A-MongOs

No sé… ese File-System es medio sospechoso

Cátedra de Sistemas Operativos

Trabajo práctico Cuatrimestral

-1C2021 -
Versión 1.0

Índice

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: 80 días
- Fecha de comienzo: 24/04/2021
- Fecha de primera entrega: 17/07/2021
- Fecha de segunda entrega: 24/07/2021
- Fecha de tercera entrega: 07/08/2021
- Lugar de corrección: Discord y Google Meet

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[1]. 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*.

[1] Durante este cuatrimestre será de modalidad virtual a menos que la facultad defina lo contrario.

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.

Definición del Trabajo Práctico

Esta sección se compone de una introducción y definición de carácter global sobre el trabajo práctico. Posteriormente se explicarán por separado cada uno de los distintos módulos que lo componen, pudiéndose encontrar los siguientes títulos:

- Lineamiento e Implementación: Todos los títulos que contengan este nombre representarán la definición de lo que deberá realizar el módulo y cómo deberá ser implementado. La no inclusión de alguno de los puntos especificados en este título puede conllevar a la desaprobación del trabajo práctico.
- Archivos de Configuración: En este punto se da un archivo modelo y que es lo mínimo que se pretende que se pueda parametrizar en el proceso de forma simple, en caso de que el grupo requiera de algún parámetro extra podrá agregarlo.

Cabe destacar que en ciertos puntos de este enunciado se explicaran exactamente como deben ser las funcionalidades a desarrollar mientras que en otros no se definirá específicamente, quedando su implementación a decisión y definición del equipo. Se recomienda en estos casos siempre consultar a sus ayudantes o en el foro de github.

¿Qué es el trabajo práctico y cómo empezamos?

El objetivo del trabajo práctico consiste en desarrollar una solución que permita la simulación de un sistema distribuido que se basa en la idea del juego Among Us, donde los grupos tendrán que planificar tareas, resolver sabotajes y organizar la información de los tripulantes así como sus bitácoras de trabajo.

Para el desarrollo del mismo se decidió la creación de un sistema bajo la metodología Iterativa Incremental donde se solicitarán en una primera instancia la implementación de ciertos módulos para luego poder realizar una integración total con los restantes.

Recomendamos seguir el lineamiento de los distintos puntos de control que se detallan al final de este documento para su desarrollo. Estos puntos están planificados y estructurados para que sean desarrollados a medida y en paralelo a los contenidos que se ven en la parte teórica de la materia. Cabe aclarar que esto es un lineamiento propuesto por la cátedra y no implica impedimento alguno para el alumno de realizar el desarrollo en otro orden diferente al especificado.

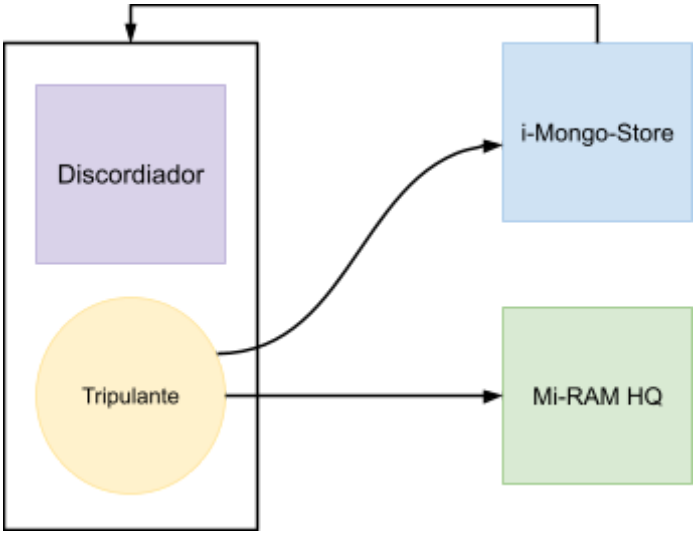
Arquitectura del sistema

Cabe aclarar que esta es una distribución de trabajo estimativa y tentativa que creemos que debe tener el trabajo práctico para que todos los alumnos puedan trabajar equitativamente y aprender conceptos de la materia. Dado que el trabajo funciona bajo el concepto iterativo incremental recomendamos modificar dichos números para que todos los integrantes del grupo participen en varios módulos y no estén sin trabajo hasta la segunda iteración.

- **Discordiador: 1,5**
- **Mi-RAM HQ: 2,**
- **i-Mongo-Store: 1,5**

Por último, se recuerda que *desarrollar únicamente temas de conectividad, serialización y sincronización es insuficiente para poder entender y aprender los distintos conceptos de la materia por lo que será un motivo de desaprobación*.

Diagrama:



Módulo Discordiador

El módulo Discordiador es el encargado de organizar, planificar y ordenar la ejecución de las tareas de los tripulantes dentro del universo del trabajo práctico, por lo cual a fines teóricos se relaciona mucho con los conceptos aplicados a la planificación de procesos y/o hilos[2].

