Sistemas Operativos

Trabajo Práctico Nº 1: Inter Process Communication

Introducción

El trabajo práctico consiste en aprender a utilizar los distintos tipos de IPCs presentes en un sistema POSIX. Para ello se implementará el juego ChompChamps.

ChompChamps

ChompChamps es un juego multijugador del género snake. El tablero de juego es una grilla rectangular donde cada celda contiene recompensas. Al inicio, los jugadores son ubicados en diferentes posiciones y a medida que se desplazan, obtienen las recompensas de las celdas que visitan.

Dinámica

ChompChamps no es un juego por turnos. Se podría dar el caso que un mismo jugador se mueva múltiples veces consecutivas, sin embargo, esto sólo es posible si el resto de jugadores no solicitan movimientos. Esta característica hace de ChompChamps un juego más dinámico y motiva a los jugadores a moverse rápidamente, evitando quedar bloqueados.

Movimientos

Los jugadores pueden moverse de 1 celda a la vez, en cualquiera de las 8 direcciones posibles, sin embargo, no pueden cruzar los límites del tablero, (no aparecen del otro lado) y tampoco pueden pasar por encima de otros jugadores ni por encima de sí mismos. Por lo tanto, solo pueden moverse a celdas adyacentes libres. Una vez que una celda es capturada, permanece en ese estado hasta la finalización del juego.

Cualquier intento de movimiento inválido simplemente se ignora, pero se registra para posibles criterios de desempate,

Recompensas

Los jugadores inician con un puntaje de 0. Las celdas donde se ubican inicialmente no otorgan recompensas. Las recompensas de las celdas que visitan se suman a sus respectivos puntajes y las mismas varían entre 1 y 9.

Ganador

El juego termina cuando ningún jugador puede moverse o transcurre una cierta cantidad de tiempo sin movimientos válidos. El ganador es aquel jugador con el mayor puntaje al finalizar el juego. En caso de empate se elige aquel con la menor cantidad de movimientos válidos realizados. Esto es un indicador de la eficiencia de sus movimientos. Si el empate persiste, se elige aquel con la

menor cantidad de movimientos inválidos. Finalmente se establece un empate si todos estos parámetros son iguales.

Requerimientos

Desarrollar en lenguaje C, la lógica, comunicación, interfaz e inteligencia artificial de ChompChamps en 3 binarios diferentes: máster, vista, y jugador.

Deben implementar el 100% de la funcionalidad indicada, respetando ciertas decisiones de diseño ya establecidas. Para ello, junto con este enunciado se provee un binario que implementa la funcionalidad del máster. La vista y jugador que implementen deberán interactuar con este máster de forma correcta, además, el máster que implementen deberá comportarse de la misma manera que el provisto, con ciertas excepciones.

Proceso Máster

Parámetros

A continuación se listan los parámetros que acepta el máster. Los parámetros entre corchetes son opcionales y tienen un valor por defecto.

- [-w width]: Ancho del tablero. Default y mínimo: 10
- [-h height]: Alto del tablero. Default y mínimo: 10
- [-d delay]: milisegundos que espera el máster cada vez que se imprime el estado. Default: 200
- [-t timeout]: Timeout en segundos para recibir solicitudes de movimientos válidos. Default: 10
- [-s seed]: Semilla utilizada para la generación del tablero. Default: time(NULL)
- [-v view]: Ruta del binario de la vista. Default: Sin vista.
- -p player1 player2: Ruta/s de los binarios de los jugadores. Mínimo: 1, Máximo: 9.

