UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

char*mander

Ingeniería en Sistemas de Información
Cátedra de Sistemas Operativos

- 2C2016 -

Table of Contents

char *mander

Introducción

El trabajo práctico de este cuatrimestre consiste en desarrollar un sistema distribuido, que mediante la metáfora pretende simular algunos aspectos de un Sistema Operativo y mostrar aspectos internos de la interacción entre los mismos.

Se encuentran, entre ellos, la planificación de procesos, con la posibilidad de bloqueos indefinidos, sincronización de aplicaciones, interconectividad y la implementación de un modelo de Sistema de Archivos.

El sistema estará compuesto por un conjunto de procesos que tendrán que competir por la obtención de recursos para cumplir con su objetivo.

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

Para amenizar la dinámica del trabajo el sistema utilizará metáforas del universo Pokémon.

Objetivos del Trabajo Práctico

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

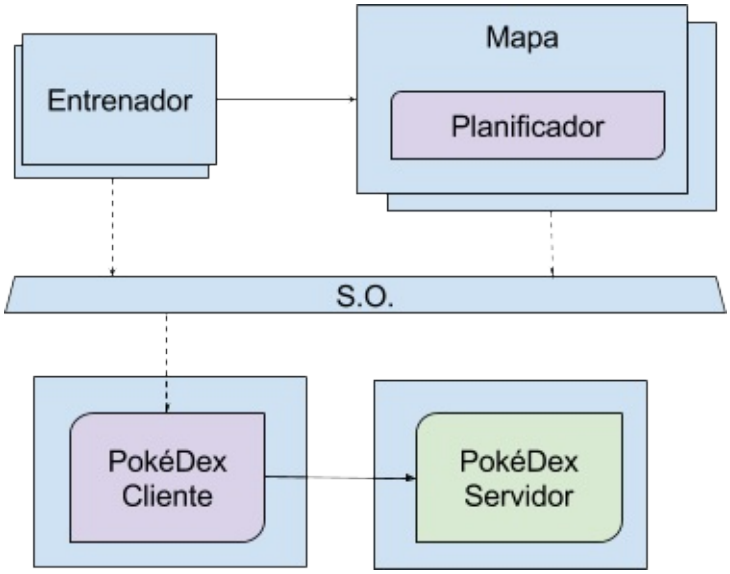
- Adquirir los conocimientos prácticos del uso y aplicación de un conjunto de servicios que ofrecen los sistemas operativos
- Desarrollar la habilidad del trabajo en equipo, el manejo de las problemáticas de un grupo y las responsabilidades que esto implica
- Evaluar la factibilidad y ponderar las distintas 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

Evaluación del Trabajo Práctico

Para evaluar este trabajo se le proveerá al grupo diversos escenarios de simulación y se validará el correcto desempeño del sistema ante los casos planteados, pudiendo también el grupo desarrollar sus propios casos para realizar sus pruebas previas a la presentación.

Se proveerá también durante el transcurso del cuatrimestre archivos de datos con valores de ejemplo y los resultados esperados.

Descripción General



El sistema consiste en una simulación de un juego de Pokémon, mediante la cual representaremos un pequeño sistema operativo capaz de planificar procesos asignándole recursos, administrar datos mediante un sistema de archivos, y comunicar sus componentes a través de una conexión de red.

Contaremos con distintos tipos de procesos para plantear la analogía: Entrenadores, Mapas y PokéDex. El proceso Entrenador se encargará de cumplir una Hoja de Viaje, que contendrá la información de distintos Mapas que deberá recorrer, junto con distintos Pokémons que deberá atrapar en cada uno, para cumplir sus objetivos.

Por su parte, distintos Entrenadores podrán jugar un mismo Mapa de forma concurrente. Será responsabilidad de cada Mapa administrar y planificar las operaciones de cada uno de los procesos Entrenador, implementando los distintos algoritmos de planificación que vemos en la teoría.

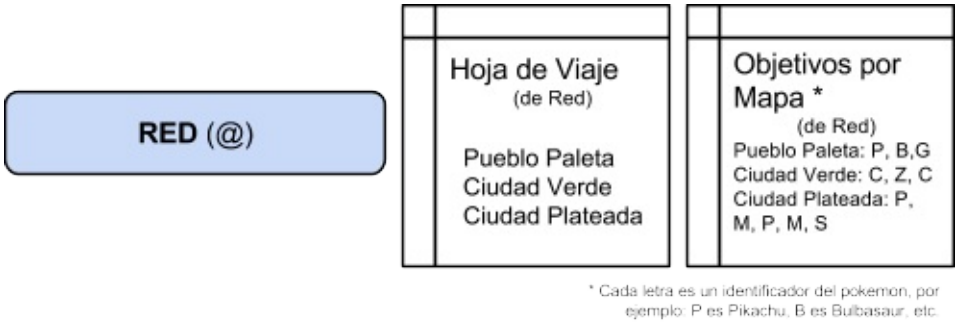
El proceso PokeDex estará encargado de administrar los datos del sistema. Tendrá información de cada uno de los Pokémon disponibles y la información de cada uno de los Entrenadores.