Lineamiento e Implementación

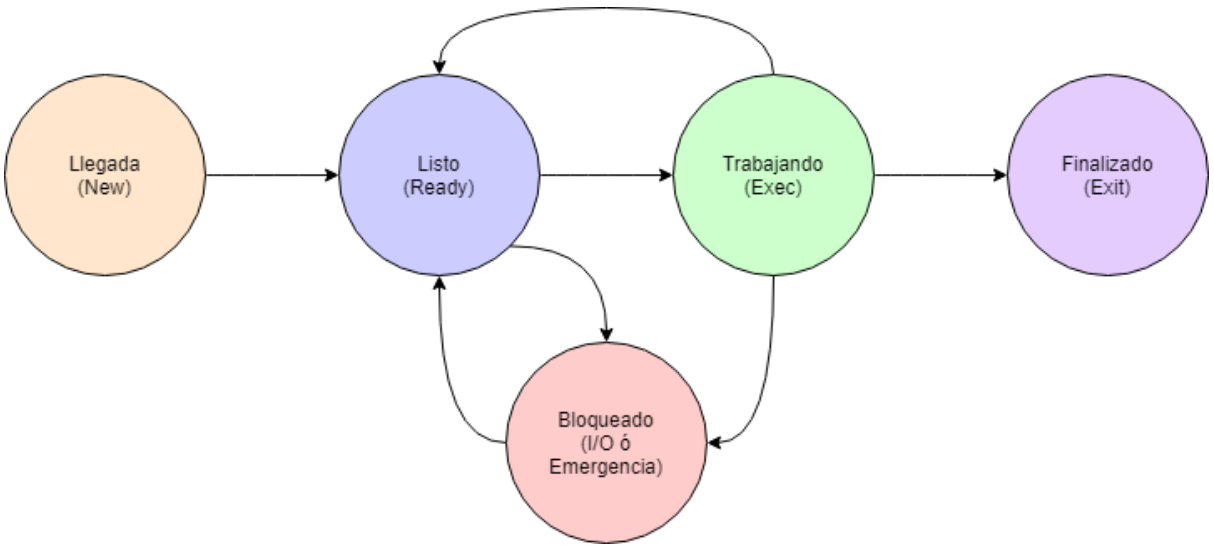
Será un proceso multihilo[3] que tendrá las siguientes funciones:

1. Atenderá de forma paralela a los diferentes tripulantes, coordinando sus tiempos de trabajo (ciclos).
2. Mantendrá el nivel de multiprocesamiento de acuerdo a lo indicado.
3. Cuando reciba por la consola del discordiador el mensaje "iniciar planificación", dará comienzo a la administración de los tripulantes, la cual debe poder ser pausada ("pausar planificación") y reanudada en cualquier momento para poder revisar el estado de los mismos en forma ordenada.
4. Organizará la ejecución de las reparaciones ante situaciones de sabotaje dentro del i-Mongo-Store.

Inicialización

Al iniciar el proceso, el mismo realizará las conexiones necesarias para su funcionamiento (ya sean como cliente o como servidor) y generará las estructuras administrativas necesarias para poder planificar los estados de los diferentes tripulantes.

Diagrama de Estados



[2] Stallings. Capitulo nro. 3: descripción y control de procesos; Silberschatz. Capitulo nro. 3: procesos.
[3] **Cualquier implementación que no sea multihilo será motivo de desaprobación directa.**

Administración de los tripulantes

Cada tripulante, al iniciarse, se conectará al Discordiador y comenzará sus tareas de preparación. Una vez que se encuentre listo, le informará al mismo dicha situación.
Mientras la planificación esté activada, a medida que vaya recibiendo la confirmación de los tripulantes, el Discordiador moverá la cantidad necesaria de los mismos hacia el estado "trabajando" (exec). Para ello, se tendrá en cuenta un límite de tripulantes permitidos para trabajar en simultáneo dictado por archivo de configuración (también conocido como *grado de multiprocesamiento*).

Los tripulantes podrán planificarse bajo uno de los dos esquemas de trabajo diferentes soportados (*elegido por archivo de configuración*):

- Primero en llegar, primero en trabajar (FIFO).
- Turnos Rotativos (RR).

En caso de que se utilice un esquema de Turnos Rotativos, habrá un quantum que indicará la cantidad de acciones que podrá realizar un tripulante antes de ser enviado a descansar (devolverlo al estado de Listo). Es importante tener en cuenta que las acciones incluyen tanto desplazamiento (donde cada movimiento es sólo horizontal en el eje X o vertical en el eje Y, consumiendo 1 quantum), como los tiempos de "espera" en ciertas tareas donde no exista entrada/salida.

Por otra parte, mientras la planificación esté detenida no habrá cambios de estado entre las colas de planificación. Solamente se podrán iniciar patotas, pero todos sus tripulantes quedarán en estado NEW. Aquellos que estén en EXEC deberán detener sus acciones y sus Quantum dejarán de correr.

Administración de las entrada/salida

Dentro de las tareas que pueda ejecutar un tripulante, habrá algunas que requieren de un acceso al módulo de File System, y lo que estas harán será bloquear al tripulante a la espera de que se concrete la tarea en cuestión.

Para poder darle un control más claro a dicho cambio de estado, este vendrá dado como un parámetro de dicha tarea (lo veremos más adelante en el submódulo tripulante). Por lo tanto, tendrá que considerarse que, a fines prácticos, hay dos tipos de colas de bloqueados: uno que es para las tareas de Entrada/Salida y otro para los casos de los sabotajes o emergencias. A este último es donde va a estar el tripulante que resuelve el sabotaje hasta que se finalice el estado de emergencia.

Administración de los sabotajes

En momentos aleatorios[4] llegará un aviso de sabotaje en una región (posición X e Y). Todos los tripulantes deberán dejar sus tareas en ese momento, pasando a estado bloqueado en el siguiente orden:

[4] Serán aleatorios para el Discordiador ya que el que envía el mensaje de sabotaje es el i-Mongo-Store.

1. Los tripulantes que se encontraban en EXEC
2. Los tripulantes que se encontraban en Ready
3. Los tripulantes que finalicen sus Bloqueos por I/O a medida que vayan concluyendo.

En caso de tener más de un tripulante en Exec o Listo, el desempate será por ID de Tripulante (TID) en orden ascendente.

Por último, el sistema deberá buscar al tripulante más cercano a la ubicación del sabotaje y planificarlo para que se mueva a dicha ubicación y resuelva el sabotaje invocando al FSCK[5] del i-Mongo-Store, pasándolo al final de la cola de bloqueados de emergencias.

Una vez solucionado el sabotaje, es decir, que termine su tiempo de duración (definido por archivo de configuración), todos los tripulantes deberán ser desbloqueados en el orden en el cual fueron bloqueados.

Consola

Para poder gestionar algunas tareas, el módulo discordiador contará con una consola que permitirá realizar ciertas acciones sobre los tripulantes:

Iniciar patota

Recibirá como parámetro la cantidad de tripulantes que tendrá la patota, un archivo de tareas que deberán ejecutar los tripulantes de la patota (se indica el formato en la siguiente sección) y una lista de posiciones iniciales de los tripulantes. Por defecto, si no se especifica la posición de la totalidad de los tripulantes, se asume que los restantes inician en 0|0.

