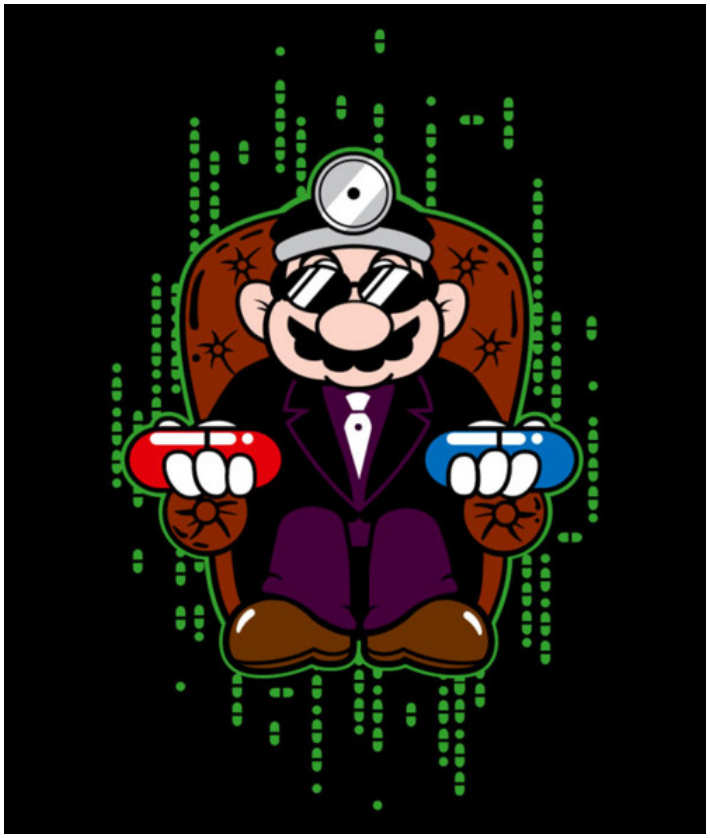
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Super Mario Proc RELOADED



Ingeniería en Sistemas de Información
Cátedra de Sistemas Operativos

- 2C2013 -
Versión 0.9.0.41c

Objetivos del Trabajo Práctico

El trabajo práctico está diseñado para que el alumno pueda mediante la ejercitación:

- Adquirir los conocimientos prácticos del uso y aplicación de un conjunto de servicios que ofrecen los sistemas operativos.
- Evaluar la factibilidad y ponderar las distintas soluciones para un mismo problema.
- Experimentar la problemática inherente a la concurrencia y su solución mediante la sincronización
- Comprender la importancia de una norma o protocolo estándar en la comunicación entre procesos.
- Desarrollar la habilidad del trabajo en equipo, el manejo de las problemáticas de un grupo y las responsabilidades que esto implica.

Temas desarrollados en el trabajo

1. Deadlock
2. Memoria compartida
3. Comunicación entre procesos
4. Sincronización de procesos e hilos
5. Planificación de procesos
6. Bibliotecas compartidas
7. Aplicaciones distribuidas multiprogramadas y multihilo.
8. Sistemas de Archivos

Índice

Introducción

El trabajo práctico de este cuatrimestre consiste en desarrollar un sistema distribuido basado en el popular Super Mario Bros, que mediante la metáfora pretende simular un planificador de procesos y mostrar aspectos internos de la interacción entre los mismos, haciendo particular foco en la sincronización y la posibilidad de bloqueos indefinidos.

El sistema estará compuesto por un conjunto de personajes que tendrán que cumplir su plan de niveles para poder enfrentar al malvado rey Koopa, quien ha secuestrado a la princesa Peach del Reino Champiñón junto con la posibilidad de aprobación de la materia.

