Tecnológico de Costa Rica

IC4700 - Lenguajes de Programación

Proyecto 1: Paradigma Imperativo

Integrantes:

Kevin Núñez Camacho

Yeri Porras Víquez

Roy Silva

II Semestre – 2025

Tabla de contenido

Enlace al repositorio en GitHub 4
Instalación del programa 4
Requisitos previos a la instalación: 4
Paso 1: Instalación del editor de código 4
Paso 2: Actualizar repositorios del sistema 4
Paso 3: Instalar el compilador y herramientas de desarrollo 4
Paso 4: Instalación de bibliotecas adicionales 5
Paso 5: Instalar GitHub Desktop 5
Instalación de dependencias 6
Compilación del programa 6
Paso 1: Clonar el repositorio de GitHub 6
Paso 2: Compilar el proyecto 6
Manual de Usuario 7
Conectarse como Usuario 8
Funcionalidades Principales 8
Listar Usuarios Conectados 8
Enviar Mensajes a Sí Mismo 9
Conectar Múltiples Usuarios 10
Cerrar el Sistema 11
Arquitectura Lógica Utilizada 11
Visión General 11
Componentes Principales 11
Estructura de Archivos 11
Explicación Detallada por Componente 12
1. Servidor (server.cpp) 12
2. Cliente (cliente.cpp) 13
3. Biblioteca de Configuración (configReader.h) 13
4. Archivo de Configuración (config.txt) 14

Flujo de Datos 14
Registro de Usuario: 14
Envío de Mensajes: 14
Mecanismos de Sincronización 14
Manejo de Excepciones y Errores 15
Seguridad 15
Escalabilidad 15
Limitaciones 15

Enlace al repositorio en GitHub

El código fuente del proyecto está disponible en el siguiente repositorio:
https://github.com/Santiago146/Proyecto1_Lenguajes_de_Programacion

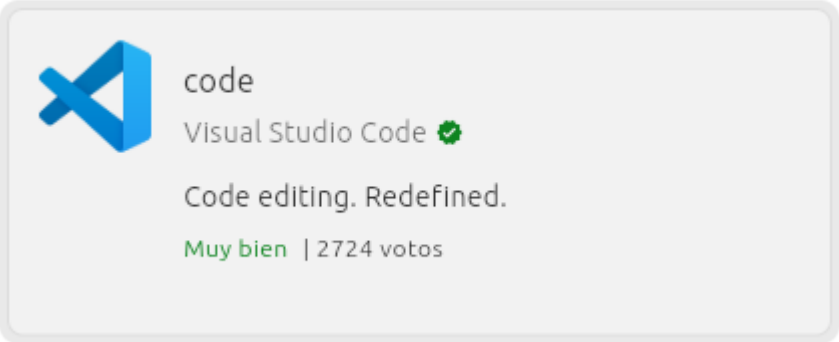
Instalación del programa

Requisitos previos a la instalación:

Este proyecto está hecho para poder correr en sistemas basados en Ubuntu/Debian, en específico Linux

Paso 1: Instalación del editor de código

Debemos tener en claro que el programa se debe ejecutar en Linux, luego para poder ejecutarlo bien es recomendable instalar el editor de código con el cual se hicieron todas las pruebas, depuraciones y creación del programa, dicho editor de código es Visual Studio Code, el cuál se instala en la tienda del sistema operativo y se busca con el nombre *Code*.



Paso 2: Actualizar repositorios del sistema

Debemos abrir la terminal de sistema y actualizar los repositorios del sistema con el siguiente comando:

```
sudo apt update
```

Paso 3: Instalar el compilador y herramientas de desarrollo

Instalar el compilador g++ que nos ayuda a compilar los archivos .cpp en donde tenemos implementado el programa y las herramientas de desarrollo esenciales, con el siguiente comando:

sudo apt install build-essential

Este comando instala:

- El compilador g++
- El compilador gcc
- make y otras utilidades de desarrollo
- Bibliotecas básicas de desarrollo