Responsabilidades

- Crear e inicializar 2 memorias compartidas, detalladas más adelante.
- Crear los canales de comunicación para recibir solicitudes de movimientos de los jugadores.
- Crear los procesos de los jugadores y la vista. Ambos procesos reciben como parámetros el ancho y el alto del tablero.
- Distribuir los jugadores en el tablero. No es necesario que sea igual al binario provisto pero deberá ser determinístico y ofrecerle un márgen de movimiento similar a cada jugador.
- Recibir solicitudes de movimientos de los jugadores, validarlas y modificar el estado acorde a las mismas en caso de serlo.
- La selección del siguiente jugador a atender deberá seguir la política round-robin, entre los jugadores con solicitudes pendientes por atender.
- Tras un cambio en el estado, esperar a que la vista termine de imprimir y luego respetar el tiempo de espera configurado.
- Registrar el paso del tiempo entre solicitudes de movimientos válidas. Si se supera el timeout configurado finaliza el juego. Este tiempo incluye la espera a la vista, es decir, que no tiene sentido establecer un delay mayor al timeout.
- Al finalizar el juego, registrarlo en la variable correspondiente del estado y esperar a que todos los jugadores y la vista finalicen su ejecución, imprimiendo el valor de retorno de cada proceso y en el caso de los jugadores, su puntaje.

Proceso Vista

Responsabilidades

- Recibir como parámetros el ancho y alto del tablero.
- Conectarse a ambas memorias compartidas
- Cuando el máster lo indique, la vista deberá imprimir el estado del juego. Todos los campos de las estructuras especificadas más adelante en la sección **Estado**, deberán ser visibles, excepto el identificador de cada proceso. Esto significa, por ejemplo, que el estado de cada celda del tablero (a quién le pertenece o qué recompensa tiene), como así también la posición actual de cada jugador y su puntaje deberán ser visibles.

Proceso Jugador

Responsabilidades

- Recibir como parámetros el ancho y alto del tablero.
- Conectarse a ambas memorias compartidas.
- Mientras no esté bloqueado, consultar el estado del tablero de forma sincronizada entre el máster y el resto de jugadores y enviar solicitudes de movimientos al máster.

Estructuras de datos y protocolos

El máster organiza la información como se muestra en las estructuras a continuación. El nombre de los campos de cada struct ha sido reemplazado por "?" y se agregó una breve descripción de su uso ya que asignarles nombres apropiados es parte de la evaluación. A su vez, el nombre de los typedef ha sido reemplazado por "XXX" o "YYY".

Estado

Las siguientes estructuras son almacenadas en una memoria compartida cuyo nombre es "`/game_state`" (shm_overview(7)).

```
typedef struct {
  char ?[16]; // Nombre del jugador
  unsigned int ?; // Puntaje
  unsigned int ?; // Cantidad de solicitudes de movimientos inválidas realizadas
  unsigned int ?; // Cantidad de solicitudes de movimientos válidas realizadas
  unsigned short ?, ?; // Coordenadas x e y en el tablero
  pid_t ?; // Identificador de proceso
  bool ?; // Indica si el jugador tiene movimientos válidos disponibles
} XXX;

typedef struct {
  unsigned short ?; // Ancho del tablero
  unsigned short ?; // Alto del tablero
  unsigned int ?; // Cantidad de jugadores
  XXX ?[9]; // Lista de jugadores
```

```
  bool ?; // Indica si el juego se ha terminado
  int ?[]; // Puntero al comienzo del tablero. fila-0, fila-1, ..., fila-n-1
} YYY;
```

Las coordenadas x e y tienen como origen (0, 0) la esquina superior izquierda del tablero. Es decir, que el primer `int` del tablero corresponde a (0, 0).

Cada celda del tablero contiene un `int` entre -8 y 9. Los valores entre 1 y 9 corresponden a celdas libres y el valor es su recompensa. Los valores entre -8 y 0 corresponden a celdas capturadas y el valor de la celda es -id, donde id es el índice correspondiente al jugador dentro de la lista de jugadores. Por ejemplo, si una celda contiene el valor -3, significa que pertenece al jugador almacenado en el índice 3 de la lista de jugadores del struct `YYY`.