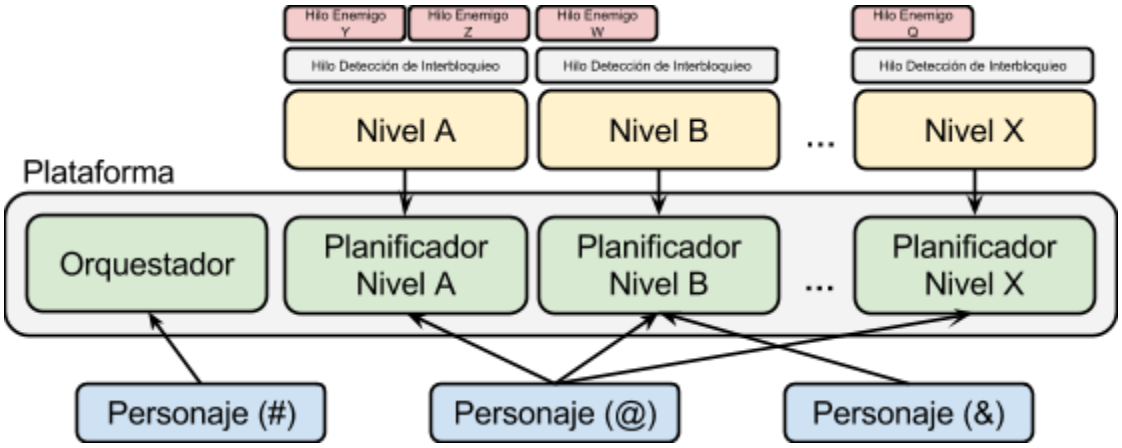
En cada nivel, el reto consistirá en conseguir un determinado conjunto de recursos, a fin de completarlo. Solo cuando todos los personajes hayan completado sus niveles, juntos, podrán enfrentar a Koopa.

Debido a que varios personajes podrán solicitar recursos dentro del mismo nivel y que las instancias de los recursos son limitadas, esto podría generar problemas que deberán ser resueltos.

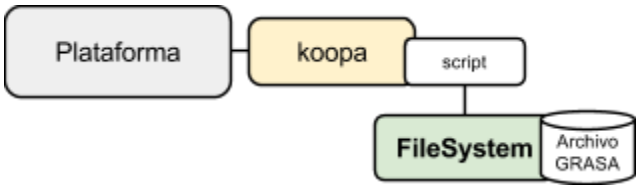
Arquitectura general

El sistema cuenta con dos clases de procesos: Personajes y Niveles, y una instancia de los procesos Plataforma, FileSystem y Koopa.

El proceso Plataforma será el centro del sistema, actuando de facilitador entre los procesos Personaje y Nivel. Una vez iniciada, la Plataforma quedará a la espera de establecer comunicación con los distintos procesos Niveles y Personajes, e irá gestionando el desarrollo del sistema desde el comienzo hasta el fin.
La Plataforma dividirá sus responsabilidades en distintos *hilos*: un hilo Orquestador y varios hilos Planificador, tantos como Niveles tenga el sistema.



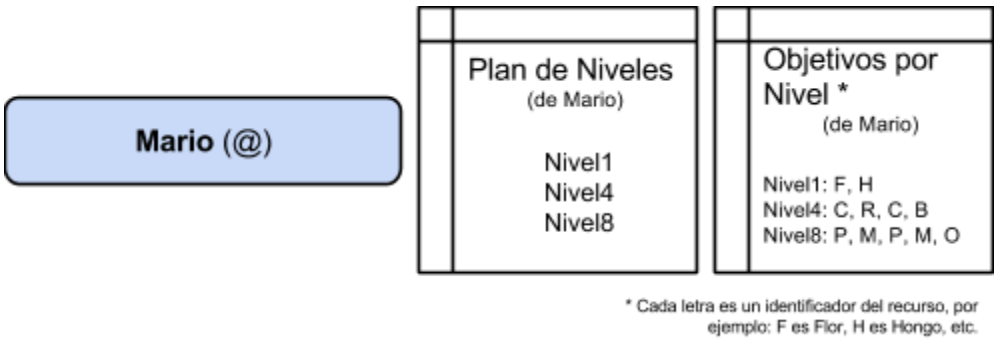
Paralelamente a la Plataforma existirá el proceso FileSystem que expondrá un sistema de archivos, utilizado por el proceso Koopa para poder concluir la simulación de forma satisfactoria.



Proceso Personaje

Existirá una instancia de este proceso por cada personaje que se encuentre participando del juego. No hay un límite de personajes ni un momento definido donde el personaje puede ingresar o abandonar el sistema.

Además de tener como atributos un nombre, un identificador y una cantidad inicial de vidas, cada personaje tendrá asociado su plan de niveles. El mismo será una lista de los niveles que este personaje deberá completar. Tendrá definido también los objetivos por nivel: una lista ordenada de los recursos que deberán ser obtenidos para poder *pasar* el nivel.



Al iniciar, el proceso leerá estos atributos de su archivo de configuración y por cada nivel que deba completar lanzará un hilo (thread), los que, de **manera concurrente**[1]**,** realizarán las siguientes acciones:

1) Conectarse al proceso Plataforma, siendo atendido por el hilo Orquestador, quien delegará la conexión al hilo Planificador del nivel correspondiente.

