
PROYECTO DE UNIDAD I MINI-SHELL

INTEGRANTES:

Josue David Poma Sucso 2022-119090

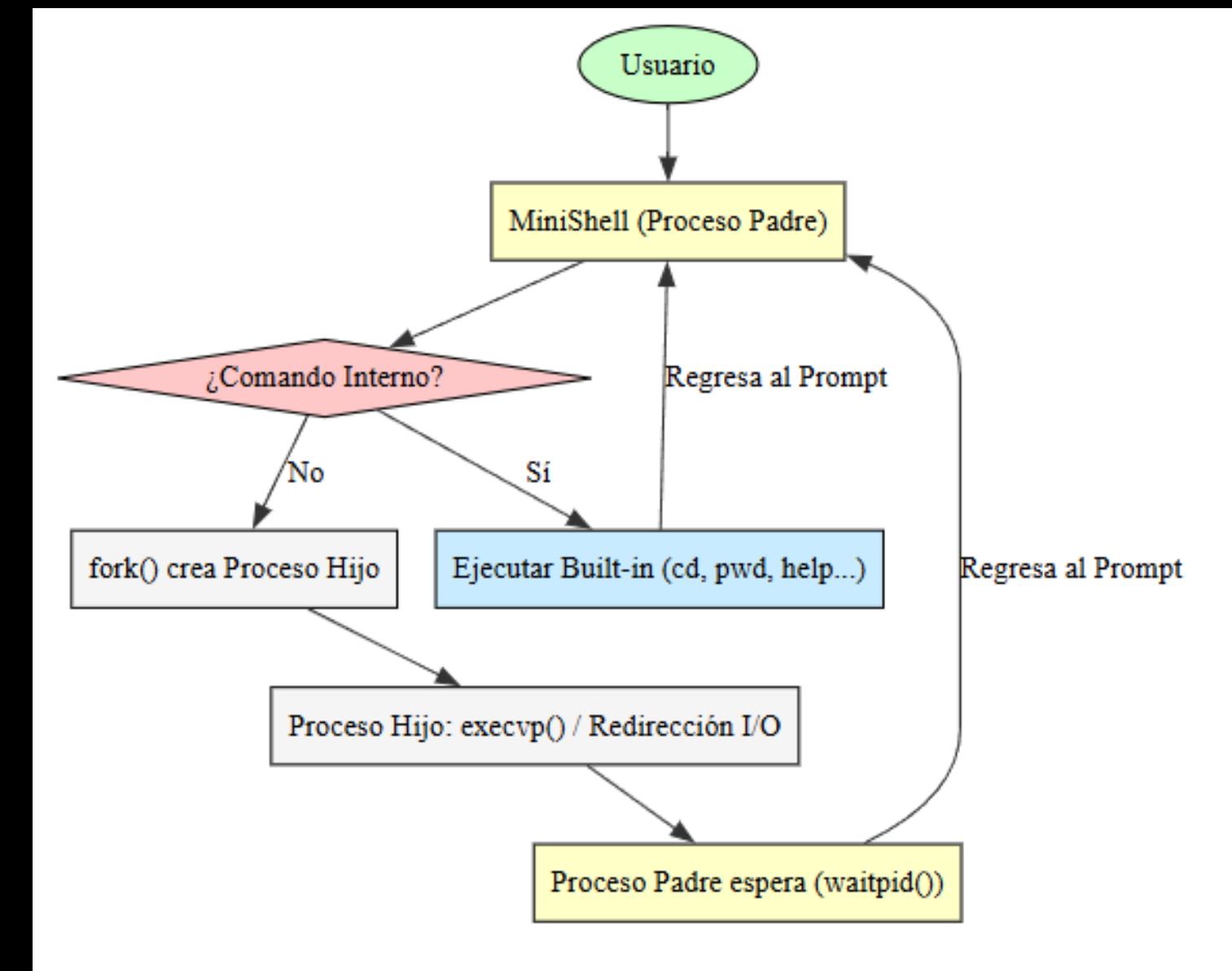
Hilder Elvis Robles Gonzales 2022-119014

Arquitectura general de la MiniShell

La MiniShell actúa como un intérprete de comandos, encargada de recibir las instrucciones del usuario, analizarlas y determinar si se trata de un comando interno (implementado dentro de la shell) o un comando externo (programa del sistema).