Este proceso implementará un Sistema de Archivos para poder realizar el almacenamiento. Al tener varios Entrenadores jugando en un Mapa, podrían darse situaciones donde un Entrenador no pueda cumplir su objetivo, debido a que la instancia requerida esté retenida por otro Entrenador, generando una situación de interbloqueo (deadlock). En ese caso se deberá resolver el conflicto con una *Batalla Pokémon*.

Proceso Entrenador

Como vimos anteriormente el objetivo del proceso Entrenador es el de completar su Hoja de Viaje. Existirá una instancia de este proceso por cada entrenador que se encuentre participando del sistema. **No hay un límite de entrenadores ni un momento definido donde el entrenador puede ingresar o abandonar la simulación.**

Además de tener como atributos un nombre, un identificador y una cantidad inicial de vidas, cada entrenador tendrá asociada su *Hoja de Viaje*. La misma será una lista de los mapas que deberá completar de manera **secuencial y ordenada**. Tendrá definido también los *objetivos por mapa*: una **lista ordenada** de los Pokémons que deberán ser obtenidos para poder completar dicho Mapa.



El proceso Entrenador obtendrá de la línea de comandos su nombre y la ruta del Pokédex, por ejemplo: `./entrenador Ash /mnt/pokedex`

En el Pokédex existirá un subdirectorio con el nombre de cada Entrenador, por lo que el proceso deberá leer su archivo de metadata del Pokédex y así conocer su Hoja de Viaje.

La Hoja de Viaje especificará la lista de Mapas que el Entrenador deberá recorrer. Para comenzar su aventura Pokémon, el Entrenador obtendrá la dirección IP y puerto del primer Mapa leyendo el correspondiente *archivo de metadata*. Una vez establecida la conexión, el Entrenador enviará al Mapa la primera de sus solicitudes de operación. Cada vez que obtenga una respuesta del Mapa, el Entrenador realizará la siguiente solicitud que tenga, hasta cumplir los objetivos del Mapa.

Dependiendo del estado en que se encuentre, el Entrenador enviará alguno de estos pedidos al Mapa:

1. **Solicitará la ubicación** de la PokeNest del próximo Pokémon que desea obtener, en caso de aún no conocerla.
2. **Avanzará una posición** hacia la siguiente PokeNest[1], informando el movimiento al Mapa, en caso de aún no haber llegado a ella.
3. Al llegar a las coordenadas de una PokeNest, **solicitará atrapar un Pokémon**.

Cuando reciba la confirmación de captura del último Pokemon del Mapa, el Entrenador **copiará la medalla del Mapa a su directorio**, se desconectará del mismo, y procederá a repetir toda la operatoria con el siguiente Mapa de su Hoja de Viaje.

Una vez cumplidos los objetivos del último Mapa de su Hoja de Viaje, el Entrenador se habrá convertido en un *Maestro Pokémon*. El proceso Entrenador informará el logro por pantalla, indicando el Tiempo Total que le tomó toda la aventura, cuánto tiempo pasó bloqueado en las PokeNests, en cuántos Deadlocks estuvo involucrado, y cuántas veces Murió durante la hazaña (ver apartado siguiente).

Muerte del Entrenador y vidas

Un Entrenador puede ser elegido como víctima en el caso de generarse un interbloqueo (más información en la sección “Mapa”). En este caso, el Entrenador mostrará por pantalla el motivo de su muerte, borrará todos los archivos en su Directorio de Bill, se desconectará del Mapa y este le expropiará todos los Pokémons que había capturado, pudiendo estos ser otorgados a algún otro Entrenador en espera.

En caso de tener vidas disponibles, el Entrenador se descontará una vida y volverá a conectarse al Mapa para reiniciar su objetivo. Además, el Entrenador 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 Entrenador en cada uno de los Mapas no se verá afectado, exceptuando el caso en que no tuviera vidas disponibles.

