TP1: Sistemas Operativos

Chomp Champs

Grupo 15

Agostino Alfieri (64607)

aalfieri@itba.edu.ar

Lola Diaz Varela (62792)

ldiazvarela@itba.edu.ar

Lucas Di Candia (63212)

ldicandia@itba.edu.ar

72.11 Sistemas Operativos

Primer cuatrimestre 2025

Decisiones Tomadas Durante el Desarrollo:

1. Arquitectura Basada en Procesos

Decisión: Dividir el programa en múltiples procesos independientes: master, players y view.

Motivación: Facilitar la concurrencia y la separación de responsabilidades:

- master: Coordina la lógica del juego y sincroniza los procesos.
- players: Ejecutan movimientos aleatorios y comunican sus acciones al master.
- view: Muestra el estado del juego en tiempo real.

Impacto: Permite una arquitectura modular y escalable, pero introduce complejidad en la sincronización.

2. Uso de Memoria Compartida

Decisión: Utilizar memoria compartida (shm_open y mmap) para almacenar el estado del juego y los semáforos de sincronización.

Motivación: Compartir datos entre procesos de manera eficiente sin necesidad de comunicación explícita constante.

Impacto: Reduce la sobrecarga de comunicación, pero requiere un manejo cuidadoso de la concurrencia.

3. Sincronización con Semáforos

Decisión: Implementar semáforos (sem_t) para coordinar el acceso a la memoria compartida y sincronizar los procesos.

Motivación: Evitar condiciones de carrera y garantizar la consistencia del estado del juego.

Impacto: Asegura un comportamiento predecible, pero introduce complejidad en el manejo de sincronización.

4. Diseño Modular

Decisión: Dividir el código en múltiples archivos y módulos (game_logic, shmemory, arg_parser, etc.).

Motivación: Mejorar la organización, la legibilidad y la mantenibilidad del código.

Impacto: Facilita la colaboración y el desarrollo incremental, pero requiere una estructura clara de dependencias.

5. Uso de Pipes para Comunicación

Decisión: Utilizar pipes para que los players envíen sus movimientos al master.

Motivación: Proporcionar un canal de comunicación unidireccional simple y eficiente.

Impacto: Simplifica la comunicación, pero requiere un manejo cuidadoso de los descriptores de archivo.

6. Manejo de Errores

Decisión: Incluir verificaciones de errores en operaciones críticas (e.g., shm_open, fork, pipe).

Motivación: Evitar fallos silenciosos y proporcionar mensajes de error claros.

Impacto: Mejora la robustez del programa, pero puede hacer que el código sea más verboso.

7. Uso de Colores y Formato en la Consola

Decisión: Utilizar secuencias ANSI para colorear la salida de la consola y mejorar la experiencia visual.

Motivación: Hacer que la interfaz sea más atractiva y fácil de interpretar.

Impacto: Mejora la experiencia del usuario, pero puede no ser compatible con todas las terminales.

8. Control de Procesos:

Se optó por usar fork para crear procesos independientes para los jugadores y la vista, y execl para ejecutar los binarios correspondientes.

9. Validación de Argumentos:

Se implementó un parser de argumentos (arg_parser) para validar los parámetros de entrada, como dimensiones del tablero, tiempo de espera, y rutas de los ejecutables.

Instrucciones de Compilación y Ejecución

Compilación:

Para compilar el proyecto, utiliza el comando:

make all

Ejecución:

Para ejecutar el juego con parámetros predeterminados:

make run

Para probar con diferentes configuraciones:

make test

Para ejecutar con el binario de Chomp Champs provisto por la cátedra:

make run_chomp

Limitaciones

1. Número Máximo de Jugadores:

El juego soporta un máximo de 9 jugadores debido a la definición de MAX_PLAYERS.

2. Tamaño del Tablero:

El tamaño mínimo del tablero es de 10x10, lo cual está validado en el parser de argumentos.

3. Sincronización:

Aunque se utilizan semáforos para evitar condiciones de carrera, el diseño actual podría no escalar bien con un número muy alto de procesos.

4. Movimientos Aleatorios:

Los jugadores realizan movimientos aleatorios, lo que puede no ser óptimo para estrategias avanzadas.

Problemas Encontrados y Soluciones

1. Condiciones de Carrera:

Inicialmente, se detectaron condiciones de carrera al acceder al estado del juego. Esto se solucionó utilizando semáforos para proteger las secciones críticas.

2. Bloqueo de Jugadores:

Algunos jugadores quedaban bloqueados al no tener movimientos válidos. Se implementó la función check_player_timeouts para detectar y manejar estos casos.

3. Errores en Pipes:

Hubo problemas al leer movimientos de los jugadores a través de pipes. Se solucionó verificando el retorno de read y marcando al jugador como bloqueado si la lectura fallaba.

4. Fugas de Memoria:

Se detectaron fugas de memoria en pruebas con valgrind. Se corrigieron liberando correctamente la memoria compartida y los recursos al final del programa.

Citas de Fragmentos de Código Reutilizados

1. Memoria Compartida y Semáforos:

El archivo provisto por la cátedra de sincronización sirvió como referencia para la implementación de memoria compartida y semáforos.

2. Parser de Argumentos:

La estructura del parser de argumentos se basó en ejemplos de uso de getopt disponibles en la documentación oficial de GNU.

3. Ordenamiento de Puntajes:

El algoritmo de ordenamiento utilizado en view.c para mostrar los puntajes se basó en el método de burbuja, ampliamente documentado en recursos educativos.

Conclusión

El proyecto Chomp Champs implementa un juego funcional que utiliza conceptos avanzados de sistemas operativos, como memoria compartida, semáforos y procesos. A pesar de las limitaciones, se logró un diseño modular y eficiente que cumple con los requisitos planteados.