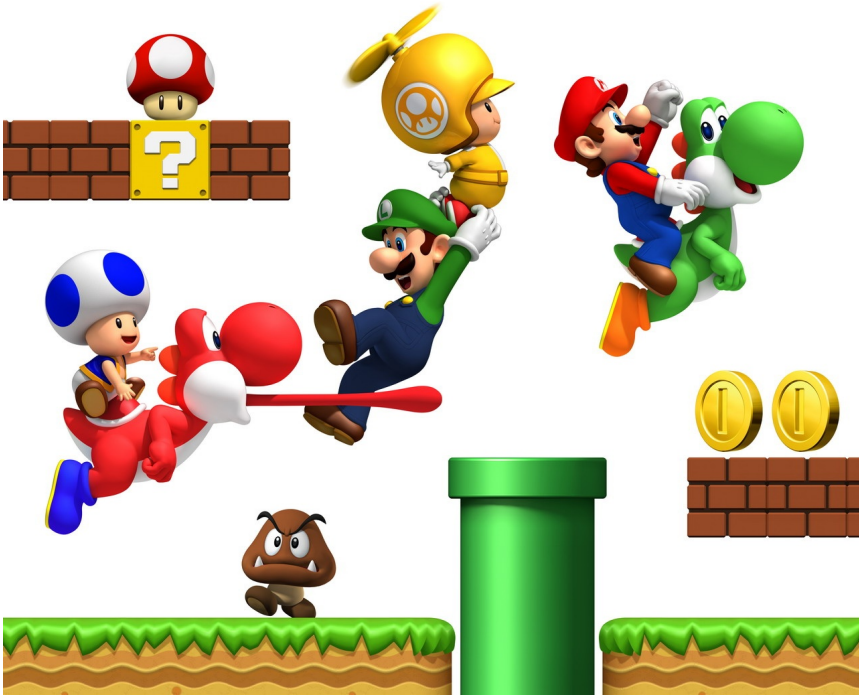
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Super Mario Proc.



Ingeniería en Sistemas de Información

Cátedra de Sistemas Operativos

- 1C2013 -
Versión 1.0

Indice

Introducción

El trabajo práctico de este cuatrimestre consiste en desarrollar un juego de plataformas, basado en el popular Super Mario Bros.

El sistema estará compuesto por un conjunto de fontaneros (Mario, Luigi, etc). Ellos tendrán como objetivo rescatar a la princesa Peach del Reino Champiñón, quien fue secuestrada por el malvado rey Koopa.

Para ello, los personajes irán superando una serie de niveles predefinidos, hasta llegar a enfrentar a Koopa. En cada nivel, ellos deberán conseguir una cantidad específica de recursos, que les permitirán pasar al siguiente nivel. Solo cuando todos los personajes hayan completado todos sus niveles, juntos, podrán enfrentar a Koopa.

Cada nivel tendrá una cantidad limitada de recursos, y los irá otorgando a los personajes que lo visiten a medida que estos se los vayan solicitando.
Si bien cada personaje tendrá su propio plan de niveles, podría ocurrir que algunos coincidan en un momento determinado en la visita de un mismo nivel. Dado que los recursos por nivel serán limitados, esto podría generar potenciales problemas que deberán ser resueltos inmediatamente.

Se pretende mediante la metáfora propuesta, 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.

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.

