

**UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN**  
**FACULTAD DE INGENIERÍA**  
**ESCUELA PROFESIONAL DE INGENIERÍA EN INFORMÁTICA Y**  
**SISTEMAS**



**PROYECTO DE UNIDAD I**  
**“Minishell en c++ para Linux”**

**ASIGNATURA:**

Ingeniería de software I

**DOCENTE:**

MSc. Ing. Hugo Manuel Barraza Vizcarra

**INTEGRANTES:**

- Lizeth Hanco Calizaya      2023-119022
- Néstor André Alarcón Luque      2023-119002

**SECCIÓN: B**

**FECHA:** 15 de Octubre del 2025

**TACNA - PERÚ**

**2025**

## **PROYECTO DE UNIDAD I: MiniShell en C++ para sistemas Linux**

### **1. Objetivos y alcance**

El objetivo principal de este proyecto es implementar un intérprete de comandos tipo mini-shell en C++ sobre sistemas Linux, capaz de ejecutar comandos del sistema, manejar procesos y realizar redireccionamiento salida y manejo de errores y rutas de comandos.

#### **Objetivos específicos:**

- Implementar procesos hijo y controladores desde el proceso padre.
- Diseñar un programa modular utilizando cabeceras separadas.
- Validar el correcto funcionamiento de los comandos del sistema en un minishell.

El alcance del proyecto comprende de la implementación de las características base de una mini-shell: un prompt personalizado, la resolución de rutas tanto absolutas como relativas, la ejecución de comandos mediante procesos utilizando `fork()` y `exec()`, el manejo de errores utilizando `perror()`, la redirección de la salida estándar (`>`) y el comando interno 'salir' para terminar la ejecución de la mini-shell.

De esta manera, el proyecto busca demostrar el dominio de los conceptos fundamentales de la unidad I del curso de Sistemas Operativos, que se relaciona con la gestión de procesos, la concurrencia y la manipulación de entradas/salidas en entornos POSIX.

### **2. Arquitectura y diseño**

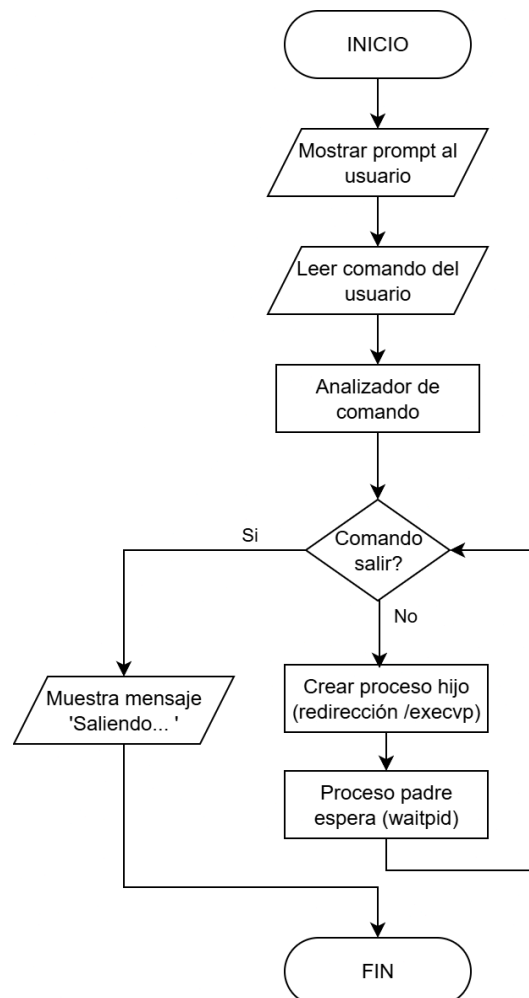
La arquitectura y el diseño del proyecto MiniShell se basa en una estructura modular escrita en C++, la cuál se dividirá en la siguientes carpetas:

- src: Donde se incluirá los archivos de código fuente como el `main.cpp`, `analizador.cpp` y `ejecutor.cpp`
- include: Donde se incluirá los archivos como el `analizador.hpp` y `ejecutor.hpp`
- docs: Donde se incluirá el informe técnico en pdf, diapositivas en pdf y una

carpeta de imágenes donde estarán las capturas de las pruebas realizadas del minishell.

