UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ingeniería en Sistemas de Información

Delibird

Enviando mensajes sin salir de casa ***#QuedateEnCasa***

Cátedra de Sistemas Operativos

Trabajo práctico Cuatrimestral

-1C2020 -
Versión 1.0

Versión de Cambios

Índice

Objetivos y Normas de resolución

Objetivos del Trabajo Práctico

Mediante la realización de este trabajo se espera que el alumno:

- Adquiera conceptos prácticos del uso de las distintas herramientas de programación e interfaces (APIs) que brindan los sistemas operativos.
- Entienda aspectos del diseño de un sistema operativo.
- Afirme diversos conceptos teóricos de la materia mediante la implementación práctica de algunos de ellos.
- Se familiarice con técnicas de programación de sistemas, como el empleo de makefiles, archivos de configuración y archivos de log.
- Conozca con grado de detalle la operatoria de Linux mediante la utilización de un lenguaje de programación de relativamente bajo nivel como C.

Características

- Modalidad: grupal (5 integrantes +- 0) y obligatorio
- Tiempo estimado para su desarrollo: 90 días
- Fecha de comienzo: 03 de Abril
- Fecha de primera entrega: 25 de Julio ***(fecha tentativa a la espera de actualización de calendario académico)***
- Fecha de segunda entrega: 1 de Agosto ***(fecha tentativa a la espera de actualización de calendario académico)***
- Fecha de tercera entrega: 22 de Agosto ***(fecha tentativa a la espera de actualización de calendario académico)***
- Lugar de corrección: Laboratorio de Medrano

Evaluación del Trabajo Práctico

El trabajo práctico consta de una evaluación en 2 etapas.

La primera etapa consistirá en las pruebas de los programas desarrollados en el laboratorio. Las pruebas del trabajo práctico se subirán oportunamente y con suficiente tiempo para que los alumnos puedan evaluarlas con antelación. Queda aclarado que para que un trabajo práctico sea considerado evaluable, el mismo debe proporcionar registros de su funcionamiento de la forma más clara posible.

La segunda etapa se dará en caso de aprobada la primera y constará de un coloquio, con el objetivo de afianzar los conocimientos adquiridos durante el desarrollo del trabajo práctico y terminar de definir la nota de cada uno de los integrantes del grupo, por lo que se recomienda que la carga de trabajo se distribuya de la manera más equitativa posible.

Cabe aclarar que el trabajo equitativo no asegura la aprobación de la totalidad de los integrantes, sino que cada uno tendrá que defender y explicar tanto teórica como prácticamente lo desarrollado y aprendido a lo largo de la cursada.

La defensa del trabajo práctico (o coloquio) consta de la relación de lo visto durante la teoría con lo

implementado. De esta manera, una implementación que contradiga a lo visto en clase o lo escrito en el documento ***es motivo de desaprobación del trabajo práctico***.

Deployment y Testing del Trabajo Práctico

Al tratarse de una plataforma distribuida, los procesos involucrados podrán ser ejecutados en diversas computadoras. La cantidad de computadoras involucradas y la distribución de los diversos procesos en estas será definida en cada uno de los tests de la evaluación y **es posible cambiar la misma en el momento de la evaluación**. Es responsabilidad del grupo automatizar el despliegue de los diversos procesos con sus correspondientes archivos de configuración para cada uno de los diversos tests a evaluar.

Todo esto estará detallado en el documento de pruebas que se publicará cercano a la fecha de Entrega Final. Archivos y programas de ejemplo se pueden encontrar en el repositorio de la cátedra.

Finalmente, recordar la existencia de las Normas del Trabajo Práctico donde se especifican todos los lineamientos de cómo se desarrollará la materia durante el cuatrimestre.

Aclaraciones

Debido al fin académico del trabajo práctico, los conceptos reflejados son, en general, versiones simplificadas o alteradas de los componentes reales de hardware y de sistemas operativos modernos, a fin de resaltar aspectos de diseño.

Invitamos a los alumnos a leer las notas y comentarios al respecto que haya en el enunciado, reflexionar y discutir con sus compañeros, ayudantes y docentes al respecto.

Abstract

El objetivo del trabajo práctico consiste en desarrollar una solución que permita la simulación de un sistema distribuido que utiliza el concepto de Colas de Mensajes (o Message Queue).

Los componentes incluidos dentro de la arquitectura del sistema deberán trabajar en conjunto para la planificación y ejecución de distintas operaciones, entre las que se encuentran, por ejemplo: leer y escribir valores. Las operaciones que conforman estos mensajes están asociadas y vinculadas al mundo de Pokémon.

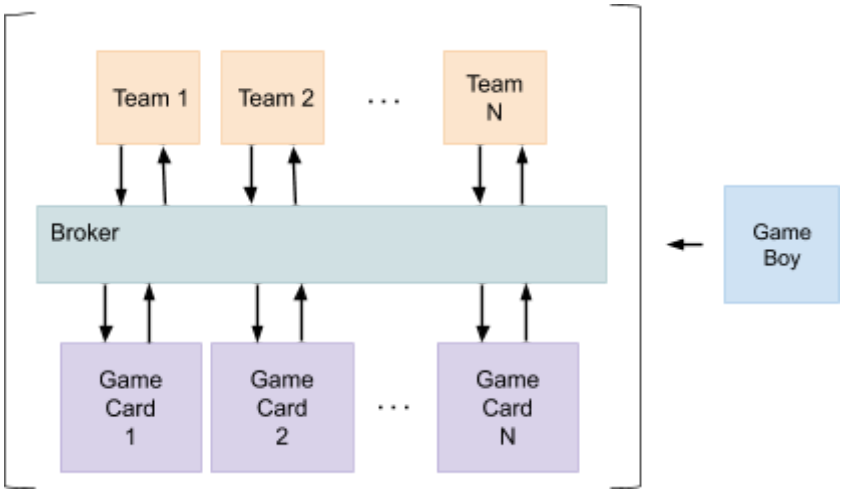
Message Queue (a partir de ahora MQ) es una técnica de software utilizada para la comunicación entre procesos (IPC) basada en el concepto de Colas (Queue). En ella, distintos procesos dejan mensajes y otros los leen de manera asincrónica. De esta manera, se permite el desarrollo de un sistema completamente distribuido, escalable e independiente.

Los componentes del sistema serán:

- Un proceso publisher que ingrese mensajes al sistema (Game Boy).
- Un proceso administrador de las Colas de Mensajes (Broker).
- Procesos que obtengan los mensajes y planifiquen en función de ellos (Team).
- Procesos filesystem que se encarguen de mantener los archivos en el tiempo (Game Card).

Arquitectura del Sistema

Como dijimos anteriormente el sistema consta de 4 módulos independientes los cuales interactúan entre sí como se muestra en el siguiente diagrama.



El Game Boy será nuestro punto de partida y asimismo, que conocerá y permitirá el envío de mensajes a distintos módulos de nuestro sistema.

El proceso Broker será el encargado de administrar las distintas Colas de Mensajes existentes en el sistema. Para esto, el mismo mantendrá distintas características y funcionalidades propias de un sistema de Cola de Mensajes real, encargándose de mantener, entender y distribuir los distintos mensajes.

El proceso Team contendrá una lista de entrenadores con distintos objetivos de captura de Pokemons. Nuestro proceso deberá planificar a los entrenadores correctamente para que cumplan sus objetivos cooperativamente dentro de un mapa. De esta manera, los Pokemon seran recursos que los distintos entrenadores deberán atrapar moviéndose por el mapa (apropiarse) y luego intercambiar en caso de ser necesario (se produzca un deadlock).

Por último, nuestro proceso Game Card será el encargado de conocer qué Pokemon se encuentran disponibles dentro del mapa y en qué posición está cada uno. Deberá mantener distintos permisos y atribuciones para que solo un proceso pueda acceder al mismo tiempo a él. Al estar este proceso conectado al Broker y asociarse a una cola de mensajes específica nos permite generar redundancia de Game Card y poder replicar con distintas demoras o distintos file system los mismos archivos.

Proceso Broker

Será el encargado de administrar las colas de mensajes de nuestro sistema. Como tal tendrá la responsabilidad de:

1. Administrar los suscriptores (Teams, Game Cards) asociados a las distintas colas.
2. Administrar la recepción, envío y confirmación de todos los mensajes a los múltiples suscriptores.
3. Mantener un registro de los últimos mensajes recibidos de las colas indicadas para futuros suscriptores.
4. Mantener e informar en todo momento los estados de las colas, con sus mensajes y suscriptores.

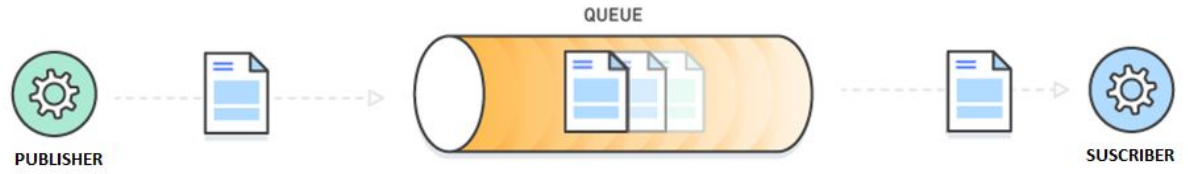
Para explicar esto, primero nos enfocaremos en el conceptos técnicos de qué es una Cola de mensajes (o Message Queue) para luego abordar los aspectos y lineamientos técnicos que tendrá nuestra implementación.