Si no le quedaran vidas disponibles, el Entrenador deberá mostrar un mensaje preguntando al usuario si desea reiniciar el juego, informando también la cantidad de reintentos que ya se realizaron. De aceptar, el Entrenador incrementará su contador de reintentos y reiniciará su Hoja de Viaje, **borrando también todas las medallas y Pokémons conseguidas hasta el momento**. En caso negativo, el Entrenador se cerrará, abandonando el juego. Adicionalmente, el proceso Entrenador 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 Entrenador abandonó el juego.

Archivos y directorios del Entrenador

Medallas

- Nombre: `medallas`
- Tipo: Directorio

- Descripción: Ubicación donde el Entrenador copiará la medalla correspondiente a cada Mapa que concluya
- Ruta: `/Entrenadores/[nombre]/medallas`

Directorio de Bill

- Nombre: `Dir de Bill`
- Tipo: Directorio
- Descripción: Ubicación donde el Entrenador copiará el archivo de metadata de cada Pokémon que capture
- Ruta: `/Entrenadores/[nombre]/Dir de Bill/`

Metadata

- Nombre: `metadata`
- Tipo: Archivo
- Descripción: configuración del Entrenador, donde está almacenada información como su Hoja de Viaje
- Ruta: `/Entrenadores/[nombre]/metadata`

El archivo de metadata del proceso Entrenador contendrá al menos los siguientes parámetros:

- **Símbolo**: caracter que representará visualmente al Entrenador en el Mapa
- **Hoja de Viaje**: lista de los Mapas que debe completar el Entrenador
- **Objetivos por Mapa**: por cada Mapa de su Hoja de Viaje, una lista de los Pokémons que deberá obtener de forma ordenada para completarlo
- **Vidas**: cantidad de vidas restantes
- **Reintentos realizados**: cantidad de veces que el Entrenador reintentó el juego

Restricciones del archivo de Metadata

- No hay una cantidad definida de Mapas, de objetivos por Mapa o de Pokémons por Mapa y por juego
- Los identificadores de Pokémon pueden reutilizarse en niveles diferentes
- Un Pokémon ubicado fuera de los márgenes del área de juego se considera un error
- Las PokeNests deben estar espaciadas al menos por dos posiciones en cada eje
- Dos Pokémons del mismo tipo de manera consecutiva en el objetivo de un Mapa se considera un error de sintaxis

> Ejemplo:
>
> `obj[Mapa8]=[P,P,F]` (error!)
>
> `obj[Mapa8]=[P,F,P,F,P,F]` (ok!)

Ejemplo de archivo de Metadata de un Entrenador

```
nombre=Red
simbolo=@
hojaDeViaje=[PuebloPaleta,CiudadVerde,CiudadPlateada]
obj[PuebloPaleta]=[P,B,G]
obj[CiudadVerde]=[C,Z,C]
obj[CiudadPlateada]=[P,M,P,M,S]
vidas=5
reintentos=0
```

---

[1] El movimiento del personaje deberá ser, cuando fuera posible, alternando eje X y eje Y (uno y uno) de a una unidad por vez. No existe posibilidad de realizar movimientos en diagonal.

Proceso Mapa

Cada instancia del proceso Mapa será encargado de gestionar la interacción entre los diversos procesos Entrenador y los Pokémons disponibles en el Mapa.

Para lograrlo, será indispensable el uso de estructuras que reflejen qué Entrenadores están listos para moverse, cuáles se encuentran esperando capturar un Pokémon y cuáles han finalizado anormalmente (víctimas de interbloqueos, finalizados por muerte, etc).

Los diversos Entrenadores ingresarán al mapa en busca de determinados Pokémons, los cuales se encuentran en las diversas PokeNest. El Entrenador solicitará la ubicación de la PokeNest que tiene el Pokémon que está buscando y se dirigirá al mismo. Al llegar, solicitará una instancia del Pokémon al Mapa el cual le será otorgado siempre y cuando hubiera disponibles. Cada PokeNest tendrá Pokémons de una única especie.

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

El proceso Mapa al ser ejecutado **obtendrá de la línea de comandos su nombre y la ruta del Pokédex**. En el directorio `/Mapas` del Pokédex existirá un subdirectorio con el nombre de cada Mapa, por lo que **el proceso deberá leer su archivo de metadata y los diversos subdirectorios dentro del directorio PokeNests para así poder conocer los recursos que tiene disponibles y crearlos**.

Hilo Planificador

El hilo Planificador será el encargado de contestar las peticiones de los Entrenadores que actúan dentro del Mapa, ordenándolos **según un algoritmo de planificación de corto plazo Round Robin o SRDF** (ver Algoritmos de Planificación).