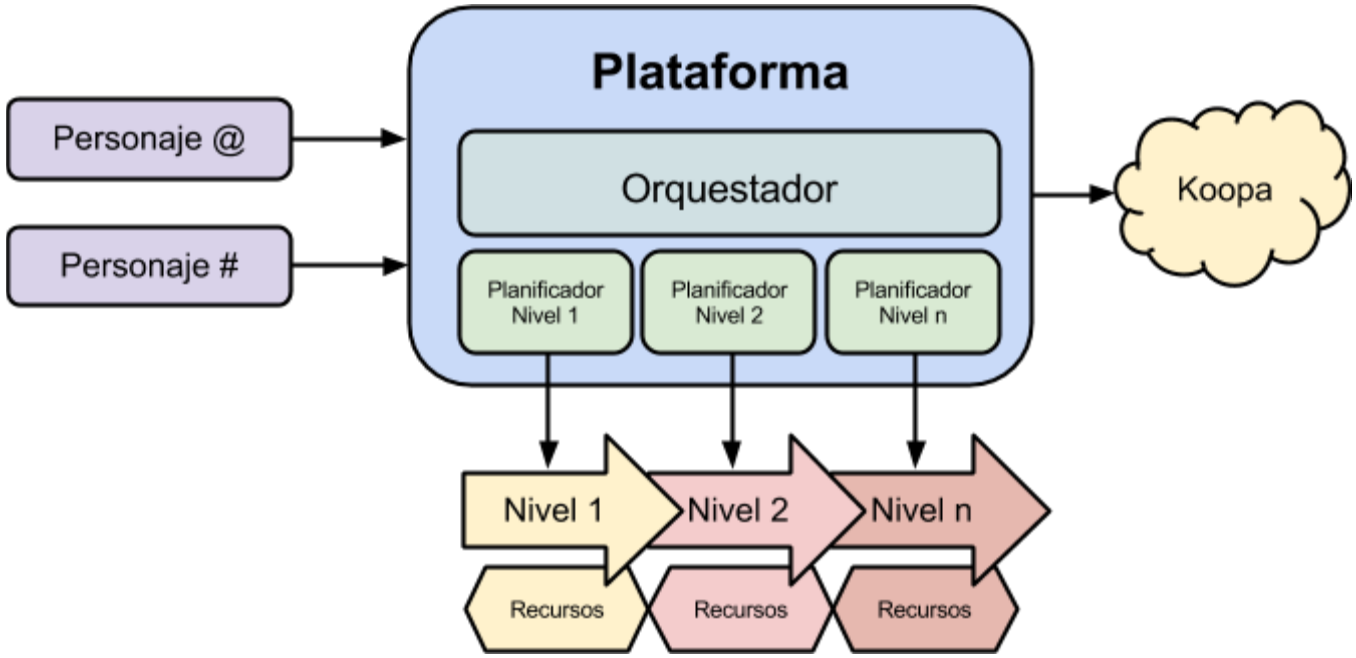
Temas desarrollados en el trabajo

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

Arquitectura general

El sistema contará con cuatro tipos de *procesos*[1]: Personaje, Plataforma, Nivel y Koopa.

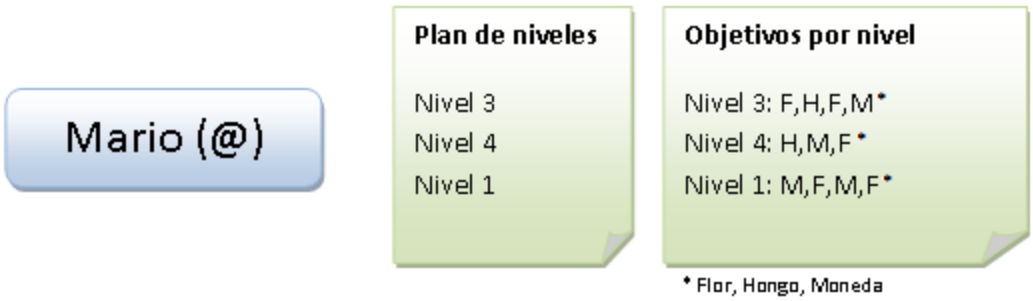
El proceso Plataforma será el centro del sistema, actuando de facilitador entre los procesos Personaje, Nivel. Una vez iniciada, la Plataforma quedará a la espera de establecer comunicación con los distintos procesos Niveles y Personajes, e irá gestionando el desarrollo del sistema desde el comienzo hasta el fin.
La Plataforma dividirá sus responsabilidades en distintos *hilos*[2] (Orquestador, Planificador), y los Personajes se comunicarán directamente con ellos según el servicio que requieran de la misma.



[1] Proceso: unidad básica de trabajo del sistema operativo. El concepto se desarrollará en la clase teórica asociada.
[2] Hilo (thread): proceso liviano. El concepto se desarrollará en la clase teórica asociada.

Proceso Personaje

Existirá una instancia de este proceso por cada personaje que se encuentre participando del juego.
Además de tener como atributos un nombre y una cantidad inicial de vidas, cada personaje tendrá asociado un plan de niveles. El mismo será una lista ordenada de los niveles que el personaje deberá completar.
Tendrá definido también los objetivos por nivel, los cuales consistirán en una lista ordenada de los recursos que deberán ser obtenidos para poder concluir el nivel.