Abstract - Message Queue (MQ)

Las colas de mensajes son software que permiten la comunicación entre procesos (IPC) de manera asíncrona, lo que significa que el emisor y el receptor del mensaje no necesitan interactuar con la cola de mensajes al mismo tiempo.

Cada mensaje colocado en una cola se almacena hasta que el/los destinatarios los recuperen y/o lean. Las colas de mensajes tienen límites implícitos o explícitos sobre el tamaño de los datos que pueden ser transmitidos en un solo mensaje y el número de mensajes que pueden quedar pendientes en la cola.

De esta manera, vamos a tener varios procesos que van a funcionar como publicadores (o Publishers) que van a ser los encargados de dejar mensajes en una Cola de mensajes en particular mientras que otros procesos van a funcionar como suscriptores (o Suscribers) que van a recibir aquellos mensajes que lleguen a las colas de mensajes donde estén suscriptos.



En el ámbito de nuestro trabajo practico implementaremos una serie de Colas de mensajes que utilizaremos para distintos propósitos que explicaremos más adelante. Dichas colas de mensajes deben cumplir con los siguientes aspectos técnicos:

1. Durabilidad: Todo mensaje debe permanecer en la cola de mensajes hasta que todos los Suscribers lo reciban.

2. Notificación de recepción: Todo mensaje entregado debe ser confirmado por cada Suscriptor para marcarlo y no enviarse nuevamente al mismo.
3. Mantenibilidad: Cada cola de mensaje debe mantener su estado y borrar aquellos mensajes que ya fueron entregados a todos sus suscriptores.
4. Asincronismo: La recepción y notificación de mensajes pueden diferir en el tiempo. No deben notificarse inmediatamente a los componentes suscritos a dicha cola.

Lineamiento e Implementación

El Broker se encarga, como dijimos anteriormente, de la administración de MQ de nuestro sistema, simulando algunos aspectos técnicos de sus implementaciones en la realidad. Por otra parte, se incorporan conceptos de la materia como administración de memoria, mensajería y sincronización.

La funcionalidad principal del Broker es la de administrar las distintas colas de mensajes con sus distintos suscriptores. Para esto, esperará las solicitudes de los distintos módulos para asociarse a las distintas colas que él mismo administra. De esta manera, cada uno de los clientes/módulos se deberá comunicar con él indicando cual es la cola a la que se desea suscribir (En este punto llamaremos al otro módulo como suscriptor).

Una vez informado esto, el Broker dispondrá de una lista de suscriptores por cada cola que el mismo administre. En otra instancia de tiempo, un mensaje llegará con el destino a dicha cola de mensajes y el Broker distribuirá dicho mensaje a los suscriptores (enviará dicho mensaje a cada uno de los suscriptores). Al realizar esto se deberá tener en cuenta lo siguiente:

1. Todo mensaje debe ser cacheado dentro de la memoria interna del Broker.
2. Todo mensaje debe saber a cuales suscriptores fue enviado y si el mismo fue recibido (confirmación, ACK ó *acknowledgement*).
3. Todo mensaje debe tener un identificador unívoco generado por el Broker que debe ser informado al módulo que generó el mismo.

Esta funcionalidad deberá ***ser implementada por medio de multi-hilos***. Esta arquitectura permitirá al Broker poder enviar y transaccionar mensajes en simultáneo a los distintos suscriptores. **Cualquier otra implementación que no esté bajo este concepto será motivo de desaprobación del trabajo práctico.**

Administración de mensajes

Como dijimos anteriormente, el Broker mantendrá una memoria interna en la cual se cachean los últimos mensajes recibidos de las distintas colas de mensajes. En el mismo deberá registrar:

1. Identificador único del mensaje dentro del sistema.
2. El tipo de mensaje (a que cola de mensajes pertenece).
3. Los suscriptores a los cuales ya se envió el mensaje.
4. Los suscriptores que retornaron el ACK del mismo.

Se implementarán dos esquemas de Administración de Memoria: Particiones dinámicas con compactación, y Buddy System (descritos más adelante). Se elegirá por archivo de configuración cual

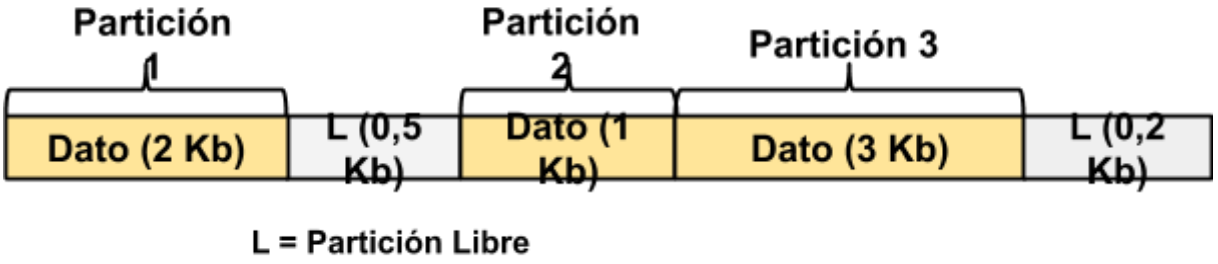
estará activa al iniciar la caché. Para ambos, se definirá por parámetro del Broker el tamaño mínimo de partición y un tamaño máximo (que será el de toda la memoria).

Uno de los requerimientos obligatorios que va a tener el Broker es que una vez inicializado ya no se podrá reservar más memoria dinámica para guardar los datos. Por lo tanto, ***toda la memoria que vaya a ser necesaria*** para el almacenamiento de los datos deberá ser pre-reservadas dinámicamente en el inicio. Solo se deberá guardar en la porción de datos el contenido del mensaje dejando el tipo, identificador y demás flags del mismo en estructuras auxiliares.

Cada vez que un proceso se suscriba a una cola de mensajes **deberá recibir todos los mensajes cacheados de dicha cola de mensajes**.

Particiones dinámicas con compactación[1]

En este esquema, se reservará una porción de memoria por cada valor almacenado, del tamaño exacto de dicho valor. De esta manera, la cantidad de particiones y su tamaño es variable. Por ejemplo:



En dicho ejemplo, en el caso de almacenar un nuevo valor de 0,2 Kb en el espacio de la primera partición libre, se tendría una nueva "partición 4" de 0,2 Kb, y al lado una nueva partición libre de 0,3 Kb.

Procedimiento para almacenamiento de datos

1. Se buscará una partición libre que tenga suficiente memoria continua como para contener el valor. En caso de no encontrarla, se pasará al paso siguiente (si corresponde[2], en caso contrario se pasará al paso 3 directamente).
2. Se compactará la memoria y se realizará una nueva búsqueda. En caso de no encontrarla, se pasará al paso siguiente.
3. Se procederá a eliminar una partición de datos, y luego se volverá al paso 2 o al 3, según corresponda.

Algoritmos para elección de partición libre y elección de víctima

Para seleccionar una partición libre, se deberá implementar los siguientes pares de algoritmos:

- First Fit (primer ajuste) y Best Fit (mejor ajuste).

[1] Referencias bibliográficas: sección 7.2, cap. 7, Stallings 6° ed.; sección 8.3.2/3, cap. 8, Silberschatz 7° ed.
[2] Se deberá poder configurar la frecuencia de compactación (en la unidad "cantidad de búsquedas fallidas"). El valor -1 indicará compactar solamente cuando se hayan eliminado todas las particiones.

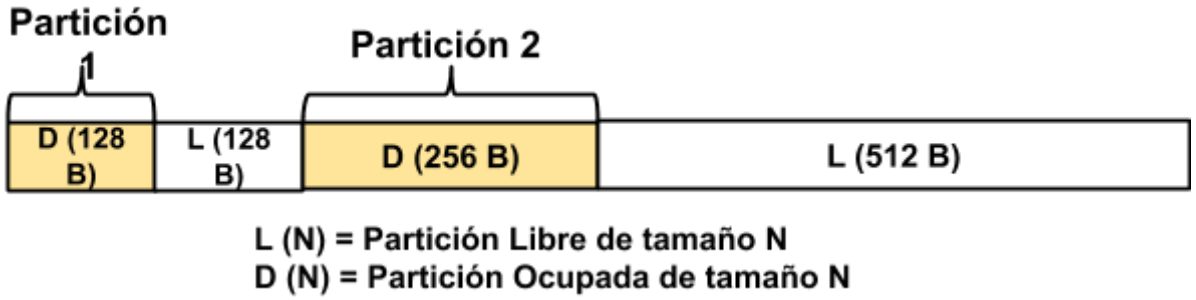
En el caso de tener que eliminar una partición, los algoritmos a implementar serán:

- FIFO (First In First Out) y LRU (Least Recently Used).

En ambos casos, el algoritmo a utilizar se definirá por archivo de configuración.

Buddy System[3]

En este esquema, se reservará una partición de memoria por cada valor almacenado, del tamaño potencia de 2 que sea más cercano a dicho valor. Por ejemplo:



L (N) = Partición Libre de tamaño N
D (N) = Partición Ocupada de tamaño N

En dicho ejemplo, en caso de almacenar un nuevo valor de 63 B en el espacio de la primer particion libre, generaría una nueva "partición 3" de 64 B, y al lado quedaría una nueva partición libre de 64 B.

