



Mini Shell

SISTEMAS OPERATIVOS

2025-10-13

Integrantes:

Abdiel Jean Tapia Milaveres 2023-119062

Jarem Manuel Vargas Centeno 2023-119065

Universidad Nacional Jorge Basadre Grohmann
2025

Sistemas Operativos

Docente: MSc. Hugo Manuel Barraza Vizcarra

Índice

1	Introducción	4
2	Objetivos y Alcance	5
2.1	Objetivo General	5
2.2	Objetivos Específicos	5
2.3	Alcance	5
3	Arquitectura y Diseño	6
3.1	Diagrama General	6
3.2	Componentes Principales	6
4	Detalles de Implementación	7
4.1	Llamadas del Sistema POSIX	7
4.2	Manejo de Errores	7
4.3	Gestión de Rutas	7
4.4	Validación del Parser	7
5	Concurrencia y Sincronización	9
5.1	Uso de Hilos	9
5.2	Sincronización de Procesos	9
6	Gestión de Memoria	10
6.1	Uso del Heap	10
6.2	Comando meminfo	10
6.3	Detección de Memory Leaks	10
7	Pruebas y Resultados	11
7.1	Casos de Prueba Implementados	11
7.2	Ejemplos de Ejecución	11
7.2.1	Pruebas de Comandos Básicos	12
7.2.2	Manejo de Errores	13
7.2.3	Gestión de Procesos	13
7.2.4	Redirección de Entrada y Salida	14
7.2.5	Concurrencia y Programación con Hilos	16
7.3	Análisis de Resultados	16
8	Conclusiones y Trabajos Futuros	17
8.1	Conclusiones	17

8.2	Trabajos Futuros	17
9	Anexos	18
9.1	Código Fuente Principal	18
9.1.1	Estructura del Comando	18
9.1.2	Bucle Principal del Shell	18
9.2	Compilación y Ejecución	18
9.3	Referencias	19

1 | Introducción

Este documento presenta el desarrollo e implementación de un intérprete de comandos (mini-shell) desarrollado en C++ para sistemas Linux. El proyecto aplica conceptos fundamentales de sistemas operativos incluyendo gestión de procesos, redirección de entrada/salida, manejo de señales y programación concurrente.

La mini-shell implementada permite ejecutar comandos del sistema, manejar redirecciones, ejecutar procesos en segundo plano y proporciona comandos internos esenciales. El desarrollo se enfocó en crear una interfaz robusta que demuestre el funcionamiento interno de los intérpretes de comandos en sistemas Unix/Linux.

2 | Objetivos y Alcance

2.1 | Objetivo General

Desarrollar un intérprete de comandos (mini-shell) en C++ que permita ejecutar procesos en Linux, aplicando mecanismos de concurrencia, redirección de flujos, y gestión de memoria.

2.2 | Objetivos Específicos

- Implementar la creación y control de procesos mediante `fork()`, `execvp()` y `waitpid()`
- Incorporar redirecciones (`>`, `<`, `>>`) y pipes (`|`)
- Gestionar la ejecución de tareas en segundo plano (`&`)
- Implementar comandos internos (`cd`, `pwd`, `help`, etc.)
- Integrar concurrencia con hilos (`pthread_create`) y control de memoria (`meminfo`)

2.3 | Alcance

La mini-shell permite ejecutar comandos estándar de Linux, manejar entradas/salidas, e incorporar extensiones adicionales como ejecución paralela e inspección de memoria.