Luego verificamos la versión de g++ para ver si se instaló correctamente, con el siguiente comando:

g++ --version

Si se instala de manera correcta debe de enviar un mensaje siguiendo la misma estructura que el siguiente ejemplo:

```
kevin@Kevin:~/Documentos/GitHub/Proyecto1_Lenguajes_de_Programacion$ g++ --versi
on
g++ (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Debemos asegurarnos que la versión sea igual o superior a la 7.0 para garantizar el soporte completo de C++.

Paso 4: Instalación de bibliotecas adicionales

Ejecutar en la misma terminal los siguientes comandos uno por uno:

sudo apt install libstdc++6

sudo apt install gdb

Paso 5: Instalar GitHub Desktop

Para poder clonar el proyecto debemos instalar GitHub Desktop, para lo cual debemos ejecutar en la terminal los siguientes comandos:

1. wget -qO - https://mirror.mwt.me/shiftkey-desktop/gpgkey | gpg --dearmor | sudo tee /usr/share/keyrings/mwt-desktop.gpg > /dev/null
2. sudo sh -c 'echo "deb [arch=amd64 signed-by=/usr/share/keyrings/mwt-desktop.gpg] https://mirror.mwt.me/shiftkey-desktop/deb/ any main" > /etc/apt/sources.list.d/mwt-desktop.list'
3. sudo apt update && sudo apt install github-desktop

Y listo tenemos GitHub Desktop instalado en nuestro sistema Linux

Instalación de dependencias

Nuestro programa necesita las siguientes bibliotecas estándar de Linux que generalmente ya vienen instaladas desde el momento de la configuración del sistema operativo, pero debemos asegurarnos que estén por lo tanto debemos ejecutar en la misma terminal los siguientes comandos:

sudo apt-get update

sudo apt-get install build-essential

Compilación del programa

Paso 1: Clonar el repositorio de GitHub

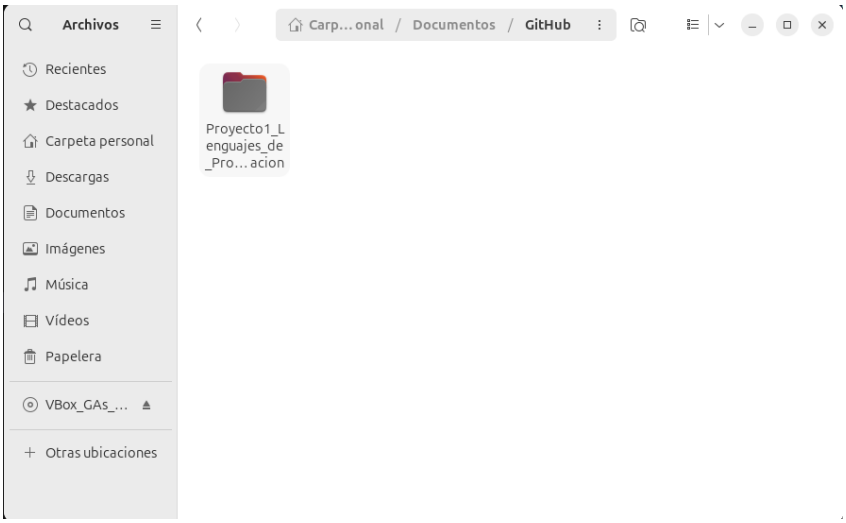
Debemos ir al siguiente link:

https://github.com/Santiago146/Proyecto1_Lenguajes_de_Programacion

y darle a la opción ver en GitHub Desktop, para luego de esto, clonar nuestro proyecto en algún lugar en específico y darle a la opción una vez ya clonado en la ruta seleccionada, presionar en la opción Open in Visual Studio Code

Paso 2: Compilar el proyecto

Una vez se abra el proyecto en el editor de código, debemos abrir una terminal e irnos directos a la ruta en donde gurdamos el proyecto, para ello podemos ir al explorador de archivos, buscar la carpeta llamada “Proyecto1 Lenguajes de Programacion”, y copiar la ruta que sale en la parte de arriba.



Una vez copiada esta ruta debemos irnos de nuevo a la terminal abierta previamente en Visual Studio y escribir cd y copiar la ruta que previamente encontramos.

Una vez nos encontramos en dicha ruta debemos compilar los archivos necesarios para que nuestro programa corra exitosamente, los cuales son, el servidor y el cliente, con los siguientes comandos:

g++ -o server server.cpp -std=c++11

g++ -o cliente cliente.cpp -std=c++11

Y ya podemos comenzar a ejecutar el programa, su ejecución se especifica más a fondo en el siguiente apartado

Manual de Usuario

Para poder usar el programa debemos correr el servidor de primero y luego el cliente, en el cual se realizarán todas las acciones y funcionalidades del programa.

Una vez compilados ambos archivos, debemos abrir una nueva terminal en la ruta de la carpeta que encontramos en paso 2 de la compilación del programa y corremos el servidor con el siguiente comando:

./server

Y ejecutamos el cliente en el cual ya se realizarán las funcionalidades y acciones que el usuario desee, debemos abrir una nueva terminal en la ruta de la carpeta que encontramos en paso 2 de la compilación del programa y ejecutar el cliente con el comando:

./cliente

Una vez hechos estos pasos procedemos a la ejecución del programa:

Antes de poder utilizar la aplicación, es necesario iniciar el servidor.

1. Abra una terminal en el directorio del proyecto
2. Inicie el servidor ejecutando:

   ./server

   Debería ver una salida similar a:

```
kevin@Kevin:~/Documentos/GitHub/Proyecto1_Lenguajes_de_Programacion$ ./server
Memoria compartida inicializada correctamente
Puerto configurado: 8080
Configuración cargada: IP=0.0.0.0 (automático), Puerto=8080
Socket creado exitosamente
Socket enlazado correctamente
Servidor escuchando en el puerto 8080...
```

3. Mantenga esta terminal abierta mientras usa el sistema

Conectarse como Usuario

Una vez que el servidor esté en funcionamiento, puede conectarse como usuario:

1. Abra una nueva terminal en el directorio del proyecto
2. Inicie el cliente ejecutando:

   ./cliente

3. Verá un mensaje de bienvenida:

```
kevin@Kevin:~/Documentos/GitHub/Proyecto1_Lenguajes_de_Programacion$ ./cliente
Configuración cargada: IP=127.0.0.1, Puerto=8080
¡Conectado al servidor 127.0.0.1:8080!
===== BIENVENIDO AL SISTEMA DE MENSAJERÍA =====
USUARIOS CONECTADOS:
- No hay usuarios conectados actualmente.
Ingrese su nombre de usuario:asda
```

4. Ingrese un nombre de usuario (por ejemplo, “Usuario1”) y presione Enter
5. Si el registro es exitoso, verá un mensaje de bienvenida y el menú principal:

```
Bienvenido, asda!


===== MENÚ PRINCIPAL =====
1. Ver lista de usuarios conectados
2. Enviar mensaje
3. Salir
Seleccione una opción:
```

Funcionalidades Principales

Listar Usuarios Conectados

Para ver la lista de usuarios actualmente conectados al sistema:

1. En el menú principal, seleccione la opción 1:

Seleccione una opción: 1

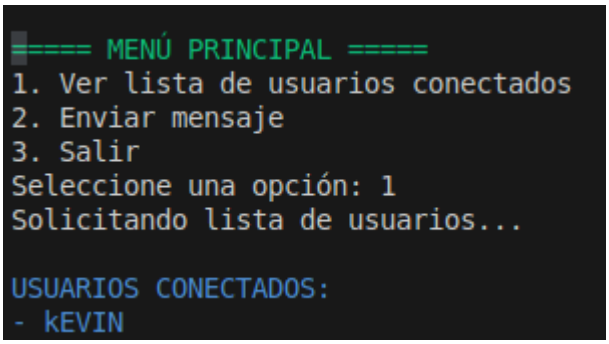
2. Verá un mensaje indicando que se está solicitando la lista y luego aparecerá:

   Solicitando lista de usuarios...

   USUARIOS CONECTADOS:
   - Usuario1

3. Esta lista se actualizará automáticamente cada vez que un usuario se conecte o desconecte

4. Se vería de la siguiente manera:



Enviar Mensajes a Sí Mismo

También puede enviarse mensajes a sí mismo:

1. Seleccione la opción 2 en el menú principal

   Seleccione una opción: 2

2. Ingrese su propio nombre como destinatario:

   Ingrese el nombre del destinatario: Usuario1

3. Escriba su mensaje:

   Ingrese su mensaje: Esto es una prueba

4. Recibirá la confirmación y verá el mensaje en su pantalla:

   Mensaje enviado a Usuario1

   [MENSAJE RECIBIDO] DE:Usuario1 MENSAJE:Esto es una prueba

Se vería de la siguiente manera:

```
Ingrese el nombre del destinatario: kEVIN
Ingrese su mensaje: ASDWE
Mensaje enviado a kEVIN

===== MENÚ PRINCIPAL =====
1. Ver lista de usuarios conectados
2. Enviar mensaje
3. Salir
Seleccione una opción:
[ MENSAJE RECIBIDO ] DE:kEVIN MENSAJE:ASDWE
Mensaje enviado a kEVIN
```

Conectar Múltiples Usuarios

Para simular una conversación entre varios usuarios:

1. Con el servidor en ejecución, abra una nueva terminal en el directorio del proyecto
2. Inicie otra instancia del cliente:

   ./cliente
3. Ingrese un nombre diferente (por ejemplo, “Usuario2”)
4. Ahora puede enviar mensajes entre “Usuario1” y “Usuario2”
5. Cada vez que se conecta un nuevo usuario, todos los usuarios conectados recibirán una actualización de la lista de usuarios
6. Para verificar esto, puede seleccionar la opción 1 en el menú principal de cualquier cliente:

   USUARIOS CONECTADOS:
   - Usuario1
   - Usuario2
7. Puede repetir este proceso para conectar tantos usuarios como desee

Y se vería de la siguiente manera:

```
1
Solicitando lista de usuarios...