Gestionará una cola de Listos y una cola de Bloqueados. Ante cada modificación de estas colas, el Planificador **deberá informar por archivo de log la modificación realizada, y la lista de los Entrenadores en cada una de las colas**.

Mientras haya Entrenadores listos, el Planificador seleccionará a cuál atenderá según el algoritmo de planificación activo. Una vez seleccionado, el Planificador contestará una por una las peticiones que el Entrenador le haga, hasta que este solicite capturar un Pokemon, o hasta que el algoritmo de planificación indique expropiarlo.

Las posibles operaciones a atender son:

1. **Conocer la ubicación de una PokeNest** - El Mapa contestará al Entrenador las coordenadas en que se encuentra la PokeNest de un Pokémon determinado
2. **Moverse en alguna dirección** - El Mapa registrará dicho movimiento y contabilizará el uso de una unidad de tiempo al Entrenador
3. **Capturar un Pokémon** - El Planificador moverá a dicho personaje a la cola de bloqueados. Descartará, si quedara, el quantum de tiempo restante del Entrenador y planificará al siguiente que se encuentre listo.

Eventualmente, el Planificador podrá detectar que un Entrenador se desconectó (cumplió los objetivos del Mapa, o murió). Cuando esto ocurra, deberá liberar todos los Pokémons que éste había capturado y, si correspondiera, otorgarlos a los Entrenadores bloqueados por ellos, desbloqueándolos. Al mismo tiempo, el Planificador eliminará al Entrenador de sus listas de Planificación.

El hilo Planificador tendrá un **algoritmo de planificación, valor de quantum**[3] **y tiempo de retardo entre turnos**[4] que será notificado por el Mapa al iniciar.Cuando reciba la señal `SIGUSR2`, el proceso Mapa deberá releer su archivo de Metadata, actualizando estos parámetros durante la ejecución. Junto con los parámetros del Planificador mencionados, el Mapa deberá actualizar, también, el **tiempo de chequeo de interbloqueo**, descrito a continuación.

Algoritmos de Planificación

El Planificador podrá ir alternando el algoritmo de planificación entre Round Robin (con quantum configurable) y Shortest Remaining Distance First (SRDF) a medida que se relea su archivo de configuración. Es válido esperar a finalizar la ráfaga de ejecución actual antes de efectivizar el cambio de algoritmo.

El SRDF es un algoritmo que sigue dos reglas:

1. Atiende una única operación del primer Entrenador Listo que no conozca su distancia a la próxima PokeNest (ya sea por recién haberse conectado, o por haber recién capturado un Pokemon); o, si no hubiera ninguno,
2. Atiende todas las operaciones que necesite el Entrenador Listo con menor distancia a su PokeNest destino, hasta que éste se bloquee por solicitar capturar un Pokemon.

Cualquiera de estas dos opciones cuenta como una ráfaga de ejecución, por lo que, para poder volver a ejecutar una siguiente ráfaga, el Entrenador tendrá primero que pasar por el final de la cola de Listos. En otras palabras, si el algoritmo de planificación cambiara de SRDF a RR mientras un Entrenador está consultando la ubicación de su próxima PokeNest al haber sido elegido por la regla 1 del SRDF, ese Entrenador irá al final de la cola de Listos, afectando su orden según Round Robin.

La distancia de un Entrenador a una PokeNest se mide en cantidad de movimientos que debería realizar el Entrenador - es decir, sin contar pasos en diagonal.

Detección de deadlock

Cada instancia del proceso Mapa deberá analizar periódicamente la existencia de un interbloqueo mediante una frecuencia de chequeo llamada **tiempo de chequeo de interbloqueo**. En caso de detectarlo, informará por archivo de log dicha situación, **indicando los Entrenadores involucrados en el interbloqueo y las tablas utilizadas para su detección**.

Adicionalmente, si se encontrara activado por archivo de configuración el modo *batalla*, seleccionará y mostrará por archivo de log el Entrenador que fue seleccionado como víctima para resolver dicha situación.

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

Resolución del deadlock

En caso de detectar un deadlock, el Mapa de manera ordenada según el tiempo de ingreso al mapa les notificará a los Entrenadores involucrados en el interbloqueo que deben elegir su Pokémon más fuerte, es decir, al Pokémon de mayor nivel, para la batalla.

Luego, el mapa deberá efectuar la simulación de una Batalla Pokémon enfrentando entre sí a los Pokémons, de a dos, dado el orden según el tiempo de ingreso del Entrenador.