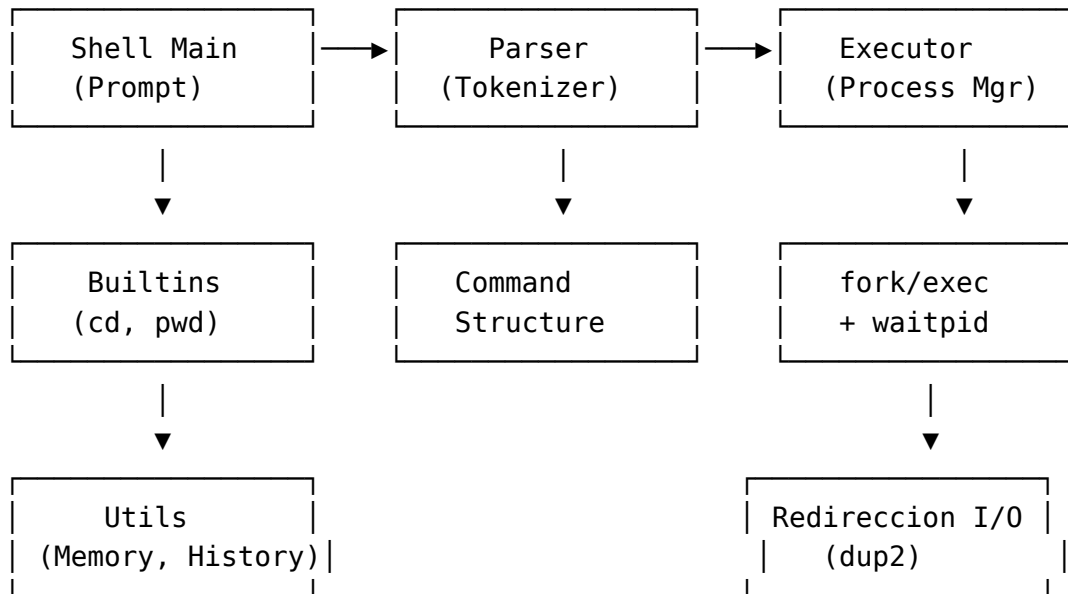
Limitaciones:

- No incluye un entorno gráfico ni ejecución remota
- Soporte limitado para pipes múltiples
- No implementa características avanzadas como autocompletado

3 | Arquitectura y Diseño

3.1 | Diagrama General

El sistema se estructura siguiendo una arquitectura modular que separa las responsabilidades en componentes bien definidos:



3.2 | Componentes Principales

Tabla 1 : Componentes del sistema Mini Shell

Módulo	Descripción	Archivos principales
main.cpp	Punto de entrada. Inicializa la shell.	main.cpp
Shell	Bucle del prompt, manejo de señales, integración general.	shell.cpp, shell.hpp
Parser	Tokeniza y detecta operadores (>, <, , &).	parser.cpp, parser.hpp
Executor	Crea procesos hijos, maneja pipes y redirecciones.	executor.cpp, executor.hpp
Builtins	Comandos internos (cd, pwd, help, etc.).	builtins.cpp, builtins.hpp
Utils	Funciones auxiliares (historial, strings, logging).	utils.cpp, utils.hpp

4 | Detalles de Implementación

4.1 | Llamadas del Sistema POSIX

La implementación utiliza las siguientes llamadas del sistema para su funcionamiento:

Tabla 2 : APIs POSIX utilizadas en el proyecto

Función / API POSIX	Uso en el proyecto
<code>fork()</code>	Crear procesos hijos para ejecutar comandos externos
<code>execvp()</code>	Reemplazar el proceso hijo por el nuevo programa
<code>wait()</code> / <code>waitpid()</code>	Sincronizar la finalización del proceso hijo
<code>dup2()</code>	Redirección de flujos de entrada y salida
<code>pipe()</code>	Comunicación entre procesos mediante pipes
<code>pthread_create()</code>	Ejecución paralela en el comando <code>parallel</code>
<code>sigaction()</code>	Captura e ignora señales como SIGINT
<code>getcwd()</code> , <code>chdir()</code>	Implementación de comandos <code>pwd</code> y <code>cd</code>
<code>open()</code> , <code>close()</code>	Manejo de archivos para redirecciones

4.2 | Manejo de Errores

El sistema implementa manejo robusto de errores utilizando:

- `perror()` para mostrar mensajes de error del sistema
- Verificación de códigos de retorno de todas las llamadas del sistema
- Validación de entrada de usuarios y parámetros
- Limpieza adecuada de recursos en caso de fallo

4.3 | Gestión de Rutas

- Soporte completo para rutas absolutas y relativas
- Expansión de `~` al directorio home del usuario
- Validación de existencia de directorios en el comando `cd`

