



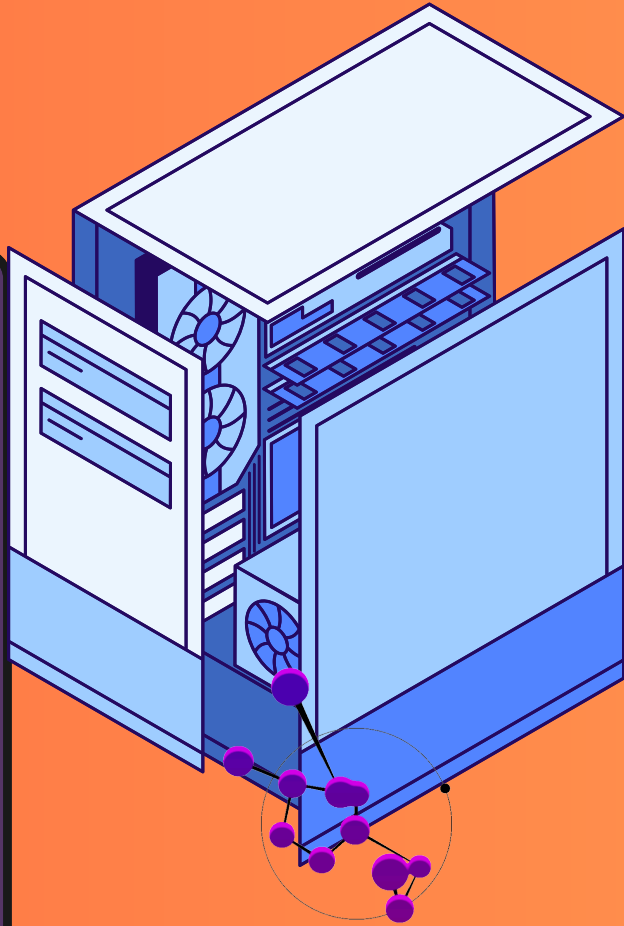
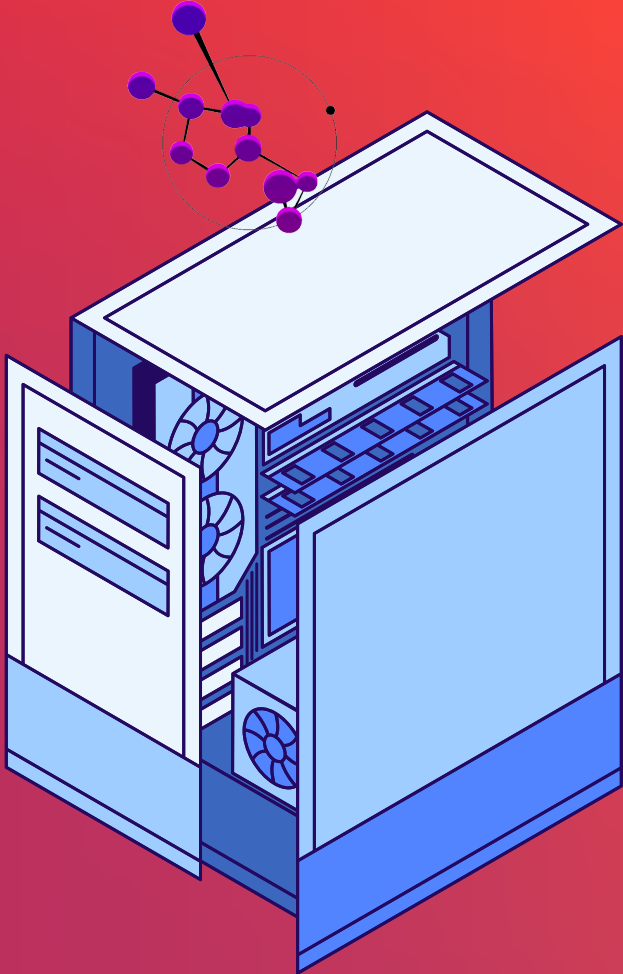
MINI- TERMINAL EN C++

INTEGRANTES:

- Nicolás Arturo Marin Gonzales
- Roger Cristian Huanca Pozo



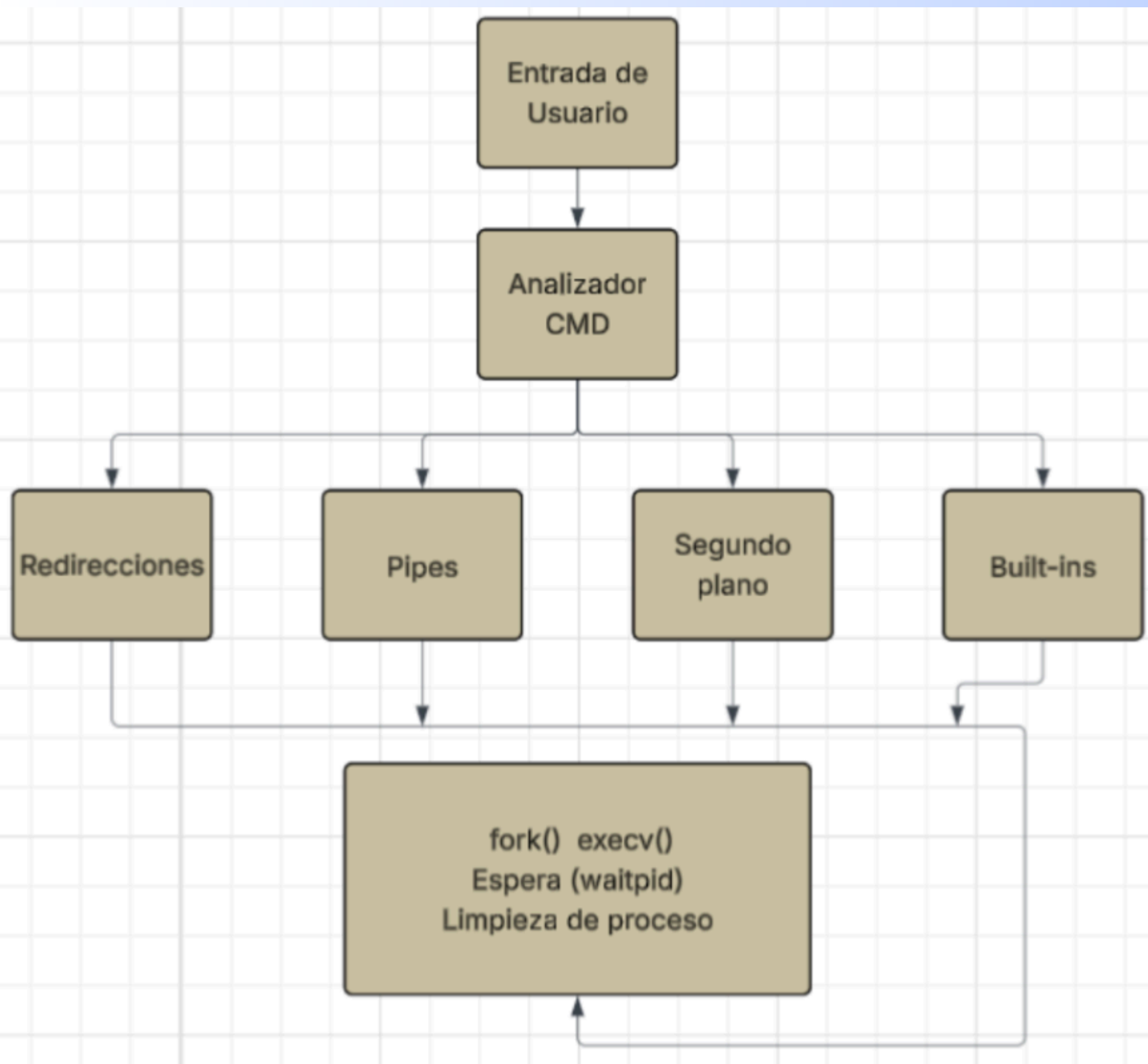
INTRODUCCION



Implementar una MiniShell funcional en C++ que replique el comportamiento básico de una terminal de Linux, ya sea en sus distros como Ubuntu y Fedora, gestionando procesos, redirecciones, pipes y comandos internos, aplicando conceptos de concurrencia y manejo de E/S del sistema operativo. Así mismo incorporar comandos internos básicos como cd, ls, help, alias, history, todo esto utilizando llamados al sistema POSIX.

La MiniShell ejecuta comandos externos del sistema (ubicados en /bin y /usr/bin), soporta redirecciones (>, >>), ejecución concurrente en segundo plano (&), y la conexión entre procesos mediante pipe (|). Además, incluye funcionalidades internas sin necesidad de crear nuevos procesos.