El Pokémon perdedor de la batalla, enfrentará en un Encuentro al Pokémon del Entrenador siguiente. Finalmente, el Entrenador cuyo Pokémon haya perdido en la última batalla deberá ser elegido como víctima para solucionar el deadlock. Cada vez que se efectúe un Encuentro, el mapa deberá notificar a los Entrenadores el resultado de dicho encuentro.

> Ejemplo:
>
> Entrenadores ordenados según tiempo de ingreso:
>
> Entrenador 1: Pikachu Nivel 10
>
> Entrenador 2: Squirtle Nivel 5
>
> Entrenador 3: Rhyhorn Nivel: 100
>
> Pikachu Vs Squirtle => Perdedor: Squirtle
>
> Squirtle Vs Rhyhorn => Perdedor: Squirtle
>
> => El Entrenador 2 será elegido como víctima.

Para simplificar el algoritmo de batalla Pokémon, **la cátedra proveerá una biblioteca[5] para la realización del Encuentro entre 2 Pokémons** y evitar el cálculo innecesario de los factores de efectividad de los ataques. Un ejemplo práctico de aplicación de la misma se detalla en el Anexo III - Algoritmo de Batalla.

Atrapar un Pokémon

Cada PokeNest tendrá una cantidad limitada de Pokémons de una determinada especie. Estos se representarán como archivos en un directorio.

Cuando un entrenador atrapa un Pokémon, copiará como prueba de dicha operación el archivo de metadata a su Directorio de Bill.

Dibujado del Mapa

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

Archivos y directorios del Mapa

Metadata

- **Nombre**: `metadata`
- **Tipo:**: Archivo
- **Descripción**: archivo donde están almacenados los parámetros del mapa
- **Ruta**: `/Mapas/[nombre]/metadata`
- **Ejemplo de Ruta: `/Mapas/Ciudad Paleta/metadata`
- **Contenido**:
  - Tiempo de chequeo de interbloqueo (en ms)
  - Batalla (on/off)
  - Algoritmo de planificación
  - Quantum
  - Retardo entre quantums (en ms)
  - IP/Puerto en que recibirá conexiones de Entrenadores
- **Ejemplo**:

```
TiempoChequeoDeadlock=10000
Batalla=1
algoritmo=RR
quantum=3
retardo=500
IP=127.0.0.1
Puerto=5001
```

Medalla

- **Nombre**: Medalla del Mapa
- **Tipo:**: Archivo
- **Descripción**: medalla del Mapa, que se otorgará a cada Entrenador que lo concluya
- **Ruta**: `/Mapas/[nombre]/medalla-[nombre].jpg`
- **Ejemplo de Ruta: `/Mapas/Ciudad Paleta/medalla-Ciudad Paleta.jpg`
- **Contenido**: no determinado

PokeNest

- **Nombre**: PokeNest
- **Tipo:**: Directorio
- **Descripción**: directorio que representa y contiene la información de una PokeNest. Puede haber más de uno por Mapa.
- **Ruta**: `/Mapas/[nombre]/PokeNests/[nombre-de-PokeNest]/`
- **Ejemplo de Ruta: `/Mapas/Ciudad Paleta/PokeNests/Pikachu/`

Metadata de un PokeNest

- **Nombre**: `metadata`
- **Tipo:**: Archivo
- **Descripción**: información de la PokeNest
- **Ruta**: `/Mapas/[nombre]/PokeNests/[nombre-de-PokeNest]/metadata`
- **Ejemplo de Ruta**: `/Mapas/Ciudad Paleta/PokeNests/Pikachu/metadata`
- **Contenido**:
  - Tipo de Pokemon
  - Posición X;Y de la PokeNest
  - Caracter identificador
- **Ejemplo de contenido**:

```
Tipo=Electrico
Posicion=23;18
Identificador=P
```

Metadata de un Pokemon

- **Nombre**: Metadata de un Pokemon
- **Tipo:**: Archivo
- **Descripción**: información de un Pokemon dentro de la PokeNest
- **Ruta**: `/Mapas/[nombre]/PokeNests/[PokeNest]/[PokeNest]NNN.dat`
- **Ejemplos de Ruta**:
  - `/Mapas/Ciudad Paleta/PokeNests/Pikachu/Pikachu001.dat`
  - `/Mapas/Ciudad Paleta/PokeNests/Pikachu/Pikachu002.dat`
  - `/Mapas/Ciudad Paleta/PokeNests/Bulbasaur/Bulbasaur001.dat`
- **Contenido**:
  - Nivel del Pokemon
  - Imagen del Pokemon en Ascii Art
- **Ejemplo de Contenido**:

```
Nivel=33
[Ascii Art]
```

Restricciones del Mapa

- No hay una cantidad definida de PokeNests. El proceso Mapa al iniciar debe recorrer el árbol y crear las instancias correspondientes.
- No se instanciarán dos Mapas con el mismo nombre en el sistema.
- Las PokeNests deberán estar dentro de los márgenes del Nivel y no podrán superponerse.
- Los archivos necesarios para ejecutar una simulación serán provistos siempre en su

totalidad y sin errores de sintaxis o semántica. Es correcto considerar un error abortivo la carencia o error de alguno de ellos.