Lógica principal

Al iniciar, el proceso leerá su archivo de configuración y en base a este tendrá su lógica de ejecución, la cual consiste en completar una serie de niveles. La secuencia para realizar estas tareas son presentadas a continuación.

Ingresar al Nivel

Le solicitará al Hilo Orquestador información de contacto sobre el próximo nivel, para así poder establecer la comunicación y empezar a consumir los recursos. Para ello:

1) Se conectará al Hilo Orquestador y le solicitará la información de contacto sobre el próximo nivel.

2) El Hilo Orquestador responderá con dicha información (IP/Puerto del proceso Nivel correspondiente y del hilo Planificador asignado a ese Nivel).

3) Se conectará a ambos (desconectándose del hilo orquestador) y quedará a la espera de una notificación del Planificador para ejecutar la primer movida.

Realizar un movimiento *(en busca del primer o próximo recurso)*

Cuando sea su turno, el personaje realizará un movimiento hacia el próximo recurso de la lista:

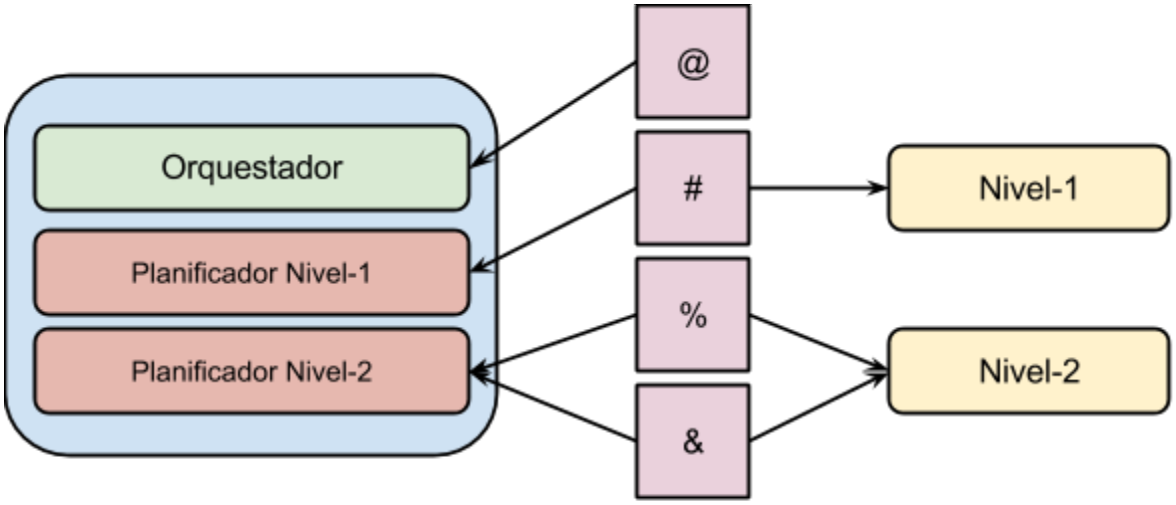
4) Una vez recibida la notificación[3] de “movimiento permitido”, realizará un movimiento un eje a la vez en dirección al recurso que requiere y evaluará si ha llegado al mismo. En caso afirmativo *le solicitará* al proceso Nivel una instancia del recurso

<u>Nota:</u> En caso de no tener destino dentro del mapa para la búsqueda de recursos, deberá solicitar al nivel la ubicación (posición x,y) del próximo recurso antes de realizar el movimiento de este turno para fijarlo como destino.

5) Notificará al Planificador que ha concluido el turno (indicando si quedó bloqueado o no) y evaluará si ha alcanzado el total de los recursos requeridos:

a) En caso afirmativo: notificará al nivel y luego se desconectará del mismo y del planificador. Luego irá al punto 1)

b) En caso negativo, volverá al punto 4).



Cada proceso personaje se conecta a los proceso requeridos

[3] Al recibir dicha notificación, el personaje asumirá, en caso de que existiera una solicitud pendiente de un recurso, que el mismo ya ha sido otorgado.

Muerte del personaje (terminación del proceso)

Existen dos motivos por los cuales el personaje puede morir: al recibir la señal SIGTERM o porque el Hilo Orquestador lo requiera para resolver una situación de interbloqueo[4].
En ambos casos el proceso mostrará por pantalla el motivo de su muerte y notificará al nivel para que este libere los recursos asignados.
En caso de que el contador de vidas del proceso sea mayor que cero, descontará una vida y le notificará al orquestador su intención de reiniciar el nivel. Si no quedaran vidas disponibles, el proceso deberá incrementar nuevamente el contador de vidas al valor inicial y reiniciar el plan de niveles.

