Sistemas Operativos

Trabajo Práctico Nº 1:

Inter Process Communication

Grupo 18

- 62013 - Agustín Julián Brunero
  abrunero@itba.edu.ar
- 64399 - Ignacio Ferrero Puértolas
  iferreropuertolas@itba.edu.ar
- 65624 - Nicolás Pérez Stefan
  nperezstefan@itba.edu.ar

Fecha de entrega: 15/9/25

1. Introducción

El presente informe corresponde al **Trabajo Práctico Nº1 de Sistemas Operativos**, cuyo objetivo es **aprender y aplicar distintos mecanismos de comunicación entre procesos (IPC) en un sistema POSIX.**

Para lograrlo, se implementó el juego **ChompChamps**, un juego multijugador del género *snake* en el cual los jugadores se desplazan en un tablero rectangular para obtener recompensas.

2. Descripción del Juego

2.1 Dinámica

El juego no es por turnos: un jugador puede moverse varias veces seguidas si los demás no realizan acciones. Esto fomenta la rapidez y la estrategia para evitar bloqueos.

2.2 Movimientos

Los jugadores pueden moverse una celda por vez en cualquiera de las 8 direcciones posibles. Estos movimientos tienen como restricciones no cruzar límites del tablero, no ocupar celdas ya tomadas y no superponerse con otros jugadores. En caso de violar alguna de estas restricciones, el movimiento se contabiliza como invalido.

2.3 Recompensas

Cada celda libre contiene entre 1 y 9 puntos. Los jugadores comienzan con puntaje 0 y suman el valor de las celdas visitadas.

2.4 Fin del juego y ganador

El juego termina cuando ningún jugador puede moverse o se alcanza un tiempo límite sin movimientos válidos. Gana el jugador con mayor puntaje. En caso de empate, se aplican criterios de eficiencia (menos movimientos válidos, luego menos inválidos).

3. Requerimientos del Trabajo

El proyecto se divide en tres ejecutables en C, cada uno con una función específica:

- **Máster**: se encarga de crear las memorias compartidas, gestionar los mecanismos de comunicación y coordinar tanto a los jugadores como a la vista.

- **Vista**: tiene como responsabilidad imprimir en pantalla el estado del tablero en cada actualización recibida.

- **Jugador**: genera de manera automática las solicitudes de movimiento, interactuando con el máster a través de los canales de comunicación definidos.

Para la interacción entre los procesos se implementaron encabezados con estructuras compartidas entre los distintos ejecutables. Asimismo, se emplearon pipes anónimos, memorias compartidas y semáforos, asegurando una correcta comunicación y sincronización entre los componentes.

Un aspecto central del proyecto fue garantizar la limpieza adecuada de los recursos utilizados, evitando deadlocks, race conditions o memory leaks, lo que asegura un funcionamiento robusto y confiable del sistema.

4. Proceso Máster

4.1 Parámetros principales

- `-w`, `-h`: dimensiones del tablero.
- `-d`: delay entre impresiones.
- `-t`: timeout de inactividad.
- `-s`: semilla aleatoria.
- `-v`: ruta de la vista.
- `-p`: rutas de los jugadores.

4.2 Responsabilidades implementadas

- Creación de memoria compartida para estado y sincronización.
- Generación de procesos hijos (jugadores y vista).
- Recepción y validación de movimientos mediante pipes.
- Política de atención round-robin.

- Registro y notificación del estado a la vista.
- Control del tiempo de inactividad y finalización de partida.

5. Proceso Vista

Responsabilidades

- Conectarse a la memoria compartida.
- Imprimir en pantalla el tablero, posiciones de jugadores y puntajes.
- Notificar al máster tras imprimir.

6. Proceso Jugador

Responsabilidades

- Conectarse a la memoria compartida.
- Consultar el estado y decidir un movimiento.
- Enviar solicitud al máster por pipe.
- Esperar confirmación antes de enviar otro movimiento.