---

[3] El valor de quantum aplica únicamente para el algoritmo de RR

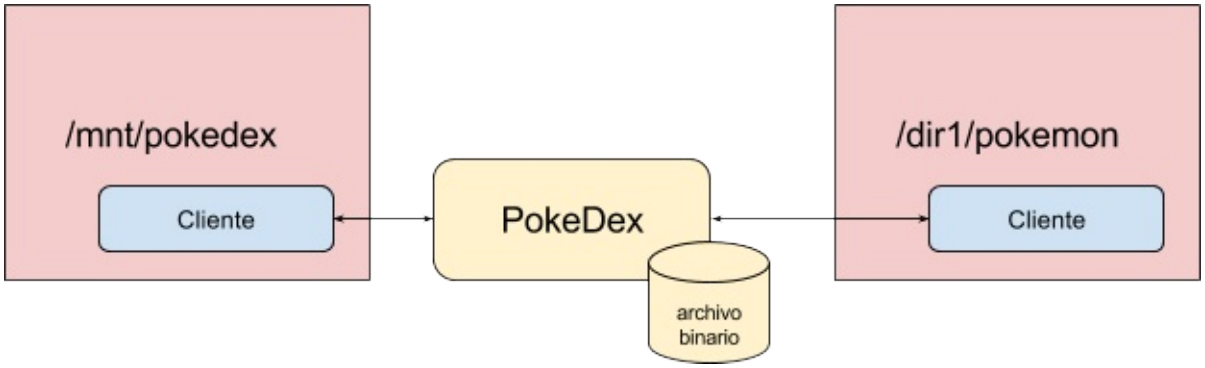
[4] Valor en milisegundos que cada planificador de manera independiente deberá esperar entre cada asignación de turno

[5] https://github.com/sisoputnfrba/so-pkmn-utils

Proceso PokeDex

La configuración del sistema junto con cierta información de estado estará almacenada de manera centralizada en el proceso PokeDex, siguiendo la estructura de un árbol de directorios de un filesystem. Todos los procesos del sistema accederán a dicha información mediante un punto de montaje y la actualizarán cuando corresponda utilizando su instancia del "Cliente del PokeDex".

Al ser el PokeDex un filesystem distribuído, este componente estará compuesto por un proceso servidor que atiende las peticiones de manera concurrente y mantiene las estructuras administrativas, y un cliente que permitirá montar dicho filesystem en los diversos equipos remotos.



Servidor PokeDex

Este proceso gestionará un Filesystem en formato Otro Sistema Académico de Archivos (OSADA) (ver Anexo I - Otro Sistema Académico de Archivos) almacenado en un archivo binario y se lo presentará a los diversos procesos PokeDex Cliente que se conecten.

La implementación deberá permitir que se pueda:

- Leer archivos
- Crear archivos
- Escribir y modificar archivos
- Borrar archivos
- Crear directorios y subdirectorios.
- Borrar directorios vacíos
- Renombrar archivos Se deberá tener especial cuidado en proteger con semáforos las estructuras administrativas del sistema de archivos ya que se pretende que el proceso pueda recibir solicitudes de manera concurrente de varios procesos Pokedex Cliente.

Proceso Cliente del PokeDex

Este proceso funcionará como interfaz entre las solicitudes que un proceso o el Sistema Operativo quieran realizar sobre el punto de montaje y el PokeDex.

Al ser iniciado obtendrá la IP y Puerto del PokeDex por variable de entorno, se conectará y quedará montado en el directorio recibido por argumento.

Todas aquellas solicitudes que se realicen sobre ese punto de montaje, por ejemplo listar archivos, las reenviará al Pokedex para que este le devuelva el resultado correspondiente, de manera análoga al cliente de un filesystem basado en la nube como Dropbox o Google Drive, teniendo en cuenta que no se almacenará ningún contenido en la computadora que ejecute el cliente.