USUARIOS CONECTADOS:
- asda
- asdads
```

Cerrar el Sistema

1. Para cerrar un cliente, seleccione la opción 3 en el menú principal:

   Seleccione una opción: 3
   Cerrando aplicación...

2. Cabe recalcar que si se cierra una sola terminal ese usuario se desconecta.

3. Para detener el servidor, presione Ctrl+C en la terminal donde se ejecuta el servidor

4. Todos los usuarios conectados serán desconectados automáticamente cuando el servidor se cierre

Notas adicionales: - Las conexiones se realizan a la dirección IP 127.0.0.1 (localhost) y al puerto 8080 por defecto - La configuración del sistema puede modificarse editando el archivo config.txt - Los mensajes aparecen en color para facilitar la lectura y diferenciar los tipos de notificaciones

Arquitectura Lógica Utilizada

Visión General

El sistema de mensajería implementa una arquitectura cliente-servidor con procesamiento concurrente, utilizando el paradigma imperativo en C++. Está diseñado para permitir la comunicación en tiempo real entre múltiples usuarios a través de una red.

Componentes Principales

La aplicación se divide en dos componentes principales:

5. **Servidor**: Maneja múltiples conexiones de clientes concurrentemente y coordina la comunicación entre ellos.
6. **Cliente**: Proporciona la interfaz de usuario para conectarse al servidor y comunicarse con otros usuarios.

Estructura de Archivos

El sistema está compuesto por los siguientes archivos:

7. server.cpp - Implementación del servidor

8. cliente.cpp - Implementación del cliente
9. configReader.h - Biblioteca para leer configuración
10. config.txt - Archivo de configuración
11. ManualUsuario.md - Guía de usuario
12. README.md - Documentación general

Explicación Detallada por Componente

1. Servidor (server.cpp)

El servidor implementa un modelo de comunicación multi-proceso con memoria compartida para coordinar los datos entre los procesos hijos que atienden a cada cliente.

Características Principales:

- **Manejo Concurrente de Conexiones**: Utiliza la llamada al sistema fork() para crear un proceso hijo por cada cliente conectado.
- **Memoria Compartida**: Implementa un mecanismo de memoria compartida entre procesos usando mmap() y sincronización con mutexes de POSIX.
- **Comunicación Entre Procesos (IPC)**: Utiliza sockets internos para permitir la comunicación entre usuarios atendidos por diferentes procesos.
- **Estructura de Usuarios**: Almacena información de cada usuario conectado incluyendo ID único, IP, socket, puerto y proceso que lo maneja.

Flujo de Ejecución:

13. El servidor inicia y carga la configuración del archivo config.txt
14. Inicializa la memoria compartida y estructuras de datos
15. Crea un socket y lo enlaza al puerto configurado
16. Entra en un bucle infinito donde:
    - Acepta nuevas conexiones de clientes
    - Crea un proceso hijo para manejar cada nueva conexión
    - El proceso hijo ejecuta la función manejarCliente()
17. Dentro de manejarCliente():
    - Verifica si el mensaje es interno (IPC)
    - Registra al usuario en la memoria compartida
    - Entra en un bucle para procesar los mensajes del cliente
    - Maneja diferentes tipos de mensajes (listado de usuarios, mensajes a otros usuarios)

Comunicación Entre Procesos:

Un aspecto clave del diseño es cómo se comunican mensajes entre usuarios manejados por diferentes procesos:

18. Cuando un usuario envía un mensaje a otro que es manejado por otro proceso:
    - Se crea un socket temporal conectado al servidor
    - Se envía un mensaje con formato especial INTERNAL_MSG:destinatario:remitente:contenido
    - El servidor recibe este mensaje y lo redirige al proceso que maneja al destinatario
    - Se confirma la entrega del mensaje

2. Cliente (cliente.cpp)

El cliente implementa una interfaz de usuario basada en consola con capacidad de procesamiento simultáneo de envío y recepción de mensajes.

Características Principales:

- **Procesamiento Concurrente**: Utiliza fork() para crear dos procesos: uno para enviar mensajes y otro para recibirlos.
- **Interfaz de Usuario**: Implementa un menú interactivo para las funciones principales.
