Sistemas Operativos

TP1: Inter-Process Communication

Integrantes:

Nicolás Kuyumciyan	55165
Magdalena Vega	55206
Lucas Casagrande	55302

IDEA

Nuestra idea consiste en un juego donde varios jugadores (2 a 4) cada uno manejando una víbora tienen que lograr derrotar a los demás jugadores, para esto tienen que lograr que la cabeza de los demás jugadores choque contra una parte del cuerpo de la víbora.

Manual de Operación

Para poder jugar al juego es necesario seguir los siguientes pasos:

1. Compilar el cliente/servidor/log utilizando *make sockets* o *make fifo*
2. Ejecutar el servidor y, preferiblemente el log
3. Es necesario cargar el archivo *config* con los parámetros de la conexión, ya sea la IP del servidor y el puerto al que quiere conectarse o la dirección del FIFO del servidor y un iniciador para el contador del servidor.
4. Ejecutar el cliente, indicando el número de lobby al que se quiere conectar (del 1 al 10)
5. Ingresar el nombre de usuario
6. De no existir el usuario el programa le indica que ingrese una contraseña para guardar el usuario
7. Si el usuario existe se requiere que el cliente ingrese correctamente la contraseña asociada a dicho usuario
8. Una vez conectado el jugador se va a encontrar en el lobby donde va a poder ver a los demás jugadores conectados
9. Cuando el jugador quiera empezar a jugar tiene que decir que está listo para empezar. Cuando todos los jugadores estén listos se empieza el juego (aunque sean menos de 4)
10. Durante el juego el jugador va a poder mover la víbora con las flechas del teclado.
11. Si el jugador pierde se le avisa en pantalla y va a poder seguir viendo la partida hasta que termine
12. Una vez que haya un ganador se actualizarán los puntajes de los jugadores

Asumpciones

- El cliente es capaz de generar el archivo de configuración, esto es, puede conseguir la IP del servidor en caso de sockets y el puerto.
- La única forma de cerrar el cliente de forma repentina es utilizando la SIGINT (CTRL + C), utilizando SIGSTOP o SIGKILL puede generar que en el caso de FIFO los archivos no sean eliminados y el Servidor no se dé cuenta que un cliente se desconectó.

Decisiones

- Para Manejar cada uno de los juegos decidimos que el servidor se *forkee*, generando que un servidor se encargue de manejar hasta 4 usuarios. Cada servidor es el encargado de ejecutar la lógica del juego y los clientes lo único que hacen es mostrar el estado del juego y enviar al servidor cualquier *feedback* por parte del jugador.
- Debido a que hay muchos servidores simultáneos se eligió SharedMemory para comunicarse con la DB, ya que de haber usado pipes cada servidor hijo tendría que comunicarse con el servidor padre para que este le envíe la información a la base, o tener un pipe por cada hijo que se genera. Para manejar la concurrencia con el proceso de la base de datos se utilizó un *mutex* con 2 variables de condición. El mutex bloquea que no haya más de un proceso intentando acceder a la base de datos. Las 2 variables de condición generan que la base de datos esté en estado de espera hasta que un pedido se hace, y luego es el que hizo el pedido el que espera hasta que la base de datos complete su tarea. Esto también evita que la base de datos esté utilizando el procesador debido al Busy Waiting que debería hacer.
- Para la implementación de FIFOs se decidió que el peer servidor cree un solo FIFO en donde recibe las nuevas conexiones, y el peer cliente cree 2 FIFOs uno donde se va a escribir y el otro donde lee, pasándole la dirección al peer servidor para que este se "conecte" a los FIFOs. Esto genera que para cada peer cliente, el peer servidor tenga una lectura y escritura independientes. También el cliente es el encargado de eliminar sus FIFOs cuando se cierra.
- Una vez que el "SuperServidor" se *forkea*, el hijo crea un *thread* que es el encargado de leer nuevas conexiones, mientras que el *thread* principal es el encargado de leer nuevos datos enviados por los clientes conectados. El *thread* que escucha nuevos clientes se ejecuta hasta que el juego haya empezado (ya que no pueden ingresar nuevos jugadores).
- Cuando el juego empieza nuevamente se crea un *thread*. Este *thread* es el encargado de leer la comunicación a la espera de que algún jugador presione una tecla, y actualiza la dirección de forma acorde. Mientras tanto el *thread* principal está encargado de la lógica del juego y de enviar periódicamente el estado del juego a los jugadores. Esto permite que se pueda aprovechar de mejor manera los núcleos del procesador y además delimita de forma clara las tareas que tiene cada *thread*.
- Para el *marshalling* y envío de datos se decidió que sea un String lo que se envíe. Dicho String está formado por un identificador en la primera posición que le indica al *unmarshalling* que tipo de dato es que representa ese String, permitiéndole volver a generar el tipo de dato que representa. Esto genera que en caso de tener que enviar otro tipo de dato diferente se tiene que crear una función especial que lo transforme, y tiene límite en la cantidad de estructuras diferentes debido a que usa un solo carácter para determinar el tipo, pero consideramos que es más que suficiente para lo que se requiere.
- Se decidió Sobrescribir la *signal* SIGINT para que en caso de que el usuario quiera cerrar el cliente de forma prematura los archivos de FIFOs sean eliminados y que le avise al servidor que el cliente se desconecta

- Dado que se pueden enviar datos varias veces antes de que el *peer* llame a *receiveData* se creó una wrapper para dicha función que la llama y se guarda en una estructura todos los datos recibidos (vendrían separados por un \0). Cada vez que se llama a esta wrapper devuelve datos almacenados en dicha estructura hasta que no le queden más, momento en el que vuelve a llamar a *receiveData*.
- Debido a que es un juego que requiere de múltiples clientes y personas la implementación de FIFOs y de sockets de forma local en la misma computadora resulta impráctica para jugar de forma correcta al juego. Lo recomendado es que se utilice la comunicación por sockets dentro de una LAN donde cada jugador tiene acceso a su propia computadora y teclado.