7. Estructuras y Sincronización

- **Memoria compartida `/game_state`**: guarda jugadores, tablero y estado de partida.
- **Memoria compartida `/game_sync`**: administra semáforos para coordinación.
- **Semáforos**: empleados para sincronizar máster-vista y máster-jugadores (incluyendo el problema de lectores/escritores).
- **Pipes**: usados para envío de movimientos de jugadores hacia el máster.
- **Archivo de encabezado `defs.h`**: para definiciones y estructuras que se comparten entre los ejecutables

8. Decisiones de Diseño

Para el desarrollo del trabajo se creó la librería `ChompChampsUtils.h` en la que implementamos todas las funciones que se requerían en más de un proceso. Funciones

para abrir/crear la memoria compartida, chequeos de alocaciones de memoria, obtener datos del juego y los jugadores, etc.

Las flags al momento de correr el programa se resolvieron correctamente, pueden recibirse en distinto orden y al no encontrarlas se utiliza el valor default del enunciado, a excepción de el parámetro `-p`, que obligatoriamente debe estar al final.

Finalmente, formateamos el código con clang-format para tener una mejor legibilidad y presentación.

9. Dificultades Encontradas

Durante el desarrollo del trabajo nos topamos con varios obstáculos técnicos que requirieron tiempo y prueba-error para resolver. La creación de las memorias compartidas resultó más compleja de lo esperado: comprender la configuración correcta, permisos y el manejo de tamaños y offsets fue necesario para evitar lecturas/escrituras erróneas entre procesos. También, nos demoraron notoriamente los errores al querer cerrarlas correctamente.

En la implementación de los semáforos, encontramos la mayor dificultad del trabajo, comprender el flujo de comunicación y el orden correcto de los wait/post para evitar condiciones de carrera y bloqueos inesperados, fue en lo que más tiempo invertimos.

La implementación de pipes presentó problemas similares: definirlos y enlazarlos adecuadamente entre procesos no triviales exigió atención al orden de apertura y cierre de descriptores para que los datos circulen correctamente. Además, la comunicación con el master de la cátedra fue fuente de fricción: la forma de pasar parámetros y la escasa retroalimentación sobre los errores complicaron la localización de fallos en nuestro código y cómo solucionarlos.

10. Limitaciones

El trabajo presenta algunas limitaciones que condicionaron el alcance de las funcionalidades previstas.

No pudimos ejecutar nuestro programa utilizando el master de la cátedra, a pesar de haber dedicado un día extra posterior a la fecha de entrega con el objetivo de lograrlo. Esto impidió validar la compatibilidad y el comportamiento del sistema en el entorno oficial.

Tampoco se implementaron múltiples algoritmos para los jugadores, por lo que la variedad estratégica y las pruebas comparativas quedaron restringidas.

Por último, la vista se limita a tableros de tamaño pequeño y dependiendo del tamaño de la consola. Es necesario correr el programa en pantalla completa y no superar los 20-25 cuadros de alto y 55-60 cuadros de ancho para que se muestre correctamente. Aún así, diseñamos una representación por terminal con colores y caracteres que mejora la legibilidad y la experiencia de usuario.

11. Instrucciones de Compilación y Ejecución

- Compilar con:

  `make`

  Los binarios se generan en la carpeta `/bin`

- Ejecutar con:

  `./bin/master -w width -h height -d delay -t timeout -s seed -v ./bin/view -p ./bin/player ./bin/player` (min: 1 max: 9)

  Ejecución de ejemplo más sencilla (con 3 jugadores):

  `./run 20 20 200 20 123`

- En caso de ser necesario, usar la imagen Docker provista por la cátedra:

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

12. Conclusiones

El trabajo nos permitió comprender y aplicar distintos mecanismos de comunicación entre procesos (IPC), tales como pipes, memoria compartida y semáforos. A lo largo de su desarrollo, fue necesario enfrentar y resolver diversos problemas de sincronización y concurrencia, lo que brindó una experiencia práctica en el manejo de situaciones críticas propias de la programación concurrente. Además, la implementación nos dio la oportunidad de experimentar con un diseño modular y la coordinación de múltiples procesos, fortaleciendo así tanto la organización del código como la comprensión de arquitecturas distribuidas en un mismo sistema.