2) Esperar un mensaje del Planificador que le indique que es su turno de realizar un movimiento.

3) Al recibir el mensaje, realizar **un movimiento**, que puede constar de todas o algunas de estas operaciones, en función del estado actual del Personaje:

a) **Solicitar la ubicación de la caja de recursos** del próximo recurso a obtener, en caso de no conocerla. Para esto, deberá enviar un mensaje al

[1] En otras palabras se debe poder ver al personaje Mario (@) actuando en varios niveles de manera simultánea.

Planificador del Nivel actual.

b) Calcular, en función de su posición actual (x,y), la dirección en la que debe realizar su próximo movimiento[2] para alcanzar la caja de recursos y **avanzar una posición**, notificando al Planificador con un mensaje.

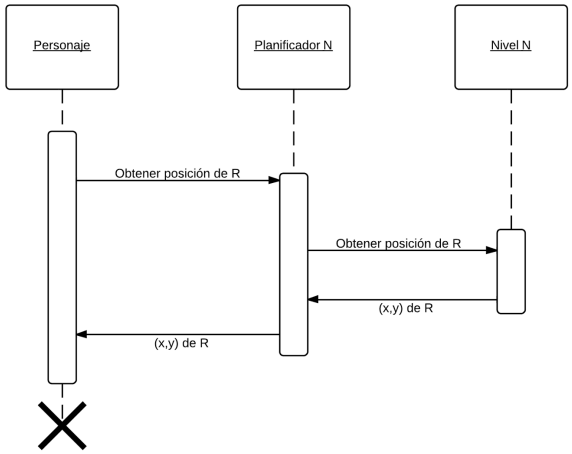
c) **Solicitar una instancia del recurso** en caso de estar en la posición de la caja correspondiente, enviándole un mensaje al Planificador. Luego de esto, deberá **esperar a la confirmación de la asignación** del mismo.

Cuando el recurso sea asignado, el hilo analizará si necesita otro recurso y volverá al punto 3), o, si ya cumplió sus objetivos del Nivel, continuará al punto 4)

4) Notificar a su Planificador que completó los objetivos de ese nivel y desconectarse.

Al concluir todos los niveles, se conectará al Orquestador y notificará que concluyó su plan de niveles. Este lo moverá a una cola de finalizados y lo dejará a la espera de que los demás Personajes terminen.

Solicitud de posicion del recurso



[2] El movimiento del personaje deberá ser cuando fuera posible alternando eje X y eje Y (uno y uno). No existe la posibilidad de movimientos en diagonal.

Muerte del personaje y vidas

Existen varios motivos por los cuales el Personaje puede morir:

- Al ser alcanzado por un Enemigo
- Al ser elegido como víctima en la resolución de un Interbloqueo

En ambos casos, el Personaje mostrará por pantalla el motivo de su muerte y se desconectará del Planificador.

En caso de tener vidas disponibles, el Personaje se descontará una vida, volverá a conectarse al hilo Orquestador y le notificará su intención de iniciar nuevamente el Nivel en que estaba jugando.

Además el Personaje puede **recibir vidas** a través del envío de la señal **SIGUSR1** al proceso, y puede **perder vidas** mediante la señal **SIGTERM**. En estos casos, el desempeño del Personaje en cada uno de los Niveles no se verá afectado, exceptuando el caso en que no tuviera vidas disponibles.

Si no le quedaran vidas disponibles, el Personaje deberá interrumpir todos sus planes de niveles y mostrar en pantalla un mensaje preguntando al usuario si desea reiniciar el juego, informando también la cantidad de reintentos que ya se realizaron. De aceptar, el Personaje incrementará su contador de reintentos y reiniciará su Plan de Niveles. En caso negativo, el Personaje se cerrará, abandonando el juego.

Adicionalmente, el proceso Personaje podría ser interrumpido por el administrador (kill, CTRL+C, etc) o por una situación anómala. En este caso **el sistema deberá reaccionar de forma favorable** asumiendo que el Personaje murió y no desea continuar.

Tanto al completar su Plan de Niveles como al ganar o perder vidas, el Personaje deberá loggear el evento, informando también la cantidad de vidas restantes y de reintentos hechos hasta el momento.

Archivo de configuración

El archivo de configuración del proceso Personaje contendrá al menos los siguientes parámetros.

