Master of Files

I'm querying your blocks

Twisting your data and smashing your duplicates

Cátedra de Sistemas Operativos - UTN FRBA

Trabajo práctico Cuatrimestral

- 2C2025 -

Versión 1.1

Índice

Historial de Cambios

v1.0 (02/09/2025) Publicación del enunciado

v1.1 (03/10/2025) Ajustes de redacción y aclaraciones

- *Se agrega que todos los módulos reciban el archivo de config como parámetro de ejecución.*
- *Se ajustaron algunos logs obligatorios para facilitar su implementación.*
- *Se corrigieron algunos logs obligatorios donde debería decir File en lugar de archivo.*

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.

Debido al fin académico del trabajo práctico, los conceptos que se verán reflejados son, en general, versiones simplificadas o alteradas de los componentes reales de hardware y de sistemas operativos vistos en las clases, a fin de resaltar aspectos de diseño o simplificar su implementación.

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.

Características

- Modalidad: grupal (5 integrantes ± 0) y obligatorio
- Fecha de comienzo: 02/09/2025
- Fecha de primera entrega: 29/11/2025
- Fecha de segunda entrega: 06/12/2025
- Fecha de tercera entrega: 20/12/2025
- Lugar de corrección: Laboratorio de Sistemas - 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 o logs de su funcionamiento de la forma más clara posible, para ello se les proveerá en cada módulo un listado de logs mínimos y obligatorios.

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 lo visto en clase o lo escrito en

el documento *es motivo de desaprobación del trabajo práctico*. Esta etapa al ser la conclusión del todo el trabajo realizado durante el cuatrimestre *no es recuperable*.

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, es mandatoria la lectura y entendimiento de las Normas del Trabajo Práctico donde se especifican todos los lineamientos de cómo se desarrollará la materia durante el cuatrimestre.

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**: Contendrá la definición funcional y aspectos de implementación técnica obligatorios del módulo en cuestión. La no inclusión de alguno de los puntos especificados en este título puede conllevar a la desaprobación del trabajo práctico.

- **Archivos de Configuración**: Se describirán los parámetros mínimos requeridos para ajustar el comportamiento de cada módulo ante cada prueba, sin recompilar. Durante la evaluación, deberá alcanzar con detener la ejecución, modificar el archivo de configuración y volver a ejecutar el módulo. En caso de que el grupo requiera de algún parámetro extra, podrá agregarlo.

Cada módulo contará con un listado de **logs mínimos y obligatorios** los cuales deberán realizarse utilizando la biblioteca de so-commons-library provista por la cátedra y los mismos deberán estar como LOG_LEVEL_INFO, pudiendo ser extendidos por necesidad del grupo utilizando LOG_LEVEL_DEBUG.

En caso de no cumplir con los logs mínimos y/o no guardarlos en archivo, *se considerará que el TP no es apto para ser evaluado* y por consecuencia el mismo estará *desaprobado*.

Cabe destacar que en ciertos puntos de este enunciado se explicarán exactamente cómo 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 en el foro de github, dado que las justificaciones deberán ser expuestas en el coloquio.

¿Qué es el trabajo práctico y cómo empezamos?