Sincronización

Para la sincronización entre los procesos involucrados, se utilizan los siguientes semáforos anónimos (sem_overview(7)). En este caso, los nombres se reemplazaron por letras mayúsculas ya que es necesario identificarlos para explicar su uso, sin embargo, asignarles nombres apropiados también es parte de la evaluación. Las siguiente estructura se almacena en una memoria compartida cuyo nombre es "`/game_sync`" (shm_overview(7)).

```
typedef struct {
    sem_t A; // Se usa para indicarle a la vista que hay cambios por imprimir
    sem_t B; // Se usa para indicarle al master que la vista terminó de imprimir
    sem_t C; // Mutex para evitar inanición del master al acceder al estado
    sem_t D; // Mutex para el estado del juego
    sem_t E; // Mutex para la siguiente variable
    unsigned int F; // Cantidad de jugadores leyendo el estado
} ZZZ;
```

La sincronización entre el máster y la vista involucra los semáforos A y B siguiendo un patrón simple de señalización bidireccional. La sincronización entre el máster y los jugadores es más compleja. Involucra los semáforos C, D y E y la variable F. En este caso, es necesario instanciar el conocido problema de sincronización denominado *lectores y escritores*, en particular, la versión que previene la inanición del escritor.

Solicitudes de movimientos

Las solicitudes de movimientos que envía el jugador corresponden a un `unsigned char` en el rango [0-7] que representan las 8 posibles direcciones, comenzando por 0 hacia arriba y avanzando en sentido horario. Cualquier solicitud de movimientos por fuera de este rango, como así también aquellas que sobrepasen los límites del tablero, o que correspondan a celdas que no están libres, serán contabilizadas como inválidas.

Estas solicitudes se envían por un pipe anónimo (pipe(7)) al máster. El máster garantiza que el extremo de escritura de este pipe esté asociado al descriptor de archivo 1. El máster deberá leer de múltiples pipes simultáneamente, para esto sugerimos el uso de select (select(2) y select_tut(2)).

La detección de end-of-file en el pipe correspondiente a un jugador, se registrará como que el jugador está bloqueado.

Consideraciones generales

- El sistema deberá estar libre de deadlocks, condiciones de carrera y espera activa.

- El proyecto deberá tener un Makefile para las tareas de compilación.
- Para el desarrollo deberán utilizar algún servicio de control de versiones desde el principio del desarrollo. No se admitirán repositorios sin un historial desde el comienzo del TP.
- El repositorio utilizado no deberá tener binarios ni archivos de prueba.
- La compilación con todos los warnings activados (-Wall) no deberá arrojar warnings.
- El análisis con Valgrind no deberá reportar problemas de memoria en ninguno de los procesos implicados desarrollados por ustedes.
- El análisis con PVS-studio no deberá reportar errores.

Informe

Se deberá presentar un informe en formato pdf, en el cual se desarrollen de forma breve, los siguientes puntos:

- Decisiones tomadas durante el desarrollo.
- Instrucciones de compilación y ejecución.
- Limitaciones.
- Problemas encontrados durante el desarrollo y cómo se solucionaron.
- Citas de fragmentos de código reutilizados de otras fuentes.

Entorno de desarrollo y ejecución

Es un requisito obligatorio tanto para el desarrollo y compilación como para la ejecución del sistema, utilizar la imagen provista por la cátedra. El binario provisto fue compilado en esta misma imagen.

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

Evaluación

La evaluación incluye y no se limita a los siguientes puntos:

- [1] Deadline.
- [4] Funcionalidad (Mandatorio).
- [3] Calidad de código.
- [1] Limpieza de recursos del sistema.
- [1] Informe.
- Defensa.

Entrega

La entrega será grupal y se habilitará la actividad “TP1” en el campus donde se podrá subir el material requerido. En caso de tener algún problema con la entrega en Campus, enviarlo por mail a los docentes.

Entregables: Link del repositorio especificando el hash del commit correspondiente a la entrega y el informe.

Defensa del trabajo práctico: Individual y obligatoria TBD.

Competencia

Al finalizar el trabajo práctico, los grupos que así lo deseen podrán participar de 2 competencias. Una en la que se votará por los procesos vista participantes y otra donde los jugadores desarrollados por cada grupo se enfrentarán en un mismo tablero. Los detalles de esta competencia serán publicados más adelante.