Ejemplo,
`INICIAR_PATOTA 5 /home/utnso/tareas/tareasPatota5.txt 1|1 3|4` [6]

Listar tripulantes

Este mensaje devolverá el listado de tripulantes con su estado actual y la patota a la que pertenecen (incluso si son parte de una patota unipersonal)

Ejemplo,
`LISTAR_TRIPULANTES`

Ejemplo de output,

```
---------------------------------------------------------------------
Estado de la Nave: 09/07/21 10:15:01
Tripulante: 1     Patota: 1   Status: EXEC
Tripulante: 2     Patota: 2   Status: EXEC
```

[5] El funcionamiento del FSCK está detallado en el módulo de i-Mongo-Store
[6] En este ejemplo solo tenemos las posiciones de 2 tripulantes, el resto inicia en 0,0

```
Tripulante: 3     Patota: 2   Status: BLOCK I/O
Tripulante: 4     Patota: 3   Status: READY
----------------------------------------------------------------
```

Expulsar tripulante

Con este comando se busca finalizar un tripulante y avisarle a Mi-RAM HQ que este tripulante es eyectado para que deje de mostrarlo en el mapa y en caso de que sea necesario elimine su segmento de tareas. Recibirá como parámetro el id del tripulante.

Ejemplo:
`EXPULSAR_TRIPULANTE 5`

Iniciar Planificación

Con este comando se dará inicio a la planificación, la idea es que hasta este punto no hayan movimientos entre las colas de planificación ni de los tripulantes. Este mensaje no contiene ningún parámetro.

Ejemplo:
`INICIAR_PLANIFICACION`

Pausar Planificación

Este comando lo que busca es detener la planificación en cualquier momento. Este mensaje no contiene ningún parámetro.

Ejemplo:
`PAUSAR_PLANIFICACION`

Obtener Bitácora

Este comando obtendrá la bitácora del tripulante pasado por parámetro a través de una consulta a i-Mongo-Store.

Ejemplo:
`OBTENER_BITACORA 1`

Sub-módulo Tripulante

El sub-módulo Tripulante es el encargado de obtener y ejecutar todas las tareas que se le definan, y mostrar los resultados de estas al usuario. Este submódulo se implementará mediante hilos creados bajo demanda en el Módulo Discordiador.

Instructivo

El submódulo tripulante será una pieza clave dentro de la planificación de nuestro sistema, ya que será el encargado de recorrer la nave y realizar una serie de tareas detalladas para lograr el objetivo de llegar a salvo.
Al momento de iniciar sus operaciones, el tripulante deberá hacer una serie de comprobaciones básicas:

1. Informar al módulo Mi-RAM HQ que desea iniciar, indicando a qué patota pertenece.
    a. En caso de que Mi-RAM HQ no tenga las tareas de la patota, enviarle al módulo de memoria el listado de tareas que va a realizar.
2. Solicitar la primera tarea a realizar.

Además, se deberá interactuar con el módulo de Mi-RAM HQ ante las siguientes situaciones:

1. Informar cada movimiento dentro de la nave.
2. Luego de realizar cada tarea, solicitar la próxima.

El submódulo tripulante contará además una bitácora que se almacenará en el i-Mongo-Store (proceso que se detalla más adelante), en esta bitácora irán registrando todos los acontecimientos que suceden en su viaje en la nave. Estas acciones serán:

- Desplazamiento, indicando origen y destino
- Inicio de una tarea
- Fin de una tarea
- Atender un sabotaje (si el tripulante es planificado para dicha tarea)
- Resolución del sabotaje.

Tareas

Las tareas que realizará un tripulante estarán determinadas en el archivo de tareas que obtendrá como parámetro al momento de recibir el mensaje INICIAR_PATOTA en la consola del discordiador.
Este archivo de tareas tendrá una sola tarea por línea y cada línea tendrá los siguientes parámetros separados por punto y coma (;)

1. Nombre y parámetros de la tarea
2. Ubicación en X
3. Ubicación en Y
4. Duración de la tarea (expresada en ciclos de CPU; en caso de que sea una tarea de Entrada Salida, este tiempo deberá transcurrir en la cola de bloqueados)

El detalle de que realiza cada tarea se encuentra en el Anexo II - Descripción de las tareas del tripulante

Archivo de Configuración

Campo	Tipo	Descripción
IP_MI_RAM_HQ	[String]	El IP del módulo Mi-RAM HQ

PUERTO_MI_RAM_HQ	[Numérico]	El puerto del módulo Mi-RAM HQ
IP_I_MONGO_STORE	[String]	El IP del módulo I-Mongo-Store
PUERTO_I_MONGO_STORE	[Numérico]	El puerto del módulo I-Mongo-Store
GRADO_MULTITAREA	[Numérico]	Cantidad de tripulantes que pueden estar trabajando al mismo tiempo (grado de Multiprocesamiento).
ALGORITMO	[String]	Algoritmo de asignación de espacios de trabajo (FIFO - RR)
QUANTUM	[Numérico]	Cantidad de acciones que podrá realizar un tripulante antes de que sea enviado a descansar
DURACION_SABOTAJE	[Numérico]	Cantidad de ciclos de CPU que llevará resolver el sabotaje en el i-Mongo-Store
RETARDO_CICLO_CPU	[Numérico]	Tiempo en segundos para el retardo de la ejecución de cada ciclo de cpu.

Ejemplo de Archivo de Configuración

```
IP_MI_RAM_HQ=127.0.0.1
PUERTO_MI_RAM_HQ=5001
IP_I_MONGO_STORE=127.0.0.1
PUERTO_I_MONGO_STORE=5001
GRADO_MULTITAREA=2
ALGORITMO=RR
QUANTUM=3
DURACION_SABOTAJE=20
RETARDO_CICLO_CPU=1
```

Módulo Mi-RAM HQ

El módulo Mi-RAM HQ es el encargado de almacenar toda la información relacionada a las instrucciones, PCB's (Patota Control Block) y TCB's (Tripulante Control Block). A su vez, se encargará de mantener un mapa actualizado con las posiciones de cada tripulante durante la ejecución de los mismos.