Para simplificar el desarrollo de la interacción con el sistema operativo el alumno deberá implementar el cliente utilizando la biblioteca FUSE la cual ejecutará en modo single thread.

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`)

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 - Entrega Obligatoria

Tiempo estimado: 2 semanas

Fecha: 10 de Septiembre

Hitos

- Estar familiarizado 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 recibir conexiones de diversos clientes y transmitir mensajes paquetizados (Mapa)
- Implementar la biblioteca `libnivel.so` y desarrollar un proceso que permita dibujar elementos en pantalla (Mapa) investigando el código de ejemplo
- Desarrollar un código capaz de reconocer las estructuras administrativas de un filesystem OSADA

Recursos necesarios

- 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: 3 semanas

Fecha: 01 de Octubre

Hitos

- Comprender la lógica de FUSE y trabajar con el ejemplo.
- Desarrollar las funciones para leer el contenido del árbol de directorios del filesystem OSADA.
- Desarrollar la lógica del proceso Entrenador conectándose a los diversos procesos Mapa e interactuando con los mismos. Implementar los algoritmos de planificación.
- Desarrollar las estructuras del Mapa para manejar los Pokemones disponibles y asignados y los diversos estados de un Entrenador.
- Crear y representar mediante `libnivel.so` los elementos y personajes del Mapa.
- Activar rutinas en los procesos mediante el uso de señales.

Recursos necesarios

- SO FUSE Example - Link
- Guia de FUSE por Joseph J. Pfeiffer (en inglés) - Link
- Traducción de la Guía de FUSE de Joseph J. Pfeiffer, por Matías García Isaia - Link
- Linux POSIX Threads (pthread) - Link
- Gcc, Makefile - Link parte 1 y Link parte 2

Tercer checkpoint

Tiempo estimado: 2 semanas **Fecha**: 15 de Octubre

Hitos

- Desarrollar el hilo que detecta interbloqueos (deadlocks) en el Mapa
- Desarrollar las operaciones de escritura y lectura de archivos de OSADA
- Integrar `readdir()` y `getattr()` en PokeDex
- Crear el proceso Cliente de Pokedex que permita realizar por un socket `readdir()` y `getattr()` sobre el PokeDex mediante mensajes paquetizados
- Desarrollar la lógica de muerte de un Entrenador y reiniciado de un Mapa

Cuarto checkpoint

Tiempo Estimado: 2 semanas

Fecha: 29 de Octubre

Hitos

- Iniciar fase de testing de integración y stress
- Integrar las operaciones de lectura y escritura de archivos al Pokedex
- Permitir realizar operaciones de lectura y escritura desde el cliente de PokeDex

Entregas

Entrega final: 19 de Noviembre

Primer recuperatorio: 3 de Diciembre

Segundo recuperatorio: 17 de Diciembre

> *Todas las fechas son estimativas y pueden ser modificadas con su debida notificación*

Anexo I - Otro Sistema Académico de Archivos (OSADA)

El OSADA 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 asemejando características de sistemas de archivos modernos.

- Tamaño máximo de disco soportado: 4GB
- Tamaño máximo de archivo: 4GB
- Soporte de directorios y subdirectorios
- Tamaño máximo de nombre de archivo: 17 caracteres
- Cantidad máxima de archivos/directorios: 2048

Características técnicas

- Tamaño de bloque: 64 bytes
- Tamaño de la tabla de archivos: 1024 bloques
- Tabla de bloques libres: Bitmap
- Estructura de gestión: Tabla de asignación. Bloques enlazados.

Arquitectura

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

Para un disco de tamaño T [bytes] y el BLOCK_SIZE de 64 las estructuras se definen de esta manera:

F: Tamaño en Bloques del Filesystem

> Por ejemplo: si tengo un filesystem de 1,584 KB tengo 25344 bloques de 64B… ¿Cuánto ocuparía el bitmap?
>
> Hagamos cuentas: Para representar 25344 bloques necesito 25344 bits. Esa cantidad termina resultando en 3168 bytes (Si 8 bits = 1 byte, entonces 25344 bits = 25344/8 = 3168 bytes).
>
> 3168 bytes resultan en 49.5 bloques. Como no podemos ocupar “49 bloques y medio” el bitmap ocupará 50 bloques.