Cuando se ejecuta un comando externo, se crea un nuevo proceso con `fork()` y se sustituye su contenido con `execvp()`.

El proceso padre espera a que el hijo termine antes de mostrar nuevamente el prompt.

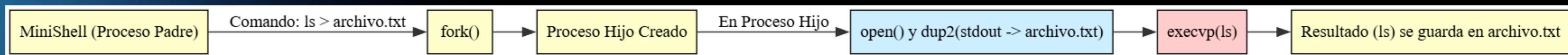


Redirección de salida (I/O)

Permite redirigir la salida estándar de un comando hacia un archivo.

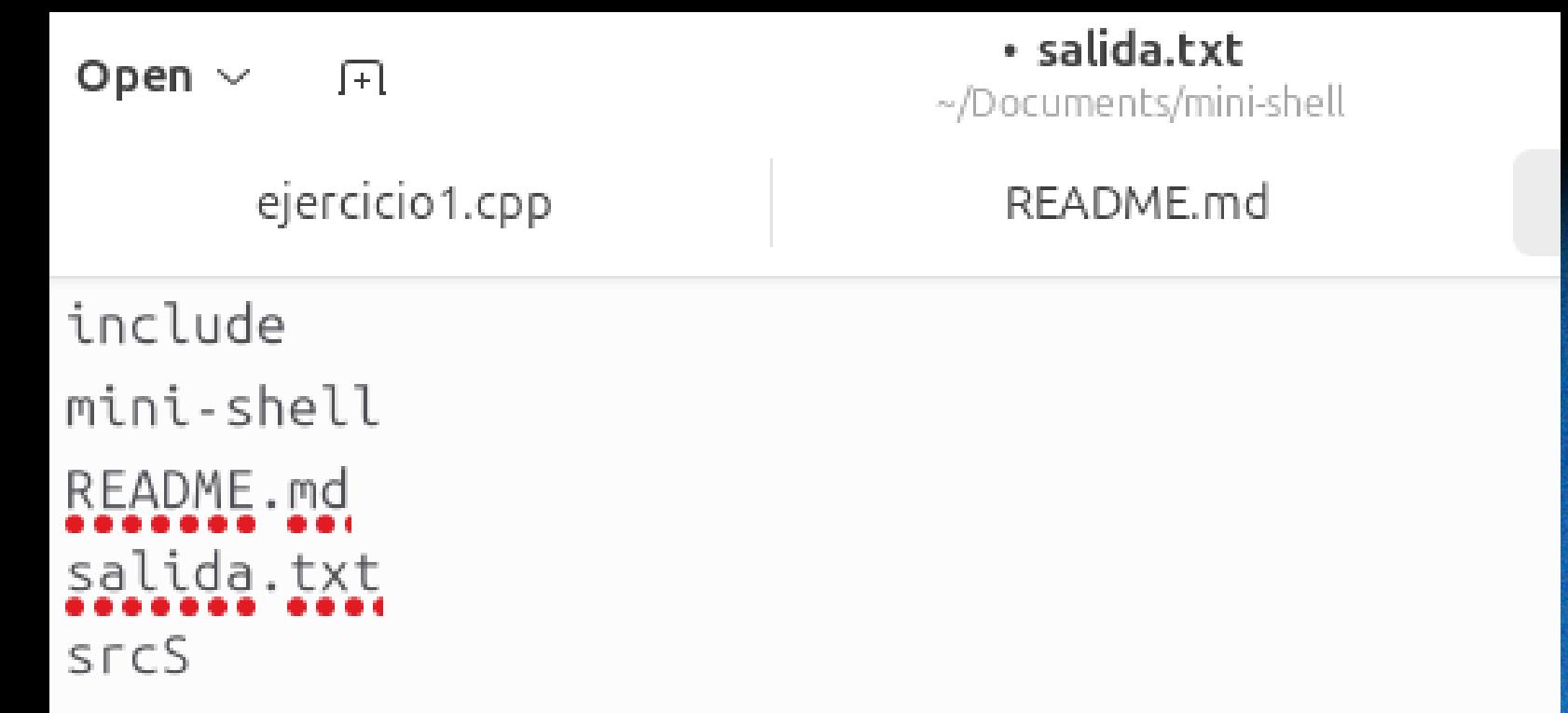
En lugar de mostrar el resultado en la terminal, se escribe en el archivo indicado.