El procedimiento de almacenamiento de datos será similar al de las particiones dinámicas, con la salvedad que la compactación se realizará de acuerdo a las reglas del propio algoritmo Buddy System. FIFO y LRU serán los algoritmos a implementar para la elección de víctima en un reemplazo (al igual que en el algoritmo previo, modificable por archivo de configuración)

Dump de la Caché

Será requerimiento del motor de administración de memoria que éste pueda depositar en un archivo el estado actual de la memoria en la caché según el esquema seleccionado. Para solicitar dicho dump, se enviará una señal SIGUSR1 que deberá ser manejada e inicializada.

No se pretende ver el contenido de la información almacenada, sino las particiones asignadas/libres, indicando su dirección de comienzo y fin, su tamaño en bytes, tiempos de LRU, el tipo de cola de mensajes que pertenece y su identificador.

Ejemplo:

```
---------------------------------------------------------------------------------------------------------------------
Dump: 14/07/2012 10:11:12
Partición 1: 0x000 - 0x3FF.     [X]     Size: 1024b     LRU:<VALOR>  Cola:<COLA>  ID:<ID>
Partición 2: 0x400 - 0x409.     [L]     Size: 9b
Partición 3: 0x40A - 0x40B.     [L]     Size: 1b
---------------------------------------------------------------------------------------------------------------------
```

[3] Referencias bibliográficas: sección 7.2, cap. 7, Stallings 6° ed.;.

Tipos de Suscribers

Todo mensaje en una implementación real tiene dos atributos claves: Su identificador y su identificador correlacional (o correlation id). El primero es un identificador único que asigna el administrador de colas de mensajes (como ya explicamos anteriormente) mientras que el segundo es el identificador correlacional del mensaje al cual está asociado.

Esto significa que si se manda el mensaje A en una cola de mensajes asignándole el identificador "1" quienes estén suscritos a esa cola recibirán dicho mensaje, junto con su Id. En caso de que algún suscriptor deba responder este mensaje, enviará un mensaje B a otra cola de mensajes donde se le asignará un nuevo identificador (por ejemplo "2"), pero el emisor asigna en el valor correlacional el identificador del mensaje al cual está respondiendo (en este caso "1").

De esta manera un módulo puede enviar un mensaje a una cola y esperar en otra cola de mensajes la respuesta asociada al primero que envió. Para esto hay que tener en cuenta varias cosas:

- Tiene que existir un protocolo de comunicación en el cual el proceso que envía el primer mensaje sabe que tiene que ir a buscar la respuesta a otra cola y el que genere la respuesta debe también conocer este protocolo.
- El administrador de colas de mensajes debe informar al emisor siempre el identificador del mensaje para que este último sepa cual es el mensaje correlativo que debe ir a buscar.
- El administrador de colas de mensajes debe mantener una lógica de conocer qué mensajes con correlativos ya fueron informados en una cola para no generar redundancia en la misma. Esto quiere decir que si hay varios suscriptores en la cola de mensajes inicial puede haber varias respuestas al mismo (que productivamente deberían ser la misma respuesta) por lo que es el administrador de cola de mensajes el que sabe que ese mensaje ya fue agregado a la cola destino y debe ignorarlo.

Una vez explicado esto diremos que tendremos dos tipos de suscriptores:

1. Suscriptores globales.
2. Suscriptores globales por mensajes correlativos.

Suscriptor global

Son suscriptores que se asocian globalmente a una cola de mensajes. Esto implica que todo mensaje que el Broker recíba a dicha cola de mensajes deberá ser enviado al suscriptor.

Cada vez que un proceso se suscriba globalmente a una cola de mensajes, el broker deberá validar en su memoria principal si tiene algún mensaje de dicha cola de mensajes y enviarles los mismos.

Para esto, el Broker manejara una lista de procesos dentro de cada mensaje en memoria indicando a qué procesos ya fue enviado el mismo. Es responsabilidad del grupo interiorizarse y resolver cómo se identifica a un proceso específico frente a una caída y recuperación.

Suscriptor globales por mensajes correlativos

Son suscriptores globales que se asocian a una cola de mensajes en espera de mensajes específicos con identificadores correlativos que él conoce. El Broker envía todos los mensajes que lleguen a la cola suscrita a este suscriptor y este último verificará si es un mensaje que él requiera (por medio de

su identificador correlativo) y de ser así actuará en acción a el. En caso que no sea un mensaje que él espera, lo ignorará.

Listado de Message Queues

El Broker deberá administrar las siguientes colas de mensajes:

- NEW_POKEMON
- APPEARED_POKEMON
- CATCH_POKEMON
- CAUGHT_POKEMON
- GET_POKEMON
- LOCALIZED_POKEMON

Tipos de datos

Dado que el trabajo práctico mantendrá seis tipos de mensajes distintos (uno por cada cola de mensajes) se normalizarán los tipos de datos a utilizar para que el tamaño que ocupe cada uno dentro de la memoria principal sea homogéneo a todos los grupos. El objetivo de esto es otorgar una manera rápida de verificación y validación de lo desarrollado por los alumnos.

Todo dato numérico será representado por un uint_32 mientras que todo dato de caracteres dinámicos será representado por un uint_32 para indicar su tamaño seguido de los caracteres que lo componen. Para ejemplificar cada tipo de mensaje les recomendamos leer el Anexo II.

Logs obligatorios

Para permitir la verificación/validación del módulo se exigirá tener un archivo de log específico e independiente que contenga la información indicada en esta sección. ***No se permite la inclusión de otros mensajes y exclusión de ninguno de los mismos*** (la falta o agregado de alguno puede implicar la desaprobación del grupo). Dado que el log será el medio de validación del trabajo práctico ***se exige no utilizar la consola como medio de logueo***. Cada operación debe loguearse en una única línea indicando la misma y sus datos, en caso que el grupo desee loguear contenido extra deberá realizarlo en un archivo independiente.

Las acciones a loguear en este archivo son:

1. Conexión de un proceso al broker.
2. Suscripción de un proceso a una cola de mensajes.
3. Llegada de un nuevo mensaje a una cola de mensajes.
4. Envío de un mensaje a un suscriptor específico.
5. Confirmación de recepción de un suscripción al envío de un mensaje previo.
6. Almacenado de un mensaje dentro de la memoria (indicando posición de inicio de su partición).
7. Eliminado de una partición de memoria (indicado la posición de inicio de la misma).
8. Ejecución de compactación (para particiones dinámicas) o asociación de bloques (para buddy system). En este último, indicar que particiones se asociaron (indicar posición inicio de ambas particiones).

9. Ejecución de Dump de cache (solo informar que se solicitó el mismo).

Archivo de Configuración

Campo	Tipo	Descripción
TAMANO_MEMORIA	[Numérico]	Tamaño de la memoria en bytes
TAMANO_MINIMO_PARTICION	[Numérico]	Tamaño mínimo de la partición en bytes
ALGORITMO_MEMORIA	[String]	El tipo de algoritmo de administración de memoria que se va a utilizar (PARTICIONES/BS)
ALGORITMO_REEMPLAZO	[String]	El tipo de algoritmo de reemplazo de memoria que se va a utilizar(FIFO/LRU)
ALGORITMO_PARTICION_LIBRE	[String]	El tipo de algoritmo de seleccion de particion libre a utilizar(FF/BF)
IP_BROKER	[String]	El IP del servidor del proceso Broker
PUERTO_BROKER	[Numérico]	El puerto del servidor del proceso Broker
FRECUENCIA_COMPACTACION	[Numérico]	Cantidad de búsquedas fallidas previa compactación
LOG_FILE	[String]	Path del archivo de log donde se almacenará el log obligatorio

Queda a decisión del grupo el agregado de más parámetros al mismo.

Ejemplo de Archivo de Configuración

```
TAMANO_MEMORIA=2048
TAMANO_MINIMO_PARTICION=32
ALGORITMO_MEMORIA=BS
ALGORITMO_REEMPLAZO=FIFO
ALGORITMO_PARTICION_LIBRE=FF
IP_BROKER=127.0.0.1
PUERTO_BROKER=6009
FRECUENCIA_COMPACTACION=3
```

Proceso Game Card

Este módulo nos permitirá **implementar nuestro propio almacenamiento de archivos**, que almacene los datos de los distintos Pokemon que se encuentren en el mapa. Para esto, se deberá implementar el FileSystem TALL_GRASS explicado en los siguientes apartados.

Este proceso se comunicará con los demás de dos posibles maneras:

1. A través de la conexión con el Broker asociándose globalmente a la cola de mensajes NEW_POKEMON, CATCH_POKEMON y GET_POKEMON.
2. A través de un socket de escucha en el cual podrá recibir mensajes de las colas de mensajes mencionadas en el punto anterior.

Cabe aclarar que el Proceso Game Card debe poder ejecutarse sin haber establecido la conexión con el Broker. Es decir, si el Broker se encuentra sin funcionar o se cae durante la ejecución, el proceso Game Card debe seguir procesando sus funciones sin el mismo.

En caso que la conexión no llegue a realizarse o se caiga, el proceso Game Card deberá contar con un sistema de reintento de conexión cada X segundos configurado desde archivo de configuración. Esto permitirá que en caso de caerse el Broker o se inicie posteriormente al mismo, el proceso Game Card pueda asociarse a las colas sin necesidad de reiniciarse.

Tall Grass

El FileSystem Tall Grass es un componente creado con propósitos académicos para que el alumno comprenda el funcionamiento básico de la gestión de archivos en un sistema operativo.