- ➔ Nombre: Identificador del Personaje de tipo alfanumérico.
- ➔ Símbolo: Caracter que representará visualmente al Personaje en el mapa
- ➔ Plan de Niveles: Lista de los Niveles que debe completar el Personaje
- ➔ Lista de objetivos por Nivel: Por cada Nivel de su Lista de Niveles, una lista de los

recursos que deberá obtener de forma ordenada para completarlo.

- ➔ Vidas: Cantidad inicial de vidas del Personaje.
- ➔ IP y Puerto en que acepta conexiones el hilo Orquestador.

Restricciones del archivo de configuración:

- ★ No hay una cantidad definida de Niveles, de objetivos por Nivel o de cajas de recursos por Nivel y por juego.
- ★ Los identificadores de recurso pueden reutilizarse en niveles diferentes
- ★ Una caja de recursos fuera de los márgenes del área de juego se considera un error.
- ★ Las cajas de recursos deben estar espaciadas al menos por dos posiciones en cada eje.
- ★ Dos instancias del mismo recurso de manera consecutiva en el objetivo de un nivel se considera un error de sintaxis

  Ejemplo:

  obj[Nivel8]=[F,F,M] (error!)

  obj[Nivel8]=[F,M,F,M,F,M] (ok!)

Ejemplo de archivo de conf de un personaje

```
nombre=Mario
simbolo=@
planDeNiveles=[Nivel3,Nivel4,Nivel1]
obj[Nivel1]=[F,H,F,M]
obj[Nivel3]=[C,J,C]
obj[Nivel4]=[P,Q,M]
vidas=5
orquestador=192.168.0.100:5000
```

Proceso Plataforma

El proceso Plataforma será el encargado de gestionar el desarrollo de todo el juego, existiendo una única instancia del mismo en todo el sistema.

Su principal responsabilidad será la administración de los Personajes y los distintos estados por los que estos irán pasando. Para lograrlo, será indispensable el uso de listas compartidas entre sus hilos, para reflejar qué Personajes están listos para moverse, cuáles se encuentran esperando el otorgamiento de un recurso y cuáles han finalizado anormalmente (víctimas de interbloqueos, finalizados por señales, etc).

La Plataforma tendrá un único punto de acceso por socket y, luego de realizar un intercambio de mensajes inicial (*handshake*), delegará cada conexión al hilo correspondiente.

Al notificarse de que todos los Personajes concluyeron sus Planes de Niveles, la Plataforma ejecutará el binario ***koopa*****,** quien a su vez ejecutará un script de pruebas sobre el sistema de archivos expuesto por el proceso FileSystem mediante FUSE. En función del valor de retorno (*exit code*) del proceso ***koopa***, que deberá ser recuperado, la Plataforma notificará si se ha obtenido la victoria.

Hilo Planificador

Cada hilo Planificador será el encargado de habilitar a los procesos Personaje para que puedan actuar dentro de cada Nivel, ordenándolos **según el criterio de un algoritmo de planificación de corto plazo Round Robin o SRDF**[3].

Existirá solamente un hilo por cada Nivel conectado al proceso Plataforma, y cada uno gestionará su propia cola de Listos y de Bloqueados. Ante cada modificación de estas colas, el Planificador deberá informar por pantalla y archivo de log la modificación realizada, y el estado de cada una de las colas. Debido a que deberá gestionar múltiples conexiones desde un único hilo se deberá utilizar alguna técnica de multiplexación de entrada-salida.

Una vez elegido el Personaje al que le corresponde realizar un movimiento, el hilo le enviará

[3] Shortest Remaining Distance First - Algoritmo no-expropiativo de planificación que define que el próximo personaje a ser planificado es aquel cuya distancia total al siguiente recurso es más corta. *(Algoritmo inventado para el trabajo práctico)*

el mensaje de “turno concedido”. El Personaje podrá responder solicitando:

a) La ubicación de una caja de recursos
   El Planificador consultará al Nivel la ubicación de la caja de recursos que se le solicita y le responderá al Personaje. **Esta operación al ser solo de información no insume un quantum**.
b) Moverse en alguna dirección
   El Planificador notificará al Nivel que el personaje realizó dicho movimiento y contabilizará el uso de un quantum al personaje.
c) El otorgamiento de un recurso
   El Planificador moverá a dicho personaje a la cola de bloqueados para ese nivel y notificará al Nivel de la solicitud. Descartará, si quedara, el quantum de tiempo restante del Personaje y planificará al siguiente que se encuentre listo.

