UNIDAD 01: EVALUACIÓN DE PRODUCTO

PROPÓSITO DEL PROYECTO

Los equipos de trabajo implementarán un intérprete de comandos (tipo mini-shell) en C++ sobre Linux, demostrando dominio práctico de procesos, hilos, concurrencia y gestión de memoria, contenidos de la Unidad I.

RESULTADOS DE APRENDIZAJE

- Diseñar e implementar procesos hijo con `fork/exec` y controlarlos desde el proceso padre con wait/waitpid.
- Aplicar redirección y pipes en Linux mediante dup2, `pipe` y `close`, manejando descriptores de archivo.
- Emplear mecanismos de concurrencia (p. e. hilos con `pthread`, sincronización con variables condición/`mutex`, o control de procesos concurrentes) de forma segura y reproducible.
- Integrar gestión de memoria: uso responsable de heap (`malloc`/new, `free`/`delete`), medición básica de consumo, y/o uso de mmap cuando corresponda.

ESPECIFICACIONES FUNCIONALES

La mini-shell debe cumplir las siguientes características base:

Prompt personalizado

- Mostrar un prompt propio y leer la línea de comando.

Resolución de rutas

- Si el usuario escribe una ruta absoluta (p. e. `/usr/bin/ls`), ejecútalo tal cual.
- Si no es ruta absoluta, asumir **`/bin`** (p.e. `ls` → ejecutar `/bin/ls` si existe).
- Manejar errores cuando el ejecutable no exista o no tenga permisos.

Ejecución mediante procesos

- La invocación debe realizarse con **`fork()`** y luego **`exec*()`** (variante libre), desde el proceso hijo.
- El proceso padre (intérprete) espera la finalización del hijo con **`wait()`/`waitpid()`** antes de aceptar el siguiente comando (comportamiento *foreground* por defecto).

Manejo de errores

- Mensajes claros cuando el comando no exista, la ruta no sea válida o `exec` falle (incluir `errno`/`perror` cuando sea útil).

Redirección de salida estándar (>)

- Si el usuario ingresa: `nombrePrograma arg2 > archivo`
- Redirigir stdout del proceso hijo a `archivo` (crearlo/truncarlo). Nada debe mostrarse en pantalla para esa ejecución.
- Nota: Todos los tokens (incluido >) están separados por espacios.

Comando de salida

- El intérprete termina al ingresar la palabra `salir`.

Extensiones de valor agregado

Implementar al menos dos (2) extensiones extra correctamente integradas a la mini-shell:

- **Pipes** (|) simples (una tubería): `cmd1 | cmd2`.
- **Tareas en segundo plano** (&): no bloquear el *prompt* y mostrar **`waitpid` no bloqueante** o recolección diferida.
- **Redirección de entrada** (<) y **doble redirección de salida** (>>).
- **Comandos internos (built-ins)**: `cd`, `pwd`, `help`, `history`, `alias`.
- **Concurrencia con hilos**: *built-in* `parallel` que ejecute n comandos en paralelo con `pthread_create`, con sincronización.
- **Gestión de memoria instrumentada**: *built-in* `meminfo` que muestre uso aproximado de heap o estadísticos de asignaciones/liberaciones (envolturas a `malloc/free`).

- **Manejo de señales**: ignorar o capturar `SIGINT` en el padre (no matar la shell accidentalmente); pasar señales a hijos cuando corresponda.

Requisitos técnicos y buenas prácticas

- **POSIX**: usar llamadas de sistema vistas en clase: `fork`, `execvp/execve`, `wait/waitpid`, `pipe`, `dup2`, `open/close`, `chdir`, `getcwd`, `sigaction`, etc.
- **Estructura modular** (headers `.h/.hpp`, fuentes `.c/.cpp`) y **comentarios útiles** (no redundantes).
- **Mensajes y errores** en **español claro**.
- **Manejo cuidadoso de memoria** (evitar *leaks* y *dangling pointers*).
- **Control de versiones con Git**: commits atómicos y con mensajes significativos.
- **Portabilidad**: probar en al menos **dos** shells o distros (p. ej., Ubuntu y Debian/Fedora).

ENTREGABLES

1. Repositorio GitHub público

Estructura sugerida del repositorio:

`src/` (código fuente)
`include/`
`docs/` (informe técnico en pdf, diapositivas en pdf)
`README.md`

`README.md` con:

Descripción breve, requerimientos, instrucciones de compilación/ejecución.
Tabla de características implementadas (base y extras).
Casos de prueba y ejemplos de uso (con capturas).

2. Informe técnico

Estructura sugerida (máx. 10 páginas, sin anexos):

- Portada (curso, sección, equipo, integrantes, fecha).
- Objetivos y alcance.
- Arquitectura y diseño (diagrama simple de procesos/hilos, manejo de I/O).
- Detalles de implementación (APIs POSIX usadas, decisiones clave).
- Concurrencia y sincronización: qué se paraleliza y cómo se evita la condición de carrera/interbloqueo.
- Gestión de memoria: estrategia y evidencias.
- Pruebas y resultados (casos, cómo se validó cada requisito).
- Conclusiones y trabajos futuros.
- Anexos: comandos probados, scripts de test, etc.

3. Sustentación (15-20 minutos por equipo)

- Explicación del diseño y ejecución guiada de casos (incluidos extras) usando diapositivas.
- Vestimenta formal.
- Responder preguntas del docente.

ENTREGA Y FORMATO

Plataforma: Aula virtual (una sola entrega por equipo).

- Enlace al repositorio público.