La estructura básica del mismo se basa en una estructura de árbol de directorios para representar la información administrativa y los datos de los entidades/Pokemon en formato de archivos. El árbol de directorios tomará su punto de partida del punto de montaje del archivo de configuración.

Durante las pruebas no se proveerán archivos que tengan estados inconsistentes respecto del trabajo práctico, por lo que no es necesario tomar en cuenta dichos casos.

Metadata

Este archivo tendrá la información correspondiente a la cantidad de bloques y al tamaño de los mismos dentro del File System.

Dentro del archivo se encontrarán los siguiente campos:

- **Block_size**: Indica el tamaño en bytes de cada bloque
- **Blocks**: Indica la cantidad de bloques del File System
- **Magic_Number**: Un string fijo con el valor “TALL_GRASS”

Ej:

```
BLOCK_SIZE=64
BLOCKS=5192
```

```
MAGIC_NUMBER=TALL_GRASS
```

Dicho archivo deberá encontrarse en la ruta `[Punto_Montaje]/Metadata/Metadata.bin`

Bitmap

Este será un archivo de tipo binario donde solamente existirá un bitmap[4], el cual representará el estado de los bloques dentro del FS, siendo un 1 que el bloque está ocupado y un 0 que el bloque está libre.

La ruta del archivo de bitmap es: `[Punto_Montaje]/Metadata/Bitmap.bin`

Files Metadata

Los archivos dentro del FS se encontrarán en un path compuesto de la siguiente manera:

```
[Punto_Montaje]/Files/[Nombre_Archivo]
```

Donde el path del archivo incluye el archivo Metadata.

```
Ej:

    /mnt/TALL_GRASS/Files/Pikachu/Metadata.bin
```

Dentro del archivo Metadata.bin se encontrarán los siguientes campos:

- **Directory:** indica si el archivo en cuestión es un directorio o no (Y/N).
- **Size**: indica el tamaño real del archivo en bytes (en caso de no ser un directorio).
- **Blocks**: es un array de números que contiene el orden de los bloques en donde se encuentran los datos propiamente dichos de ese archivo (en caso de no ser un directorio).
- **Open**: indica si el archivo se encuentra abierto (Y/N).

```
Ej Directorio:

DIRECTORY=Y
```

```
Ej Archivo:

DIRECTORY=N
SIZE=250
BLOCKS=[40,21,82,3]
OPEN=Y
```

De esta manera podremos formar el siguiente árbol de archivos en donde la metadata dentro del directorio Pokemon contiene un DIRECTORY con valor Y y Pikachu tiene un metadata con un DIRECTORY con el valor N.

```
/mnt/TALL_GRASS/Files/Pokemon/Metadata.bin
/mnt/TALL_GRASS/Files/Pokemon/Pikachu/Metadata.bin
```

Datos

Los datos estarán repartidos en archivos de texto nombrados con un número, el cual representará el número de bloque. (Por ej 1.bin, 2.bin, 3.bin),

[4] Se recomienda investigar sobre el manejo de los bitarray de las commons library.

Dichos archivos se encontraran dentro de la ruta:

```
[Punto_Montaje]/Blocks/[nroBloque].bin
```

Ej:

```
/mnt/TALL_GRASS/Blocks/1.bin
/mnt/TALL_GRASS/Blocks/2.bin
```

Lineamiento e Implementación

Este proceso gestionará un Filesystem que será leído e interpretado como un árbol de directorios y sus archivos utilizando el Filesystem Tall Grass.

A su vez, al iniciar el proceso Game Card se intentara suscribir globalmente al Broker a las siguientes colas de mensajes:

- NEW_POKEMON
- CATCH_POKEMON
- GET_POKEMON

Al suscribirse a cada una de las colas deberá quedarse a la espera de recibir un mensaje del Broker. Al recibir un mensaje de cualquier hilo se deberá:

1. Informar al Broker la recepción del mismo (ACK).
2. Crear un hilo que atienda dicha solicitud.
3. Volver a estar a la escucha de nuevos mensajes de la cola de mensajes en cuestión.

Todo archivo dentro del file system tendrá un valor “OPEN” dentro de su metadata que indicará si actualmente hay algún proceso que se encuentra utilizando el mismo. ***Bajo ningún concepto se permitirá a dos procesos abrir el mismo archivo en simultáneo***. En caso que suceda esto se deberá informar el error pertinente por archivo de log o consola.

Archivos Pokemon

Cada archivo de tipo pokemon tendra internamente (por cada fila) la lista de posiciones en el mapa que se encuentra con la cantidad en dicha posición. De esta manera un posible archivo de pokemon puede ser:

```
1-1=10
1-5=1
3-1=2
7-6=1000
```

La inclusión de una nueva línea o eliminación depende de la recepción de los distintos mensajes desde el Broker.

A continuación se explicará el funcionamiento que se debe realizar al recibir mensajes de alguna de estas colas.

New Pokemon

Este mensaje cumplirá la función de agregar la aparición de un nuevo pokémon al mapa. Tendrá cuatro parámetros de entrada:

1. ID del mensaje recibido.
2. Pokemon a agregar.
3. Posición del mapa.
4. Cantidad de pokémon en dicha posición a agregar.

Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Pokémon existe dentro de nuestro Filesystem. Para esto se deberá buscar dentro del directorio Pokemon si existe el archivo con el nombre de nuestro pokémon. En caso de no existir se deberá crear.
2. Verificar si se puede abrir el archivo (si no hay otro proceso que lo esté abriendo). En caso que el archivo se encuentre abierto se deberá finalizar el hilo y reintentar la operación luego de un tiempo definido por configuración.
3. Verificar si las posiciones ya existen dentro del archivo. En caso de existir se deben agregar la cantidad pasada por parámetro a la actual. En caso de no existir se debe agregar al final del archivo una nueva línea indicando la cantidad de pokémon pasadas.
4. Cerrar el archivo.
5. Conectarse al Broker y enviar el mensaje a la Cola de Mensajes ***APPEARED_POKEMON*** con los los datos:
    - ID del mensaje recibido.
    - Pokemon.
    - Posición del mapa.

En caso que no se pueda realizar la conexión con el Broker se debe informar por logs y continuar la ejecución.

Catch Pokemon

Este mensaje cumplirá la función de indicar si es posible capturar un Pokemon. Para esto se recibirán los siguientes parámetros:

1. ID del mensaje recibido.
2. Pokemon a atrapar.
3. Posición del mapa.

Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Pokémon existe dentro de nuestro Filesystem. Para esto se deberá buscar dentro del directorio Pokemon si existe el archivo con el nombre de nuestro pokémon. En caso de no existir se deberá informar un error.
2. Verificar si se puede abrir el archivo (si no hay otro proceso que lo esté abriendo). En caso que el archivo se encuentre abierto se deberá finalizar el hilo y reintentar la operación luego de un tiempo definido por configuración.

3. Verificar si las posiciones ya existen dentro del archivo. En caso de no existir se debe informar un error.
4. En caso que la cantidad del Pokémon sea “1”, se debe eliminar la línea. En caso contrario se debe decrementar la cantidad en uno.
5. Cerrar el archivo.
6. Conectarse al Broker y enviar el mensaje indicando el resultado correcto.

Todo resultado, sea correcto o no, deberá realizarse conectandose al Broker y enviando un mensaje a la Cola de Mensajes ***CAUGHT_POKEMON*** indicando:

1. ID del mensaje recibido originalmente.
2. Resultado.

En caso que no se pueda realizar la conexión con el Broker se debe informar por logs y continuar la ejecución.

Get Pokemon

Este mensaje cumplirá la función de obtener todas las posiciones y su cantidad de un Pokémon específico. Para esto recibirá:

1. El identificador del mensaje recibido.
2. Pokémon a devolver.

Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Pokémon existe dentro de nuestro Filesystem. Para esto se deberá buscar dentro del directorio Pokemon si existe el archivo con el nombre de nuestro pokémon. En caso de no existir se deberá informar el mensaje sin posiciones ni cantidades.
2. Verificar si se puede abrir el archivo (si no hay otro proceso que lo esté abriendo). En caso que el archivo se encuentre abierto se deberá finalizar el hilo y reintentar la operación luego de un tiempo definido por configuración.
3. Obtener todas las posiciones y cantidades de Pokemon requerido.
4. Cerrar el archivo.
5. Conectarse al Broker y enviar el mensaje con todas las posiciones y su cantidad.

En caso que se encuentre por lo menos una posición para el Pokémon solicitado se deberá enviar un mensaje al Broker a la Cola de Mensajes ***LOCALIZED_POKEMON*** indicando:

3. ID del mensaje recibido originalmente.
4. El Pokémon solicitado.
5. La lista de posiciones y la cantidad de cada una de ellas en el mapa.

En caso que no se pueda realizar la conexión con el Broker se debe informar por logs y continuar la ejecución.

Archivo de Configuración

Campo	Tipo	Descripción

TIEMPO_DE_REINTENTO_CONEXION	[Numérico]	Tiempo en segundos en el cual el proceso debe reintentar conectarse al broker.
TIEMPO_DE_REINTENTO_OPERACION	[Numérico]	Tiempo en segundos en el cual el proceso debe reintentar reabrir el archivo que se encontraba abierto.
PUNTO_MONTAJE_TALLGRASS	[String]	Punto en el cual se va a inicializar el file system.
IP_BROKER	[String]	El IP del servidor del proceso Broker
PUERTO_BROKER	[Numérico]	El puerto del servidor del proceso Broker

Queda a decisión del grupo el agregado de más parámetros al mismo.

