Proyecto IRC (Internet Rust Chat)

Objetivo

El objetivo del presente Trabajo Práctico consiste en el desarrollo de un servidor y un cliente de chat siguiendo los lineamientos del protocolo IRC.

Introducción

Internet Relay Chat (IRC) es un protocolo de mensajería de texto diseñado para conversaciones grupales, mediante el uso de canales, con soporte adicional para conversaciones uno a uno.

El protocolo funciona con un modelo cliente-servidor, donde los clientes se conectan a un servidor, que a su vez puede estar conectado a otros en una red de servidores.

La definición base de como funciona esta dada en documentos RFCs. El mayormente adoptado como base es el RFC1459, aunque la mayoría de las redes no se adecúan completamente a ninguna RFC.

Para el desarrollo del TP, utilizaremos la anterior mencionada RFC como el documento técnico con la descripción del trabajo. Además, dividiremos la entrega en dos etapas a desarrollar durante el cuatrimestre.

Recomendamos adicionalmente leer este pequeño documento que sirve de guía para leer las RFC

Desarrollo

Elementos del sistema

Servidor

Los servidores actúan como el lugar adonde los usuarios se conectan para hablar entre ellos. A su vez, un servidor puede conectarse con otros servidores, para armar una red IRC única, manteniendo una topología de spanning tree. La configuración de spanning tree, permite a mensajes viajar a traves de la red, sin que los servidores se tengan que preocupar porque los mismos entren en un loop y nunca dejen de ser re enviados.

Cliente

El cliente es la aplicación que permite al usuario conectarse a la red. Cada cliente en la red tiene un nickname de hasta 9 caracteres. Además, cada cliente tiene un hostname, un username, y un servidor asociado al cual esta conectado. El hostname es la IP o una dirección DNS y el username puede ser, por ejemplo, la primera parte de un email.

Canales

Los canales son grupos de clientes. Estos se crean cuando el primer cliente se une a uno, y se eliminan cuando el último sale. Cada canal tiene un nombre que debe comenzar con un & o un #, puede tener hasta 200 caracteres, y **no** puede contener espacios, el ASCII 7 (control+G) o comas, ya que estas son utilizadas como separadores por el protocolo.

Además los canales pueden ser distribuidos entre varios servidores, en cuyo caso se indican con #, o ser visibles por un solo servidor, en cuyo caso se indican con &. Además pueden tener varios modos adicionales que se indican en la especificación.

Operadores

Los operadores son los administradores (*admins*) de los servidores, pudiendo desconectarlos y conectarlos a la red, y además pueden bloquear (*ban*) y expulsar (*kick*) usuarios. El admin puede serlo de toda la red, o de un único servidor.

Operadores de canales

Los operadores de canales son admins de canales. Pueden expulsar clientes, invitarlos, y cambiar el modo del canal y su topic.

Mensajes

Los mensajes son el mecanismo de comunicación cliente-servidor y servidor-servidor.

Consisten de 3 partes: un prefijo, un comando, y sus parámetros.

Si existe un prefijo, este se indica con un caracter ":" (0x3b). El mismo se utiliza para los mensajes entre servidores para indicar el origen del mismo.

El comando puede ser cualquier comando IRC válido o 3 dígitos en ASCII.

El mensaje debe terminar con un CR-LF (Carriage return - Line Feed), y no puede exceder los 520 caracteres.

La descripción de los mensajes se da en formato BNF.

::= [':'] ``` ::= | ['!'] ['@'] ::= { } | ::= ' ' { ' ' } ::= [':' |]

::= <Any *non-empty* sequence of octets not including SPACE or NUL or CR or LF, the first of which may not be ':'> ::= <Any, possibly *empty*, sequence of octets not including NUL or CR or LF> ::= CR LF

Este formato see lee por ejemplo: "El comando es una letra y varias letras repetidas o 3 números" "El parámetro es un espacio y opcionalmente un ':' y un trailing o un middle y params"

Hitos del desarrollo del Proyecto

Entrega intermedia

Solicitamos para la primera entrega tener una aplicación servidor (actuando como servidor individual o *single server*) y una aplicación cliente que permita comunicar a múltiples usuarios conectados tanto en el mismo host (utilizando distinta configuración de puertos) como en distintos hosts.