Lineamiento e Implementación

El módulo será un proceso multihilo[7] que tendrá las siguientes funciones:

1. Atender de forma paralela a los diferentes tripulantes.
2. Mantener el mapa con las posiciones de los tripulantes actualizado.
3. En el momento en que reciba una señal (implementada mediante `signal`) realizar un dump del contenido de la memoria como se indica más adelante.

Inicialización

Al momento de inicializar el proceso se realizarán las siguientes operaciones:

1. Reservar el espacio de memoria indicado por el archivo de configuración.
2. Dibujar el mapa inicial vacío, dado de que al inicio no se encontrará ningún tripulante conectado.
3. Iniciar el servidor, al cual se conectarán los tripulantes para comenzar a ejecutar sus instrucciones.

Administración de la memoria

En Mi-RAM HQ, al momento de inicializar, se reservará un **único bloque** de memoria, el cual será administrado bajo uno de dos esquemas de manejo distintos soportados (elegido por archivo de configuración):

- Segmentación pura[8].
- Paginación simple[9] con memoria virtual.

Cabe aclarar que para **ambos casos**, cada tripulante será tratado como si fuera un **hilo parte de un proceso o patota**. En los próximos apartados se especificará de *forma simplificada* el manejo de ambas técnicas.

Memoria Compartida

Dado que en el tp siempre buscamos tener ciertas analogías con la realidad, nuestros Tripulantes van a tener su bloque de control (TCB) y nuestras Patotas van a tener su propio bloque de control (PCB). Por lo que las siglas no son al azar y los tripulantes serán análogos a los hilos y las patotas análogas a los procesos, dicho esto, en nuestra memoria vamos a contar con 3 tipos de elementos principales:

---

[7] Cualquier implementación que no sea multihilo será motivo de desaprobación directa.
[8] Stallings. Tercera parte: memoria. P. 325; Silberschatz. Capítulo 8: memoria principal. P. 269.
[9] Stallings. Tercera parte: memoria. P. 321; Silberschatz. Capítulo 8: memoria principal. P. 255.

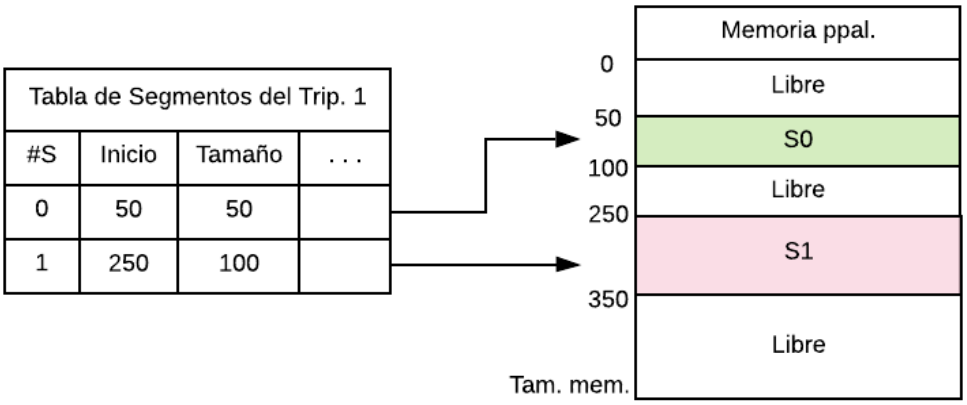
- PCB: Elemento de control de la patota.
- TCB: Elemento de control propio de cada tripulante
- Tareas: Listado de tareas de un tripulante/patota.

Segmentación pura

Este apartado se enfocará en los siguientes puntos: el manejo de memoria, la compactación y la compartición. Es importante aclarar que estos tópicos se explicarán en detalle dependiendo del calendario de su cursada teórica, lo siguiente será una breve explicación.

Manejo de memoria

Para Mi-RAM HQ, se almacenarán los PCB's, los TCB's y sus respectivas instrucciones, es decir, solamente se **utilizaran *dos segmentos* por patota** (uno para el PCB y otro para las tareas) **y *uno* para el tripulante** (para almacenar el TCB). Para las instrucciones, las cuales no tendrán un tamaño fijo, se almacenarán en memoria principal cuando sean recibidas.



Para la elección del lugar en el que se va a guardar un nuevo segmento, se deberá verificar si hay algún hueco libre con el tamaño suficiente para almacenar nuestro segmento. En el caso de que haya más de uno, se utilizará uno de los siguientes criterios[10] (definido por archivo de configuración):

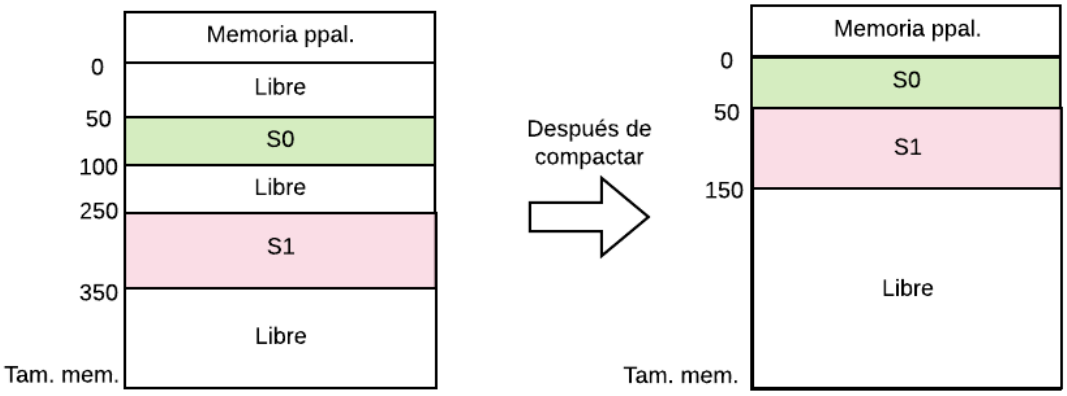
- Mejor ajuste (Best Fit): se coloca el segmento dentro del hueco libre en el que se desperdicie el menor espacio.
- Primer ajuste (First Fit): se coloca el segmento en el primer hueco libre que se encuentre.