Ejemplo de Archivo de Configuración

```
TIEMPO_DE_REINTENTO_CONEXION=10
TIEMPO_DE_REINTENTO_OPERACION=5
PUNTO_MONTAJE_TALLGRASS=/home/utnso/desktop/tall-grass
IP_BROKER=127.0.0.1
PUERTO_BROKER=6009
```

Proceso Team

Este proceso será el encargado de administrar distintos entrenadores “planificándolos” dentro de un mapa de dos coordenadas. Cada entrenador tendrá objetivos particulares en los cuales deberán atrapar distintos Pokémon, los cuales serán configurados por archivos de configuración. Cabe aclarar que un entrenador no podrá atrapar mas pokemones de los que indique su objetivo, por ejemplo si su objetivo es atrapar tres pokemones cualesquiera, no podrá atrapar más de tres, por más que no sean los tres que él necesita.

Lineamiento e Implementación

El objetivo de este proceso es verificar la aparición de un nuevo Pokémon y, en caso de que algún entrenador requiera del mismo para el cumplimiento de su objetivo, planificar al entrenador más cercano libre se mueva a dicha posición a atraparlo. Este proceso se comunicará de dos posibles maneras:

1. A través de la conexión con el Broker asociándose globalmente a la cola de mensajes APPEARED_POKEMON, LOCALIZED_POKEMON y CAUGHT_POKEMON .
2. A través de un socket de escucha en el cual podrá recibir mensajes de apariciones de Pokémon.

Cabe aclarar que el Proceso Team debe poder ejecutarse sin haber establecido la conexión con el Broker. Es decir, si el broker se encuentra sin funcionar o se cae durante la ejecución, el proceso Team debe seguir procesando sus funciones sin el mismo. Para esto, se contarán con funciones default para aquellos mensajes que el Proceso Team envíe directamente al Broker.

En caso que la conexión no llegue a realizarse o se caiga, el proceso Team deberá contar con un sistema de reintento de conexión cada X segundos configurado desde archivo de configuración.

Planificación

Como dijimos anteriormente los distintos entrenadores se configuran desde archivo de configuración. Al iniciar el proceso, se deberá crear un hilo por cada entrenador existente y el proceso Team deberá conocer cuáles y qué cantidad de Pokémon de cada especie requiere en total para cumplir el objetivo global.

Se dice que un proceso Team cumplió su objetivo global cuando todos sus entrenadores obtuvieron los Pokémon que requieren. Al realizarse esto, se debe informar, desalojar los recursos obtenidos y finalizar el proceso.

Al aparecer un Pokémon (por cualquiera de los dos métodos antes explicados) sólo se podrá planificar a un entrenador hacia dicha posición independientemente de cuántos Pokémon de dicha especie haya en la posición en la que apareció.

Al planificar un entrenador, se activará el hilo del entrenador más cercano al Pokémon. Cada movimiento en el mapa responderá a un ciclo de CPU. Para simular más a la realidad esta funcionalidad, se deberá agregar un retardo de X segundos configurado por archivo de configuración.

Para planificar a los distintos entrenadores se utilizarán los algoritmos FIFO, Round Robin y Shortest job first con y sin desalojo. Para este último algoritmo se desconoce la próxima rafaga, por lo que se deberá utilizar la fórmula de la media exponencial. A su vez, la estimación inicial para todos los entrenadores será la misma y deberá poder ser modificable por archivo de configuración

Diagrama de estados de un Entrenador

Cada entrenador al iniciar en el sistema entrará en estado New. A medida que el Team empiece a recibir distintos Pokémon en el mapa despertará a los distintos entrenadores en estado New o en Blocked (que estén esperando para procesar) pasandolos a Ready. Siempre se planificará aquel entrenador que se encuentre sin estar realizando ninguna operación activamente y, en caso de existir más de uno, sea el que más cerca se encuentre del objetivo.

A medida que cada entrenador se planifique (ya sea para moverse, intercambiar o atrapar un Pokémon) entrarán en estado exec. En el contexto de nuestro trabajo practico no contemplaremos el multiprocesamiento, esto implica que solo UN entrenador podrá estar en estado Exec en determinado tiempo.

Cuando un entrenador en estado Exec finalice su recorrido y su ejecución planificada entrará en un estado bloqueados. Este estado implica que el entrenador no tiene más tareas para realizar momentáneamente.

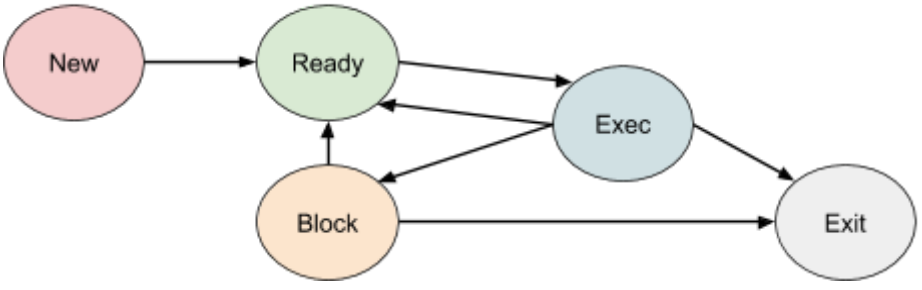
Cuando un entrenador en estado Exec cumpla todos sus objetivos, pasará a estado Exit. Cuando todos los entrenadores dentro de un Team se encuentren en Exit, se considera que el proceso Team cumplió el objetivo global.

Cuando se detecte situaciones de Deadlock deberán estar ambos en estado bloqueado. En este momento, uno de ellos se pasa a estado Ready con el objetivo que se lo planifique hasta la posición del otro. Al llegar a la misma posición, se deberá realizar el intercambio. Cada intercambio ocupara 5 ciclos de CPU. Cada intercambio solo involucra a dos Pokémon.

Al finalizar el intercambio se verificará si alguno está en condiciones de ir a Exit y de no ser así irán nuevamente a estado bloqueado. A su vez, cada acción de movimiento o envío de mensaje al Broker consumirá un ciclo de CPU.

Cada vez que un entrenador realice una operación de captura sobre un Pokémon se lo bloqueará a la espera del resultado no pudiendo volver a operar hasta obtener el mismo.

Adjuntamos un diagrama de estados con lo anteriormente mencionado.



Cabe aclarar que el diagrama antes descrito es similar al visto en la teoría pero agrega transiciones propias del contexto de este Trabajo práctico.

Competición y Deadlock

Dado que pueden existir varios procesos Team dentro de nuestro sistema, puede darse la posibilidad de que varios de ellos requieran una especie de un Pokemon y no exista la misma cantidad de ellos en el sistema. Este flujo es el esperado y nos permitirá probar los distintos algoritmos de planificación con sus ventajas y desventajas.

Para comparar los mismos el proceso Team al cumplir su objetivo deberá informar:

1. Cantidad de ciclos de CPU totales.
2. Cantidad de cambios de contexto realizados.
3. Cantidad de ciclos de CPU realizados por entrenador.
4. Deadlocks producidos y resueltos (Spoiler Alert).

Dado que el proceso Team conoce cuantos Pokemon de cada especie necesita globalmente, cuantos de cada uno ha atrapado y planifica al entrenador más cercano libre, puede darse el caso que un entrenador que no requiere una especie de Pokémon termine capturandolo, no permitiéndole a otro del mismo equipo que si lo necesita, obtenga el mismo.

En estos casos se producirá un caso de Deadlock, en el cual el proceso Team no podrá finalizar debido a que varios de sus entrenadores están en un estado de Interbloqueo. Es responsabilidad de cada grupo definir un algoritmo para detectar estos casos para poder resolverlos.

Cuando se detecte dichos casos, se deberá bloquear uno de los entrenadores y planificar al/los otro/s a la posición del primero para generar un “intercambio” (cada intercambio implica que cada entrenador entregue un Pokémon al otro uno de ellos).

Tipo de mensajes

El proceso Team manejara 5 tipos de mensajes hacia el Broker, uno será por suscripción global y los otros dos pares de mensajes de envio y suscripcion por identificador correlacional.

Appeared Pokemon

Este mensaje permitirá la inclusión en el proceso Team de un nuevo Pokémon en el mapa. Esto se podrá producir de las dos maneras indicadas anteriormente.

Al llegar este mensaje, el proceso Team deberá verificar si requiere atrapar el mismo controlando los Pokemon globales necesarios y los ya atrapados. No se debe poder atrapar mas Pokemon de una especie de los requeridos globalmente.

En caso que se requiera el mismo, se debe agregar a la lista de Pokémon requeridos y en el momento que un entrenador se encuentre en estado “Dormido” o “Libre” debe planificarlo para ir a atraparlo.

En este mensaje se recibirán los siguientes parámetros:

- Especie de Pokemon.
- Posición del Pokemon.

Get Pokemon

Este mensaje se ejecutará al iniciar el proceso Team. El objetivo del mismo es obtener todas las locaciones de una especie de Pokemon. De esta manera, al iniciar el proceso, por cada especie de Pokémon requerido se debe enviar un mensaje a la cola de mensajes GET_POKEMON del Broker.

Para esto se deben ejecutar los siguientes pasos:

1. Enviar el mensaje a la cola de mensajes GET_POKEMON indicando cual es la especie del Pokemon.
2. Obtener el ID del mensaje anterior desde el Broker.