Eventualmente, el Planificador podrá detectar que un Personaje se desconectó (murió). Cuando esto ocurra, deberá liberar todos los recursos que éste tenía asignados y, si correspondiera, otorgarlos a los Personajes bloqueados por ellos, desbloqueándolos.

Cada Hilo Planificador tendrá un **algoritmo de planificación**, **valor de quantum**[4] y **tiempo de retardo entre turnos**[5] independiente que será notificado por el nivel al iniciar. Estos parámetros podrán ser modificados en tiempo de ejecución sin afectar el normal funcionamiento del sistema.

Hilo Orquestador

Recepción de conexiones: El hilo Orquestador tendrá el socket servidor que funcionará como único punto de entrada al proceso.

Distribución de Niveles: Al conectarse un proceso, el orquestador identificará la clase del mismo mediante un intercambio de mensajes de *handshake* y tomará una acción. En caso de tratarse de un Nivel, creará el correspondiente hilo Planificador y le delegará la conexión para que la atienda. En caso de tratarse de una conexión de un Personaje, validará que el Nivel que está solicitando exista y delegará la conexión al planificador correspondiente.

Personajes finalizados: Al finalizar su plan de niveles, el Planificador enviará la conexión del personaje al Orquestador y le notificará si es el último Personaje en el juego. En ese caso, se ejecutará el proceso koopa.

[4] El valor de quantum aplica únicamente para el algoritmo de RR
[5] Valor en milisegundos que cada planificador de manera independiente deberá esperar entre cada asignación de turno

Ejemplo de archivo de conf del Proceso Plataforma

```
puerto=5000
koopa=/home/utnso/koopa
script=/home/utnso/evaluacion.sh
punto_montaje= /home/utnso/mnt/fuse
```

Proceso Nivel

Cada instancia del proceso Nivel expone una serie de recursos disponibles a los Personajes. Mediante una biblioteca (proporcionada por la cátedra) dibujará en pantalla el estado actual de los recursos y de los personajes en el mapa.

Al iniciar el proceso Nivel leerá su archivo de configuración, lo validará, creará las cajas de los recursos que en el mismo figuren y se conectará a la IP y Puerto del Proceso Plataforma. Luego de un intercambio de mensajes (handshake) le notificará el algoritmo con el que deberá planificar y quedará a la espera de mensajes del Planificador y actualizará el mapa en función de los mensajes que reciba.

Además tendrá las siguientes responsabilidades:

Chequeo de interbloqueo: Cada instancia del proceso Nivel deberá tener un hilo que analice la existencia de un interbloqueo cada cierto período de tiempo configurable. En caso de detectarlo, informará por pantalla y por archivo de log dicha situación, *indicando los Personajes involucrados en el interbloqueo*.

Adicionalmente, si se encontrara activado por archivo de configuración el *recovery*, seleccionará y mostrará por pantalla y por archivo de log el Personaje que fue seleccionado como víctima para resolver dicha situación. Luego se lo notificará al Planificador para que éste actúe en consecuencia.

Es importante aclarar que la ejecución del algoritmo de interbloqueo no deberá interrumpir el funcionamiento habitual del Nivel, excepto en aquellos casos en que ambos requieran acceder a los mismos recursos compartidos.

Cambio de algoritmo, retardo o quantum: Es responsabilidad del nivel validar cambios en el archivo de configuración y notificar cuando sea posible al planificador para que actualice

los criterios de planificación.

Gestión de Enemigos: Cada Nivel contará con una serie de enemigos (ver *Enemigos*) definidos en el archivo de configuración, que funcionarán de manera concurrente entre sí y respecto de los Personajes.

Enemigos

El Nivel será el encargado de gestionar enemigos (símbolo *) que de manera concurrente e independiente del Planificador se moverán dentro del mapa según el algoritmo descrito debajo cada una cantidad de milisegundos definido por archivo de configuración. Iniciarán en una posición random la cual no podrá ser (0,0) y no podrán pasar por las cajas de recursos ni atacar a los personajes que se encuentren bloqueados en las mismas.
Al pasar por la misma posición que un Personaje lo informará por pantalla y por archivo de log y le notificará al Planificador que el Personaje perdió una vida.

