Trabajo Práctico Nº 1

Inter Process Communication (IPC)

Primer cuatrimestre de 2025

Comisión S - Grupo 4

Integrantes:

Cuneo Gima, Matias	64492
Buela, Mateo	64680
Prado, Nahuel Ignacio	64276

Introducción

El siguiente informe detalla el desarrollo del trabajo práctico número uno de la materia de sistemas operativos. Este trata de los métodos de comunicación entre procesos en sistemas POSIX. Para demostrar su funcionamiento, se creó un juego llamado Chomp Champs que consiste en que jugadores avanzan por una matriz con puntajes en cada casilla que una vez que se pasa por cada una se le suman los puntos al jugador en particular y se ocupa esa posición. Se distinguen tres procesos, el del comportamiento del jugador, el encargado de la vista y, por último, el maestro encargado de gestionar el juego.

Desarrollo

A continuación, se explican las decisiones tomadas durante el desarrollo.

Decisiones tomadas durante el desarrollo.

En primer lugar, la validación de parámetros recibidos del máster. Se optó por aceptar todos los aquellos explicitados en la consigna del trabajo y cualquier otro argumento se ignora y si hay algún error se toman los valores default definidos en la consigna. Entonces, por ejemplo para la siguiente llamada:

```
:~# ./bin/master -p player1 player2 -w 15 -h -10 player3 -d 0 -s semilla player4 -v view
```

Llamada ejemplo

Se interpretan los siguientes argumentos:

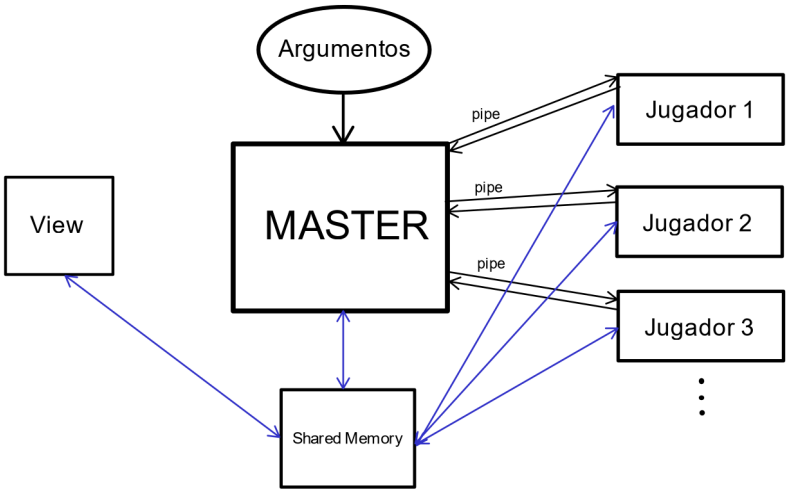
```
width: 15
height: 10
delay: 200
timeout: 10
seed: 1744583077
view: view
num_players: 2
  player1
  player2
```

Interpretación de los argumentos mandados en la llamada ejemplo anterior

Luego, para la comunicación entre procesos se siguió la indicada en la consigna. Esta sigue las siguientes especificaciones:

- El master crea pipes anónimos para recibir los movimientos de los jugadores donde estos tienen el extremo de escritura vinculado al file descriptor 1.
- El master crea memorias compartidas que contienen los detalles de la partida y tanto el jugador como la vista tienen permisos de lectura para ver su estado.

- El master también crea la memoria compartida para los semáforos que utilizan tanto la vista como los players y el master.



Estructura de comunicación entre procesos

Además, aunque el enunciado menciona basarse en el sistema POSIX para IPC, se consideró correcto implementar una alternativa para los usuarios que no dispongan del mismo, de manera se pueden utilizar las funciones de System V.

Otra decisión tomada fue para las posiciones iniciales de los jugadores en el tablero. Esta se definió con el siguiente criterio:

- Primero se traza una elipse que depende de la altura y ancho del tablero.
- Luego esta se le dividen los trescientos sesenta ángulos por la cantidad de jugadores.
- Finalmente, se ubican los jugadores arrancando por el primero en la parte inferior y el resto en sentido antihorario en intervalos iguales.