Ambas aplicaciones deberan soportar los siguientes mensajes:

Conexión y Registración (RFC 1459 sección 4.1)

- Password message 4.1.1
- Nickname message 4.1.2
- User message 4.1.3
- Operator message 4.1.5
- Quit message 4.1.6

Intercambio de mensajes (RFC 1459 sección 4.4)

- Private messages 4.4.1
- Notice messages 4.4.2

Channels (RFC 1459 sección 4.2)

- Join message 4.2.1
- Part message 4.2.2
- Names message 4.2.5
- List message 4.2.6

- Invite message 4.2.7

Informacion de Usuarios (RFC 1459 sección 4.5)

- Who query 4.5.1
- Whois query 4.5.2

Entrega final

Se incorporara el soporte para multiples servidores, que funcionaran basados en la topología de spanning tree.

Ademas de las funcionalidades de la entrega intermedia se agregara soporte a los siguientes mensajes:

Conexión y Registración (RFC 1459 sección 4.1)

- Server message 4.1.4
- Server Quit message 4.1.7

Channels (RFC 1459 sección 4.2)

- Mode message 4.2.3
- Topic message 4.2.4
- Kick message 4.2.8

Other Features

- Away message 5.1

Interfaz gráfica

Se debe implementar una interfaz gráfica utilizando la biblioteca GTK, mediante el crate gtk-rs.

La interfaz deberá permitir el uso completo de las funcionalidades solicitadas y su apariencia deberá ser similar a la de cualquier software de mensajería de múltiples canales de los usados popularmente. Esto es, deberá mostrar una lista de canales, una lista de los usuarios del canal, los mensajes enviados en el canal y un espacio para escribir un mensaje en ese canal.

Evaluaciones

El desarrollo del proyecto tendrá un seguimiento directo semanal por parte del docente a cargo del grupo.

Se deberá desarrollar y presentar los avances y progreso del trabajo semana a semana (simulando un *sprint* de trabajo). Cada semana, cada docente realizará una valoración del estado del trabajo del grupo.

El progreso de cada semana deberá ser acorde a lo que se convenga con el docente para cada sprint. Si el mismo NO cumple con la cantidad de trabajo requerido, el grupo podrá estar desaprobado de forma prematura de la materia, a consideración del docente.

Se deja constancia que las funcionalidades requeridas por este enunciado son un marco de cumplimiento mínimo y que pueden haber agregados o modificaciones durante el transcurso del desarrollo por parte del docente a cargo, que formarán parte de los requerimientos a cumplir. Cabe mencionar que estos desvíos de los requerimientos iniciales se presentan en situaciones reales de trabajo con clientes.

Finalización del Proyecto

El desarrollo del proyecto finaliza el último día de clases del cuatrimestre. En esa fecha, cada grupo deberá realizar una presentación final y se hará una evaluación global del trabajo.

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 en caso de que se quiera generar valores aleatorios, además del crate de GTK mencionado anteriormente.
- 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**.

- Cada tipo de dato implementado debe ser colocado en una unidad de compilación (archivo fuente) independiente.

Criterios de aprobación del Trabajo Práctico / Cursada de la materia

Para aprobar la cursada de la materia, se debe aprobar el Trabajo Práctico.

Estos son los criterios de aprobación:

- El software desarrollado funciona correctamente.
- Se puede testear en la red de Bit Torrent oficial.
- El proyecto debe estar debidamente documentado siguiendo los estándares de documentación de **cargo doc**.
- Se debe hace uso de las herramientas **cargo fmt** para formatear el código fuente y de **cargo clippy** para asegurar que el código es lo más idiomático posible. No debe haber warnings de clippy.
- El trabajo del grupo fue desarrollado con un esfuerzo constante y parejo a lo largo del cuatrimestre.
- Se cumplieron los objetivos planteados y comprometidos con los tutores en las reuniones de seguimiento.
- Se desarrollaron tests unitarios y de integración de las partes importantes del proyecto.
- Se hizo un uso correcto de la metodología de trabajo con Github.