En caso que el Broker no se encuentre funcionando o la conexión inicial falle, se deberá tomar como comportamiento Default que no existen locaciones para la especie requerida.

Catch Pokemon

Este mensaje se ejecutará al intentar atrapar a un Pokémon (cuando un entrenador llegue a la posición del mismo). Para esto, se enviará un mensaje a la cola de mensajes CATCH_POKEMON del Broker.

Para esto, se deben ejecutar los siguientes pasos:

1. Enviar el mensaje a la cola de mensajes CATCH_POKEMON indicando cual es la especie del Pokémon y la posición del mismo.
2. Obtener el ID del mensaje anterior desde el Broker y guardarlo a la espera de la llegada de la respuesta en CAUGHT_POKEMON.
3. Bloquear al entrenador en cuestión a la espera del resultado del mensaje. Este entrenador no podrá volver a ejecutar hasta que se reciba el resultado.

En caso que el Broker no se encuentre funcionando o la conexión inicial falle, se deberá tomar como comportamiento Default que el Pokémon ha sido atrapado con éxito.

Localized Pokémon

El proceso Team se suscribirá de manera global a esta cola de mensajes. Al recibir uno de los mismos deberá realizar los siguientes pasos:

1. Verificar si ya recibió en algún momento un mensaje de la especie del Pokémon asociado al mensaje. Si es así, descarta el mensaje (ya sea Appeared o Localized).
2. En caso de que nunca lo haya recibido, realiza las mismas operatorias que para APPEARED_POKEMON por cada coordenada del pokemon.

Caught Pokémon

El proceso Team se suscribirá de manera global a esta cola de mensajes. Al recibir uno de los mismos deberá realizar los siguientes pasos:

1. Validar si el id de mensaje correlativo del mensaje corresponde a uno pendiente de respuesta generado por la la instrucción CATCH_POKEMON antes descripta. Si no corresponde a ninguno, ignorar el mensaje.

2. En caso que corresponda se deberá validar si el resultado del mensaje es afirmativo (se atrapó el Pokémon). Si es así se debe asignar al entrenador bloqueado el Pokémon y habilitarlo a poder volver operar.

Logs obligatorios

Para permitir la verificación/validación del módulo se exigirá tener un archivo de log específico e independiente que contenga la información indicada en esta sección. ***No se permite la inclusión de otros mensajes y exclusión de ninguno de los mismos*** (la falta o agregado de alguno puede implicar la desaprobación del grupo). Dado que el log será el medio de validación del trabajo práctico ***se exige no utilizar la consola como medio de logueo***. Cada operación debe loguearse en una única línea indicando la misma y sus datos, en caso que el grupo desee loguear contenido extra deberá realizarlo en un archivo independiente.

Cabe aclarar que cada proceso Team deberá tener su propio archivo de Log. Por lo tanto, dado que puede darse el caso que se ejecuten varios procesos Team sobre el mismo ordenador se deberá indicar el archivo de log que se utilizará para cada uno por archivo de configuración.

Las acciones a loguear en este archivo son:

1. Cambio de un entrenador de cola de planificación (indicando la razón del porqué).
2. Movimiento de un entrenador (indicando la ubicación a la que se movió).
3. Operación de atrapar (indicando la ubicación y el pokemon a atrapar).
4. Operación de intercambio (indicando entrenadores involucrados).
5. Inicio de algoritmo de detección de deadlock.
6. Resultado de algoritmo de detección de deadlock.
7. Llegada de un mensaje (indicando el tipo del mismo y sus datos).
8. Resultado del Team (especificado anteriormente).
9. Errores de comunicación con el Broker (indicando que se realizará la operación por default).
10. Inicio de proceso de reintento de comunicación con el Broker.
11. Resultado de proceso de reintento de comunicación con el Broker.

Archivo de Configuración

Campo	Tipo	Descripción
POSICIONES_ENTRENADORES	[Lista de lista]	Contiene una lista de las posiciones de los entrenadores.
POKEMON_ENTRENADORES	[Lista de lista]	Contiene una lista de los pokemon de cada entrenadores.
OBJETIVOS_ENTRENADORES	[Lista de lista]	Contiene una lista de los pokemon que cada entrenador debe obtener.
TIEMPO_RECONEXION	[Numérico]	Tiempo en segundos en el cual el proceso debe reintentar conectarse al broker.

RETARDO_CICLO_CPU	[Numérico]	Tiempo en segundos para el retardo de la ejecución de cada ciclo de cpu
ALGORITMO_PLANIFICACION	[String]	El tipo de algoritmo de planificación que se va a utilizar (FIFO/RR/SJF-CD/SJF-SD)
QUANTUM	[Numérico]	El valor del quantum en caso de que el algoritmo utilice RR.
IP_BROKER	[String]	El IP del servidor del proceso Broker
ESTIMACION_INICIAL	[Numérico]	El valor de la estimación inicial para SJF en caso de que aplique
PUERTO_BROKER	[Numérico]	El puerto del servidor del proceso Broker
LOG_FILE	[String]	Path del archivo de log donde se almacenará el log obligatorio

Queda a decisión del grupo el agregado de más parámetros al mismo.

Ejemplo de Archivo de Configuración

```
POSICIONES_ENTRENADORES=[ [1, 2], [3, 7], [5, 5] ]
POKEMON_ENTRENADORES=[[Pikachu, Squirtle, Pidgey], [Squirtle, Charmander], [Bulbasaur]]
OBJETIVOS_ENTRENADORES=[[Pikachu, Pikachu, Squirtle, Pidgey], [Pikachu, Charmander, Charmander], [Squirtle, Bulbasaur]]
TIEMPO_RECONEXION=30
RETARDO_CICLO_CPU=2
ALGORITMO_PLANIFICACION=RR
QUANTUM=2
ESTIMACION_INICIAL=0
IP_BROKER=127.0.0.1
PUERTO_BROKER=5002
LOG_FILE=/home/utnso/log_team1.txt
```

```
POSICIONES_ENTRENADORES=[ [1, 2], [3, 7], [5, 5] ]
POKEMON_ENTRENADORES=[[Pikachu, Squirtle, Pidgey], [Squirtle, Charmander], [Bulbasaur]]
OBJETIVOS_ENTRENADORES=[[Pikachu, Pikachu, Squirtle, Pidgey], [Pikachu, Charmander, Charmander], [Squirtle, Bulbasaur]]
TIEMPO_RECONEXION=30
RETARDO_CICLO_CPU=2
ALGORITMO_PLANIFICACION=SJF-CD
QUANTUM=0
ESTIMACION_INICIAL=5
```

```
IP_BROKER=127.0.0.1
PUERTO_BROKER=5002
LOG_FILE=/home/utnso/log_team2.txt
```

Proceso Game Boy

Este proceso cumplira la funcion de ser un cliente que permita:

1. Enviar un mensaje al Broker, a un Proceso Team o a un Proceso Game Card.
2. Suscribirse a una cola de mensajes específica del Broker por un tiempo limitado.

Para esto se iniciará el proceso desde consola enviando los argumento necesarios para poder enviar el mensaje al proceso destino.

Lineamiento e Implementación

El objetivo del proceso Game Boy es poder probar independientemente todos los otros procesos. De esta manera, el mismo podrá enviar cualquier mensaje a cualquier cola que el proceso Broker conozca y enviar por socket al proceso Team el mensaje de aparición de un nuevo Pokemon.

Este proceso no espera ninguna lógica específica o particular, solo serializará y des serializará los mensajes requeridos por argumentos y lo enviará al proceso. Este proceso se ejecutará enviando parametros por argumento que indicara cuál será la funcionalidad que el mismo cumpla. Para esto, tendrá dos posibles variantes dependiendo si es para enviar un mensaje o suscribirse.

De esta manera, para el envío de mensajes el formato de ejecución del mismo sea el siguiente:

```
./gameboy [PROCESO] [TIPO_MENSAJE] [ARGUMENTOS]*
```

Cabe aclarar que dicho formato ***NO es modificable***. No se permite la inclusión de ningun argumento mas al mismo. De esta manera, la cátedra proveerá al momento de realizar las pruebas en las distintas instancias de evaluación scripts que contengan la ejecución de varios de dichos comandos.

Los mensajes a implementar son:

Broker - New Pokemon

Permitirá enviar un mensaje al Broker a la cola de mensajes NEW_POKEMON. Para esto, el formato del mensaje será:

```
./gameboy BROKER NEW_POKEMON [POKEMON] [POSX] [POSY] [CANTIDAD]
```

Broker - Appeared Pokemon

Permitirá enviar un mensaje al Broker a la cola de mensajes APPEARED_POKEMON. Para esto, el formato del mensaje será:

```
./gameboy BROKER APPEARED_POKEMON [POKEMON] [POSX] [POSY] [ID_MENSAJE]
```

Cabe aclarar que el ID_MENSAJE será un valor definido tanto por la cátedra como por los alumnos al realizar sus propios test. Este ID dentro de un script o entorno de ejecución deberá ser ÚNICO.