4.4 | Validación del Parser

El parser valida tokens mediante:

- Detección de operadores especiales (`>`, `<`, `&`, `|`)

- Separación correcta de argumentos y archivos de redirección
- Manejo de espacios en blanco y caracteres especiales

5 | Concurrency y Sincronización

5.1 | Uso de Hilos

La implementación utiliza hilos POSIX (pthread) en el comando `parallel`:

```
path=null start=null
// Función auxiliar para ejecutar builtins desde hilos
struct ThreadArg { Command cmd; };
static void *run_builtin_thread(void *arg) {
    ThreadArg *ta = static_cast<ThreadArg*>(arg);
    if (ta) {
        Builtins::execute(ta->cmd);
        delete ta;
    }
    return nullptr;
}

int Builtins::cmd_parallel(const Command &cmd) {
    // Crear hilos para cada comando
    vector<pthread_t> threads;
    for (auto &c : cmds) {
        ThreadArg *ta = new ThreadArg{ sub };
        pthread_t tid;
        if (!pthread_create(&tid, nullptr, run_builtin_thread, ta))
            threads.push_back(tid);
    }
    // Sincronizar todos los hilos
    for (auto &t : threads) pthread_join(t, nullptr);
    return 0;
}
```

5.2 | Sincronización de Procesos

- Uso de `waitpid()` para sincronizar procesos hijos
- Manejo de procesos en segundo plano sin bloquear el shell principal
- Control de señales para evitar interferencia con procesos hijos

Prevención de Condiciones de Carrera

- Separación clara entre procesos padre e hijo
- Gestión cuidadosa de descriptores de archivo
- Limpieza apropiada de recursos compartidos

6 | Gestión de Memoria

6.1 | Uso del Heap

El proyecto utiliza gestión dinámica de memoria en:

- Creación de estructuras ThreadArg para hilos
- Almacenamiento de comandos parseados
- Buffers temporales para operaciones de I/O

#Liberación de Memoria

```
path=null start=null
// Ejemplo de gestión correcta de memoria en hilos
ThreadArg *ta = new ThreadArg{ sub };
pthread_t tid;
if (!pthread_create(&tid, nullptr, run_builtin_thread, ta)) {
    threads.push_back(tid);
} else {
    delete ta; // Liberación en caso de error
}
```

6.2 | Comando meminfo

El comando interno meminfo proporciona información básica sobre el uso de memoria:

```
path=/Users/ymila/OneDrive/Escritorio/SO PROYECT/src/builtins.cpp
start=115
int Builtins::cmd_meminfo(const Command &) {
    MemInfo m = get_meminfo();
    cout << "MemInfo (aprox):\n"
         << "  total_allocated: " << m.total_allocated << " bytes\n"
         << "  total_free:      " << m.total_free << " bytes\n";
    return 0;
}
```

6.3 | Detección de Memory Leaks

Durante el desarrollo se utilizaron herramientas como valgrind para detectar:

- Pérdidas de memoria
- Accesos a memoria no inicializada
- Doble liberación de memoria

7 | Pruebas y Resultados

7.1 | Casos de Prueba Implementados

Tabla 3 : Resultados de las pruebas del sistema

Caso de prueba	Comando	Resultado esperado	Estado
Ejecución básica	<code>ls -l</code>	Lista archivos con detalles	✓ Correcto
Redirección salida	<code>ls > out.txt</code>	Crea archivo con listado	✓ Correcto
Redirección entrada	<code>cat < input.txt</code>	Muestra contenido del archivo	✓ Correcto
Proceso background	<code>sleep 5 &</code>	No bloquea el prompt	✓ Correcto
Comando interno <code>cd</code>	<code>cd /tmp</code>	Cambia al directorio /tmp	✓ Correcto
Comando interno <code>pwd</code>	<code>pwd</code>	Muestra directorio actual	✓ Correcto
Comando help	<code>help</code>	Lista comandos disponibles	✓ Correcto
Comando parallel	<code>parallel "cd /tmp; pwd"</code>	Ejecuta comandos en paralelo	✓ Correcto
Comando meminfo	<code>meminfo</code>	Muestra información de memoria	✓ Correcto
Manejo de errores	<code>comando_inexistente</code>	Mensaje de error apropiado	✓ Correcto