Compactación

En el caso de segmentación pura se deberá implementar la *compactación*[11] de la memoria principal para poder aprovechar mejor el espacio libre y así poder recibir más tripulantes/patotas.

Lo que se busca es reacomodar los segmentos de manera tal que todos los huecos libres se unifiquen en uno solo, al final de la memoria.

[10] Stallings. Tercera parte: memoria. P. 317; Silberschatz. Capítulo 8: memoria principal. P. 254.
[11] Stallings. Tercera parte: memoria. P. 316; Silberschatz. Capítulo 8: memoria principal. P. 255.



La compactación deberá ejecutarse en dos ocasiones:

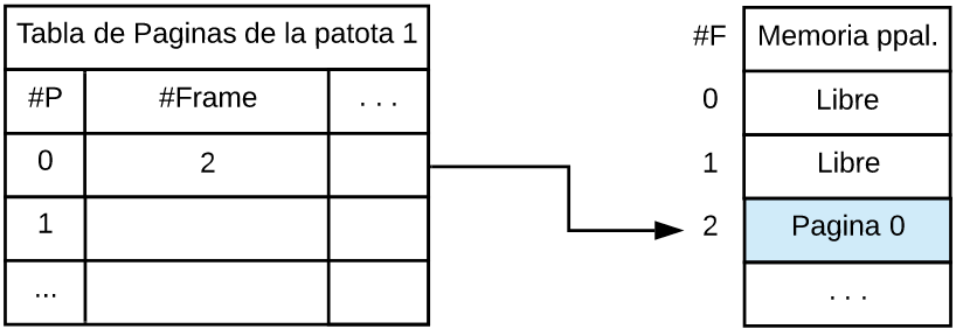
1. Mediante un `signal`.
2. Cuando se quiera escribir en la memoria y se detecte que no hay espacio para el segmento necesario. Tener en cuenta que si luego de realizar la compactación se sigue sin poder guardar, se deberá denegar la solicitud.

Paginación simple

Manejo de memoria

A diferencia del esquema anterior, en este tendremos dos estructuras importantes: las páginas, y los marcos. Las páginas son utilizadas para dividir en partes iguales al proceso que se desea almacenar en memoria principal, mientras que esta última se la dividirá en marcos. Es importante aclarar que el tamaño de la memoria siempre será múltiplo del tamaño de página.

Cada proceso tendrá su tabla de páginas, donde cada página tendrá un marco asignado, los cuales pueden o no estar contiguos, tal y como se indica en el siguiente gráfico.



En nuestra implementación se deberá mantener la información (PCB, Instrucciones y TCBs) en forma contigua a lo largo de las páginas asignadas. Hay que tener en cuenta que las páginas no se separan entre las que contienen un campo u otro, sino que puede darse el caso donde, por ejemplo, el TCB llene la mitad de una página y el resto se complete con las instrucciones.

El tamaño de página/marco estará determinado por el archivo de configuración.

Memoria Virtual (SWAP)

Para el esquema de paginación se va a implementar un área de intercambio para ampliar la memoria.

Para implementar esto se deberá generar un archivo en una ruta definida por archivo de configuración y de un tamaño definido por archivo de configuración, el cual será siempre un múltiplo del tamaño de la página. Y el proceso de swap comenzará una vez que se encuentre llena la memoria principal.

La parametrización del algoritmo de reemplazo se realizará por medio del archivo de configuración y será de alcance global (es decir que una patota puede hacer swap de páginas de otra patota).
Para el reemplazo de las páginas se podrán realizar bajo alguno de los siguiente algoritmos de reemplazo[12]:

- LRU
- Clock (O Algoritmo de Segunda Oportunidad)

En el caso de que se quiera escribir una nueva página y no haya marcos libres en la memoria ni en SWAP, se deberá denegar la solicitud.

Estructuras

Patota Control Block (PCB)[13]

Elemento	Descripción	Tipo de Dato
PID	Identificador de la Patota	uint32_t
Tareas	Dirección base del segmento de Tareas	void*

Tripulante Control Block (TCB)[14]

Elemento	Descripción	Tipo de Dato
TID	Identificador del tripulante	uint32_t
Estado	Estado del tripulante (N/R/E/B)	char
Posición X	Posición del tripulante en el Eje X	uint32_t
Posición Y	Posición del tripulante en el Eje Y	uint32_t
Próxima instrucción	Identificador de la próxima instrucción a ejecutar	uint32_t

[12] Stallings. Tercera parte: memoria. P. NNN; Silberschatz. Capítulo 9: memoria virtual. P. 296
[13] El tamaño total de un PCB en memoria es de 8 bytes.
[14] El tamaño total de un TCB en memoria es de 17 bytes.

Servidor

El módulo de Mi-RAM HQ al ser el encargado de atender las peticiones de los tripulantes, es por esto que el módulo va a tener que responder al menos a los siguientes métodos:

- Iniciar tripulante: será el encargado de crear la o las estructuras administrativas necesarias para que un tripulante pueda ejecutar. En caso de que no se encuentre creada la patota a la que pertenece, deberá solicitar el listado de tareas.
- Recibir tareas de una patota: recibirá el listado de tareas de la patota y los almacenará en la memoria.
- Recibir la ubicación del tripulante: se encarga de obtener la nueva posición del tripulante y de actualizarla en el mapa.
- Enviar próxima tarea: el objetivo es facilitarle al tripulante su próxima tarea a ejecutar.
- Expulsar tripulante: eliminará un tripulante tanto de las estructuras administrativas de la memoria, como también del mapa. Puede incluir liberar algunos frames de la memoria y del swap.

Estos métodos podrán extenderse si el grupo requiere de alguna comprobación extra.

