Universidad del Valle de Guatemala

Facultad de Ingeniería

Sistemas Operativos

PROYECTO 1

CHAT

Gabriel Paz 221087

José Gramajo 22907

Angel Herrarte 22873

28 de marzo del 2025, Guatemala de la Asunción

Link de repositorio : https://github.com/Abysswalkr/Proyecto-Chat.git

Introducción

Su finalidad es reforzar los conocimientos en procesos, hilos, concurrencia y comunicación entre procesos. Se desarrollará una aplicación de chat en C/C++ que constará de dos partes: un servidor y uno o más clientes.

Objetivos

- **Objetivo General:**
  Desarrollar un sistema de chat que permita a múltiples clientes conectarse a un servidor, enviarse mensajes de forma general (broadcast) y directa (DM), y gestionar estados (ACTIVO, OCUPADO, INACTIVO).

- **Objetivos Específicos:**

  - Implementar el registro y liberación de usuarios en el servidor.

  - Permitir la comunicación en grupo (broadcasting) y la comunicación privada entre usuarios.

  - Utilizar multithreading para manejar de forma concurrente las conexiones de múltiples clientes.

  - Implementar cambios de estado de usuario y actualización automática por inactividad.

  - Definir un protocolo de comunicación que permita la interoperabilidad entre clientes y servidores de distintos grupos.

Descripción del Proyecto

El sistema de chat se compone de dos programas:

Servidor

- **Funcionalidad:**

  - Mantener una lista de clientes/usuarios conectados.

  - Registrar y liberar usuarios.

  - Enviar mensajes broadcast a todos los usuarios o mensajes directos a uno en específico.

  - Proveer información de usuario y actualizar el estado (ACTIVO, OCUPADO, INACTIVO).

  - Manejar conexiones de clientes mediante multithreading.

- **Ejecución:**
  El servidor se ejecuta en una máquina Linux (aunque se ha adaptado la versión para Windows usando MinGW, MSYS2 o WSL también es viable) con el comando:

```
gabriel@DELL-GABRIEL:/mnt/c/UVG/SISTEMAS OPERATIVOS/ProyectoChat/Proyecto-Chat/server$ .\server.exe
```

Cliente

- **Funcionalidad:**
  - Conectarse al servidor mediante la IP y puerto especificados.
  - Registrarse en el servidor enviando su nombre de usuario.
  - Enviar mensajes de broadcast y mensajes directos (DM).
  - Solicitar y mostrar el listado de usuarios conectados.
  - Consultar información de un usuario en particular.
  - Permitir el cambio de estado (ACTIVO, OCUPADO, INACTIVO) y mostrar una interfaz de chat con comandos de ayuda.
- **Ejecución:**
  El cliente se ejecuta con el comando:

<nombredelcliente> <nombredeusuario> <IPdelservidor> <puertodelservidor>

```
gabriel@DELL-GABRIEL:/mnt/c/UVG/SISTEMAS OPERATIVOS/ProyectoChat/Proyecto-Chat/client$ .\client.exe NombreUsuario  127.0.0.1 50213
```

Definición del Protocolo de Comunicación

El protocolo de comunicación se basa en el intercambio de mensajes en formato JSON. Cada mensaje enviado desde un cliente al servidor (y viceversa) contiene campos que indican el **tipo** o **acción** de la comunicación. A continuación se describen los componentes básicos:

Mensajes de Registro

- **Tipo:** "REGISTRO"
- **Campos:**
  - "usuario": Nombre del usuario.
  - "direccionIP": Dirección IP del cliente (se envía "0.0.0.0" para que el servidor la detecte).

Mensajes de Salida (Desconexión)

- **Tipo:** "EXIT"

- **Campos:**
    - "usuario": Nombre del usuario que cierra la sesión.

Cambio de Estado

- **Tipo:** "ESTADO"
- **Campos:**
    - "usuario": Nombre del usuario.
    - "estado": Nuevo estado, entre "ACTIVO", "OCUPADO" o "INACTIVO".

Mensajes de Consulta

- **Listado de Usuarios:**
    - **Acción:** "LISTA"
    - **Opcional:** Se puede enviar el nombre del usuario solicitante.
- **Información de Usuario:**
    - **Tipo:** "MOSTRAR"
    - **Campos:**
        - "usuario": Nombre del usuario del que se solicita la información.