Además el personaje puede recibir vidas, a través de la señal SIGUSR1, la cual no afecta a la posición actual del personaje en el mapa, solo hace incrementar el contador de vidas en 1.

Archivo de configuración

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

- Nombre: Identificador del proceso 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
- Objetivos ordenados por nivel: Lista de los objetivos que debe cumplir por cada nivel, donde cada objetivo es un recurso del nivel.
- Vidas: Cantidad inicial de vidas del personaje.
- IP y Puerto del hilo orquestador.

Ejemplo de archivo de conf de un personaje

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

[4] Interbloqueo: bloqueo permanente de un conjunto de procesos que compiten por recursos del sistema o bien simplemente se comunican entre sí. El concepto se desarrollará en la clase teórica asociada

Proceso Plataforma

Este proceso, para cual existirá una única instancia en todo el sistema, se encargará de gestionar todo el juego.
Una de sus principales responsabilidades pasará por la administración de los personajes y los distintos estados por los que irán pasando. Para ello, será indispensable el uso de listas compartidas entre sus hilos; para reflejar los personajes listos para moverse, los que se encuentran esperando el otorgamiento de un recurso y los que han sido finalizados anormalmente (interbloqueo, señal, etc).

Hilo Planificador

Este hilo será el encargado de habilitar a los procesos personaje para que puedan actuar dentro de cada nivel, ordenando a los mismos **según el criterio del algoritmo de planificación de corto plazo Round Robin**. Existirá un hilo por cada Nivel conectado al proceso plataforma, el cual gestionará su propia cola de listos.

Una vez elegido el personaje al que le corresponde realizar un movimiento, el hilo le enviará el mensaje de “movimiento permitido”. Luego deberá recibir inmediatamente de parte el personaje una indicación de que el movimiento fue realizado (“turno concluido”). Luego deberá esperar una cantidad de segundos definida por archivo de configuración antes de realizar la próxima operación. El Planificador podría también ser notificado en dicho mensaje del bloqueo del personaje ante el no otorgamiento de un recurso solicitado. En caso de que ello suceda, deberá sacar de la cola de listos al Proceso Personaje bloqueado, y pasarlo a una cola de bloqueados a la espera de que se liberen instancias de ese recurso.

El hilo Planificador utilizará un valor de quantum[5] que se le indique por archivo de configuración pudiendo ser modificado de forma inmediata en tiempo de ejecución sin que esto afecte el resto del funcionamiento del sistema.

Hilo Orquestador

[5] Un personaje consume quantum solo cuando realiza un “movimiento permitido” o cuando solicita un recurso.

Este hilo es el encargado de realizar tres tareas fundamentales:

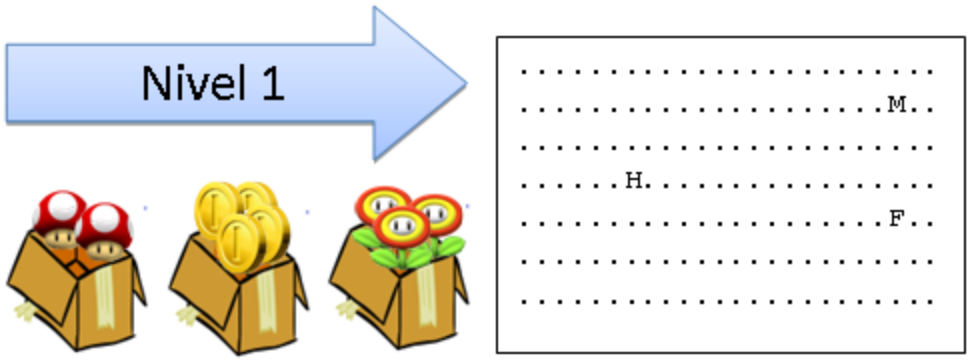
Distribución de niveles: Cuando el personaje termina un nivel (por haber logrado su objetivo) le notifica al orquestador que terminó, y que quiere pasar al siguiente nivel. El orquestador entonces le envia la informacion necesaria (ip y puerto) del hilo planificador correspondiente al nivel solicitado. Es importante aclarar que por cada avance de nivel el personaje deberá desconectarse del orquestador luego de recibida la información, para ahorrar recursos, y cuando termine el nivel deberá re-conectarse para obtener la información del siguiente nivel.