El diagrama simple de procesos sería como se muestra a continuación:

Figura 1: Diagrama de procesos



### 3. Detalles de implementación

Las APIS POSIX utilizadas fueron las siguientes:

- `fork()`, para crear un proceso hijo.
- `execvp`, para ejecutar el comando solicitado reemplazando el proceso hijo.
- `waitpid`, para que el proceso padre espera la finalización del hijo.
- `access()`, para la verificación de existencia del comando y sus permisos.
- `open()`, para crear y abrir el archivo para la redirección.
- `dup2()`, para redirigir la salida estándar al archivo.

- `close()`, para cerrar el descriptor de archivo.
- `perror()`, para imprimir la descripción del error según `errno`.
- `pipe()`, conectar la salida de un proceso con la entrada de otro.

Las decisiones que se consideraron clave son:

- Todos los comandos que no tiene ruta absoluta, se asumirán que estarán en la carpeta `/bin`.
- Los tokens se separarán por espacios
- Se utilizará la función `execvp()` para reemplazar el proceso hijo por otro del sistema, lo cual es utilizado para sistemas operativos Unix/Linux.
- El comando salir será para terminar la ejecución del minishell por medio de `exit(0)`.

#### **4. Concurrencia y sincronización: qué se paraleliza y cómo se evita la condición de carrera/interbloqueo.**

Este proyecto, hasta el cumplimiento de las características básicas es de ejecución secuencial, ya que el proceso padre esperará a que el proceso hijo termine utilizando `waitpid()`, antes de aceptar otro comando. Desde la implementación de las extensiones se empezó a usar procesos en segundo plano, pero no una paralelización que ejecute funciones como `pthread_create()`

#### **5. Gestión de memoria: estrategia y evidencias.**

Para la memoria se utilizó estructuras estándar de C++ como los vectores, ya que no se sabe el tamaño para almacenar, pues el usuario es quien digita el comando

#### **6. Pruebas y resultados**

Para validar que el programa cumple con los requisitos funcionales, se diseñaron y ejecutaron los siguientes casos de prueba, cada uno alineado con un requisito funcional.

##### **Caso 1: ls**

El requisito validado es la ejecución de comandos básicos del sistema con ruta relativa. En la validación, se ejecutó el comando `ls` desde la minishell desarrollada y

se observó que muestra correctamente los archivos del directorio actual, igual que en la terminal estándar. Por lo que se confirma que `execvp` funciona para comandos simples.

#### **Caso 2: `ls > salida.txt`**

El requisito validado es la redirección de salida. En la validación, se ejecutó el comando y redirigió correctamente la salida estándar al archivo `salida.txt`. Se comprobó que en la pantalla no mostró salida y que el archivo `salida.txt` fue creado con el contenido esperado.

#### **Caso 3: `/bin/ls`**

El requisito validado es comandos con rutas absolutas. En la validación, mostró el listado de archivos correctamente.

#### **Caso 4: Comando inexistente**

El requisito validado es el manejo de errores ante comandos inválidos. En la validación, se ingresó el comando y el programa respondió: "Error al ejecutar el comando: No such file or directory". Confirmando que maneja correctamente el error.

#### **Caso 5: salir**

El requisito validado es la implementación de comando interno para finalizar la shell. En la validación, al escribir `salir`, el programa muestra un mensaje "Saliendo..." y termina la ejecución correctamente.

#### **Caso 6: `ls >`**

El requisito validado es el manejo de errores de sintaxis. En la validación, se ingresó el comando con redirección de mal forma y el programa mostró el mensaje de error indicando el error que falta el nombre del archivo.

La validación por entorno, las pruebas se realizaron en Ubuntu 22.04 y Debian 13. En ambos entornos, los resultados del programa fueron consistentes y correctos, por lo que se confirmó que el programa es funcional y portátil entre distribuciones comunes de Linux. Donde las pruebas de su validación se mostrarán en los anexos 1 y 2.