Movimiento de los enemigos

El movimiento de los enemigos será siempre en función de acercarse al jugador no bloqueado que se encuentre más cerca moviéndose por los ejes de manera alternada. Si en el camino dirección del jugador se encontrara una caja de recursos deberá **esquivarla**.
En caso que no hubiera personajes en el nivel el movimiento deberá ser en dirección aleatoria simulando el movimiento de un caballo en el ajedrez (en forma de L), como siempre de a una posición por vez.

Dibujado del mapa

Para mostrar el mapa, la cátedra proveerá una biblioteca encargada de dibujar en pantalla el mapa, los personajes y las cajas de recursos. La misma se encuentra descrita en el Anexo II - Interfaz Gráfica.

Archivo de configuración

- ➔ Nombre
- ➔ Recursos disponibles
- ➔ IP y Puerto del orquestador
- ➔ Tiempo de chequeo de interbloqueo (en ms)
- ➔ Recovery (on/off): indica si estará activa la elección de víctima y su consecuente finalización.
- ➔ Cantidad de hilos enemigos

- ➔ Tiempo de movimientos de enemigos
- ➔ Stream de movimiento de enemigos

Ejemplo de archivo de configuración de un Nivel

```
Nombre=nivel1
# CajaNNN=[Nombre],[Simbolo],[Instancias],[PosX],[PosY]
Caja1=Flores,F,3,23,5
Caja2=Hongos,H,2,7,4
Caja3=Caramelos,C,4,21,2
Plataforma=192.168.0.100:5000
TiempoChequeoDeadlock=10000
Recovery=1
Enemigos=3
Sleep_Enemigos=2000
algoritmo=RR
quantum=3
retardo=500
```

Restricciones del archivo de configuración:

- ★ No hay una cantidad definida de cajas de recursos
- ★ No podrá haber dos niveles con el mismo nombre en el sistema. Esta condición deberá validarse al hacer el handshake con la Plataforma.
- ★ Las cajas de recursos deberán estar dentro de los márgenes del Nivel y no podrán superponerse.

Proceso FileSystem (FS)

Este proceso gestionará un Filesystem en formato GRAn Sistema de Archivos (ver Anexo I - GRAn Sistema de Archivos) almacenado en un archivo binario y se lo presentará al sistema operativo mediante el uso de la biblioteca FUSE.

La implementación deberá permitir que se pueda:

- ★ Leer archivos
- ★ Crear archivos
- ★ Escribir archivos
- ★ Borrar archivos
- ★ Crear directorios y dos niveles de subdirectorios.
- ★ Borrar directorios vacíos

Se deberá tener especial cuidado en proteger con semáforos las estructuras administrativas del sistema de archivos ya que se pretende que el proceso FS pueda implementar FUSE en su modo normal (multihilo)

Comandos recomendados:

- **md5sum**: devuelve la reducción criptográfica MD5 del contenido de un archivo. Si este fuera modificado aunque sea un bit, el valor se modificaría.
- **touch**: crea un archivo vacío (create)

Proceso Koopa (Pelea Final)

Mario se encuentra absorto en un cuarto vacío, observando que la computadora que abre la puerta del calabozo donde se encuentra prisionera la Princesa Peach no responde.
Un escalofrío lo paraliza al escuchar la risa del malvado Koopa, quien se burla de él y le explica que ha sido víctima de una emboscada... Que él personalmente formateó el disco que tenía los archivos para accionar el mecanismo de la puerta

Mario comprende que su derrota es inminente y se dispone a promulgar su rendición, cuando nota que su enemigo, en su torpeza, olvidó borrar el script que genera los archivos. Renovado de energías nuestro héroe reconstruye el filesystem, rescata a la princesa y encierra a su contrincante en la celda donde pasará el resto de sus días en agonía y desesperación. (?)

Proceso Koopa

Una vez que todos los personajes hayan cumplido su plan de niveles el proceso Plataforma deberá ejecutar el binario **koopa**.

Este programa será para el alumno una caja negra. Al ejecutarlo se le debe pasar como argumento un punto de montaje en que se encuentra la implementación del GRAn Sistema

de Archivos sobre FUSE desarrollado por el alumno, y un programa[6] que realizará operaciones sobre archivos (crear archivos y directorios, borrar archivos y directorios, leer archivos, escribir archivos y truncar archivos)
Luego de ejecutar este programa, **koopa** validará que el sistema de archivos se encuentre en un estado consistente, y de ser así, devolverá un *exit code* 0.

Entregas y checkpoints

A modo de guía para los alumnos se proponen una serie de puntos de control, cada uno con un conjunto de objetivos a desarrollar y un plazo en el que deberían estar finalizados.