Esto resulta en que los jugadores tengan todos las mismas condiciones iniciales y, por ende, las mismas probabilidades de victoria.

Luego, es requisito que se implementara una mecánica de round-robin. Esta consta de una mecánica de emparejamiento de las condiciones de los jugadores. Para ello se implementa el siguiente algoritmo:

Se cicla a través de los pipes leyendo de aquellos en los que se pueda una jugada. En caso de que no haya mandado una jugada se sigue por el arreglo de pipes. En caso contrario, se lee esa jugada y se actualiza el estado del juego y se continua por el siguiente jugador.

Por otro lado, se pide que se renombren los campos en las estructuras preestablecidas en la consigna. Estas finalmente resultaron siendo así:

```c
typedef struct {
    char name_player[MAX_PLAYER_LENGTH]; // Nombre del jugador
    unsigned int points; // Puntaje
    unsigned int amount_invalid_movements; // Cantidad de solicitudes de movimientos inválidas realizadas
    unsigned int amount_valid_movements; // Cantidad de solicitudes de movimientos válidas realizadas
    unsigned short x, y; // Coordenadas x e y en el tablero
    pid_t pid; // Identificador de proceso
    bool cant_move; // Indica si el jugador no tiene movimientos válidos disponibles
} Tplayer_state;
```

Estructura de los jugadores

```c
typedef struct {
  unsigned short width; // Ancho del tablero
  unsigned short height; // Alto del tablero
  unsigned int amount_players; // Cantidad de jugadores
  Tplayer_state players[MAX_NUM_PLAYERS]; // Lista de jugadores
  bool can_end; // Indica si el juego no se ha terminado
  int board[]; // Puntero al comienzo del tablero. fila-0, fila-1, ..., fila-n-1
} Tgame_state;
```

Estructura del tablero y estado de juego

```c
typedef struct {
  sem_t show_needed;  // Indica a la vista que hay cambios por imprimir
  sem_t show_done;  // Indica al máster que la vista terminó de imprimir
  sem_t master_mutex;  // Evita inanición del máster al acceder al estado
  sem_t game_state_mutex;  // Protege el estado del juego contra modificaciones concurrentes
  sem_t player_read_count_mutex;  // Protege la variable 'player_reading_status'
  unsigned int player_reading_status;  // Cantidad de jugadores leyendo el estado
} Tgame_sync;
```

Estructura que engloba a los semáforos usados

Finalmente, la última decisión tomada en la implementación del trabajo fue la correspondiente a la estructuración de archivos. Esta está constituida de la siguiente manera:

- Dos carpetas principales: src y include
- En src se encuentran los tres archivos de código C correspondientes a master, un jugador y la view y, también, la carpeta de librerías usadas.
- En esta carpeta se encuentran los archivos de código C de las librerías como la encargada de generar números random, la de procesado de argumentos del master, las de manejo de memoria compartida y semáforos, la de creación de procesos y sus pipes y la de la lógica del juego.
- Luego, en la carpeta de include, se encuentran los archivos encabezados de las librerías anteriormente mencionadas y el archivo con las estructuras usadas, algunos tipos de datos definidos y los códigos de error.

Se priorizo ocultamiento en donde sea posible como la correcta modularización en todo momento.

Instrucciones de compilación y ejecución.

Para la correcta ejecución del proyecto se deben seguir los siguientes pasos:

- Primero se clona el repositorio de GitHub utilizando el comando $> git clone <repositorio>
- Luego se inicializa el contenedor de docker corriendo $> ./dockerRun.sh
- Ya dentro del contenedor se puede compilar el proyecto con el makefile ubicado en la root ejecutando $> make
- Los binarios generados se guardan en la carpeta /root/bin, en especial el ejecutable principal llamado master. Ahora simplemente con el comando $> ./bin/master [parámetros] corre el proyecto.

Limitaciones.