Esto se logra duplicando el descriptor de archivo (dup2) dentro del proceso hijo antes de ejecutar el comando.



Ejemplo:

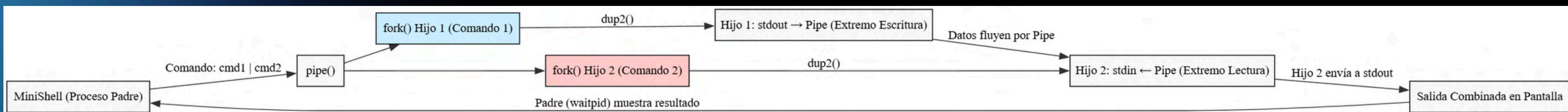
```
mini-shell$ ls > salida.txt
mini-shell$ cat salida.txt
include
mini-shell
README.md
salida.txt
src
mini-shell$
```



Pipes (tuberías)

Los pipes conectan la salida de un comando con la entrada de otro, permitiendo la comunicación entre procesos.

La MiniShell crea una tubería unidireccional con pipe(), y dos procesos: uno escribe en el pipe y el otro lee.



Ejemplo:

```
mini-shell$ ls | grep cpp
mini-shell$
```



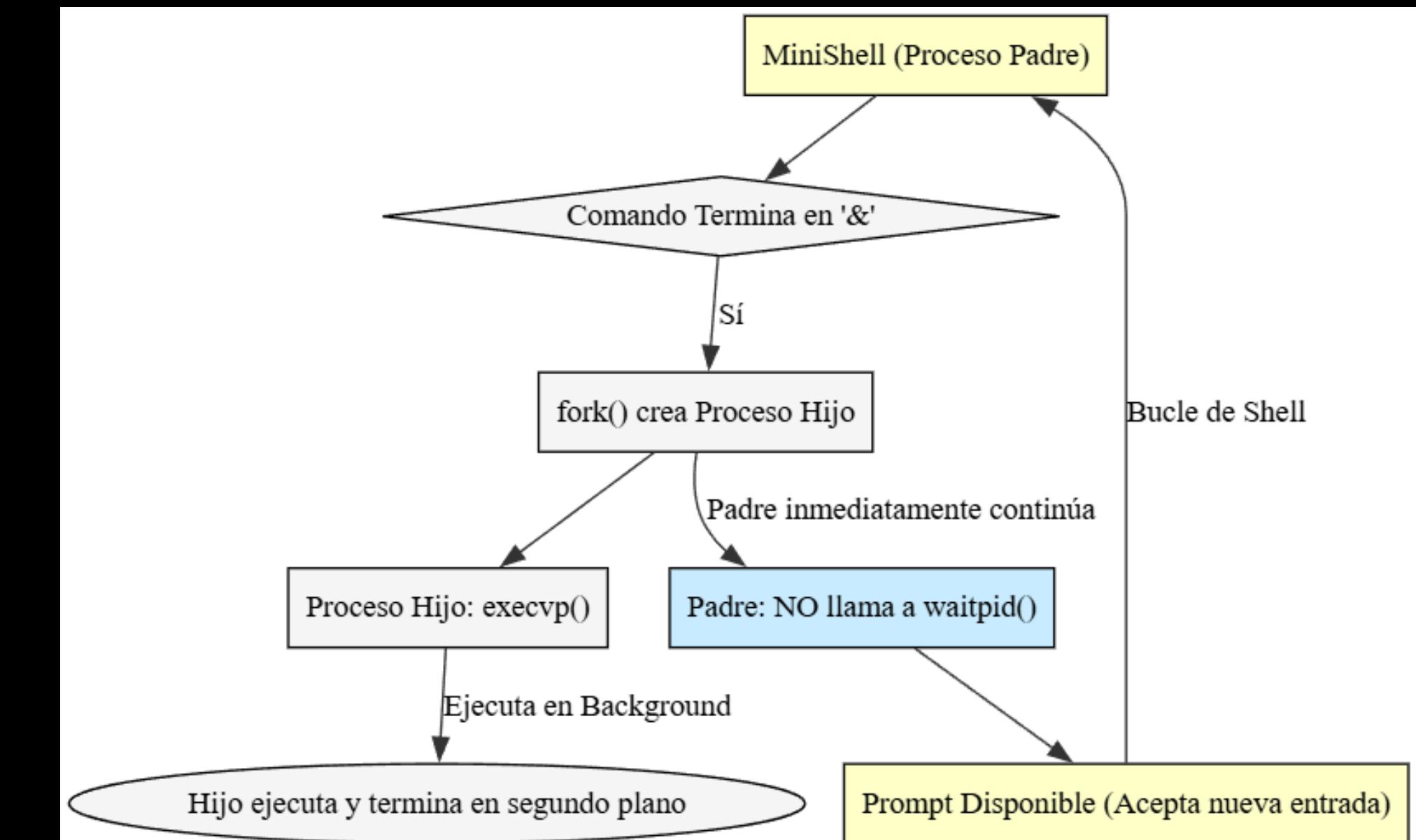
```
mini-shell$ cd src
mini-shell$ ls | grep cpp
builtins.cpp
executor.cpp
main.cpp
parser.cpp
signals.cpp
mini-shell$ █
```

Ejecución en segundo plano (&)

Permite ejecutar procesos sin bloquear el prompt.
Cuando un comando finaliza con &, el proceso hijo se ejecuta en background, y la MiniShell muestra su PID sin esperar a que termine.

Ejemplo:

```
mini-shell$ sleep 6 &
[background pid 4625]
mini-shell$
```





Concurrencia y sincronización

La mini-shell ejecuta varios comandos al mismo tiempo mediante el comando parallel, que crea hilos (pthread) independientes para cada orden. Así se logra un paralelismo real, aprovechando los recursos del sistema.

Para evitar condiciones de carrera, se usa un mutex (`std::mutex`) que bloquea el acceso a datos compartidos (como historial o alias) cuando un hilo los modifica.

Además, el programa sincroniza los hilos con `pthread_join()` para esperar su finalización antes de continuar, y no usa múltiples bloqueos, evitando interbloqueos (deadlocks).