Gestión de los procesos bloqueados a la espera de un recurso: Cuando el personaje realiza un movimiento y llega al depósito de recursos, éste le pide al nivel una instancia de ese recurso. Si ese recurso no esta disponible, se bloqueará y el hilo Planificador lo moverá a la cola de bloqueados correspondiente.
Cuando un proceso personaje termina un nivel correctamente, el orquestador recibirá del nivel una notificación de la cantidad de instancias y los tipos de recursos que fueron liberados como consecuencia de la salida del personaje del nivel. El orquestador seleccionará los procesos se desbloquearan por cada recurso liberado (si hubiera procesos bloqueados), y le informará al nivel, la cantidad de recursos asignados como consecuencia del desbloqueo de los procesos para que el nivel actualice de ser necesario su contador de instancias disponibles.
Una vez hecho esto el orquestador deberá sacar de las colas de bloqueados a los procesos en orden, insertándolos en la cola de listos para que sean planificados por el Planificador.

Recupero ante un interbloqueo: Cuando el proceso Nivel detecte un interbloqueo, podría decidir solicitarle al orquestador que realice un recupero (recovery) de la situación. En caso que así sea, es responsabilidad del orquestador decidir qué proceso personaje morirá en función del orden de llegada de los mismos al proceso nivel. Una vez definida la víctima, actualizará sus listas y le notificará al proceso Nivel cual es el personaje víctima, para que el nivel pueda analizar los recursos que se liberarán con su muerte. También deberá notificar al proceso Personaje, para que actualice su estado interno apropiadamente.

Proceso Nivel

Cada instancia del proceso nivel expone una serie de recursos disponibles a los personajes. Mediante la biblioteca proporcionada por la cátedra deberá dibujar el estado actual de los recursos y de los personajes en el mapa en pantalla.



Además del dibujo del mapa, el proceso nivel tiene las siguientes responsabilidades:

Movimiento del personaje: El nivel estará a la espera de conexiones de los personajes que ingresen al nivel. Una vez ingresado el personaje le pedirá al nivel que le informe la posición del depósito del recurso que el personaje necesita para poder completar su objetivo, y luego el nivel comenzará a recibir solicitudes de movimiento del personaje, en las cuales debe validar que se quede dentro de los márgenes del nivel. Se debe tener en cuenta que para facilitar el desarrollo, dos personajes pueden compartir un mismo punto dentro del mapa.

