado a un puerto configurable. A continuación, realiza *broadcasting* del nickname a la subred, para después quedar escuchando mensajes.

Si recibe un mensaje de broadcast con un nickname, lo agrega a la lista de usuarios de la sala de chat, junto a la dirección IP de quien se lo envía.

Si recibe otro mensaje, se lo imprime por pantalla.

Para recibir un mensaje del usuario, se debe leer de la estrada estándar (*stdin*). Si se lo antecede con el nickname de destino, se lo envía a ese destinatario en particular. Con *Enter*, se transmite el mensaje.

Ejercicio 3 - FTP Honeypot

Introducción

Un "honeypot" es una aplicación que simula ser un servidor de otra aplicación mayor, para que cuando un usuario malicioso se conecte, ataque este servidor falso, permitiendo tomar registro de las técnicas de ataque que se utilizan en la red. En este trabajo práctico prototiparemos un honeypot de un servidor FTP.

Protocolo FTP

El protocolo FTP es un **protocolo de texto**, formado por mensajes delimitados por un **salto de línea**.

Los comandos son palabras en su mayoría de 4 letras, seguidos de un espacio que los separa de sus argumentos. El servidor responde con un código numérico y un texto descriptivo.

Cliente FTP

Los comandos que puede ejecutar el cliente son los siguientes

- `USER <username>`: Envía un nombre de usuario para realizar login.
- `PASS <password>`: Envía un password para el usuario.
- `SYST`: Consulta información del sistema
- `LIST`: Consulta los archivos contenidos en el directorio actual
- `HELP`: Consulta los comandos disponibles
- `PWD`: Consulta el directorio actual
- `MKD`: Crea un directorio
- `RMD`: Elimina un directorio

Para simplificar el trabajo práctico, la transferencia de datos se realizará respondiendo en el mismo puerto por el cual se conectan inicialmente los clientes.

Esto implica que los comandos `PASV` y `PORT` no serán necesarios para transmitir información (por ejemplo con el comando `LIST`).

Servidor FTP

El servidor FTP responde con los siguientes mensajes:

- Cuando un cliente recién se conecta: 220 `<newClient>`
- Cuando un cliente quiere operar sin haber hecho login: 530 `<clientNotLogged>`
- Luego de que un cliente envía un comando `USER`: 331 `<passRequired>`
- Si el usuario envía un comando que no es `PASS` luego de un comando `USER`: 530 `<clientNotLogged>`
- Si el usuario realiza un login válido (el usuario enviado con `USER` y la contraseña enviada con `PASS` son válidas, es decir, concuerdan con las configuradas): 230 `<loginSuccess>`
- Si el usuario realiza un login inválido: 530 `<loginFailed>`

Una vez que el cliente realizó un login exitoso, se habilitan varios comandos al cliente:

- Si el usuario envía el comando `SYST`: 215 `<systemInfo>`
- Si el usuario envía el comando `HELP`: 214 `<commands>`
- Si el usuario envía el comando `LIST`, se realizará una respuesta en 3 partes

La primera línea será 150 `<listBegin>`
Luego enviará una línea por cada directorio cargado, con el siguiente formato:

```
drwxrwxrwx 0 1000 1000 4096 Sep 24 12:34 <nombre directorio>
```

Los directorios serán listados en orden alfabético.
Luego de enviar las líneas, enviará 226 `<listEnd>`.

- Si el usuario envía el comando `PWD`: 257 `<currentDirectoryMsg>`
- Si el usuario envía el comando `MKD <nombreDir>`, intenta agregar "`<nombreDir>`" a la lista de directorios existentes responde:
    - 257 "`<nombreDir> <mkdSuccess>`" en caso de que el directorio no existía.

    - 550 <mkdFailed> si ya existía

- Si el usuario envía el comando RMD <nombreDir> intenta remover "<nombreDir>" de la lista de directorios y responde:
    - 250 <rmdSuccess> si el directorio existe.
    - 550 <rmdFailed> si no existía.
- Finalmente, el usuario puede pedirle al servidor que termine la conexión enviando la palabra QUIT. En tal caso el servidor responde con 221 <quitSuccess> y ambos cierran sus conexiones ordenadamente.

Cabe destacar que la entrada de comandos puede no terminar con QUIT, debiendo el cliente cerrar su conexión al llegar al final del stream y el servidor liberar sus recursos adecuadamente.

Configuración de mensajes

El servidor FTP lee un archivo de configuración las siguientes variables

- user: Usuario para realizar login
- password: Password para realizar login
- newClient: Mensaje enviado a un cliente recién conectado
- clientNotLogged: Mensaje enviado a un cliente que quiere operar sin haber hecho login
- passRequired: Mensaje de solicitud de password
- loginSuccess: Mensaje de password aceptado
- loginFailed: Mensaje de password rechazado
- systemInfo: Mensaje de información del sistema
- commands: Mensaje con los comandos disponibles
- unknownCommand: Mensaje de comando inválido
- quit: Mensaje de desconexión del usuario

El archivo de configuración posee un formato "clave=valor", y debe ser cargado al iniciar la aplicación. No se validará que se encuentren todas las claves necesarias, en caso de faltar una clave necesaria, es a decisión del desarrollador cómo contemplar este caso.

Formato de Línea de Comandos

Servidor

```
./server <puerto/servicio> <configuracion>
```

Donde <puerto/servicio> es el puerto TCP (o servicio) en donde estará escuchando la conexiones entrantes, <configuracion> el archivo con las variables del servidor.

Cliente

El cliente se ejecuta utilizando el siguiente formato de línea de comandos

```
./client <ip/hostname> <puerto/servicio>
```

El cliente se conectará al servidor corriendo en la máquina con dirección IP <ip> (o <hostname>), en el puerto (o servicio) TCP <puerto/servicio>.

Requerimientos no funcionales

Los siguientes son los requerimientos no funcionales para la resolución de los ejercicios:

- El proyecto deberá ser desarrollado en lenguaje Rust, usando las herramientas de la biblioteca estándar.
- No se permite utilizar *crates* externos. El único crate autorizado a ser utilizado es *rand*[1] en caso de que se quiera generar valores aleatorios.
- El código fuente debe compilarse en la versión stable del compilador y no se permite utilizar bloques unsafe.
- El código deberá funcionar en ambiente Unix / Linux.
- El programa deberá ejecutarse en la línea de comandos.
- La compilación no debe arrojar *warnings* del compilador, ni del linter *clippy*.
- Las funciones y los tipos de datos (*struct*) deben estar documentadas siguiendo el estándar de *cargo doc*.
- El código debe formatearse utilizando *cargo fmt*.
- Las funciones no deben tener una extensión mayor a 30 líneas. Si se requiriera una extensión mayor, se deberá particionarla en varias funciones.
- Cada tipo de dato implementado debe ser colocado en una unidad de compilación (archivo fuente) independiente.

[1] https://crates.io/crates/rand