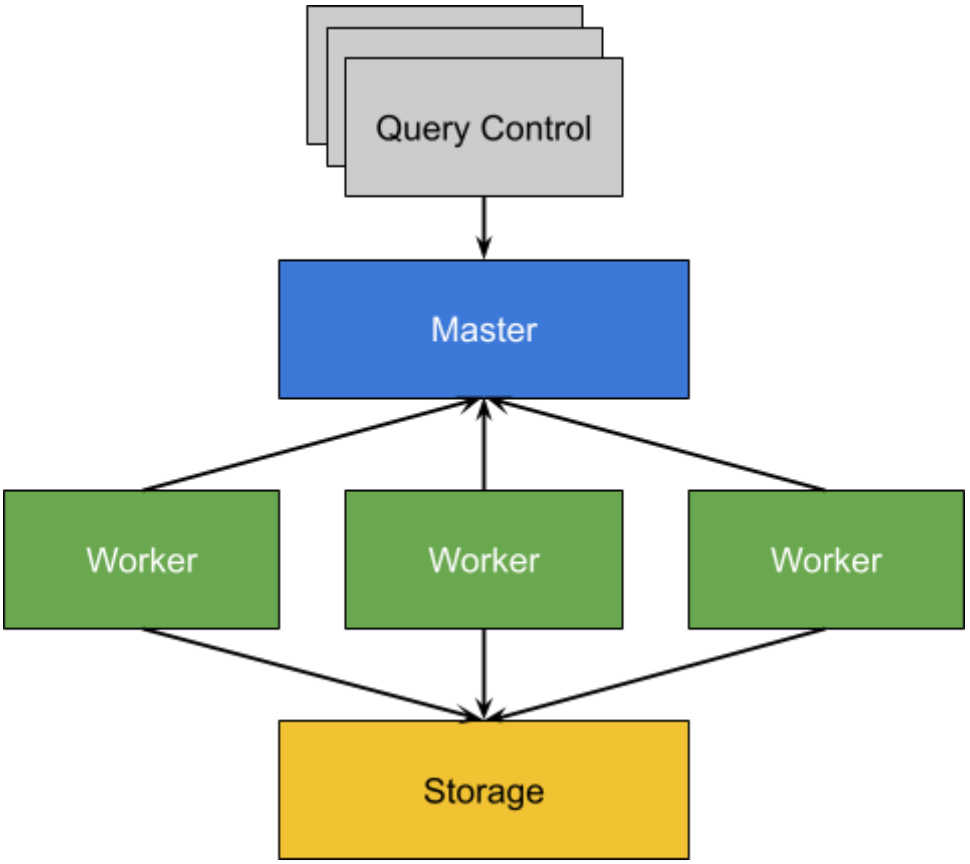
El objetivo del trabajo práctico consiste en desarrollar una solución que permita la simulación de un sistema distribuido, donde los grupos tendrán que planificar procesos, resolver peticiones al sistema, y administrar de manera adecuada una memoria bajo los esquemas explicados en sus correspondientes módulos.

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

Este trabajo práctico correrá una serie de módulos, los cuales deberán ser capaces de correr en diferentes computadoras y/o máquinas virtuales.



NOTA: El sentido de las flechas indica dependencias entre módulos. Ejemplo: al momento de iniciar la ejecución del Worker es necesario contar con el Master y el Storage previamente iniciados.

Aclaración importante

Desarrollar únicamente temas de conectividad, serialización, sincronización o el módulo Query Control es insuficiente para poder entender y aprender los distintos conceptos de la materia. Dicho caso será un motivo de desaprobación directa.

Módulo: Query Control

El módulo Query Control es el encargado de enviar a ejecutar una Query al sistema. Dicha Query estará representada por un archivo que incluirá una serie de instrucciones relativas al manejo de archivos (de ahora en más, "**Files**") que serán operados en el sistema.

Lineamiento e Implementación

Cada Query Control al momento de iniciarse deberá poder recibir como mínimo los siguientes parámetros de ejecución[1]:

- Archivo de configuración, el cual se encuentra detallado más abajo.
- Archivo de Query, es el nombre del archivo.
- Prioridad, siendo un valor numérico mayor o igual a 0.

```
➜ ~ ./bin/query [archivo_config] [archivo_query] [prioridad]
```

Una vez iniciado el módulo, el mismo deberá conectarse al Módulo Master, enviarle el nombre del `archivo_query` y la prioridad, y deberá quedarse a la espera de los mensajes que el módulo Master le envíe. Estos mensajes podrán contener información leída por la Query o el aviso de finalización de la Query.

