UNJBG

MINI-SHELL

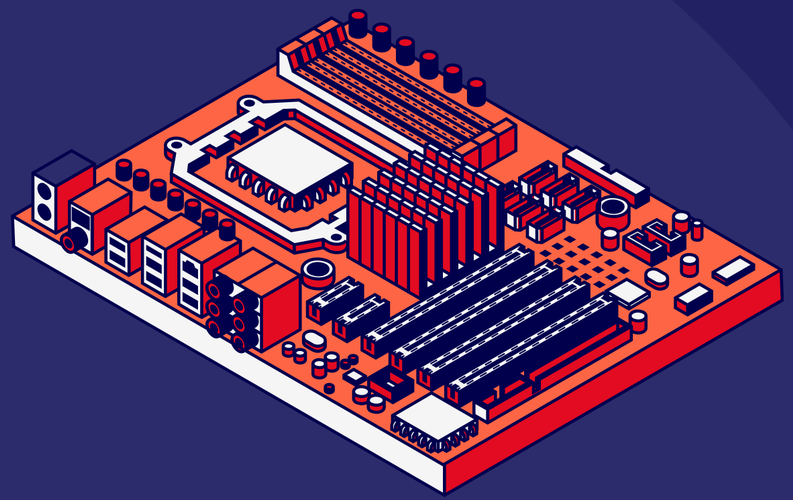
Interprete de comandos

Sistemas Operativos

INTEGRANTES

SEBASTIAN QUISPE CONDORI - 2023-119056

FABIAN VARGAS QUISPE - 2022-119095



OBJETIVOS

Implementar un intérprete de comandos (mini-shell) en C++ para sistemas Linux que demuestre el dominio práctico de los conceptos fundamentales de sistemas operativos: gestión de procesos, comunicación entre procesos, concurrencia y gestión de memoria.

Alcance

Funcionalidades Base :

- Prompt personalizado
- Resolución de rutas (absolutas y relativas)
- Ejecución mediante fork/exec
- Manejo de errores
- Redirección de salida (>)
- Comando de salida

Extensiones Implementadas:

- Pipes simples y múltiples (|)
- Comandos built-in (cd, pwd, help, history, echo, meminfo)
- Redirección avanzada (>>, <)
- Procesos en background (&)
- Manejo de señales (SIGINT, SIGCHLD)

Estadísticas del Proyecto:

- Líneas de código: ~2100
- Archivos fuente: 6 (.cpp) + 5 (.h)
- Funciones implementadas: 43
- Casos de prueba: 31
- Memory leaks: 0

Arquitectura de Software

Diagrama de Flujo

FUNCIONALIDADES

- Prompt personalizado: "Mini-Shell-ESIS:".
- Resolución de comandos usando la variable PATH (con fallback a /bin y /usr/bin).
- Ejecución con fork + exec; foreground por defecto.
- Redirecciones: >, >> y <.
- Pipes múltiples.
- Background con &, notificación y recolección no bloqueante.
- Built-ins: cd, pwd, help, history, echo, meminfo, salir/exit.
- Historial persistente en archivo (configurable).
- Limitaciones (fuera de alcance actual):
- alias y export no implementados.
- Tabla de jobs y builtins fg/bg no disponibles.
- Parallel (con pthread) no implementado en esta versión (se propone como mejora).

COMPONENTES

- main.cpp — inicialización y banner.
- shell.cpp — loop principal, prompt, historial.
- parser.cpp — tokenización (soporte de comillas, escapes y operadores) y
- construcción de estructura de comandos.
- executor.cpp — resolución de ruta, fork/exec, manejo de pipes y redirecciones.
- builtins.cpp — implementación de comandos internos.
- utils.cpp — utilidades (strings, manejo señales, recolección background).
- include/*.h — cabeceras y config.h con constantes (ruta del history, etc.).
- tests/*.sh — scripts de prueba automatizados.