Primero, la entrada del usuario pasa al analizador de comandos (CMD), encargado de dividir y reconocer los argumentos, operadores y posibles símbolos especiales. Luego, según el tipo de instrucción detectada, el flujo se dirige hacia los módulos correspondientes: redirección de salida (`>` y `>>`), pipes (`|`), ejecución en segundo plano (`&`) o comandos internos (como `cd`, `help`, `history`). Cada módulo prepara el entorno necesario y, finalmente, el proceso se ejecuta mediante las llamadas al sistema `fork()` y `execv()`, que crean y reemplazan procesos. Al concluir, la shell usa `waitpid()` para sincronizar la finalización y realizar la limpieza de procesos evitando zombies, manteniendo así un ciclo continuo de lectura y ejecución de comandos.

DETALLES DE IMPLEMENTACION

Llamas al sistema POSIX o API's POSIX usadas:

Función	Propósito
fork()	Crea un proceso hijo.
execv()	Ejecuta un programa reemplazando la imagen del proceso.
waitpid()	Espera la finalización de un proceso específico.
pipe()	Crea un canal de comunicación unidireccional entre procesos.
dup2()	Redirige descriptores de archivo para redirección y pipes.
open() / close()	Abren o cierran archivos para redirección de salida.
chdir()	Cambia el directorio actual (para cd).
stat()	Verifica permisos de ejecución de archivos.



CONCURRENCIA Y SINCRONIZACIÓN



La concurrencia se maneja mediante procesos hijos creados con la función de llamada al sistema `fork()`.

- Si un comando incluye `&`, el proceso hijo no bloquea la shell principal.
 - Si no incluye `&`, la shell espera su finalización con `waitpid()`.
- Los pipes crean dos procesos hijos conectados por un canal (`pipe(fd)`), permitiendo que la salida del primero sea la entrada del segundo. Además, se evita el interbloqueo (deadlock) asegurando que cada descriptor de archivo se cierre correctamente tras duplicarse con `dup2()`.



PRUEBAS

Categoría	Comando probado	Resultado esperado
Comando simple	ls, pwd, echo Hola	Se ejecutan correctamente y muestran salida.
Ruta absoluta	/bin/ls, /usr/bin/whoami	Ejecuta el comando directamente por su ruta.
Redirección simple	ls > lista.txt	Crea el archivo con el resultado del comando.
Redirección append	echo Hola >> lista.txt	Añade texto sin borrar contenido anterior.
Segundo plano	sleep 5 &	Devuelve control inmediato a la shell.
Built-in cd	cd /home	Cambia correctamente el directorio actual.
Built-in history	history	Lista comandos ejecutados en la sesión.
Built-in alias	alias ll='ls -l'	Crea y almacena alias funcionales.

CONCLUSIONES



En conclusión, se logró implementar una Mini-Shell funcional aplicando las principales llamadas al sistema POSIX, lo que permitió ejecutar comandos, gestionar procesos y manejar redirecciones de forma eficiente. Esta práctica reforzó de manera significativa los conceptos de concurrencia, comunicación entre procesos (IPC) y gestión de procesos en entornos tipo Unix, al trabajar directamente con funciones del sistema operativo. Además, el diseño modular del programa facilitó la integración progresiva de nuevas funcionalidades como redirecciones, ejecución en segundo plano y comandos internos sin comprometer la estabilidad ni la claridad del código. Finalmente, la experiencia demostró la importancia del control manual de memoria y procesos en C++, así como la utilidad de comprender en profundidad cómo las capas bajas del sistema operan para construir herramientas más complejas.

Figura 1 muestra la ejecución exitosa de la mini-shell con todas las funcionalidades implementadas.

```
● cristian@cristian-VirtualBox:~/mini-shell$ g++ proyecto.cpp -o proyecto
○ cristian@cristian-VirtualBox:~/mini-shell$ ./proyecto
=== MiniShell Mejorada ===
Soporta:
- Ejecución normal (ls, pwd, echo)
- Rutas absolutas (/bin/ls)
- Pipes (|)
- Redirección (> y >>)
- Segundo plano (&)
- Built-ins: cd, help, history, alias
- Comando salir

minishell> |
```