Logs mínimos y obligatorios

Conexión al master: "`## Conexión al Master exitosa. IP: <ip>, Puerto: <puerto>`"

Envío de Query: "`## Solicitud de ejecución de Query: <archivo_query>, prioridad: <prioridad>`"

Lectura de File: "`## Lectura realizada: File <File:Tag>, contenido: <CONTENIDO>`"

Finalización de la Query: "`## Query Finalizada - <MOTIVO>`"

Archivo de Configuración

Campo	Tipo	Descripción
IP_MASTER	String	IP del módulo Master
PUERTO_MASTER	Numérico	Puerto del módulo Master
LOG_LEVEL	String	Nivel de detalle máximo a mostrar. Compatible con log_level_from_string()

[1] Se recomienda consultar la guía de argumentos para el main.

Ejemplo de Archivo de Configuración

```
IP_MASTER=127.0.0.1
PUERTO_MASTER=9001
LOG_LEVEL=INFO
```

Módulo: Master

El módulo **Master** será el encargado de la gestión de las diferentes peticiones (Queries) que se envían desde los módulos Query Control.

Lineamiento e Implementación

Al momento de iniciarse deberá poder recibir como mínimo el siguiente parámetro de ejecución:

- Archivo de configuración, el cual se encuentra detallado más abajo.

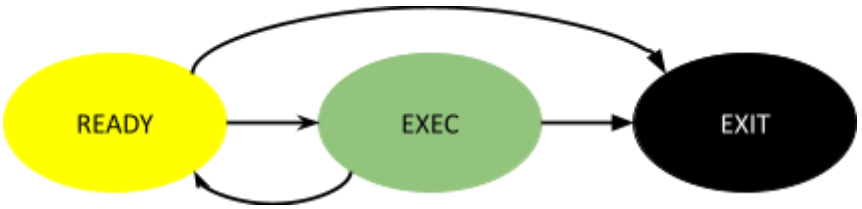
```
➜ ~ ./bin/master [archivo_config]
```

Para gestionar estas Queries, el módulo Master dispondrá de un puerto en el cual escuchará las conexiones de los Query Control. Estas conexiones **se deberán mantener** hasta que la Query enviada por el Query Control finalice.

Durante la ejecución de las Queries, el módulo Master recibirá mensajes de las lecturas realizadas por el Worker correspondiente, los cuales deberán reenviarse al Query Control asociado.

Planificación

El Módulo Master será el encargado de planificar las Queries que le lleguen de parte de los Query Control. Las mismas se pueden encontrar en alguno de los siguientes estados:



Al momento de recibir una Query por parte de un Query Control, se le asignará un identificador único en todo el sistema, que será un valor autoincremental iniciado en 0.

Una vez hecho esto, se la dejará en el estado READY, donde estará a la espera de ser enviada a un Worker en base al algoritmo de planificación elegido. Al momento de que se seleccione un Worker, la Query pasará del estado READY al estado EXEC, y se procederá a esperar los mensajes de lectura recibidos para ser reenviados al Query Control, o eventualmente el resultado final de la ejecución.

La cantidad de Workers conectados nos indicará el grado de multiprocesamiento del sistema. El proceso Master será el responsable de indicarle a cada Worker qué Query deberá ejecutar.

Algoritmos

Los algoritmos a utilizar en la planificación de corto plazo son:

- FIFO
- Prioridades con desalojo y aging

FIFO

Las ejecuciones de las Queries se irán asignando a cada Worker por orden de llegada. Una vez que todos los Workers tengan una Query asignada se encolará el resto de las Queries en el estado READY.

Prioridades con desalojo y aging

Las Queries se enviarán a ejecutar según su prioridad, siendo 0 la prioridad más alta (a mayor número, menor prioridad).

En caso que los Workers estuvieran ocupados y una Query nueva tuviera más prioridad que alguna de las que estuviera en ejecución, deberá ser *desalojada* aquella que tuviera la menor prioridad.

Para desalojar una Query de un Worker, se deberá solicitar al Worker su desalojo. Este último le devolverá al Master el Program Counter (PC) para que éste luego sepa desde dónde reanudarlo.