Primer Checkpoint

Tiempo estimado: 2 semanas
Fecha: 5 de Octubre

Hitos

- Familiarizarse con el setup de las VMs, repositorio y el entorno de desarrollo
- Implementar la commons-library para el manejo de archivos de configuración, listas/colas y logs
- Desarrollar un proceso servidor de conexiones que soporte múltiples clientes (posteriormente hilo Planificador)
- Desarrollar un proceso cliente liviano que permita enviar y recibir mensajes paquetizados (Nivel)
- Implementar la biblioteca libnivel.so y desarrollar un proceso que permita dibujar elementos en pantalla (Nivel) investigando el código de ejemplo
- Comprender la lógica de FUSE y trabajar con el ejemplo (ver a continuación)
- Desarrollar un código capaz de reconocer las estructuras de un filesystem GRASA en un archivo mapeado a memoria (ver mmap)

Recursos necesarios

Configuración de un repositorio GitHub en Eclipse - Link

[6] Un programa que ejecuta operaciones puede también ser un script en BASH o en cualquier lenguaje de scripting. Será importante validar que el mismo tenga permisos de ejecución (chmod +x script.sh)

Video debug con Eclipse - Link
Video GIT Basic (link) y GIT desde Eclipse (link)
Video creación una Shared Library - Link
Guia Beej de Programación en redes - Link
SO FUSE Example - Link

Segundo Checkpoint

Tiempo estimado: 3 semanas
Fecha: 26 de Octubre

Hitos

- Desarrollar la lógica del proceso personaje con varios hilos conectados a varios planificadores de manera simultánea.
- Integrar los procesos Personaje, Nivel y el Planificador, y permitir el paso de mensajes entre ellos. Bosquejar Personajes y recursos en el Nivel y desarrollar su interacción.
- Desarrollar el hilo Orquestador dentro del proceso Plataforma.
- Convertir el proceso *servidor* del checkpoint anterior en el hilo Planificador del proceso Plataforma.
- Desarrollar los hilos Enemigos en el Nivel
- Desarrollar un código que mediante FUSE permita operaciones de sólo lectura sobre un filesystem GRASA en el proceso Filesystem

Recursos necesarios

Linux POSIX Threads (pthread) - Link
Gcc, Makefile - Link parte 1 y Link parte 2

Tercer Checkpoint - Entrega Obligatoria

Tiempo Estimado: 2 semanas
Fecha: 9 de Noviembre

Hitos

- Detectar la modificación del archivo de configuración en tiempo de ejecución (inotify)

- Desarrollar el hilo que verifica interbloqueos en el Nivel
- Desarrollar las operaciones de escritura de GRASA en el proceso Filesystem

Recursos necesarios

Intro to Inotify - Link
PPT Procesos - Link

Cuarto Checkpoint

Tiempo Estimado: 2 semanas
Fecha: 23 de Noviembre

Hitos

- Completar la lógica de los procesos nivel, plataforma, personaje y filesystem
- Integrar todos los procesos e iniciar fase de testing de integración y stress

Entrega Final: 30 de Noviembre
Primer Recuperatorio: 7 de Diciembre
Segundo Recuperatorio: 14 de Diciembre[7]

[7] Las fechas son estimativas y pueden ser modificadas con su debida notificación.

Anexo I - GRAn Sistema de Archivos

El GRAn Sistema de Archivos es un filesystem creado con propósitos académicos para que el alumno se interiorice y comprenda el funcionamiento básico de la gestión de archivos en un sistema operativo. A pesar de esto, sus características se asemejan a sistemas de archivos modernos, y sus estructuras están optimizadas para discos de gran capacidad y la nueva propuesta de estándar Advanced Format

- ★ Tamaño máximo de disco soportado: 16 TB
- ★ Tamaño máximo de archivo 3.9GB
- ★ Soporte de directorios y subdirectorios
- ★ Tamaño máximo de nombre de archivo: 71 caracteres
- ★ Cantidad máxima de archivos: 1024

Características técnicas

- ★ Tamaño de bloque: Fijo (4096 bytes)
- ★ Tamaño de la tabla de nodos: 1024 bloques (4 MB)
- ★ Tabla de bloques libres: Bitmap

Arquitectura

Cada bloque en el sistema de archivos estará direccionado por un puntero de 4 bytes `ptrGBloque` permitiendo así un máximo de 2^32 bloques.

