Laboratorio Práctico: Mecanismos IPC

WSL (Ubuntu) - Nivel introductorio

Basado en el documento: IPC - Inter-Process Communication

OBJETIVO

Observar y analizar los tres mecanismos de IPC del documento: **Pipes**, **Señales** y **Memoria Compartida**.

PARTE 1: Pipes - Tuberías

Ejercicio 1.1: Pipe en Shell (sin programar)

bash

```
# Ejecutar comando encadenado del documento
ps aux | grep bash | wc -l

# Pregunta: ¿Cuántos procesos participan aquí?
# Respuesta: ps → grep → wc (tres procesos conectados por pipes)

# Verificar que son procesos diferentes
ps aux | grep bash | head -5
```

Ejercicio 1.2: Compilar y ejecutar pipe_ejemplo.c

bash

```
# El código ya está en el documento (página 9-10)
gcc pipe_ejemplo.c -o pipe_ejemplo

# Ejecutar
./pipe_ejemplo
```

Observar y responder:

Pregunta	Tu observación
¿Cuántos mensajes imprime el programa en total?	
¿El mensaje del padre siempre aparece antes que el del hijo? Ejecuta 3 veces y verifica.	
¿Qué pasa si el padre no hace `wait(NULL)`? (Predice, no modifiques)	

PARTE 2: Señales - Signals

Ejercicio 2.1: Señales desde teclado

bash

```
# Abrir una terminal y ejecutar:
sleep 300

# Presionar Ctrl+Z  (SIGSTOP - pausar)
# Ver estado: aparece [1]+  Stopped

# Reanudar en foreground:
fg

# Ahora presionar Ctrl+C  (SIGINT - interrumpir)
# El proceso termina
```

Ejercicio 2.2: Señales con kill

bash

```
# Terminal 1: Iniciar proceso en segundo plano
sleep 300 &
# Anotar el PID que muestra, ej: [1] 1234

# Terminal 2: Enviar señales al PID
kill -STOP 1234   # Pausar
kill -CONT 1234   # Continuar
kill -TERM 1234   # Terminar ordenadamente
```

Tabla a completar:

Señal	Número	¿Qué hace?	¿El proceso puede ignorarla?
SIGINT	2		
SIGSTOP	19		
SIGCONT	18		
SIGTERM	15		
SIGKILL	9		

Pista del documento (página 2): SIGKILL es "terminación forzada"

PARTE 3: Memoria Compartida - Shared Memory

Teoría del documento

> "Región de RAM accesible por múltiples procesos" - Más rápido que pipes/sockets (no copia datos).

Ejercicio 3.1: Compilar programas del documento

bash

```
# Compilar ambos programas del documento (página 4-5)
gcc shm_escritor.c -o shm_escritor
gcc shm_lector.c -o shm_lector
```

Ejercicio 3.2: Ejecutar en orden correcto

bash

```
# Terminal 1 - Ejecutar PRIMERO
./shm_escritor

# Terminal 2 - Ejecutar SEGUNDO (mientras escritor espera)
./shm_lector
```

Ejercicio 3.3: Verificar con comandos del sistema

bash

```
# En una tercera terminal, mientras ambos corren:
ipcs -m

# Salida esperada (similar a página 6):
# key        shmid      owner      perms      bytes      nattch     status
# 0x000004d2 65536      usuario    666        1024       2

# nattch = 2 significa: 2 procesos adjuntos (escritor y lector)
```

Ejercicio 3.4: Análisis de direcciones

```
[Escritor] Dirección de memoria: 0x7f8b3c7a2000
[Lector]   Dirección de memoria: 0x7f8a2b1c4000  ← ¡Diferente!
```

Pregunta	Respuesta del documento
¿Las direcciones son iguales?	
¿El contenido leído es el mismo?	
¿Por qué son diferentes las direcciones pero el contenido es igual?	

PARTE 4: Comparativa de Mecanismos (10 minutos)

Tabla del documento

Característica	Pipe	Memoria Compartida	Socket
Velocidad	Media	**Muy alta**	Media
Dirección	Unidireccional	Bidireccional	Bidireccional
Complejidad	Simple	Compleja	Media
Red	No	No	**Sí**

Preguntas de aplicación:

1. **Para comunicar dos procesos en la misma máquina que necesitan máxima velocidad:**
   - ¿Qué eliges según el documento?
2. **Para comandos encadenados en shell (`ls | grep`):**
   - ¿Qué usa el sistema?
3. **Para un navegador web hablando con un servidor en otra ciudad:**
   - ¿Qué necesitas?

ENTREGABLE DEL LABORATORIO

Capturas requeridas:

1. Ejecución de `pipe_ejemplo` mostrando mensajes de padre e hijo
2. Salida de `ipcs -m` con `nattch = 2`
3. Ejecución de `shm_escritor` y `shm_lector` mostrando direcciones diferentes pero contenido igual

Respuestas escritas:

- Tabla de señales (5 señales) completada
- Explicación con tus palabras: ¿por qué memoria compartida es más rápida que pipes?