Una limitación que se presenta en nuestro proyecto corresponde a la velocidad con la que manda jugadas cada jugador. Dado el caso de que uno escriba muchas jugadas por el pipe, puede llegar a ocasionarse un error. Esto logra trabar la ejecución del proceso correspondiente a ese jugador lo cual, probablemente no sea la intención de este.

A su vez, un jugador puede no hacer buen uso de los semáforos. En ciertos casos, esto puede llevar a un comportamiento egoísta de un jugador o a que este acapare mucho tiempo de ejecución.

También, cada proceso es responsable de cerrar sus memorias compartidas y finalizar su ejecución.

Problemas encontrados durante el desarrollo y cómo se solucionaron.

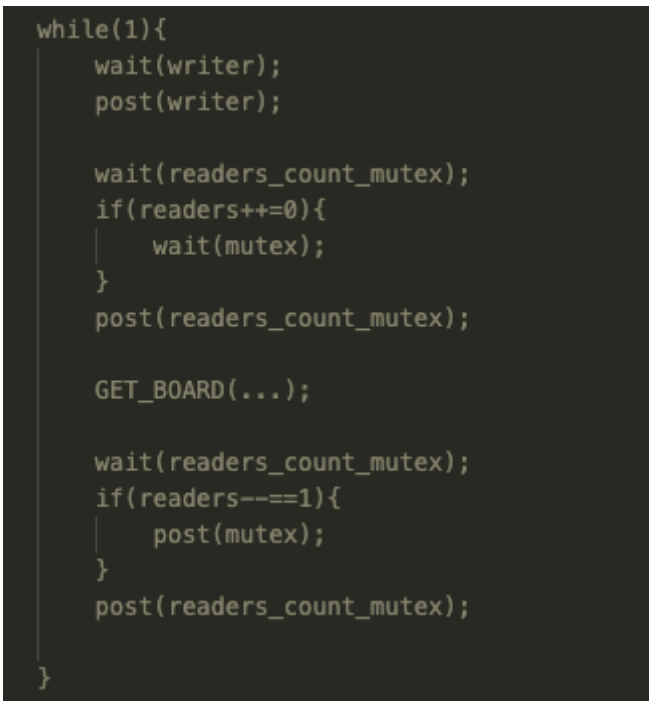
En primera instancia, se ocasionaron dificultades entendiendo la funcionalidad de cada semáforo en la estructura dada por el enunciado. Fue particularmente complejo dado que el binario de master provisto obviamente no muestra su funcionamiento interno. Tras investigación propia y ayuda de la cátedra, se terminó entendiendo correctamente cada uno.

Otro problema hallado, es que a la hora de trabajar con tiempos (por ejemplo, el uso de un sleep) distintas arquitecturas tienen distintos comportamientos. En particular, se notó que Ubuntu 24 necesita sleeps “más largos” que una Mac. Luego resultó ser innecesario el uso de estas funciones, por lo que el error no tuvo más impacto.

Finalmente, también fue un desafío entender nuevos conceptos únicos al trabajo práctico como la manera de implementar un select o como saber la terminación de un proceso. Esto se solucionó por medio de leer la documentación como el comando man u otras fuentes.

Citas de fragmentos de código reutilizados de otras fuentes.

Para entender el comportamiento de los jugadores, se usó de referencia el código mostrado en clase. A continuación se lo puede encontrar:



```
while(1){
    wait(writer);
    post(writer);

    wait(readers_count_mutex);
    if(readers++=0){
        wait(mutex);
    }
    post(readers_count_mutex);

    GET_BOARD(...);

    wait(readers_count_mutex);
    if(readers--==1){
        post(mutex);
    }
    post(readers_count_mutex);

}
```

Codigo referencia para player

Conclusión

En conclusión, se cree haber realizado el trabajo correctamente sin demasiadas complicaciones. A su vez, se halla la consigna entretenida y dinámica al tratarse de un juego. Finalmente, se agradece haber tenido la oportunidad de aprender de los métodos de comunicación entre procesos.