Header

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

Campo	Tamaño	Valor
Identificador	7 bytes	`OsadaFS` (sin `\0`)
Versión	1 byte	1
Tamaño del FS [bloques]	4 bytes	T / BLOCK_SIZE
Tamaño del Bitmap [bloques]	4 bytes	N: Tamaño del FS [bloques] / 8 / BLOCK_SIZE
Inicio Tabla Asignaciones [bloque]	4 bytes	1 + N + 1024
Tamaño Datos [bloques]	4 bytes	F - 1 - N - 1024 - A
Relleno	40 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 Archivos

La tabla de archivos es un array de tamaño fijo de 2048 posiciones de estructuras de tipo `osadaFile`. Cada estructura `osadaFile` consta de los siguientes campos:

Campo	Tamaño	Valor
Estado	1 byte	0: Borrado, 1: Ocupado, 2: Directorio
Nombre de archivo	17 bytes	Nombre del archivo
Bloque padre	2 bytes	Posición del array de la tabla de archivos donde está almacenado el directorio padre o `0xFFFF` si está en el directorio raíz
Tamaño del archivo	4 bytes	Tamaño total del archivo en bytes
Fecha última modificación	4 bytes	Timestamp de la última modificación[6]
Bloque inicial	4 bytes	Ubicación del primer elemento en la tabla de asignaciones

Tabla de Asignaciones

La tabla de asignaciones es una estructura que permite identificar los bloques que componen cada archivo.

Es un array de enteros donde cada posición representa un bloque de datos y el valor almacenado en dicha posición indica el próximo bloque.

El bloque de inicio de cada archivo es un campo de la estructura `osadaFile` de la tabla de archivos. Supongamos que sabemos que un archivo empieza en el bloque 8, vamos a la posición 8 de la tabla de asignaciones y ahí nos dirá el próximo bloque (supongamos el 12)

y sucesivamente.

Se ve en el diagrama a continuación que nuestro archivo está compuesto por los bloques de datos 8, 12, 13 y 15. *Recuerde que los arrays comienzan en cero.*

También a modo de ejemplo, si se nos indicara que un archivo empieza en el bloque 17, podríamos inferir que el archivo se compone de los bloques de datos, 17, 4 y 2

Ejemplo:

		FFFFFFFF		2
			12	
		13	15	
FFFFFFFF		4		

Osada Utils - Herramientas y estructuras

Se le proveerán al alumno dos herramientas para simplificar el desarrollo del trabajo práctico:

- `osada-format` : Recibe como parámetro un archivo e inicializa un filesystem OSADA dentro del mismo.
- `osada-dump` : Imprime por pantalla el estado de las estructuras administrativas de un filesystem OSADA.

Además, se publicarán, también, las definiciones de las estructuras básicas de un filesystem OSADA, a modo de cumplimentar la especificación. Para acceder a estos, consultar el repositorio https://github.com/sisoputnfrba/osada-utils

---

[6] Ver función `time()`

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 - Algoritmo de Batalla

Para determinar el ganador del Encuentro de una Batalla se diseñó un algoritmo simplificado, dada la complejidad de que tienen las batallas Pokémon. Dicho algoritmo consistirá en:

> Vence el Pokémon que tenga mayor en nivel, más un factor que dependerá de la efectividad del ataque ante el Pokémon contrario.

Dicho factor de efectividad será un valor entero correspondiente a los siguientes:

- `5` : para ataques super efectivos.
- `0` : para ataques comunes.
- `-5` : para ataques poco efectivos.
- `-20` : para ataques que no causen daño (ej, eléctrico a tierra).

Se deberá tener en cuenta lo siguiente:

- El tipo de los ataques de un Pokémon serán del tipo principal del Pokémon, a fines de simplificar el algoritmo.
- Si el Pokémon contrario tiene 2 tipos, se realizará la suma del factor de efectividad de cada tipo, con la sola excepción de que algún factor sea por *no causar daño*. En dicho caso *el factor de efectividad total será de no causar daño*, es decir, -20.

En caso de empate, ganará el Pokémon de mayor nivel (y en caso de empate, el ganador será el primer Pokémon, dado que empezó a atacar primero).

Por ejemplo:

> Pikachu [Electrico] Nivel: 30 Vs Rhyhorn [Tierra/Roca] Nivel: 6
>
> **Factor de Efectividad de Pikachu**: 30 + -20 (Sin Daño al Tierra y Daño Normal a Roca) = **10**
>
> **Factor de Efectividad de Rhyhorn**: 6 + 5 (Daño Doble) = **11** >
>
> Gana **Rhyhorn**

La cátedra implementó dicho algoritmo en una biblioteca compartida, que además de permitir efectuar una batalla, permite la creación de Pokémons a modo de simplificar aún más el funcionamiento de la biblioteca.

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

> https://github.com/sisoputnfrba/so-pkmn-utils.git

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