Al momento de volver a planificar una Query previamente desalojada, se deberá enviar el PC previamente recibido para poder retomar en el lugar correcto.

Aging

Para que una Query de baja prioridad no sufra starvation, se implementará un esquema de aging tal que se aumente la prioridad de aquellas Queries que estén en READY por un período prolongado de tiempo. A cada Query se le irá reduciendo en 1 (uno) su número de prioridad cada vez que se cumpla cierto intervalo en milisegundos en READY (definido por archivo de configuración), hasta que la misma sea seleccionada para ejecutar.

Desconexión de Query Control

Ante la desconexión de un módulo Query Control, la Query enviada por el mismo deberá iniciar el proceso de cancelación inmediatamente. En el caso de que la Query se encuentre en READY, la misma se deberá enviar a EXIT directamente. En caso de que la Query se encuentre en EXEC, se deberá notificar al Worker que la está ejecutando que debe desalojar dicha Query y una vez recibido su contexto se enviará la Query a EXIT.

Conexión y desconexión de Workers

Los módulos Worker se podrán conectar y desconectar en tiempo de ejecución. En caso de que un nuevo Worker se conecte al Master, se deberá planificar la siguiente Query según el algoritmo configurado.

Al momento de que un Worker se desconecte, la Query que se encontraba en ejecución en dicho Worker se finalizará con error y se notificará al Query Control correspondiente.

Logs mínimos y obligatorios

Conexión de Query Control: “## Se conecta un Query Control para ejecutar la Query <PATH_QUERY> con prioridad <PRIORIDAD> - Id asignado: <QUERY_ID>. Nivel multiprocesamiento <CANTIDAD>”

Conexión de Worker: “## Se conecta el Worker <WORKER_ID> - Cantidad total de Workers: <CANTIDAD>”

Desconexión de Query Control: “## Se desconecta un Query Control. Se finaliza la Query <QUERY_ID> con prioridad <PRIORIDAD>. Nivel multiprocesamiento <CANTIDAD>”

Desconexión de Worker: “## Se desconecta el Worker <WORKER_ID> - Se finaliza la Query <QUERY_ID> - Cantidad total de Workers: <CANTIDAD> ”

Envío de Query a Worker: “## Se envía la Query <QUERY_ID> (<PRIORIDAD>) al Worker <WORKER_ID>”

Desalojo de Query en Worker: “## Se desaloja la Query <QUERY_ID> (<PRIORIDAD>) del Worker <WORKER_ID> - Motivo: <DESCONEXION / PRIORIDAD>”

Cambio de prioridad de Query: “##<QUERY_ID> Cambio de prioridad: <PRIORIDAD_ANTERIOR> - <PRIORIDAD_NUEVA>”

Finalización de Query en Worker: “## Se terminó la Query <QUERY_ID> en el Worker <WORKER_ID>”

Envío de lectura de Query a Query Control: “## Se envía un mensaje de lectura de la Query <QUERY_ID> en el Worker <WORKER_ID> al Query Control”

Archivo de Configuración

Campo	Tipo	Descripción
PUERTO_ESCUCHA	Numérico	Puerto al cual se deberán conectar los procesos Query Control y Workers
ALGORITMO_PLANIFICACION	String	Algoritmo de planificación a utilizar FIFO / PRIORIDADES
TIEMPO_AGING	Numérico	Tiempo en milisegundos que deberán pasar antes de que se le reduzca el número de prioridad. El valor 0 (cero) implicará “sin aging”
LOG_LEVEL	String	Nivel de detalle máximo a mostrar. Compatible con log_level_from_string()

Ejemplo de Archivo de Configuración

```
PUERTO_ESCUCHA=9001
ALGORITMO_PLANIFICACION=PRIORIDADES
TIEMPO_AGING=2500
LOG_LEVEL=INFO
```

Módulo: Worker

El Módulo Worker en el contexto de nuestro trabajo práctico será el encargado de ejecutar las diferentes Queries, de una a la vez.

