Trabajo Práctico N°1 - Grupo 14

Andrés Cortese, acortese@itba.edu.ar 64612

Tomás Jerónimo Esquivel, tesquivel@itba.edu.ar 64756

Miguel David Taylor

Introducción

Este informe presenta el desarrollo del Trabajo Práctico Nº 1 de la materia Sistemas Operativos del ITBA, cuyo objetivo principal fue implementar un sistema basado en IPC (Inter Process Communication) que permitiera correr un juego multijugador denominado ChompChamps. Este juego pertenece al género "snake" y está diseñado para funcionar con tres procesos independientes: el proceso máster, que organiza y controla la ejecución del juego; el proceso jugador, que representa la lógica de cada participante del juego; y el proceso vista, encargado de visualizar el estado actual del tablero en la consola.

Objetivos del Proyecto

El trabajo tuvo como finalidad consolidar conocimientos sobre comunicación entre procesos en un entorno POSIX. Para ello, se propuso un diseño en el que la interacción entre los diferentes procesos se lograra mediante memoria compartida, semáforos y pipes. Entre los objetivos específicos se encuentran:

- Implementar la sincronización correcta entre lectores y escritores sin inanición del escritor.
- Gestionar múltiples procesos de forma concurrente y segura.
- Lograr una visualización en tiempo real del estado del juego.
- Ejecutar el juego de forma determinística y sin condiciones de carrera.

Desarrollo del Proyecto

Diseño General del Sistema

El sistema está compuesto por tres binarios: master, player y view. El proceso máster es el responsable de inicializar los recursos compartidos, lanzar los procesos de los jugadores y la vista, recibir y procesar solicitudes de movimiento, y determinar la finalización del juego. Cada jugador, representado por un proceso player, analiza el tablero en busca de movimientos válidos y comunica sus intenciones mediante pipes. La vista, por su parte, imprime periódicamente el estado del juego al recibir la orden del máster.

Los procesos comparten dos memorias: /game_state, que almacena el estado del juego (tablero, jugadores, puntajes, etc.), y /game_sync, que contiene los semáforos para

sincronización. Para evitar condiciones de carrera, se implementó el patrón de sincronización de lectores y escritores sin inanición del escritor, utilizando semáforos.

Proceso Máster

El máster inicia verificando y normalizando los parámetros de ejecución. Luego crea e inicializa las memorias compartidas, distribuye a los jugadores sobre el tablero (en posiciones determinísticas que aseguran movilidad inicial), y genera recompensas aleatorias para cada celda.

La función principal del máster es esperar movimientos de los jugadores utilizando select() para manejar múltiples pipes de entrada. Esto le permite decidir, en tiempo real, cuál jugador tiene una solicitud pendiente. Si el movimiento recibido es válido, se actualiza el estado del juego y se otorga la recompensa correspondiente. En caso contrario, el movimiento es contabilizado como inválido. Además, se coordina con la vista para mostrar el tablero actualizado tras cada acción.

Proceso Jugador

Cada jugador identifica su posición en la estructura compartida mediante su PID. Una vez reconocido, entra en un bucle de ejecución en el que intenta encontrar celdas adyacentes libres. Al detectar una celda válida, envía la dirección al máster escribiendo en stdout, que ha sido redirigido por el máster al pipe correspondiente.

Si el jugador no encuentra movimientos válidos, se declara bloqueado y abandona la ejecución. La lógica implementada evita espera activa mediante retardos y lectura sincronizada del estado compartido.

Proceso Vista

La vista espera señales del máster mediante semáforos. Una vez notificada, accede al estado compartido como lector y actualiza la representación visual del tablero y la información de cada jugador. La vista borra la pantalla y dibuja el estado actual utilizando secuencias ANSI para una experiencia fluida y clara en consola.

La visualización incluye información completa sobre el tablero (recompensas y capturas), así como puntajes, movimientos válidos/ inválidos y posiciones de cada jugador.

Estructura del Código

El código fuente está organizado de forma modular, con estructuras de datos compartidas claramente definidas. La estructura game_state_t representa el estado global del juego,

mientras que sync_t maneja los semáforos. Se definieron funciones auxiliares para simplificar tareas comunes como la inicialización de memoria compartida, la verificación de parámetros y la impresión del tablero.

Se utilizaron arrays globales para representar los desplazamientos posibles en las 8 direcciones (estilo Snake) y se manejó cuidadosamente la liberación de recursos en todos los procesos.

Compilación y Ejecución

Para compilar los binarios se incluye un Makefile que genera los tres ejecutables: master, player y view. Se recomienda usar la imagen Docker provista por la cátedra:

docker pull agodio/itba-so-multi-platform:3.0

Una vez compilado, el sistema se ejecuta desde el máster, especificando los parámetros deseados. Por ejemplo:

./master -w 15 -h 15 -d 300 -t 10 -s 123 -v ./view -p ./player ./player ./player

Limitaciones

La lógica de los jugadores está basada en una heurística muy simple: moverse a la primera celda libre disponible. Esto puede llevar a decisiones subóptimas. No se implementó ningún tipo de estrategia avanzada ni detección de obstáculos a futuro.

El tablero se imprime por consola sin capacidades gráficas extendidas, lo cual puede dificultar la visualización en tableros muy grandes. Tampoco se permite reiniciar una partida sin reiniciar completamente el sistema.

Problemas Encontrados

Uno de los desafíos principales fue lograr una sincronización correcta entre los procesos. La implementación del patrón lectores/escritores se realizó cuidadosamente para evitar condiciones de carrera y garantizar que el máster pudiera escribir sin sufrir inanición. También se debió resolver la multiplexación de pipes para permitir que el máster escuche a todos los jugadores de forma eficiente.

Otro punto crítico fue la correcta identificación de cada jugador dentro de la memoria compartida. Para ello se utilizó el PID de cada proceso, garantizando que cada jugador pudiera acceder a su propia estructura.

Conclusión

El desarrollo de ChompChamps permitió aplicar múltiples conceptos avanzados de sistemas operativos en un entorno práctico, incluyendo manejo de procesos, sincronización, comunicación y control de concurrencia. El sistema resultante es robusto, funcional y cumple con los requisitos planteados.

Como mejora futura, se podría optimizar la inteligencia de los jugadores, ofrecer una vista más rica o permitir modos de juego personalizables. No obstante, el objetivo académico y técnico fue cumplido con éxito.