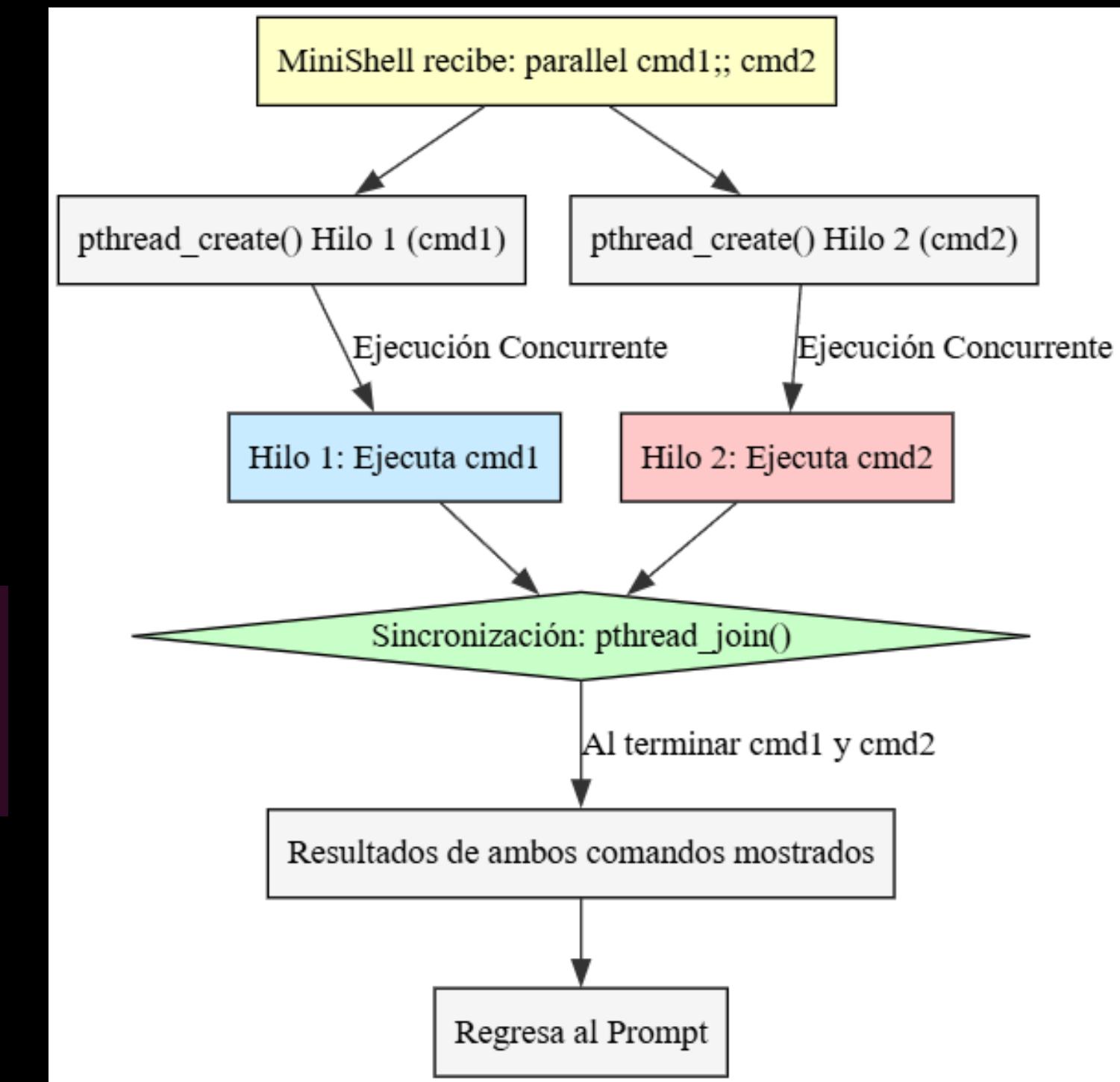
Concurrencia (Parallel con hilos)

El comando `parallel` permite ejecutar varios comandos simultáneamente usando hilos (`pthread_create`).

Cada comando se ejecuta en un hilo independiente, y la shell sincroniza todos con `pthread_join()` para evitar condiciones de carrera.