- **Comunicación con el Servidor**: Mantiene una conexión TCP con el servidor para enviar/recibir mensajes.
- **Formato de Colores**: Utiliza códigos ANSI para mejorar la visualización de mensajes en la consola.

Flujo de Ejecución:

19. El cliente inicia y carga la configuración
20. Crea un socket y se conecta al servidor
21. Recibe la lista inicial de usuarios conectados
22. Solicita y envía el nombre de usuario
23. Se bifurca en dos procesos:
    - Proceso hijo: ejecuta recibirMensajes() para mostrar mensajes entrantes
    - Proceso padre: ejecuta enviarMensajes() para manejar la interfaz de usuario
24. El proceso de envío muestra el menú principal con opciones para:
    - Ver la lista de usuarios conectados
    - Enviar un mensaje
    - Salir del sistema

3. Biblioteca de Configuración (configReader.h)

Esta biblioteca proporciona funciones para leer la configuración del sistema desde un archivo de texto.

Funciones:

- extractPortConfig(): Lee el puerto para el servidor
- extractClientConfig(): Lee la IP y puerto para el cliente

Propósito:

Permite configurar el sistema sin necesidad de recompilar el código, facilitando el despliegue en diferentes entornos.

4. Archivo de Configuración (config.txt)

Archivo de texto plano que contiene la configuración básica del sistema:

PORT=8080
IP=127.0.0.1

Flujo de Datos

Registro de Usuario:

25. Cliente se conecta al servidor
26. Servidor crea un proceso hijo para manejar al cliente
27. Cliente envía nombre de usuario
28. Servidor almacena información del usuario en memoria compartida
29. Servidor envía lista actualizada de usuarios a todos los clientes

Envío de Mensajes:

30. Cliente A selecciona enviar mensaje a Usuario B
31. Cliente A envía mensaje al servidor en formato DESTINATARIO:mensaje
32. Servidor verifica si el destinatario existe
33. Si el destinatario es manejado por el mismo proceso:
    - Envía el mensaje directamente al socket del destinatario
34. Si el destinatario es manejado por otro proceso:
    - Crea un socket temporal
    - Envía mensaje interno al servidor con formato INTERNAL_MSG:destinatario:remitente:contenido
    - El servidor redirige el mensaje al proceso que maneja al Usuario B
    - El mensaje se entrega al Usuario B
35. Cliente B recibe y muestra el mensaje

Mecanismos de Sincronización

El sistema utiliza varios mecanismos para garantizar la correcta sincronización y comunicación:

1. **Mutex POSIX**: Para sincronizar el acceso a la memoria compartida entre procesos
2. **Serialización/Deserialización**: Para mantener la consistencia de datos entre procesos
3. **Sockets Temporales**: Para la comunicación entre procesos de servidor
4. **Identificadores Únicos**: Para distinguir unívocamente a cada usuario

Manejo de Excepciones y Errores

El sistema implementa varios mecanismos para manejar errores:

1. **Validación de Entrada**: Verifica que los nombres de usuario no estén vacíos
2. **Detección de Desconexiones**: Detecta cuando un cliente se desconecta y libera recursos
3. **Manejadores de Señales**: Captura señales del sistema para limpieza de recursos
4. **Procesos Zombies**: Implementa un manejador de SIGCHLD para evitar procesos zombies

Seguridad

El sistema implementa medidas básicas de seguridad:

1. **Validación de Usuarios**: Verifica que un nombre de usuario no puede ser usado por diferentes IPs
2. **Sanitización de Entrada**: Elimina caracteres no deseados de los mensajes
3. **Mensajes de Confirmación**: Confirma la entrega de mensajes

Escalabilidad

La arquitectura del sistema permite una escalabilidad moderada:

1. El uso de procesos separados permite manejar múltiples clientes simultáneamente
2. La memoria compartida proporciona un mecanismo eficiente para compartir datos entre procesos
3. El sistema de comunicación interna permite la comunicación entre cualquier par de usuarios

Limitaciones

El diseño actual tiene algunas limitaciones:

1. El servidor maneja todo en memoria, sin persistencia de datos
2. No hay encriptación en la comunicación

3. La comunicación está limitada a una red local o a un servidor con IP conocida
4. No hay autenticación robusta de usuarios