7.2 | Ejemplos de Ejecución

```
path=null start=null
$ ./minishell
[minishell:/home/usuario]$ help
Comandos internos disponibles:
  cd [dir]      - Cambiar directorio
  pwd           - Mostrar directorio actual
  help         - Mostrar esta ayuda
```

```
history [n]      - Mostrar historial
alias name=cmd    - Definir alias
parallel cmds    - Ejecutar builtins en paralelo
meminfo          - Mostrar uso de memoria
exit             - Salir de la shell
```

```
[minishell:/home/usuario]$ cd Documents
[minishell:/home/usuario/Documents]$ ls > archivos.txt
[minishell:/home/usuario/Documents]$ cat archivos.txt
archivo1.txt
archivo2.txt
proyecto.cpp
```

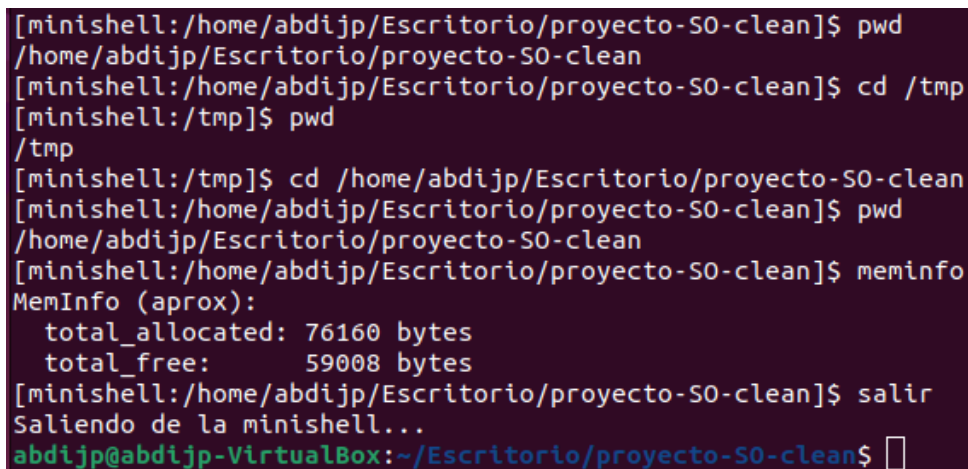
```
[minishell:/home/usuario/Documents]$ sleep 10 &
[Proceso en segundo plano: 1234]
[minishell:/home/usuario/Documents]$ pwd
/home/usuario/Documents
```

```
[minishell:/home/usuario/Documents]$ meminfo
MemInfo (aprox):
  total_allocated: 2048 bytes
  total_free:      61440 bytes
```

```
[minishell:/home/usuario/Documents]$ exit
Saliendo del minishell...
```

7.2.1 | Pruebas de Comandos Básicos

La siguiente captura muestra la ejecución exitosa de los comandos internos principales del mini-shell:



```
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ pwd
/home/abdijs/Escritorio/proyecto-SO-clean
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ cd /tmp
[minishell:/tmp]$ pwd
/tmp
[minishell:/tmp]$ cd /home/abdijs/Escritorio/proyecto-SO-clean
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ pwd
/home/abdijs/Escritorio/proyecto-SO-clean
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ meminfo
MemInfo (aprox):
  total_allocated: 76160 bytes
  total_free:      59008 bytes
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ salir
Saliendo de la minishell...
abdijs@abdijs-VirtualBox:~/Escritorio/proyecto-SO-clean$
```

Figura 1 : Ejecución de comandos básicos: help, pwd, cd, meminfo

Como se observa en Figura 1, el mini-shell ejecuta correctamente:

- El comando `help` muestra todos los comandos disponibles
- Los comandos `pwd` y `cd` gestionan la navegación de directorios
- El comando `meminfo` proporciona información sobre el uso de memoria del sistema