Mensajes de Comunicación

- **Broadcast:**
    - **Acción:** "BROADCAST"
    - **Campos:**
        - "nombre_emisor": Nombre del usuario emisor.
        - "mensaje": Texto del mensaje.
- **Mensaje Directo (DM):**
    - **Acción:** "DM"
    - **Campos:**
        - "nombre_emisor": Nombre del usuario emisor.
        - "nombre_destinatario": Nombre del usuario destinatario.

        - "mensaje": Texto del mensaje.

El protocolo se acordará en conjunto con otros grupos para garantizar la interoperabilidad entre clientes y servidores.

Investigaciones Realizadas

Para el desarrollo del proyecto se investigaron los siguientes temas:

- **Sockets en C:**
  Se consultó la documentación de Linux (por ejemplo, man 7 ip en https://linux.die.net/man/7/ip) para comprender el funcionamiento de los sockets y la comunicación entre procesos.

- **Multithreading en C:**
  Se estudió el uso de pthreads para manejar múltiples conexiones concurrentes en el servidor, así como su portabilidad en entornos Windows (utilizando pthreads-win32).

- **Formato JSON:**
  Se investigó el uso de la librería cJSON para parsear y generar mensajes en formato JSON, facilitando la estructuración de datos en la comunicación.

- **Protocolos de Comunicación:**
  Se definió un protocolo de comunicación basado en mensajes JSON para regular la interacción entre clientes y servidor (registro, mensajes, consultas, cambio de estado, etc.).

- **Portabilidad:**
  Se exploraron alternativas para compilar y ejecutar el proyecto tanto en Linux como en Windows, utilizando entornos como WSL, MSYS2 y MinGW.

Arquitectura del Sistema

Servidor

- **Responsable de:**
    - Registrar usuarios y mantener una lista de clientes conectados.
    - Distribuir mensajes broadcast y mensajes directos.
    - Atender solicitudes de cambio de estado y consultas de información.
- **Multithreading:**
  Se utiliza un hilo para cada conexión entrante, y otro hilo para verificar la inactividad de los usuarios.

Cliente

- **Responsable de:**

- Conectarse y registrarse con el servidor.
- Enviar comandos y mensajes a través de una interfaz de línea de comandos.
- Mostrar mensajes recibidos (broadcast, DM, respuestas de solicitudes).

Implementación

Detalles del Código

- **Lenguaje:** C
- **Librerías Clave:**
  - *Sockets*: Se usan funciones de sockets para la comunicación TCP.
  - *Pthreads*: Para manejar múltiples hilos en el servidor y el cliente.
  - *cJSON*: Para manejar mensajes en formato JSON.
- **Adaptaciones para Windows:**
  - Se usan bloques condicionales #ifdef _WIN32 para incluir las cabeceras de WinSock2 y Windows.
  - Se redefinen close() y sleep() para que utilicen closesocket() y Sleep() respectivamente.
  - Se llama a WSAStartup() al inicio y WSACleanup() al finalizar en Windows.

Instrucciones de Compilación

- **En Windows con MinGW:**
  - Colocar en el mismo directorio los archivos client.c, cJSON.c y cJSON.h (si no se tiene cJSON instalado globalmente).
  - Abrir una consola (CMD o PowerShell) y verificar que la ruta de MinGW esté en el PATH.
  - Ejecutar:

```
PS C:\UVG\SISTEMAS OPERATIVOS\ProyectoChat\Proyecto-Chat> gcc server.c cJSON.c -o server.exe -lws2_32 -lpthread
```

Para ejecutar:

<nombredelcliente> <nombredeusuario> <IPdelservidor> <puertodelservidor>

```
gabriel@DELL-GABRIEL:/mnt/c/UVG/SISTEMAS OPERATIVOS/ProyectoChat/Proyecto-Chat/client$ .\client.exe NombreUsuario  127.0.0.1 50213
```

- **En Linux:**

- Compilar con:

Para ejecutar:

Conclusiones y Aprendizajes

- Se aprendió a implementar un sistema de chat en C/C++ utilizando sockets y multithreading.
- El proyecto refuerza conocimientos sobre comunicación entre procesos, concurrencia y manejo de JSON.
- Se abordó la portabilidad del código, adaptándolo para que pueda compilarse en Linux y Windows.
- La definición de un protocolo de comunicación basado en JSON facilitó la estructuración de los mensajes entre clientes y servidor.