Ejemplo:

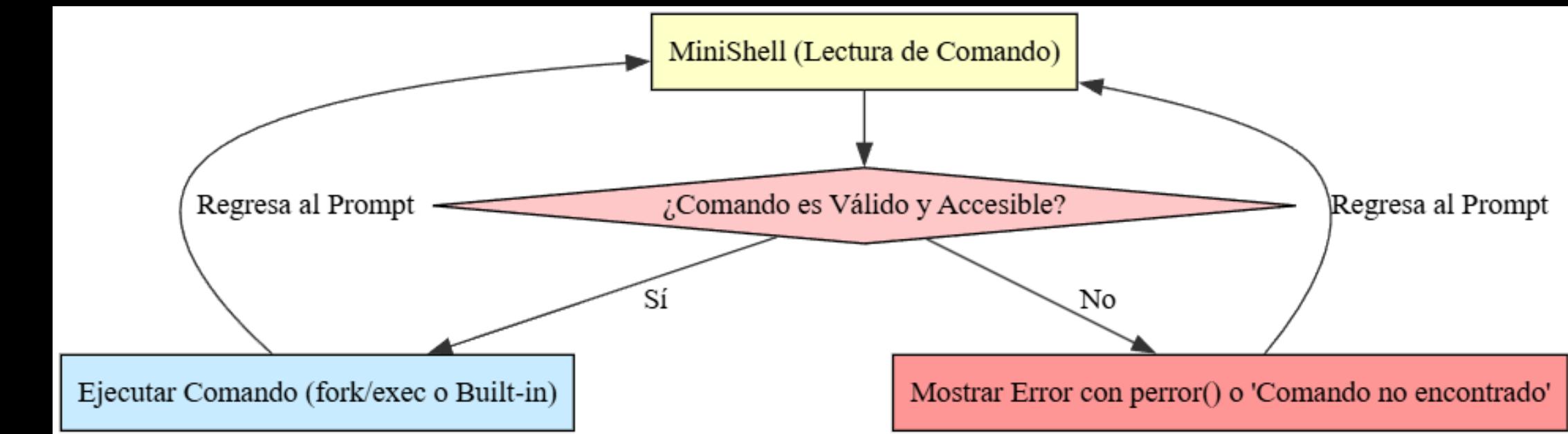
```
mini-shell$ parallel ls;;pwd  
/home/David/Documents/mini-shell/src  
builtins.cpp executor.cpp main.cpp parser.cpp signals.cpp  
mini-shell$
```



Manejo de errores

La MiniShell detecta y reporta errores cuando el comando no existe o la ruta es inválida.

Usa perror() y el código errno del sistema para mostrar mensajes claros y útiles.



Ejemplo:

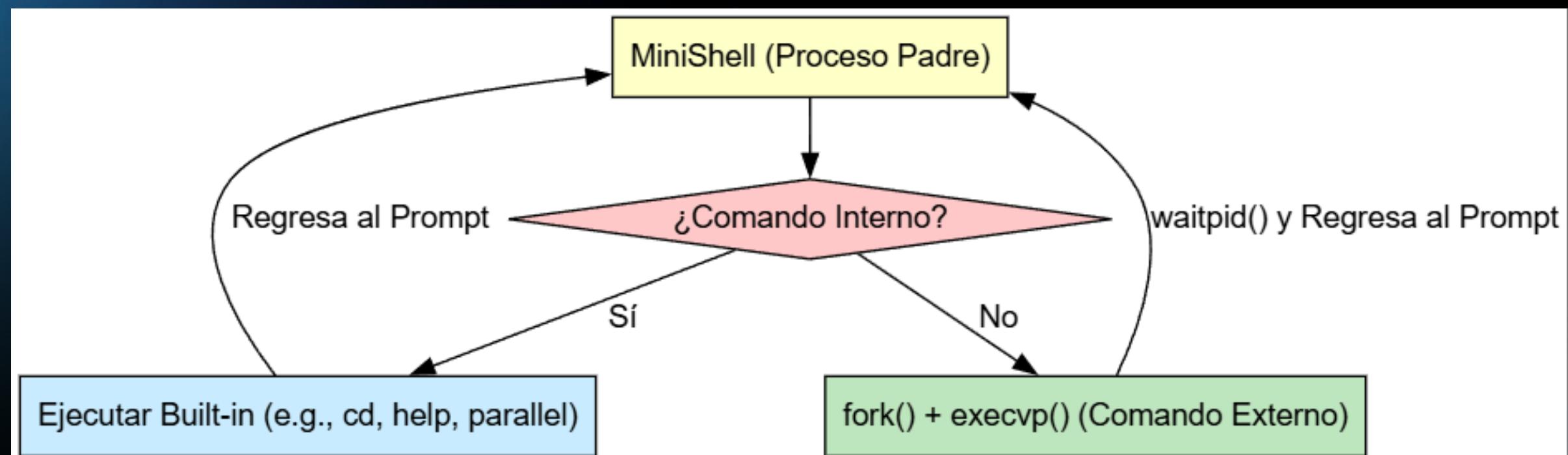
```
mini-shell$ comando_inexistente
execvp comando: No such file or directory
mini-shell$
```