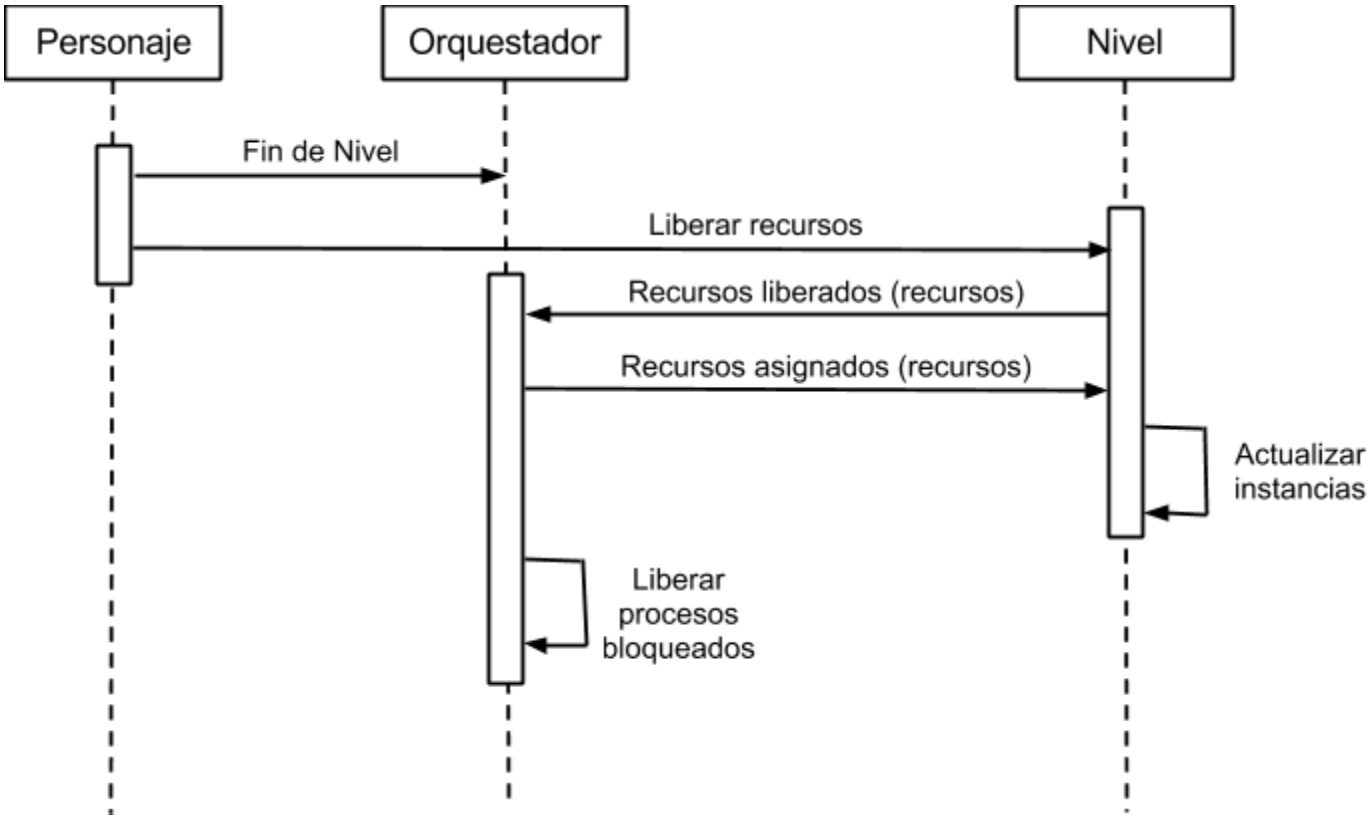
Control de recursos totales y asignados: Cuando un personaje solicita un recurso, el nivel debe validar, por un lado que el personaje efectivamente esté en el casillero que corresponde. Luego, deberá analizar si tiene recursos disponibles para darle, de ser así le responderá con una confirmación, o el rechazo (en este caso el personaje le notificará al orquestador el bloqueo).

Chequeo de interbloqueo: Cada proceso nivel deberá tener un hilo extra que cada una cierto período de tiempo configurable, analice la existencia de un interbloqueo[6]. En caso afirmativo,

[6] El algoritmo de detección de interbloqueo se podrá encontrar en la bibliografía sugerida de la cátedra

informará al usuario (por pantalla o archivo log) de dicha situación, *indicando los procesos involucrados en el interbloqueo*.
Adicionalmente, si se encuentra activado el *recovery*, informará al orquestador los procesos involucrados en el interbloqueo para que elimine a uno de ellos bajo su criterio. El orquestador responderá cual proceso será finalizado. Una vez obtenido el proceso victima, el nivel le informará al orquestador los recursos liberados a causa de la muerte del personaje, y el orquestador contestará, de esa cantidad de recursos inicialmente liberados, cuales fueron asignados a procesos antes bloqueados. Es responsabilidad del nivel mantener su lista de recursos disponibles y asignados consistente y actual, por ende deberá marcar como asignados los recursos que le diga el orquestador, y marcar los restantes como liberados.

En caso de muerte del personaje (sea por el motivo que sea), el nivel deberá detectar la desconexión del mismo y deberá liberar sus recursos (informando al orquestador) para que el personaje reinicie el nivel o el plan completo según su estado actual.



Dibujado del mapa

Para el dibujado del mapa la cátedra proveerá una biblioteca, la cual será encargada de la pantalla el mapa, los personajes y las cajas de recursos. La misma se encuentra descrita en el

Anexo A.