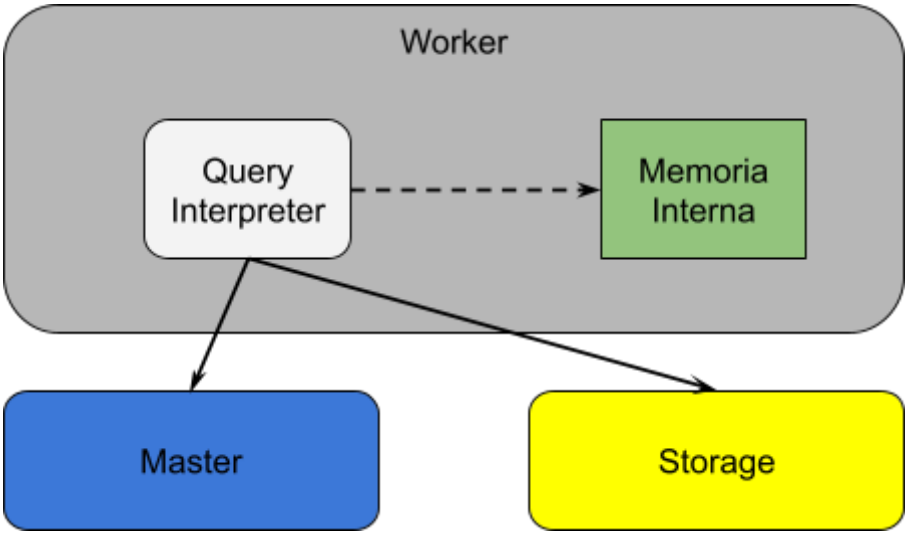
Lineamiento e Implementación

Cada Worker al momento de iniciarse deberá poder recibir como parámetro el archivo de configuración y un identificador propio del Worker, el cuál se encuentra detallado más abajo

```
➜ ~ ./bin/worker [archivo_config] [ID Worker]
```

Al momento de iniciar el Worker, el mismo deberá conectarse al módulo Storage y una vez realizada la conexión deberá conectarse al módulo Master indicando su ID a fin de que el Master pueda identificarlo.

Para poder ejecutar las Queries el módulo Worker contará con una Memoria Interna y un Query Interpreter.



Query Interpreter

El componente de Query Interpreter, será el encargado de ir ejecutando cada una de las **instrucciones** de las Queries. Estas acciones podrán interactuar directamente sobre el componente de Memoria Interna y/o comunicarse con el Módulo de Storage.

Para esto, el Worker tendrá un path indicado en el archivo de configuración donde se encontrarán todas las Queries. Al recibir del Master la solicitud de ejecutar una nueva Query, deberá buscar el archivo en este Path para parsearlo a partir del PC indicado. De ahí en más, se deberán ejecutar una por una las instrucciones, respetando el retardo configurado.

Los parámetros de las instrucciones se separan por espacios. En el caso de los tags de los Files[2], siempre se separarán del nombre del File por ":".

El Query interpreter admite las siguientes instrucciones:

Instrucciones

CREATE

Formato: `CREATE <NOMBRE_FILE>:<TAG>`

La instrucción CREATE solicitará al módulo Storage la creación de un nuevo File con el Tag recibido por parámetro y con tamaño 0.

TRUNCATE

Formato: `TRUNCATE <NOMBRE_FILE>:<TAG> <TAMAÑO>`

La instrucción TRUNCATE solicitará al módulo Storage la modificación del tamaño del File y Tag indicados, asignando el tamaño recibido por parámetro (deberá ser múltiplo del tamaño de bloque).

WRITE

Formato: `WRITE <NOMBRE_FILE>:<TAG> <DIRECCIÓN BASE> <CONTENIDO>`

La instrucción WRITE escribirá en la Memoria Interna los bytes correspondientes a partir de la dirección base del File:Tag. En caso de que la Memoria Interna no cuente con todas las páginas necesarias para satisfacer la operación, deberá solicitar el contenido faltante al módulo Storage.