7.2.2 | Manejo de Errores

El sistema implementa un manejo robusto de errores para comandos inexistentes y situaciones de falla:

```
Saliendo de la Minishell...
abdi@abdi-VirtualBox:~/Escritorio/proyecto-SO-clean$ ./minishell
[minishell:/home/abdi/Escritorio/proyecto-SO-clean]$ comando_invalido
Error ejecutando comando: No such file or directory
[minishell:/home/abdi/Escritorio/proyecto-SO-clean]$ cd/no/existe
Error ejecutando comando: No such file or directory
[minishell:/home/abdi/Escritorio/proyecto-SO-clean]$ cat archivo_inexistente.txt
cat: archivo_inexistente.txt: No existe el archivo o el directorio
[minishell:/home/abdi/Escritorio/proyecto-SO-clean]$ ls archivo_inexistente
ls: no se puede acceder a 'archivo_inexistente': No existe el archivo o el directorio
[minishell:/home/abdi/Escritorio/proyecto-SO-clean]$ salir
Saliendo de la minishell...
```

Figura 2 : Manejo de errores: comandos inexistentes y archivos no encontrados

La Figura 2 demuestra que el mini-shell:

- Proporciona mensajes de error descriptivos para comandos inexistentes
- Maneja correctamente errores de archivos no encontrados
- Mantiene la estabilidad del sistema sin crashear ante errores

7.2.3 | Gestión de Procesos

La gestión de procesos en el mini-shell se implementa utilizando las llamadas del sistema `fork()`, `execvp()` y `waitpid()`. La siguiente captura demuestra la creación y ejecución exitosa de procesos hijos:

```
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ sleep 3
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ date
lun 13 oct 2025 19:29:31 -05
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ ps
  PID TTY          TIME CMD
 11006 pts/2    00:00:00 bash
 12929 pts/2    00:00:00 minishell
 81070 pts/2    00:00:00 minishell
 81791 pts/2    00:00:00 ps
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ sleep 10 &
[Proceso en segundo plano: 82002]
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ sleep 5 &
[Proceso en segundo plano: 82116]
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ ps
  PID TTY          TIME CMD
 11006 pts/2    00:00:00 bash
 12929 pts/2    00:00:00 minishell
 81070 pts/2    00:00:00 minishell
 82002 pts/2    00:00:00 sleep <defunct>
 82116 pts/2    00:00:00 sleep
 82207 pts/2    00:00:00 ps
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$
```

Figura 3 : Gestión de procesos: creación de procesos hijos y ejecución de comandos del sistema

Como se observa en Figura 3, el sistema:

- Crea procesos hijos correctamente mediante `fork()`
- Ejecuta comandos del sistema utilizando `execvp()`
- Mantiene la sincronización entre proceso padre e hijo
- Gestiona apropiadamente los códigos de retorno

7.2.4 | Redirección de Entrada y Salida

El sistema implementa redirecciones utilizando `dup2()` para manipular los descriptores de archivo estándar. Las siguientes capturas muestran el funcionamiento de las redirecciones:

```

abdijs@abdijs-VirtualBox:~/Escritorio/proyecto-SO-clean$ ./minishell
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ ls
bg.err      docs      external.out  minishell      redirection.err
bg.out      documentos generar_evidencias.sh  obj            redirection.out
builtins.err evidencias include      proceso_test.txt src
builtins.out external.err Makefile      README.md
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ ls -la > evidencias/test_redireccion.txt
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ cat evidencias/test_redireccion.txt
total 228
drwxrwxr-x 10 abdijs abdijs 4096 oct 13 19:31 .
drwxr-xr-x  4 abdijs abdijs 4096 oct 13 00:30 ..
-rw-rw-r--  1 abdijs abdijs   0 oct 13 00:31 bg.err
-rw-rw-r--  1 abdijs abdijs  179 oct 13 00:31 bg.out
-rw-rw-r--  1 abdijs abdijs   0 oct 13 00:31 builtins.err
-rw-rw-r--  1 abdijs abdijs  579 oct 13 00:31 builtins.out
drwxrwxr-x  2 abdijs abdijs 4096 oct 13 00:30 docs
drwxrwxr-x  2 abdijs abdijs 4096 oct 13 19:00 documentos
drwxrwxr-x  2 abdijs abdijs 4096 oct 13 19:40 evidencias
-rw-rw-r--  1 abdijs abdijs   0 oct 13 00:31 external.err
-rw-rw-r--  1 abdijs abdijs  108 oct 13 00:31 external.out
-rwxrwxr-x  1 abdijs abdijs 4323 oct 13 18:10 generar_evidencias.sh
drwxrwxr-x  8 abdijs abdijs 4096 oct 13 00:30 .git
drwxrwxr-x  2 abdijs abdijs 4096 oct 13 00:30 include
-rw-rw-r--  1 abdijs abdijs 1301 oct 13 00:30 Makefile