Dibujado del Mapa

Para mostrar el mapa, la cátedra proveerá una biblioteca encargada de dibujar en pantalla el mapa, los personajes y las cajas de recursos. La misma se encuentra descrita en el Anexo I - Dibujado del Mapa de la Nave. El mapa deberá dibujarse utilizando ncurses.

Dump de la Memoria

El dump de la memoria, dado que el proceso estará constantemente mostrando el mapa en pantalla, deberá realizarse en un archivo específico, el cual deberá llamarse `Dump_<Timestamp>.dmp`[15].

El objetivo del dump no es ver el contenido de las páginas, sino ver el estado de asignación de las mismas, por lo que el formato del archivo será diferente según el esquema de memoria utilizado. Para cada esquema se pide lo siguiente[16]:

Segmentación

- Id del Proceso (patota)
- # de segmento
- Dirección de inicio
- Tamaño del segmento

Ejemplo,

[15] Consejo: ver temporal.h en las commons

[16] Es importante tener en cuenta que dado que esto se va a leer en una terminal los espacios entre cada “columna” del dump son importantes, por favor traten de hacerlos fáciles de leer

```
-------------------------------------------------------------------------
Dump: 09/07/21 10:11:12
Proceso: 1  Segmento: 1 Inicio: 0x0000    Tam: 20b
Proceso: 1  Segmento: 2 Inicio: 0x0013    Tam: 450b
Proceso: 2  Segmento: 1 Inicio: 0x01D6    Tam: 20b
-------------------------------------------------------------------------
```

Paginación

- # de Frame
- Estado (Libre u Ocupado)
- Id del proceso (patota)
- # de página

Ejemplo,

```
-------------------------------------------------------------------------
Dump: 09/07/21 10:11:12
Marco:0     Estado:Ocupado    Proceso: 1  Pagina: 2
Marco:1     Estado:Ocupado    Proceso: 3  Pagina: 5
Marco:2     Estado:Libre      Proceso: -  Pagina: -
Marco:3     Estado:Ocupado    Proceso: 2  Pagina: 1
-------------------------------------------------------------------------
```

Archivo de Configuración

Campo	Tipo	Descripción
TAMANIO_MEMORIA	[Numérico]	Tamaño en bytes de la memoria real
ESQUEMA_MEMORIA	[String]	Esquema de administración de memoria, puede ser: SEGMENTACION o PAGINACION
TAMANIO_PAGINA	[Numérico]	Tamaño en bytes de la cada página
TAMANIO_SWAP	[Numérico]	Tamaño en bytes del área de SWAP
PATH_SWAP	[String]	Ubicación del archivo de SWAP
ALGORITMO_REEMPLAZO	[String]	LRU o CLOCK - Indica el algoritmo de reemplazo a utilizar en el esquema de paginación
PUERTO	[Numérico]	El puerto del servidor al cual se conectarán los clientes.

Ejemplo de Archivo de Configuración

```
TAMANIO_MEMORIA=2048
```

```
ESQUEMA_MEMORIA=PAGINACION
TAMANIO_PAGINA=64
TAMANIO_SWAP=4096
PATH_SWAP=/home/utnso/swapFile.bin
ALGORITMO_REEMPLAZO=LRU
PUERTO=5001
```

Módulo i-Mongo-Store

El módulo i-Mongo-Store es el encargado de almacenar toda la información de manera persistente, análogamente a como funciona el File System de un sistema operativo real. Por lo tanto, será capaz de finalizar su ejecución y volver a reanudarse manteniendo la información.

Lineamiento e Implementación

Será un proceso multihilo[17] que tendrá las siguientes funciones:

1. Atenderá de forma paralela a los diferentes tripulantes.
2. Será capaz de permitir lecturas y escrituras concurrentes de los diferentes archivos.
3. Cuando reciba una señal, implementada mediante signal (SIGUSR1) le notificará de que ocurrió un sabotaje al proceso Discordiador.

Inicialización

Al momento de inicializar el proceso se realizarán las siguientes operaciones:

1. Controlar si se va a trabajar con un File System (FS) limpio o si se debe recuperar el FS existente[18].
2. En caso de tener que inicializar un FS limpio tendrá que generar todos los archivos de bloques según indique la metadata.
3. Iniciar el servidor al cual se conectarán los tripulantes para comenzar a ejecutar.

Especificación del File System

La estructura básica del mismo se basa en una estructura de árbol de directorios para representar la información administrativa y los datos 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.
Para este file system, como lo que se busca es generar los recursos que se van a utilizar en la nave, cada archivo va a tener un caracter de llenado el cual va a ser el único que deba encontrarse dentro de un archivo y el cual se van a incrementar o decrementar por medio de las tareas ejecutadas por los tripulantes. Ejemplo: En el caso de tener 5 oxígenos el archivo deberá tener “OOOOO” en su contenido.

SuperBloque

Este archivo contendrá toda la información administrativa del file system y el bitmap para poder identificar los bloques libres.
Dentro de este archivo encontraremos la siguiente información

- Block_size: Indica el tamaño en bytes de cada bloque, este dato será de tipo uint32_t
- Blocks: Indica la cantidad de bloques del File System, este dato será de tipo uint32_t
- Bitmap: Será un bitmap donde estará el estado de los bloques del FileSystem, donde un 0 indica un bloque vacío y un 1 indica un bloque ocupado.

[17] Cualquier implementación que no sea multihilo será motivo de desaprobación directa.
[18] Esto se hará verificando una serie de archivos que se detallan más abajo.

Dicho archivo deberá encontrarse en la ruta `[Punto_Montaje]/SuperBloque.ims`

Blocks

Este archivo será el encargado de almacenar todos los bloques de nuestro File System. Al inicializar el File System, si el mismo no está formateado, **deberá crearse el archivo con el tamaño de la cantidad de bloques por el tamaño del bloque.**
El archivo de bloques deberá encontrarse en `[Punto_Montaje]/Blocks.ims`

Internamente, este archivo deberá ser mapeado[19] a memoria y se le realizará una copia[20] sobre la cual se trabajará para lograr simular la concurrencia y asincronismo en las bajadas a disco de los sistemas operativos reales. El tiempo de sincronización[21] hacia el archivo físico de Blocks.ims será especificado por archivo de configuración. **Cualquier otra implementación es motivo de desaprobación directa**.