Comandos internos (built-ins)

Son funciones implementadas directamente dentro de la MiniShell.

No crean procesos nuevos, sino que se ejecutan en el mismo contexto del intérprete.



Comandos internos disponibles

Comando	Función
salir	Termina la ejecución de la
cd	Cambia el directorio actual
pwd	Muestra el directorio actual
help	Lista los comandos disponibles
history	Muestra el historial de comandos
alias	Crea o muestra alias
parallel	Ejecuta varios comandos en
meminfo	Muestra estadísticas de

Comandos internos (built-ins)

Ejemplos:

```
mini-shell$ salir
```

```
David@David:~/Documents/mini-shell$
```

```
mini-shell$ pwd
```

```
/home/David/Documents/mini-shell
```

```
mini-shell$ cd src
```

```
mini-shell$ pwd
```

```
/home/David/Documents/mini-shell/src
```

```
mini-shell$
```

```
mini-shell$ help
```

```
Mini-shell - comandos soportados (built-ins):
```

```
salir          : salir de la shell
```

```
cd <dir>       : cambiar directorio
```

```
pwd            : mostrar directorio actual
```

```
history        : lista comandos ejecutados en la sesión
```

```
alias name='cmd' : crear alias simple (sin persistencia)
```

```
parallel cmd1 ;; cmd2 ;; ... : ejecutar comandos en paralelo (separador ';;')
```

```
meminfo        : muestra uso aproximado de memoria (VmSize, VmRSS, VmData)
```

```
help           : esta ayuda
```

```
mini-shell$ history
```

```
1  pwd
```

```
2  cd src
```

```
3  pwd
```

```
4  help
```

```
5  clear
```

```
6  history
```

```
mini-shell$
```

```
mini-shell$ meminfo
```

```
VmSize:      6420 kB
```

```
VmRSS:       3472 kB
```

```
VmData:      268 kB
```

```
mini-shell$
```

```
mini-shell$ alias 11 = 'ls -l'
```

```
mini-shell$ 11
```

```
execvp 'ls: No such file or directory
```

```
mini-shell$ alias
```

```
11=' 'ls -l'
```

```
mini-shell$
```



Gestión de memoria: estrategia

PROGRAMMING FOR INSIGHTS

La mini-shell sigue una estrategia basada en asignación y liberación controlada. Cada módulo del proyecto (lectura, parsing, ejecución) es responsable de manejar su propia memoria temporal, de manera que no existan dependencias innecesarias entre funciones.



```
    resp_iter = self.stub.GetStatuses()
    statuses = {}
    for data in resp_iter:
        status = Status(
            status_id=data.id, name=data.name,
            type=data.type, value=data.value,
            created_at=data.created_at,
            updated_at=data.updated_at)
```

Gestión de memoria: estrategia

Lectura de entrada

- La entrada del usuario se obtiene mediante funciones como `getline()` o `fgets()`, que asignan memoria dinámicamente para la cadena del comando. Una vez procesada la instrucción, la memoria es liberada inmediatamente con `free()` para evitar acumulación.