READ

Formato: `READ <NOMBRE_FILE>:<TAG> <DIRECCIÓN BASE> <TAMAÑO>`

La instrucción READ leerá de la Memoria Interna los bytes correspondientes a partir de la dirección base del File y Tag pasados por parámetro, y *deberá enviar dicha información al módulo Master*.

En caso de que la Memoria Interna no cuente con todas las páginas necesarias para satisfacer la operación, deberá solicitar el contenido faltante al módulo Storage.

TAG

Formato: `TAG <NOMBRE_FILE_ORIGEN>:<TAG_ORIGEN>`
`<NOMBRE_FILE_DESTINO>:<TAG_DESTINO>`

La instrucción TAG solicitará al módulo Storage la creación un nuevo File:Tag a partir del File y Tag origen pasados por parámetro.

[2] Para diferenciar los archivos nativos de linux de los archivos que va a gestionar nuestro Filesystem, en todo el enunciado vamos a nombrarlos como "archivos" a los primeros y como "Files" a los segundos

COMMIT

Formato: `COMMIT <NOMBRE_FILE>:<TAG>`

La instrucción COMMIT, le indicará al Storage que no se realizarán más cambios sobre el File y Tag pasados por parámetro.

FLUSH

Formato: `FLUSH <NOMBRE_FILE>:<TAG>`

Persistirá todas las modificaciones realizadas en Memoria Interna de un File:Tag en el Storage.

Nota: Esta instrucción también deberá ser ejecutada implícitamente bajo las siguientes situaciones:

- Previo a la ejecución de un COMMIT.
- Previo a realizar el desalojo de la Query del Worker (para todos los File:Tag eventualmente modificados)

DELETE

Formato: `DELETE <NOMBRE_FILE>:<TAG>`

La instrucción DELETE solicitará al módulo Storage la eliminación del File:Tag correspondiente.

END

Formato: END

Esta instrucción da por finalizada la Query y le informa al módulo Master el fin de la misma.

Ejemplo de archivo de Query

```
CREATE MATERIAS:BASE
TRUNCATE MATERIAS:BASE 1024
WRITE MATERIAS:BASE 0 SISTEMAS_OPERATIVOS
FLUSH MATERIAS:BASE
COMMIT MATERIAS:BASE
READ MATERIAS:BASE 0 8
TAG MATERIAS:BASE MATERIAS:V2
DELETE MATERIAS:BASE
WRITE MATERIAS:V2 0 SISTEMAS_OPERATIVOS_2
COMMIT MATERIAS:V2
END
```

Memoria Interna

El componente de la Memoria Interna estará implementado mediante un único `malloc()` que contendrá la información que el Query Interpreter necesite leer y/o escribir a lo largo de las diferentes ejecuciones.

El tamaño de la Memoria Interna vendrá delimitado por archivo de configuración y no se podrá cambiar en tiempo de ejecución.

Se utilizará un esquema de *paginación simple* a demanda con memoria virtual, donde el tamaño de página estará definido por el tamaño de bloque del Storage, este dato se debe obtener al momento de hacer el Handshake con el Storage. De esta manera, se deberá mantener una tabla de páginas por cada File:Tag de los que tenga al menos una página presente.

Para cada referencia lectura o escritura del Query Interpreter a una página de la Memoria Interna, se deberá esperar un tiempo definido por archivo de configuración (RETARDO_MEMORIA).

Algoritmos de Reemplazo de páginas

En caso de que la Memoria Interna esté llena y deba cargar una nueva página, se deberá elegir una víctima para ser reemplazada.

La elección de la víctima se podrá realizar mediante los algoritmos LRU o CLOCK-M definido por archivo de configuración y el mismo no se modificará en tiempo de ejecución. La víctima podrá ser una página correspondiente a cualquier File:Tag que se encuentre presente en la Memoria Interna del Worker. Si la página víctima se encuentra modificada, se deberá previamente impactar el cambio en el módulo Storage, teniendo en cuenta que los números de página están directamente relacionados al número del bloque lógico dentro del File:Tag.