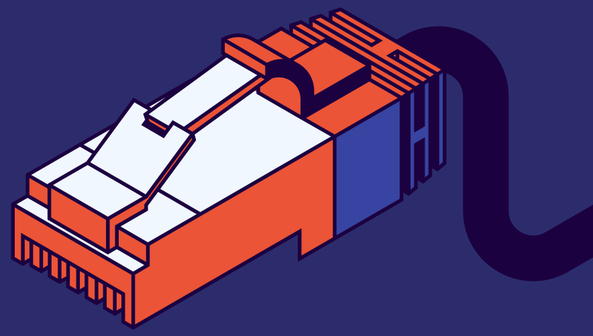
SINCRONIZACIÓN ENTRE PROCESOS

waitpid() se utiliza para que el proceso padre espere a la finalización del hijo en ejecuciones en primer plano; esto evita que el intérprete acepte un nuevo comando hasta que el hijo termine (comportamiento foreground esperado).

Cuando un proceso se ejecuta en segundo plano (con &), el proceso padre no espera: muestra el PID del hijo y sigue aceptando comandos. Luego limpia los procesos terminados con waitpid(..., WNOHANG) (en Utils::reapBackgroundProcesses() o en el manejador SIGCHLD), evitando zombies y manteniendo el prompt activo.

Las tuberías (|) conectan la salida de un proceso con la entrada del siguiente; el executor crea los pipe() necesarios y configura duplicaciones (dup2) en cada hijo según su posición en la cadena, de manera que los procesos puedan trabajar en paralelo y transferir datos sin interferencias.

GESTIÓN DE MEMORIA



Executor.cpp

En executor.cpp están las llamadas a new[] y sus liberaciones correspondientes.

Historial

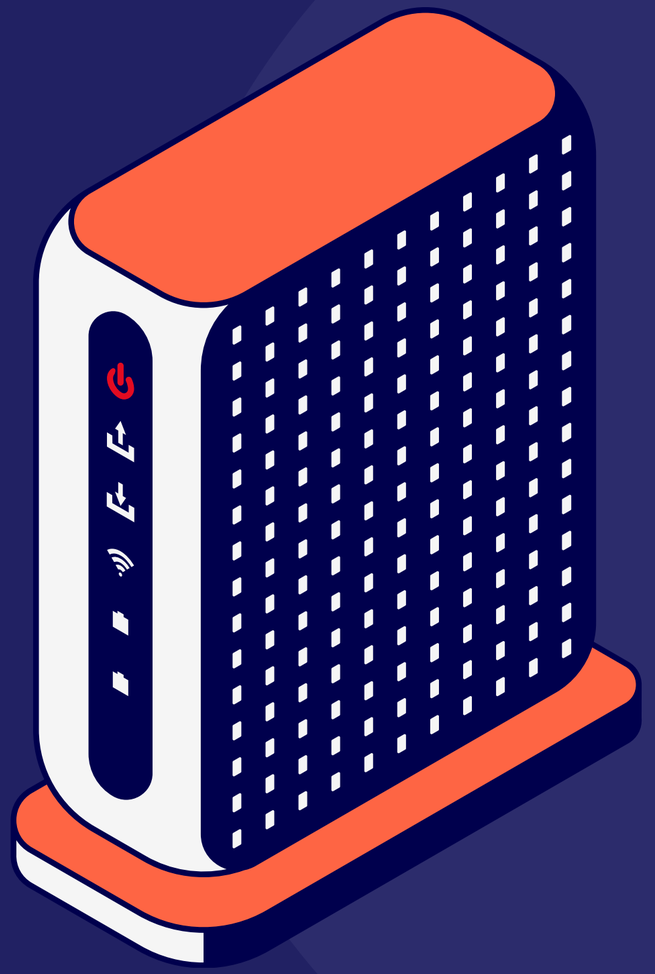
Historial en shell.cpp usa std::vector y se persiste en fichero (configurable).

Resultados

Pruebas funcionales no mostraron comportamiento indefinido en uso normal.

Studio Shodwe

DEMOSTRACIÓN DEL CÓDIGO



GRACIAS