Sistemas Operativos (SO)

72.11

2ndo Cuatrimestre 2025

TP1: Inter-Process Communication

Grupo 2

Valentin Ontivero - 60034

Jonathan Lucas Blankleder - 64660

Agustín German Ramirez Donoso - 64716

1. Decisiones tomadas durante el desarrollo

1.1. Estructura de Datos y Memoria Compartida

Se definieron dos estructuras en memoria compartida:

- **GameState**: guarda el estado del tablero, la lista de jugadores y toda la información relacionada al juego.

- **Semaphores**: contiene todos los semáforos para la sincronización y la comunicación entre procesos.

1.2. Comunicación con Jugadores

- Cada jugador envía sus movimientos al máster mediante **pipes anónimos**, redirigiendo la salida estándar (fd=1) al extremo de escritura del pipe creado por el máster.

- Los procesos de jugador y la vista se crean con `fork()` y luego se reemplazan por el binario correspondiente usando `execve()`, pasando como argumentos el ancho y alto del tablero.

1.4. Sincronización

Se utilizaron **semáforos** para:

- Coordinar la actualización del estado del juego (máster) y su visualización (vista).

- Garantizar que, mientras el máster modifica el estado, ningún jugador pueda leerlo, evitando *race conditions*.

- Avisar a cada jugador cuando se proceso su movimiento, garantizando que cada jugador espera a la señal del master para calcular su proxima jugada.

El control de concurrencia se implementó con el algoritmo **Readers/Writers** (preferencia a lectores), junto con un semáforo de *turnstile* para evitar inanición del escritor.

1.5. Precálculos antes de la Zona Crítica

En el `while` del máster (lógica del juego), antes de entrar a la zona crítica se realizan chequeos como:

- Validar que el movimiento esté dentro de los límites del tablero.
- Calcular el índice `board[]` de destino.
- Determinar si un jugador debe ser bloqueado (EOF en pipe o falta de movimientos válidos).

Esto sigue la filosofía de que dentro de la zona crítica se realice sólo lo estrictamente necesario.

1.6. Modularización

Se desarrollaron librerías específicas:

- **shmlib** → funciones para creación y cierre de memorias compartidas.
- **playerslib** → funciones para la creación y manejo de variables relacionadas a los players
- **gamelib** → funciones relacionadas a la lógica del juego.

---

2. Instrucciones de Compilación y Ejecución

2.1. Compilación

El proyecto incluye un **Makefile**. Solo es necesario ejecutar:

```
make
```

Si se compila manualmente, debe usarse el estándar **C99**.

2.2. Ejecución

El máster debe recibir al menos un **player** mediante la flag `-p`.
Si no se especifican otros parámetros, se usan valores por defecto.

Ejemplo de ejecución:

```
./master -p playerInteligente
```

Configuración estándar:

- `-w width` → ancho del tablero (mínimo y default: 10).
- `-h height` → alto del tablero (mínimo y default: 10).
- `-d delay` → delay en ms entre impresiones del estado (default: 200).
- `-t timeout` → timeout en segundos para movimientos válidos (default: 10).
- `-s seed` → semilla para generación del tablero (default: `time(NULL)`).
- `-v view` → ruta del binario de la vista (default: sin vista).
- `-p player1 player2 ...` → rutas a los binarios de jugadores (mínimo: 1, máximo: 9).

---

3. Limitaciones

- Las librerías no son muy flexibles y fueron adaptadas al máster.
- `shmlib` solo crea memorias con permisos `RDWR` y modo `0666`.
- `gamelib` funciona únicamente para tableros bidimensionales en arrays unidimensionales y movimientos en la forma `0–7`.

---

4. Problemas Encontrados

Durante el desarrollo surgieron varios **bugs** en sincronización, creación de procesos hijos y la lógica de *ChompChamps*.
Algunos ejemplos:

- Jugadores que saltaban de un borde del tablero a otro.
- Manejo incorrecto del *path* de los players.

Lo complejo fue identificar los errores, ya que el sistema tiene múltiples procesos en paralelo y los fallos ocurrían de forma no determinística, dificultando el debugging.

---

5. Citas de Fragmentos de Código

Para la memoria compartida y los semáforos se usaron como referencia:

- Manuales de Linux (`man sem_open`, `man mmap`).
- Ejemplos vistos en clase, especialmente los de sincronización Readers/Writers.

---

6. Herramientas Utilizadas

- **strace** → seguimiento de *syscalls* (pipes, semáforos, deadlocks).
- **gdb** → depuración de máster, vista y jugadores.
- **gcc** → compilador y debugger.
- **valgrind** → detección de *memory leaks*.