Files

En esta carpeta se almacenará la metadata para la reconstrucción de los distintos archivos que se crearán en el sistema. Estos archivos simularán ser un contenedor de recursos, se llenarán con un mismo caracter.
Para esto se almacenará un archivo de metadata por cada archivo existente en el sistema que contendrá:

- El tamaño del archivo expresado en bytes
- La cantidad de bloques del archivo
- La lista de bloques que lo conforman
- El caracter con el cual se llena el archivo
- MD5 del archivo, deberá actualizarse cada vez que se realicen modificaciones en el mismo.

La File metadata de los archivos deberá encontrarse en: `[Punto_Montaje]/Files`
Ej: `/home/utnso/mnt/Files/Oxigeno.ims`

Ej:

- SIZE=132
- BLOCK_COUNT=3
- BLOCKS=[1,2,3]
- CARACTER_LLENADO=O
- MD5_ARCHIVO=BD1014D173BA92CC014850A7087E254E

Bitácora

Asimismo, la bitácora de cada uno de los tripulantes del sistema será guardada dentro del File System en una carpeta aparte pero siguiendo el mismo formato de los archivos comunes, esto quiere decir que cada bitácora tendrá un valor de SIZE y la lista de BLOCKS que la conforman y estos archivos se encontraran dentro de un path específico dentro de nuestro file system.
Los archivos de bitácora deberán encontrarse en: `[Punto_Montaje]/Files/Bitacoras`
Ej: `/home/utnso/mnt/Files/Bitacoras/Tripulante1.ims`

[19] investigar la función mmap()
[20] investigar la función memcpy()
[21] investigar la función msync()

La información que se almacenará dentro de las bitácoras será información relevante a las acciones que realice cada tripulante, estas serán notificadas por el tripulante al i-Mongo-Store a medida que las realice.
Las mismas serán algunas de las siguientes opciones según correspondan:

- Se mueve de X|Y a X'|Y'
- Comienza ejecución de tarea X
- Se finaliza la tarea X
- Se corre en pánico hacia la ubicación del sabotaje
- Se resuelve el sabotaje

Sabotajes

El viaje es largo y los riesgos son muchos. Por eso es posible que, en cualquier momento dado, se pueda corromper información crucial almacenada en cualquier componente del File System. Cabe aclarar que los sabotajes se van a hacer de a un tipo a la vez.
La forma de corromper la información del File System es simple; a mano, alguien con intenciones de verificar que el file system sea capaz de recuperarse, modificará un archivo y enviará la señal SIGUSR1 al proceso i-Mongo-Store para que éste notifique al módulo Discordiador.
En algún momento, un tripulante (designado por el discordiador) solicitará que se controle el status del File System y dará inicio al proceso de fsck.

Procedimiento

1. El i-Mongo-Store afectado le debe notificar al Discordiador sobre el estado de alerta. Este moverá todos los tripulantes al estado correspondiente y ejecutará al instante la tarea de repararlo.
2. Una vez sea invocado por el Discordiador, comenzará el protocolo de **fsck** para el contenido afectado y realizar las correcciones correspondientes.
3. Al finalizar se le notificará al Dicordiador la finalización del protocolo **fsck**

fsck (File System Catastrophe Killer)

- Sabotajes en el superbloque
    - Cantidad de bloques: Se debe garantizar que la cantidad de bloques que hay en el sistema sea igual a la cantidad que dice haber en el superbloque. **Reparación**: sobreescribir "cantidad de bloques" del superbloque con la cantidad de bloques real en el disco.
    - Bitmap: Se debe garantizar que los bloques libres y ocupados marcados en el bitmap sean los que dicen ser. **Reparación**: corregir el bitmap con lo que se halle en la metadata de los archivos.
- Sabotajes en Files:

    - Size: Se debe asegurar que el tamaño del archivo sea el correcto, validando con el contenido de sus bloques. **Reparación**: Asumir correcto lo encontrado en los bloques.
    - Block_count y blocks: Al encontrar una diferencia entre estos dos se debe restaurar la consistencia. **Reparación**: Actualizar el valor de Block_count en base a lo que está en la lista de Blocks.
    - Blocks: Se debe validar que los números de bloques en la lista sean válidos dentro del FS. **Reparación**: Restaurar archivo.
- Sabotajes en los bloques
    - Para todos las inconsistencias dentro de un bloque (la cual estará marcada por tener el bloque lleno de un caracter especial, especificado por archivo de configuración). **Reparación**: Restaurar archivo.

El proceso de restauración de archivos consiste en tomar como referencia el size del archivo y el caracter de llenado e ir llenando los bloques hasta completar el size del archivo, en caso de que falten bloques, los mismos se deberán agregar al final del mismo.

Archivo de Configuración

Campo	Tipo	Descripción
PUNTO_MONTAJE	[String]	Ruta donde se montará el FS.
PUERTO	[Numérico]	El puerto en el cual se iniciará el servidor
TIEMPO_SINCRONIZACION	[Numérico]	Tiempo entre bajadas de memoria a disco.

Ejemplo de Archivo de Configuración

```
PUNTO_MONTAJE=/home/utnso/polus
PUERTO=5001
TIEMPO_SINCRONIZACION=150000
```

Descripción de las entregas

Hito 1: Conexión Inicial

Fecha: 08/05/2021

Objetivos:

- Familiarizarse con Linux y su consola, el entorno de desarrollo y el repositorio.
- Aprender a utilizar las Commons Libraries, principalmente las funciones para listas, archivos de conf y logs.
- Definir el Protocolo de Comunicación.
- Comenzar el desarrollo del módulo discordiador como cliente de Mi-RAM HQ y de i-Mongo-Store
- Atender conexiones del discordiador en Mi-RAM HQ e i-mongo-store

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: 29/05/2021

Objetivos:

- **Proceso Discordiador:** Generar estructuras para administrar las colas de planificación y permitir la planificación de tripulantes mediante FIFO
- **Proceso Mi-RAM HQ:** Generar las estructuras base de la memoria y responder a los mensajes del discordiador.
- **Proceso i-MongoStore:** Iniciar el File System y generar las estructuras en memoria necesarias para poder contestar las peticiones del Discordiador

Lectura recomendada:

- Sistemas Operativos, Stallings, William 5ta Ed. - Parte IV: Planificación
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 5: Planificación

Hito 3: Checkpoint Obligatorio Virtual - Vía pantalla compartida

Fecha: 19/06/2021

Objetivos:

- **Proceso Discordiador:** Permitir la planificación mediante FIFO y RR e implementar el manejo del protocolo ante sabotajes.
- **Proceso Mi-RAM HQ:** Administrar la memoria mediante el esquema de paginación sin swap y segmentación sin compactación..
- **Proceso i-MongoStore:** Poder iniciar con un FileSystem existente y poder operar sobre los diferentes archivos**.**

Lectura recomendada:

- Sistemas Operativos, Stallings, William 5ta Ed. - Parte IV: Planificación
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 5: Planificación
- Sistemas Operativos, Stallings, William 5ta Ed. - Parte VII: Gestión de la memoria (Cap. 7)
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 8: Memoria principal

Hito 4: Avance del Grupo

Fechas: 03/07/2021

Objetivos:

- **Proceso Discordiador:** Permitir el funcionamiento completo del proceso incluyendo el protocolo para los sabotajes, la detención y reanudación de la planificación tanto en FIFO como en RR.
- **Proceso Mi-RAM HQ:** Implementar segmentación con compactación e implementar la base de SWAP para paginación.
- **Proceso i-MongoStore:** Implementar el protocolo de FSCK.

Lectura recomendada:

- Sistemas Operativos, Stallings, William 5ta Ed. - Parte VII: Memoria virtual (Cap. 8)
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 9: Memoria virtual
- Sistemas Operativos, Stallings, William 5ta Ed. - Parte V: Gestión de ficheros (Cap. 12)
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 10: Interfaz del sistema de archivos

Hito 5: Entregas Finales

Fechas: 17/07/2021 - 24/07/2021 - 07/08/2021

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

Anexo I - Dibujado del Mapa de la Nave

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 II - Descripción de las tareas del tripulante

Hay todo tipo de tareas que puede realizar un tripulante, algunas de ellas van a crear recursos para la nave, otras se van a encargar de eliminar residuos y otras simplemente van a ser tareas de control.
Para facilitar el desarrollo del trabajo práctico vamos a detallar el set de tareas que requieren de una acción de Entrada/Salida.
Teniendo en cuenta que en nuestro archivo de tareas va a estar separado por punto y coma (;) los parámetros que reciban nuestras tareas van a estar en el primer elemento y van a estar separados por espacios.

```
TAREA PARAMETROS;POS X;POS Y;TIEMPO
```

Las tareas que no se encuentren en este anexo consistirán en llegar a la ubicación y esperar que pase el tiempo solicitado y no tienen que hacer ninguna operación sobre archivos de recursos.

Tarea	GENERAR_OXÍGENO
Parámetros	CANTIDAD: Cantidad de Oxígeno a Generar (Número)
Pasos	1. Verificar que exista un archivo llamado Oxigeno.ims en el i-Mongo-Store 2. Si existe saltar al paso 4. 3. Si no existe el archivo, crearlo y asignarle el carácter de llenado O 4. Agregar tantos caracteres de llenado del archivo como indique el parámetro CANTIDAD
Ejemplo	GENERAR_OXIGENO 12;2;3;5

Tarea	CONSUMIR_OXÍGENO
Parámetros	CANTIDAD: Cantidad de Oxígeno a Consumir (Número)
Pasos	1. Verificar que exista un archivo llamado Oxigeno.ims en el i-Mongo-Store 2. Si no existe informar que no existe el archivo y finalizar la tarea. 3. Eliminar tantos caracteres de llenado del archivo como indique el parámetro CANTIDAD. En caso de que intenten eliminarse de más, dejar el archivo completamente vacío e informar por log que se quisieron eliminar más caracteres de los existentes.
Ejemplo	CONSUMIR_OXIGENO 120;2;3;1

Tarea	GENERAR_COMIDA
Parámetros	CANTIDAD: Cantidad de Comida a Generar (Número)
Pasos	1. Verificar que exista un archivo llamado Comida.ims en el i-Mongo-Store 2. Si existe saltar al paso 4. 3. Si no existe el archivo, crearlo y asignarle el caracter de llenado C 4. Agregar tantos caracteres de llenado del archivo como indique el

	parámetro CANTIDAD
Ejemplo	GENERAR_COMIDA 4;2;3;1

Tarea	CONSUMIR_COMIDA
Parámetros	CANTIDAD: Cantidad de Comida a consumir (Número)
Pasos	1. Verificar que exista un archivo llamado Comida.ims en el i-Mongo-Store 2. Si no existe informar que no existe el archivo y finalizar la tarea. 3. Eliminar tantos caracteres de llenado del archivo como indique el parámetro CANTIDAD. En caso de que intenten eliminarse de más, dejar el archivo completamente vacío e informar por log que se quisieron eliminar más caracteres de los existentes.
Ejemplo	CONSUMIR_COMIDA 1;2;3;4

Tarea	GENERAR_BASURA
Parámetros	CANTIDAD: Cantidad de Basura a Generar (Número)
Pasos	1. Verificar que exista un archivo llamado Basura.ims en el i-Mongo-Store 2. Si existe saltar al paso 4. 3. Si no existe el archivo, crearlo y asignarle el caracter de llenado B 4. Agregar tantos caracteres de llenado del archivo como indique el parámetro CANTIDAD
Ejemplo	GENERAR_BASURA 12;2;3;5

Tarea	DESCARTAR_BASURA
Parámetros	-
Pasos	1. Verificar que exista un archivo llamado Basura.ims en el i-Mongo-Store 2. Si el archivo existe, Eliminarlo 3. Si el archivo NO existe, notificar por el log
Ejemplo	DESCARTAR_BASURA