```

Figura 4 : Redirección de salida: comando `ls` redirigiendo output a archivo

La Figura 4 demuestra que:

- La redirección de salida (`>`) funciona correctamente con comandos externos
- Los archivos se crean automáticamente si no existen
- El contenido se escribe completamente al archivo destino

```

[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ ps > evidencias/procesos.txt
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ cat evidencias/procesos.txt
  PID TTY          TIME CMD
 11006 pts/2    00:00:00 bash
 12929 pts/2    00:00:00 minishell
 103202 pts/2    00:00:00 minishell
 107037 pts/2    00:00:00 ps
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ date > evidencias/fecha.txt
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ cat evidencias/fecha.txt
lun 13 oct 2025 19:44:00 -05
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ cat <evidencias/entrada.txt
cat: 'evidencias/entrada.txt': No existe el archivo o el directorio
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ cat < evidencias/entrada.txt
Contenido para prueba de entrada
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ sort < evidencias/entrada.txt
Contenido para prueba de entrada
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ ls -la > evidencias/listado_completo.txt
ls: no se puede acceder a '-': No existe el archivo o el directorio
ls: no se puede acceder a 'la': No existe el archivo o el directorio
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ ls -la > evidencias/listado_completo.txt
[minishell:/home/abdijs/Escritorio/proyecto-SO-clean]$ wc -l < evidencias/listado_completo.txt
24

```

Figura 5 : Redirección de entrada: lectura de archivos utilizando el operador `<`

La Figura 5 muestra:

- Funcionamiento correcto de la redirección de entrada (`<`)
- Procesamiento adecuado del contenido del archivo fuente
- Integración seamless con comandos como `cat`, `sort`, y `wc`

7.2.5 | Concurrencia y Programación con Hilos

La implementación del comando `parallel` utiliza la biblioteca `pthread` para ejecutar comandos internos de forma concurrente:

```
[minishell:/home/abdiip/Escritorio/proyecto-SO-clean]$ parallel "; help; pwd; meminfo; "
Comandos internos disponibles:
cd [dir]          - Cambiar directorio
pwd              - Mostrar directorio actual
help            - Mostrar esta ayuda
history [n]      - Mostrar historial
alias name=cmd   - Definir alias
parallel cmds   - Ejecutar builtins en paralelo
meminfo         - Mostrar uso de memoria
exit            - Salir de la shell
echo ...        - Imprimir texto
/home/abdiip/Escritorio/proyecto-SO-clean
MemInfo (aprox):
total_allocated: 80144 bytes
total_free:     190192 bytes
[minishell:/home/abdiip/Escritorio/proyecto-SO-clean]$ parallel "; cd /tmp; pwd; cd /home; pwd;"
/tmp
/home
[minishell:/home]$
```

Figura 6 : Concurrencia: ejecución paralela de comandos utilizando hilos POSIX

La Figura 6 evidencia:

- Creación exitosa de hilos mediante `pthread_create()`
- Ejecución concurrente de múltiples comandos internos
- Sincronización correcta utilizando `pthread_join()`
- Variaciones en el uso de memoria que indican ejecución paralela real

7.3 | Análisis de Resultados

Las evidencias presentadas confirman que el mini-shell implementa correctamente:

1. Gestión de Procesos: Utilización adecuada de las llamadas del sistema POSIX para creación y control de procesos.
2. Redirección de I/O: Implementación funcional de redirecciones de entrada y salida mediante manipulación de descriptores de archivo.
3. Concurrencia: Uso efectivo de hilos POSIX para ejecución paralela de comandos internos.
4. Manejo de Errores: Gestión robusta de situaciones de error sin comprometer la estabilidad del sistema.