Para un disco de tamaño T [bytes] y el BLOCK_SIZE de 4096 las estructuras se definen de esta manera:


```
1 bloque

n bloques = T / BLOCK_SIZE / 8

1024 bloques

x bloques = T / BLOCK_SIZE - 1 - n - 1024
```

Header

El encabezado del sistema de archivos estará almacenado en el primer bloque. Contará con los siguientes campos:

Campo	Tamaño	Valor
Identificador	`5 char`	“GRASA”
Versión	`4 bytes`	1
Bloque de inicio del bitmap	`ptrGBloque`	1
Tamaño del Bitmap (en bloques)	`4 bytes`	n
Relleno	`4073 bytes`	[no definido]

Bitmap

El Bitmap, también conocido como bitvector, es un array de bits en el que cada bit identifica si el bloque ubicado en su posición se encuentra ocupado (1) o no (0).

Por ejemplo, un bitmap con este valor 000001010100 nos indicaría que los bloques 5, 7 y 9 están ocupados y los restantes libres (recordar que la primer ubicación es la 0).

Cuando necesite localizar un bloque libre, el filesystem utiliza esta estructura para encontrar uno, lo marca como ocupado y lo utiliza.

Es importante recalcar que los bloques utilizados por las estructuras administrativas deben siempre estar marcados como utilizados.

Tabla de Nodos

La tabla de nodos del GRAn Sistema de Archivos es de tamaño fijo: un array de 1024 posiciones de estructuras de tipo `GFile`

Cada nodo consta de los siguientes campos:

Campo	Tamaño	Valor
Estado	`1 byte`	0: Borrado, 1: Ocupado, 2: Directorio
Nombre de Archivo	`71 bytes`	[Nombre del archivo]
Bloque padre	`ptrGBloque`	El número de bloque donde está almacenado el directorio padre o cero si está en el directorio raiz
Tamaño del archivo	`4 bytes`	(Máximo 4 GB)
Fecha de creación	`8 bytes`	Timestamp de creación del archivo
Fecha de modificación	`8 bytes`	Timestamp de la última modificación del archivo
Array de punteros a bloques indirectos simples	`1000 x ptrGBloque`	Cada posición del array contiene la dirección del bloque que almacena un array de 1024 direcciones de bloques de datos

Herramientas - Grasa Tools

Se le proveerán al alumno dos herramientas para simplificar el desarrollo del trabajo práctico: https://github.com/sisoputnfrba/grasa-tools

- **grasa-format**: Recibe como parámetro un archivo e inicializa un filesystem GRASA dentro del mismo.

- **grasa-dump**: Imprime por pantalla el estado de las estructuras administrativas de un filesystem GRASA.

Definición de las estructuras

https://github.com/sisoputnfrba/grasa-tools/blob/master/grasa.h

Anexo II - Interfaz gráfica

Para simplificar el desarrollo de una interfaz de usuario, la cátedra proveerá una primitiva y sencilla biblioteca para graficar el mapa en ventanas, desde la consola. Dicha biblioteca hará uso de otra biblioteca llamada ncurses (solo será necesario dicho conocimiento para la compilación y linkeo del programa).

La biblioteca, junto con un programa de ejemplo, se encuentran en el siguiente repositorio:

https://github.com/sisoputnfrba/so-nivel-gui-library.git

También se provee un breve tutorial en la wiki de dicho repositorio.

Anexo III - Especificaciones Técnicas

Compilación y errores

Se deberá utilizar el compilador gcc. El proceso de compilación no debe arrojar errores.

Comunicaciones – Sockets y otros

Para la comunicación cliente/servidor se utilizarán los protocolos especificados en el trabajo práctico y se implementarán utilizando sockets orientados a la conexión del tipo AF_INET para TCP/IPv4.

Cuando sea necesario utilizar una función para multiplexar I/O (o sea, manejar múltiples descriptores al mismo tiempo), el grupo deberá optar por utilizar alguna/s de las siguientes opciones:

- `select()`
- `poll()`
- `epoll()`

Queda prohibido el uso de las funciones de la familia `sleep()` para cuestiones de sincronización en la comunicación.

Process Managment y Multithreading

En caso de ser requerido, solo se podrá utilizar la API de la biblioteca pthreads que forma parte del estándar POSIX (NPTL – Native POSIX Thread Library).

Manejo de Errores

Queda prohibido el uso de cualquier mecanismo de exception handling (__try __except) o funciones/bibliotecas de C++.