## 7. Conclusiones

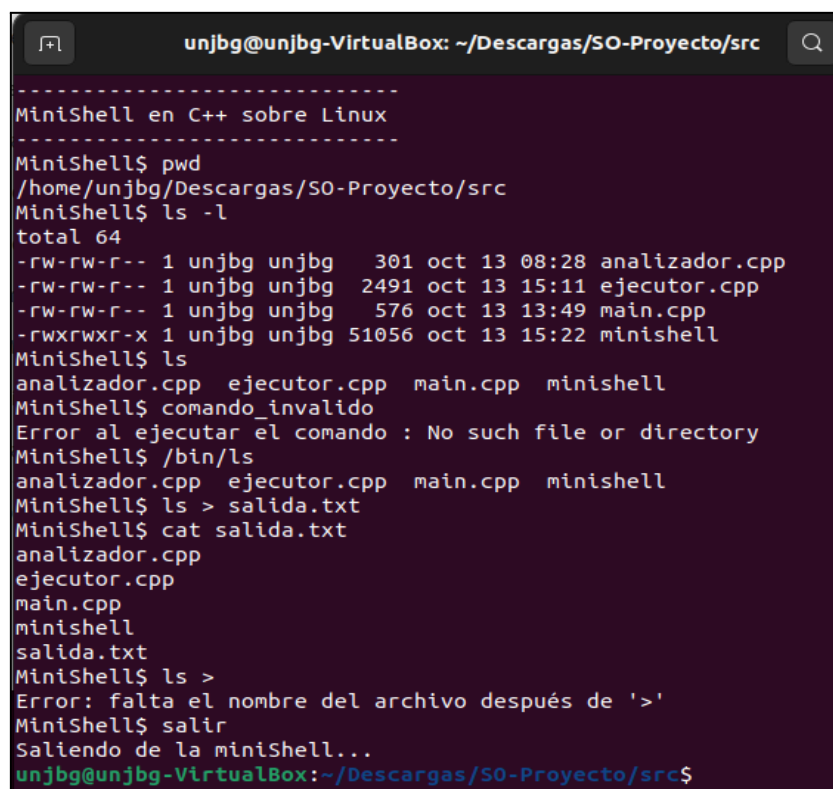
- La minishell se logró implementar correctamente con las funciones base de prompt, ejecución de comandos, la redirección de salida y el manejo de errores.
- Se utilizó APIs POSIX para demostrar la comprensión de la creación y control de procesos

## 8. Trabajos futuros

Para los trabajos futuros se consideró los comandos internos (built-ins), donde se incluirá el cd, pwd, help, history, alias

## 9. Anexos

### Anexo 1: Comando probados en Ubuntu 22.04



```
unjbg@unjbg-VirtualBox: ~/Descargas/SO-Proyecto/src
-----
MiniShell en C++ sobre Linux
-----
MiniShell$ pwd
/home/unjbg/Descargas/SO-Proyecto/src
MiniShell$ ls -l
total 64
-rw-rw-r-- 1 unjbg unjbg  301 oct 13 08:28 analizador.cpp
-rw-rw-r-- 1 unjbg unjbg 2491 oct 13 15:11 ejecutor.cpp
-rw-rw-r-- 1 unjbg unjbg  576 oct 13 13:49 main.cpp
-rwxrwxr-x 1 unjbg unjbg 51056 oct 13 15:22 minishell
MiniShell$ ls
analizador.cpp ejecutor.cpp main.cpp minishell
MiniShell$ comando_invalido
Error al ejecutar el comando : No such file or directory
MiniShell$ /bin/ls
analizador.cpp ejecutor.cpp main.cpp minishell
MiniShell$ ls > salida.txt
MiniShell$ cat salida.txt
analizador.cpp
ejecutor.cpp
main.cpp
minishell
salida.txt
MiniShell$ ls >
Error: falta el nombre del archivo después de '>'
MiniShell$ salir
Saliendo de la miniShell...
unjbg@unjbg-VirtualBox:~/Descargas/SO-Proyecto/src$
```

### Anexo 1: Comando probados en Debian 13

```
pc2@pc2: ~/Descargas/SO-Proyecto/src
-----
MiniShell en C++ sobre Linux
-----
MiniShell$ pwd
/home/pc2/Descargas/SO-Proyecto/src
MiniShell$ ls -l
total 64
-rw-rw-r-- 1 pc2 pc2 301 oct 14 15:17 analizador.cpp
-rw-rw-r-- 1 pc2 pc2 2491 oct 14 15:17 ejecutor.cpp
-rw-rw-r-- 1 pc2 pc2 576 oct 14 15:17 main.cpp
-rwxrwxr-x 1 pc2 pc2 49024 oct 14 15:28 minishell
-rw-rw-r-- 1 pc2 pc2 58 oct 14 15:33 salida.txt
MiniShell$ ls
analizador.cpp ejecutor.cpp main.cpp minishell salida.txt
MiniShell$ /bin/ls
analizador.cpp ejecutor.cpp main.cpp minishell salida.txt
MiniShell$ comando_invalido
Error al ejecutar el comando : No such file or directory
MiniShell$ ls > salida.txt
MiniShell$ cat salida.txt
analizador.cpp
ejecutor.cpp
main.cpp
minishell
salida.txt
MiniShell$ ls >
Error: falta el nombre del archivo después de '>'
MiniShell$ salir
Saliendo de la miniShell...
pc2@pc2:~/Descargas/SO-Proyecto/src$
```