Figura 2 Ejecución de Comandos Básicos en la Mini-Shell

como ls (listar archivos), pwd(mostrar directorio actual), echo. Se observa el correcto manejo de procesos fork/exec y la salida estándar sin redirección.

```
minishell> ls
a          archivo.txt  copia          lista.txt     mini.cpp     output       proyect.cpp   salida.txt
a.cpp      b          directorio.txt mensaje.txt   minishell    ox.cpp       proyecto     saludo.txt
archivos.txt b.cpp     listas.txt     mini          on.txt       para         proyecto.cpp  texto.txt
minishell> pwd
/home/cristian/mini-shell
minishell> echo

minishell> █
```

Figura 2 y 3 Ejecución de Comandos Básicos en la Mini-Shell y con rutas absolutas.

Como ls (listar archivos), pwd (mostrar directorio actual), echo. Se observa el correcto manejo de procesos fork/exec y la salida estándar sin redirección.

```
minishell> ls
a          archivo.txt  copia          lista.txt     mini.cpp     output       proyect.cpp   salida.txt
a.cpp      b          directorio.txt mensaje.txt   minishell    ox.cpp       proyecto     saludo.txt
archivos.txt b.cpp     listas.txt     mini          on.txt       para         proyecto.cpp  texto.txt
minishell> pwd
/home/cristian/mini-shell
minishell> echo

minishell> █
```

```
minishell> /bin/ls
a          archivo.txt  copia          lista.txt     mini.cpp     output       proyect.cpp   salida.txt
a.cpp      b          directorio.txt mensaje.txt   minishell    ox.cpp       proyecto     saludo.txt
archivos.txt b.cpp     listas.txt     mini          on.txt       para         proyecto.cpp  texto.txt
minishell> /usr/bin/whoami
cristian
minishell> █
```

Prueba de pipes simples con dos casos: (1) `ls | grep cpp` filtró 6 archivos `.cpp` del directorio actual, y (2) `cat /etc/passwd | wc -l` retornó 48 líneas de usuarios del sistema. La implementación utiliza `pipe()`, `fork()` doble y `dup2()` para redirigir `stdout` del primer comando a `stdin` del segundo.

```
minishell> ls | grep cpp
a.cpp
b.cpp
mini.cpp
ox.cpp
proyect.cpp
proyecto.cpp
minishell> cat /etc/passwd | wc -l
48
minishell> □
```

Figura 5. Redirección de Salida con Operador >> (Append)

Nota. Secuencia de comandos mostrando: (1) manejo de error cuando se intenta ejecutar un archivo .txt como comando, y (2) redirección exitosa con echo Hola >> lista.txt que agrega contenido al archivo sin mostrarlo en pantalla.

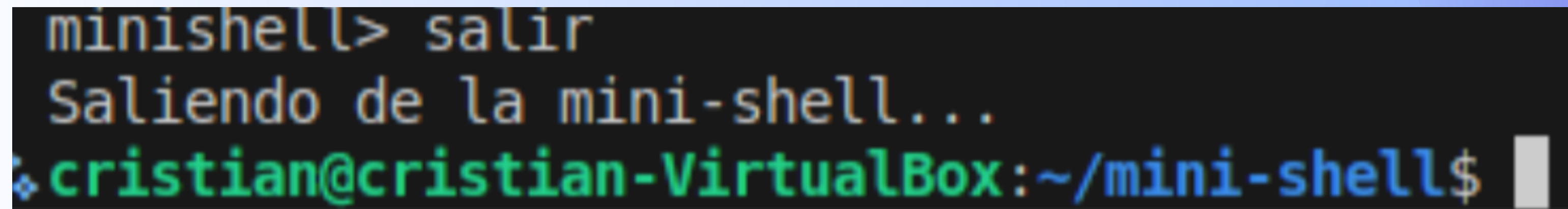
```
minishell> lista.txt
Error: comando 'lista.txt' no encontrado en /bin o /usr/bin
minishell> echo Hola >> lista.txt
minishell> 
```

Figura 6 Ejecución de Proceso en Segundo Plano con Operador &

```
minishell> sleep 5 &  
[Proceso en segundo plano] PID: 4794  
minishell> 
```