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.[7]

Ejemplo de archivo de configuración de un Nivel

```
Nombre=nivel1
# CajaX=[Nombre],[Simbolo],[Instancias],[PosX],[PosY]
Caja1=Flores,F,3,23,5
Caja2=Hongos,H,2,7,4
Caja3=Monedas,M,4,21,2
orquestador=192.168.0.100:5000
TiempoChequeoDeadlock=10000
Recovery=1
```

Proceso Koopa (Pelea Final)

Una vez que los personajes cumplan su Plan de Niveles el proceso planificador deberá ejecutar el binario **koopa** el cual tiene como requisito una biblioteca compartida provista por el alumno. El programa Koopa malvado y hambriento por cpu, le tiende una trampa al valiente Mario llevándose el módulo de asignaciones de memoria del sistema que acciona el puente

Mario, entiende que debe ser él quien ordene las solicitudes de particiones de memoria variables en el limitado espacio disponible utilizando el algoritmo que se le fue indicado **(best-fit,**

[7] Esto significa que si bien la detección se realizará siempre, en caso de estar deshabilitado el recovery lo único que hará el nivel es detectarlo y notificarlo al usuario.

next-fit, first-fit, worst-fit).

De hacerlo bien podrá activar el mecanismo para que Koopa caiga a un río de roca volcánica ardiente y muera de una forma espantosa (?).

Así Mario habrá salvado una vez más a su princesa.

Proceso Koopa

El proceso Koopa es un binario que será provisto por la cátedra sin acceso a su código fuente. El mismo pregunta por pantalla el tamaño del segmento de memoria a crear y el nombre del archivo que contiene una lista de solicitudes de almacenamiento de partición con formato: [operación]:[char id]:[int tamanio]:[t_memoria contenido], por ejemplo:

```
GRABAR:A:1024:[cadena de 1024 bytes]
GRABAR:B:100:[cadena de 100 bytes]
BORRAR:A
GRABAR:C:300:[cadena de 300 bytes]
BORRAR:C
```
[8]

Luego de leer el archivo, invoca a la función *crear_memoria()* y le envía como parámetro el tamaño del segmento a crear.

Por cada solicitud leída del archivo invoca a la función *almacenar_particion()* o *eliminar_particion()* según corresponda.

Una vez llegado al final de la lista de solicitudes, solicitará las *particiones()* y con esto imprimirá un listado como el siguiente:

```
Segmento de Memoria:
---------------------------------------
C[0:299:300]:[cadena de 300 bytes]
VACIO[300:1023:724]
B[1024:1123:100]:[cadena de 100 bytes]
VACIO[1124:2047:924]
---------------------------------------
```

[8] Donde dice “[cadena de x bytes]” en la práctica la solicitud tendrá realmente una cadena de x bytes. Estos se deberán grabar dentro de la partición.

```
Total: 2048 bytes, libres: 1648 bytes
```

Por último invocará *liberar_memoria()* y si el orden de las particiones es el correcto mostrará un mensaje por pantalla y se habrá ganado el juego.

Integración

Una vez que todos los personajes hayan concluído su plan de niveles el proceso Planificador deberá solicitar por pantalla una confirmación y mediante una función de la familia exec() ejecutar el proceso Koopa con los parámetros correspondientes definidos por archivo de configuración.

Desarrollo

El alumno deberá descargar el proceso Koopa de este link y desarrollar la biblioteca dinámica libMemory.so requerida por el proceso respetando el algoritmo que le fue asignado al grupo y la interfaz descripta en el Anexo B - koopa.h

Dada la naturaleza del proceso (un solo hilo de ejecución) no será necesario que la biblioteca soporte llamadas concurrentes (thread-safe).

Entregas y checkpoints

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

Primer Checkpoint

Tiempo estimado: 3 semanas
Fecha: 4 de Mayo

Hitos

- Familiarizarse con el setup de la VM, del repositorio y el entorno de desarrollo
- Implementar la commons-library para el manejo de archivos de configuración,

listas y logs.
- Desarrollar un proceso servidor de conexiones que soporte múltiples clientes (planificador)
- Desarrollar un proceso cliente liviano que permita enviar y recibir mensajes paquetizados (personaje y nivel)
- Implementar la biblioteca libnivel.so y desarrollar un proceso que permita dibujar elementos en pantalla (nivel)

Lectura recomendada

Configuración de un repositorio GitHub en Eclipse - Link
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

Segundo Checkpoint

Tiempo estimado: 4 semanas
Fecha: 1 de Junio

Hitos

- Dividir la responsabilidad del proceso servidor en dos hilos independientes implementando pthreads (planificador)
- Desarrollar libMemoria.so respetando el algoritmo asignado (koopa)
- Desarrollar la lógica del proceso personaje.
- Integrar los procesos personaje, nivel y planificador y permitir el paso de mensajes entre ellos. Bosquejar personajes y recursos en el nivel y desarrollar su interacción.