Broker - Catch Pokemon

Permitirá enviar un mensaje al Broker a la cola de mensajes CATCH_POKEMON. Para esto, el formato del mensaje será:

```
./gameboy BROKER CATCH_POKEMON [POKEMON] [POSX] [POSY]
```

Broker - Caught Pokemon

Permitirá enviar un mensaje al Broker a la cola de mensajes CAUGHT_POKEMON. Para esto, el formato del mensaje será:

```
./gameboy BROKER CAUGHT_POKEMON [ID_MENSAJE] [OK/FAIL]
```

Broker - Get Pokemon

Permitirá enviar un mensaje al Broker a la cola de mensajes GET_POKEMON. Para esto, el formato del mensaje será:

```
./gameboy BROKER GET_POKEMON [POKEMON]
```

Team - Appeared Pokemon

Permitirá enviar un mensaje al Team a la cola de mensajes APPEARED_POKEMON. Para esto, el formato del mensaje será:

```
./gameboy TEAM APPEARED_POKEMON [POKEMON] [POSX] [POSY]
```

Game Card - New Pokemon

Permitirá enviar un mensaje al Game Card a la cola de mensajes NEW_POKEMON. Para esto, el formato del mensaje será:

```
./gameboy GAMECARD NEW_POKEMON [POKEMON] [POSX] [POSY] [CANTIDAD]
```

Game Card - Catch Pokemon

Permitirá enviar un mensaje al Game Card a la cola de mensajes CATCH_POKEMON. Para esto, el formato del mensaje será:

```
./gameboy GAMECARD CATCH_POKEMON [POKEMON] [POSX] [POSY]
```

Game Card - Get Pokemon

Permitirá enviar un mensaje al Game Card a la cola de mensajes GET_POKEMON. Para esto, el formato del mensaje será:

```
./gameboy GAMECARD GET_POKEMON [POKEMON]
```

Modo Suscriptor

En este modo, el proceso GameBoy deberá conectarse como suscriptor durante un tiempo definido en segundos pasado por parámetro. Para esto se mantendrá la siguiente nomenclatura en su ejecución:

```
./gameboy SUSCRIPTOR [COLA_DE_MENSAJES] [TIEMPO]
```

Este modo permitirá obtener los mensajes actuales que contiene en memoria el Broker y probar efectiva y correctamente el algoritmo de reemplazo y la compactación.

Logs obligatorios

Para permitir la verificación/validación del módulo se exigirá tener un archivo de log específico e independiente que contenga la información indicada en esta sección. ***No se permite la inclusión de otros mensajes y exclusión de ninguno de los mismos*** (la falta o agregado de alguno puede implicar la desaprobación del grupo). Dado que el log será el medio de validación del trabajo práctico ***se exige no utilizar la consola como medio de logueo***. Cada operación debe loguearse en una única línea indicando la misma y sus datos, en caso que el grupo desee loguear contenido extra deberá realizarlo en un archivo independiente.

Las acciones a loguear en este archivo son:

1. Conexión a cualquier proceso.
2. Suscripción a una cola de mensajes.
3. Llegada de un nuevo mensaje a una cola de mensajes.
4. Envío de un mensaje a un suscriptor específico.

Archivo de Configuración

El proceso deberá poseer un archivo de configuración en una ubicación conocida donde se deberán especificar, al menos, los siguientes parámetros:

Campo	Tipo	Ejemplo
IP_BROKER	[String]	127.0.0.1
IP_TEAM	[String]	127.0.0.2
IP_GAMECARD	[String]	127.0.0.3
PUERTO_BROKER	[Numérico]	5003
PUERTO_TEAM	[Numérico]	5002
PUERTO_GAMECARD	[Numérico]	5001

Queda a decisión del grupo el agregado de más parámetros al mismo.

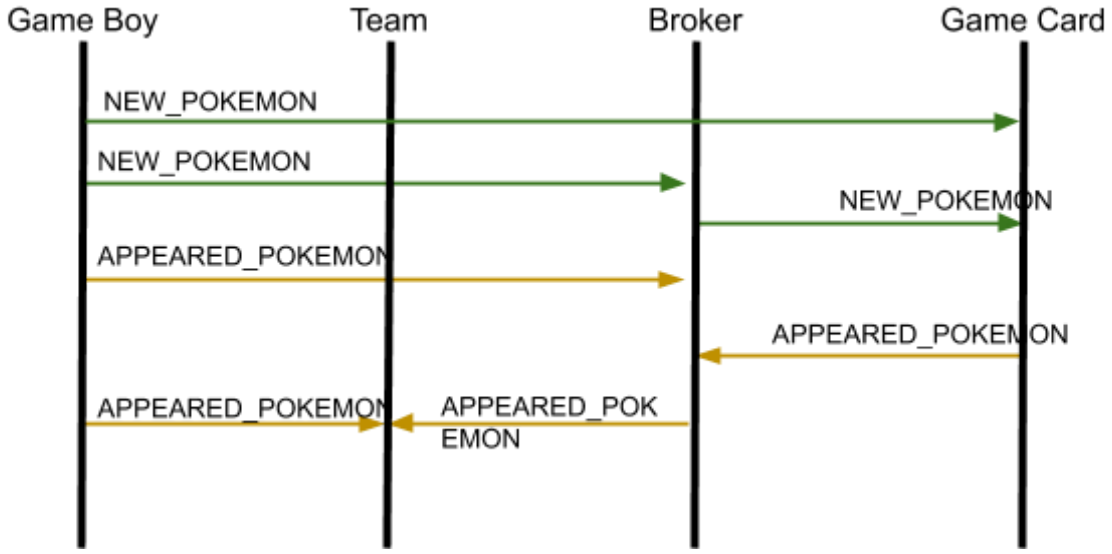
Ejemplo de Archivo de Configuración

```
IP_BROKER=127.0.0.1
IP_TEAM=127.0.0.2
IP_GAMECARD=127.0.0.3
PUERTO_BROKER=5003
```

```
PUERTO_TEAM=5002
PUERTO_GAMECARD=5001
```

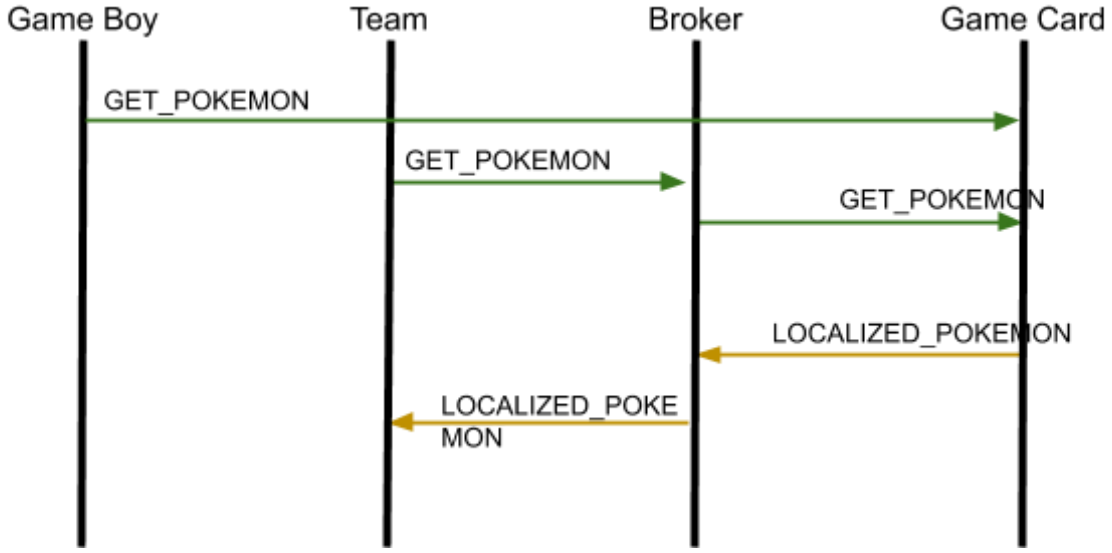
Anexo I - Ejemplos de Flujos

Flujo New Pokemon - Appeared Pokemon



El Game Boy va a ser nuestro punto de partida para crear un nuevo pokemon. Este le va a enviar un mensaje al Broker a través de la cola NEW_POKEMON indicando el pokemon, su posición y su cantidad. El broker luego, le deberá informar a todos los procesos Game Card que estén suscritos a la cola de mensajes. Los procesos Game Card avisarán en la cola de mensajes de APPEARED_POKEMON que los pokemones fueron agregados correctamente. Nuevamente el proceso Broker va a notificar a todos los procesos suscritos a dicha cola del nuevo evento.

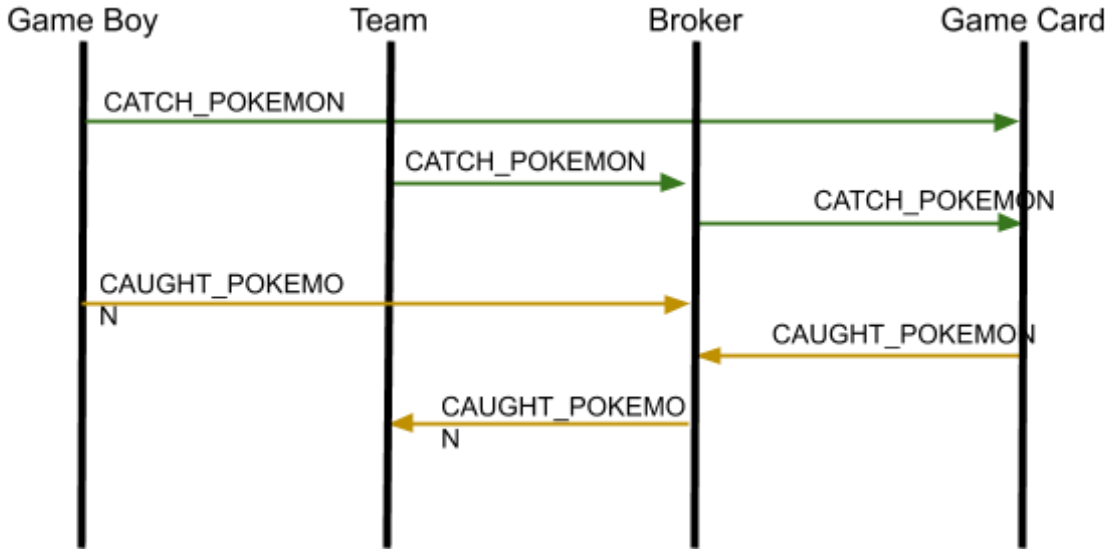
Flujo Get Pokemon - Localized Pokemon



Al iniciar el proceso Team, va a enviar un mensaje por cada pokemon que requiera capturar, a través de la cola de GET_POKEMON. El Broker luego va a redirigir el mensaje a los procesos suscritos. Una

vez que la Game Card recibe el mensaje, va a recopilar la información que tenga de ese pokemon (cantidad y posición) y va a enviarlos por la cola de LOCALIZED_POKEMON. El proceso Game Boy tambien podra enviar el mensaje de GET_POKEMON al Game Card. Este flujo nos permitirá la evaluación del trabajo práctico.