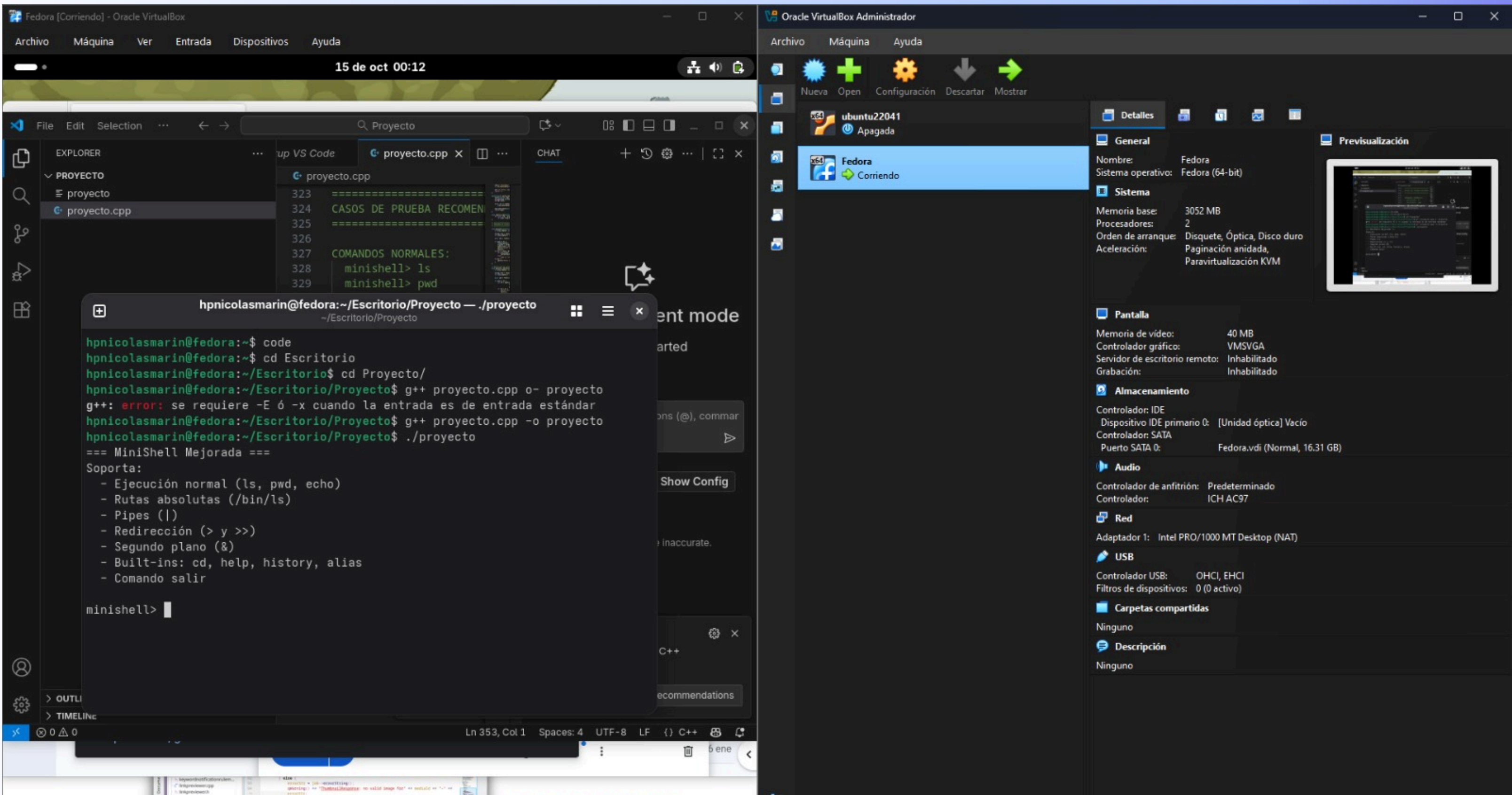
Prueba del comando `sleep 5 &` que suspende la ejecución por 5 segundos sin bloquear el intérprete. El sistema muestra el mensaje "[Proceso en segundo plano] PID: 4794" y retorna el prompt inmediatamente. La implementación utiliza `fork()` sin `waitpid()` bloqueante, aplicando `WNOHANG` para recolectar procesos zombie.

Figura 7 Finalización de la terminal con Comando "salir"



```
minishell> salir
Saliendo de la mini-shell...
❖ cristian@cristian-VirtualBox:~/mini-shell$
```


Figura 8 Compilación en otro distro de Linux Fedora 39





THANK YOU!