Lectura recomendada

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

Tercer Checkpoint - Entrega Obligatoria

Tiempo Estimado: 3 semanas
Fecha: 22 de Junio

Hitos

- Desarrollar la lógica de los procesos nivel y el hilo planificador
- Permitir la modificación del quantum en tiempo de ejecución (inotify)
- Desarrollar el hilo que verifica interbloqueos en el nivel

Lectura recomendada

Intro to Inotify - Link

Cuarto Checkpoint

Tiempo Estimado: 2 semanas
Fecha: 6 de Julio

Hitos

- Desarrollar la lógica del hilo orquestador.
- Iniciar fase de testing de integración y stress

Entrega Final: 13 de Julio
Primer Recuperatorio: 27 de Julio
Segundo Recuperatorio: 3 de Agosto

Especificaciones técnicas

Compilación y errores

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

Comunicaciones – Sockets

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, el grupo deberá optar por utilizar alguna/s de las siguientes opciones:

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

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++.

Archivo Log

Cada instancia de cada proceso contará con su correspondiente archivo de log donde registrará los eventos más importantes de la ejecución de la aplicación y los valores necesarios para conocer el estado del sistema en un determinado momento.

Los archivos de Log deberán respetar un formato y se deberá permitir distintos niveles de detalle configurable por el usuario.

Formato: *[TIPOLOG] [TIMESTAMP] [NOMBRE_PROCESO]/(PID:TID): MENSAJE*

- ***Tipo de Log:*** Nivel de detalle a criterio. INFO, WARN, ERROR ó DEBUG
- ***Fecha:*** Fecha del sistema. Deberá respetar el siguiente formato [HH:mm:ss.SSS].
- ***Thread ID:*** Cuando se trate de un proceso multihilo se deberá registrar el identificador del thread que escribe en el archivo.
- ***Mensaje:*** Descripción del evento ó cualquier información que se considere apropiada.

Ejemplo: `[DEBUG] 15:05:10:1123 Nivel/(3043:15): Creo el socket`

Eventos de logueo obligatorio

- Todos: Conexiones y desconexiones de todos procesos (incluido el motivo si existe). Agregar información de conección (ip/puerto origen / destino)
- Nivel:
  - Detección de deadlock: Se deberá incluir los nombres de los personajes involucrados en el mismo.
  - Recovery: incluyendo el nombre del personaje elegido como víctima
- Personaje: por cada turno, su posición actual, la solicitud realizada (si existiera) y la respuesta recibida
- Planificador
  - Cada turno otorgado al personaje (indicando el personaje)
  - Ante cada modificación de todas las listas utilizadas, una impresión completa de las mismas.
    - Ej: Listos: Mario <- Luigi; Bloqueados: <- Yoshi1 <- Yoshi 1; Ejecutando: Yoshi3

Anexo A - Interfaz gráfica del Nivel

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 B - koopa.h

Se describe en este anexo la interfaz que debe cumplir la biblioteca desarrollada por el alumno para poder ser utilizada por el proceso koopa.

koopa.h

```
typedef char* t_memoria;

typedef struct t_particion {
	char id;
	int inicio;
	int tamanio;
	t_memoria dato;
```

```
	bool libre;
} particion;
	// estructura que define una particion


t_memoria crear_memoria(int tamanio);
	// Crea el segmento de memoria a particionar

int almacenar_particion(t_memoria segmento, char id, int tamanio, t_memoria
contenido);

	/* Crea una particion dentro del segmento de memoria de tamaño,
	identificador y contenido especificado. Devuelve el valor numerico -1
	en caso de error (ej: tamaño de la particion mayor que el tamaño total
	del segmento, id duplicado, etc.), 1 en caso de exito y 0 en caso de
	no encontrar una particion libre lo suficientemente grande para
	almacenar la solicitud. */

int eliminar_particion(t_memoria segmento, char id);

	/* Esta funcion elimina la particion dentro del segmento de memoria
	correspondiente al identificador enviado como parametro. Devuelve el
	valor numérico 1 en caso de exito y 0 en caso de no encontrar una
	particion con dicho identificador */

void liberar_memoria(t_memoria segmento);

	// Esta funcion libera los recursos tomados en crear_memoria()

t_list* particiones(t_memoria segmento);

	/* Esta funcion devuelve una lista en el formato t_list de las
	commons-library con la siguiente descripcion por cada particion */
```