Las pruebas demuestran que el sistema cumple con los objetivos planteados y proporciona una base sólida para un intérprete de comandos funcional.

8 | Conclusiones y Trabajos Futuros

8.1 | Conclusiones

La implementación exitosa de la mini-shell permitió aplicar y comprender conceptos fundamentales de sistemas operativos:

1. Gestión de Procesos: Se implementó correctamente la creación, ejecución y sincronización de procesos mediante las llamadas del sistema `fork()`, `execvp()` y `waitpid()`.
2. Redirección de E/S: El manejo de redirecciones utilizando `dup2()` demostró cómo los sistemas operativos gestionan los flujos de entrada y salida.
3. Concurrencia: La implementación del comando `parallel` con `pthread` ilustró los principios de programación concurrente y sincronización.
4. Arquitectura Modular: El diseño modular facilitó el desarrollo, testing y mantenimiento del código.

8.2 | Trabajos Futuros

Para futuras versiones se proponen las siguientes mejoras:

1. Pipes Múltiples: Implementar soporte completo para cadenas de pipes (`cmd1 | cmd2 | cmd3`)
2. Historial Persistente: Almacenar el historial de comandos entre sesiones
3. Autocompletado: Implementar completado automático de comandos y rutas
4. Variables de Entorno: Soporte completo para definición y uso de variables
5. Scripts: Capacidad de ejecutar archivos de script con comandos
6. Mejores Redirecciones: Soporte para `>>` (append) y redirecciones más complejas
7. Control de Jobs: Implementar control completo de trabajos con `fg`, `bg`, `jobs`
8. Configuración: Archivo de configuración personalizable para el usuario

La mini-shell desarrollada demuestra la comprensión de los conceptos fundamentales de sistemas operativos y proporciona una plataforma extensible para futuras mejoras.

9 | Anexos

9.1 | Código Fuente Principal

9.1.1 | Estructura del Comando

```
path=/Users/ymila/OneDrive/Escritorio/SO PROYECTO/include/parser.hpp
start=6
struct Command {
    vector<string> argv;    // lista de argumentos del comando
    bool background = false; // indica si se ejecuta en segundo plano
    (&)
    string outputFile;      // archivo para redirección de salida (>)
    string inputFile;      // archivo para redirección de entrada (<)
};
```

9.1.2 | Bucle Principal del Shell

```
path=/Users/ymila/OneDrive/Escritorio/SO PROYECTO/src/shell.cpp start=17
void Shell::run() {
    string line;
    signal(SIGINT, SIG_IGN); // Ignorar Ctrl+C en el proceso padre

    while (true) {
        showPrompt();          // Mostrar el prompt
        if (!getline(cin, line)) break; // Leer línea de comando
        if (line.empty()) continue; // Si está vacía, continuar
        if (line == "salir") break; // Comando para terminar la
shell
        handleCommand(line);    // Procesar comando
    }

    cout << "Saliendo de la minishell..." << endl;
}
```

9.2 | Compilación y Ejecución

El proyecto incluye un Makefile para facilitar la compilación:

```
path=null start=null
# Compilar el proyecto
make all

# Ejecutar la mini-shell
```

```
./minishell
```

```
# Limpiar archivos objeto  
make clean
```

9.3 | Referencias

- Tanenbaum, A. S., & Bos, H. (2014). **Modern Operating Systems** (4th ed.). Pearson.
- Stevens, W. R., & Rago, S. A. (2013). **Advanced Programming in the UNIX Environment** (3rd ed.). Addison-Wesley.
- Love, R. (2013). **Linux System Programming** (2nd ed.). O'Reilly Media.
- Manual páginas del sistema Linux (man 2 fork, man 2 execvp, etc.)