Flujo Catch Pokemon - Caught Pokemon



Este flujo es iniciado por la aparición de nuevos pokemons (cuando el Team recibe un mensaje en APEARED_POKEMON). Una vez que los procesos se enteran que apareció un nuevo pokémon y determinaron si que lo necesitan, van a intentar capturarlo (utilizando el proceso explicado en el proceso Team). Para esto cada proceso Team va a enviar un mensaje a través de la cola de CATCH_POKEMON al Boker, el cual va a reenviar a todos los procesos subscriptos a dicha cola, el mensaje. Una vez que el Game Card resuelva las peticiones, el Broker va a informar a todos los procesos teams todos los mensajes de CAUGHT_POKEMON, los cuales deberá ignorar todos los mensajes que no correspondan con su id. El proceso Game Boy tambien podra enviar el mensaje de CATCH_POKEMON y CAUGHT_POKEMON al Broker. Este deberá distribuir correctamente los nuevos mensajes a los suscriptores. Este flujo nos permitirá la evaluación del trabajo práctico.

Anexo II - Mensajes en memoria

El objetivo de este anexo es definir con ejemplos cómo se guardan los distintos tipos de mensaje en memoria. El objetivo de esto es normalizar el consumo de memoria principal para los distintos mensajes.

Tamaño de New Pokemon

Este mensaje tendrá el nombre del Pokemon a enviar, las coordenadas de la posición donde se encuentra y la cantidad de pokémon de esta especie que habra en dicha posicion. Un ejemplo del mensaje es:

```
‘Pikachu’ 5 10 2
```

Viéndolo a nivel tipo de dato vamos a tener un uint_32 para saber el largo del nombre del pokémon, más el nombre del pokemon y tres uint_32 indicando la posición y la cantidad. Este ejemplo tendrá un tamaño de 23 bytes en memoria principal (lo que ocupan cuatro uint_32 más el largo del nombre del pokemon).

Tamaño de Localized Pokemon

Este mensaje tendrá el nombre del pokémon, un entero para la cantidad de pares de coordenadas y los pares de coordenadas donde se encuentra el mismo. Un ejemplo del mensaje es:

```
‘Pikachu’ 3 4 5 1 5 9 3
```

Viéndolo a nivel tipo de dato vamos a tener un uint_32 para saber el largo del nombre del pokémon, el nombre del pokemon, un uint_32 indicando la cantidad de posiciones donde se encuentra y un par de uint_32 para cada posición donde se encuentre. Este ejemplo tendrá un tamaño de 39 bytes en memoria principal (lo que ocupa un uint_32 multiplicado por los 8 que tenemos más el largo del nombre del pokemon).

Tamaño de Get Pokemon

Este mensaje tendrá el nombre del pokemon. Un ejemplo del mensaje es:

```
‘Pikachu’
```

Viéndolo a nivel tipo de dato vamos a tener un uint_32 para saber el largo del nombre del pokemon y luego el nombre del pokemon. Este ejemplo tendrá un tamaño de 11 bytes en memoria principal (lo que ocupa un uint_32 más el largo del nombre del pokemon).

Tamaño de Appeared Pokemon

Este mensaje tendrá el nombre del pokemon y para indicar la posición en X y en Y. Un ejemplo del mensaje es:

```
‘Pikachu’ 1 5
```

Viéndolo a nivel tipo de dato vamos a tener un uint_32 para saber el largo del nombre del pokémon, el nombre del pokemon y dos uint_32 indicando la posición. Este ejemplo tendrá un tamaño de 19 bytes en memoria principal (lo que ocupan tres uint_32 más el largo del nombre del pokemon).

Tamaño de Catch Pokemon

Este mensaje tendrá el nombre del pokemon y la posición en X y en Y. Un ejemplo del mensaje es:

```
‘Pikachu’ 1 5
```

Viéndolo a nivel tipo de dato vamos a tener un uint_32 para saber el largo del nombre del pokémon, el nombre del pokemon y luego dos uint_32 indicando la posición. Este ejemplo va a tener un tamaño de 19 bytes en memoria principal (lo que ocupan tres uint_32 más el largo del nombre del pokemon).

Tamaño de Caught Pokemon

Este mensaje tendrá un valor para indicar si se pudo o no atrapar al pokemon (0 o 1). Un ejemplo del mensaje es:

```
0
```

Viéndolo a nivel tipo de dato vamos a tener un uint_32 para saber si se puedo o no atrapar al pokemon. Este ejemplo va a tener un tamaño de 4 bytes en memoria principal (lo que ocupa un uint_32).

Descripción de las entregas

Hito 1: Conexión Inicial

Fecha: 25/04

Objetivos:

- ★ Familiarizarse con Linux y su consola, el entorno de desarrollo y el repositorio.
- ★ Aplicar las Commons Libraries, principalmente las funciones para listas, archivos de conf y logs
- ★ Definir el Protocolo de Comunicación

Implementación mínima:

- ★ Una biblioteca muy básica que permita enviar strings hasta un proceso que actúe de servidor.

Lectura recomendada:

- Tutorial de "Cómo arrancar" de la materia: http://faq.utnso.com.ar/arrancar
- Beej Guide to Network Programming - https://beej.us/guide/bgnet/
- SO UTN FRBA Commons Libraries - https://github.com/sisoputnfrba/so-commons-library
- Guía de Punteros en C - http://faq.utnso.com.ar/punteros

Hito 2: Avance del Grupo

Fecha: 16/05

Objetivos:

- ★ **Proceso Team**: Permitir solamente planificar de forma FIFO un conjunto de entrenadores.
- ★ **Proceso Broker**: Implementación completa de la administración de las colas de mensajes. Aceptar suscripciones a una cola de mensajes específica.
- ★ **Proceso GameBoy**: Permitir el envío de varios mensajes al proceso Broker y el mensaje Appeared Pokemon al proceso Team.

Lectura recomendada:

- Sistemas Operativos, Silberschatz, Galvin - Capítulo 3: Procesos y Capítulo 4: Hilos
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 8: Memoria Principal
- Ejemplo de Implementación de FUSE - https://github.com/sisoputnfrba/so-fuse_example

Hito 3: Checkpoint Presencial en el Laboratorio

Fecha: 13/06 ***(fecha tentativa a la espera de actualización de calendario académico)***

Objetivos:

- ★ **Proceso Team**: Permitir el envío de mensajes al Broker para Catch Pokemon y Get Pokemon.
- ★ **Proceso Broker**: Implementación del sistema de Particiones Dinámicas. Administrar flujo de mensajes y envío de los mismos a los distintos suscriptores.
- ★ **Proceso GameCard**: Comenzar implementación de Tall Grass. Creación de archivos y directorios.
- ★ **Proceso GameBoy**: Finalizar el desarrollo del módulo.

Lectura recomendada:

- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 6: Sincronización de Procesos
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 8: Memoria Principal
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 10: Interfaz del Sistema de Archivos

Hito 4: Avance del Grupo

Fechas: 27/06 ***(fecha tentativa a la espera de actualización de calendario académico)***

Objetivos:

- ★ **Proceso Team**: Implementación del algoritmo RR, desarrollo de métricas.
- ★ **Proceso Broker**: Sistema de reemplazo y compactación.
- ★ **Proceso GameCard**: Avances sobre la implementación del FileSystem de Tall Grass. Poder leer y escribir archivos.

Lectura recomendada:

- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 9: Memoria Virtual
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 11: Implementación de Sistemas de Archivos

Hito 5: Entregas Finales

Fechas: 25/7 - 1/8 - 22/8
(fechas tentativas a la espera de actualización de calendario académico)

Objetivos:

- ★ Probar el TP en un entorno distribuido
- ★ Realizar pruebas intensivas
- ★ Finalizar el desarrollo de todos los procesos
- ★ Todos los componentes del TP ejecutan los requerimientos de forma integral, bajo escenarios de stress.

Lectura recomendada:

- Guías de Debugging del Blog utnso.com - https://www.utnso.com.ar/recursos/guias/
- MarioBash: Tutorial para aprender a usar la consola - http://faq.utnso.com.ar/mariobash
- Tutoral de como desplegar un proyecto - https://github.com/sisoputnfrba/so-deploy