Tokenización y parsing

- Al separar el comando en argumentos, se utiliza una lista de punteros (`char **args`) generada con `malloc()` o `calloc()`. Estos arreglos permiten manipular los tokens de forma flexible y son liberados tras la ejecución del comando, ya sea interno o externo.

Ejecución de comandos

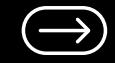
- En los comandos externos, el uso de `fork()` y `execvp()` crea un nuevo espacio de memoria aislado para el proceso hijo. Este proceso hereda únicamente la información esencial del padre, ejecuta el binario correspondiente y libera sus recursos automáticamente al finalizar.
- El proceso padre, mientras tanto, utiliza `waitpid()` para sincronizar la finalización del hijo, asegurando que la memoria asociada al proceso hijo sea recuperada por el sistema operativo.

Comandos internos (built-in)

- Estos comandos se ejecutan dentro del proceso principal. En este caso, la mini-shell libera cualquier estructura temporal una vez completada la operación (por ejemplo, al cambiar de directorio o mostrar variables del entorno).



shutterstock.com · 2177671221



Pruebas y resultados



Pruebas y resultados

Tipo	Comando / Operador	Descripción	Ejemplo de uso	Resultado esperado
Interno	cd	Cambia el directorio de trabajo actual.	cd /home	Cambia al directorio /home.
Interno	salir	Termina la ejecución de la MiniShell.	salir	Cierra la shell.
Interno	pwd	Muestra el directorio actual.	pwd	Imprime la ruta actual.
Interno	help	Muestra ayuda general con los comandos internos disponibles.	help	Lista de comandos con breve descripción.
Interno	history	Muestra el historial de comandos ejecutados.	history	Lista numerada de comandos anteriores.
Interno	alias	Crea alias para comandos o muestra los existentes.	alias ls="ls -l"	Define un alias para ls -l.
Interno	meminfo	Muestra información de memoria usada por la shell (malloc/free).	meminfo	Estadísticas de memoria dinámica.
Interno	parallel	Ejecuta varios comandos simultáneamente usando threads (pthread_create).	parallel ls;; pwd;;date	Ejecuta ls, pwd y date en paralelo.



Pruebas y resultados

Tipo de Comando	Comando	Descripción	Ejemplo de uso	Salida esperada	→
Externo	ls	Lista los archivos del directorio actual.	ls	Muestra los nombres de archivos y carpetas.	
Externo	pwd	Muestra el directorio de trabajo actual.	pwd	/home/usuario	
Externo	cat	Muestra el contenido de un archivo.	cat archivo.txt	Imprime el contenido del archivo.	
Externo	echo	Imprime texto o variables.	echo "Hola"	Hola	
Externo	grep	Busca texto dentro de archivos.	grep "hola" archivo.txt	Muestra líneas que contienen "hola".	
Externo	ps	Muestra los procesos activos del sistema.	ps	Lista de procesos en ejecución.	
Externo	sleep	Detiene la ejecución por unos segundos.	sleep 3	Pausa durante 3 segundos.	
Externo	date	Muestra la fecha y hora actual.	date	Ejemplo: Sat Oct 11 10:05:00 2025	
Externo	whoami	Muestra el usuario actual.	whoami	Ejemplo: david	



A través de este proyecto, se pudo evidenciar la importancia de los procesos, los hilos y las llamadas al sistema POSIX en la ejecución de tareas y en la administración eficiente de los recursos del sistema.

Este trabajo permitió profundizar en el funcionamiento interno de un intérprete de comandos, entendiendo cómo se crean y controlan los procesos, cómo se gestionan las entradas y salidas, y cómo se mantiene la sincronización entre tareas concurrentes. Además, se demostró que una correcta gestión de memoria y la prevención de condiciones de carrera son esenciales para garantizar la estabilidad y confiabilidad de cualquier entorno de ejecución.



Conclusiones



THANK YOU