Logs mínimos y obligatorios

Recepción de Query: “## Query <QUERY_ID>: Se recibe la Query. El path de operaciones es: <PATH_QUERY>”

Fetch Instrucción: “## Query <QUERY_ID>: FETCH - Program Counter: <PROGRAM_COUNTER> - <INSTRUCCIÓN>[3]”.

Instrucción realizada: “## Query <QUERY_ID>: - Instrucción realizada: <INSTRUCCIÓN>[4]”

Desalojo de Query: “## Query <QUERY_ID>: Desalojada por pedido del Master”

Lectura/Escritura Memoria: “Query <QUERY_ID>: Acción: <LEER / ESCRIBIR> - Dirección Física: <DIRECCION_FISICA> - Valor: <VALOR LEIDO / ESCRITO>”.

Asignar Marco: “Query <QUERY_ID>: Se asigna el Marco: <NUMERO_MARCO> a la Página: <NUMERO_PAGINA> perteneciente al - File: <FILE> - Tag: <TAG>”.

Liberación de Marco: “Query <QUERY_ID>: Se libera el Marco: <NUMERO_MARCO> perteneciente al - File: <FILE> - Tag: <TAG>”.

Reemplazo Algoritmo: “## Query <QUERY_ID>: Se reemplaza la página <File1:Tag1>/<NUM_PAG1> por la <File2:Tag2><NUM_PAG2>”

[3] Los parámetros no se deben loguear
[4] Los parámetros no se deben loguear

Bloque faltante en Memoria: "`Query <QUERY_ID>: - Memoria Miss - File: <FILE> - Tag: <TAG> - Pagina: <NUMERO_PAGINA>`"

Bloque ingresado en Memoria: "`Query <QUERY_ID>: - Memoria Add - File: <FILE> - Tag: <TAG> - Pagina: <NUMERO_PAGINA> - Marco: <NUMERO_MARCO>`"

Archivo de Configuración

Campo	Tipo	Descripción
IP_MASTER	String	IP del módulo Master
PUERTO_MASTER	Numérico	Puerto del módulo Master
IP_STORAGE	String	IP del módulo Storage
PUERTO_STORAGE	Numérico	Puerto del módulo Storage
TAM_MEMORIA[5]	Numérico	Tamaño expresado en bytes de la Memoria Interna.
RETARDO_MEMORIA	Numérico	Tiempo en milisegundos que deberá esperarse ante cada lectura y/o escritura en la Memoria.
ALGORITMO_REEMPLAZO	String	Algoritmo de reemplazo para las páginas. LRU / CLOCK-M
PATH_QUERIES	String	Path donde se encuentran los archivos de Queries
LOG_LEVEL	String	Nivel de detalle máximo a mostrar. Compatible con log_level_from_string()

Ejemplo de Archivo de Configuración

```
IP_MASTER=127.0.0.1
PUERTO_MASTER=9001
IP_STORAGE=127.0.0.1
PUERTO_STORAGE=9002
TAM_MEMORIA=4096
RETARDO_MEMORIA=1500
ALGORITMO_REEMPLAZO=LRU
PATH_SCRIPTS=/home/utnso/queries
LOG_LEVEL=INFO
```

[5] El tamaño de memoria siempre va a ser un múltiplo de tamaño de página.

Módulo: Storage

Este módulo representará un sistema de archivos (File System) sobre el que los Workers pueden operar de forma concurrente.

Lineamiento e Implementación

Al iniciar leerá su archivo de configuración, de ser necesario se crearán los archivos y estructuras necesarias para inicializar el FS, de lo contrario leerá los archivos ya existentes para poder atender las peticiones de los Workers. Creará un servidor multihilo y quedará a la espera de las conexiones de los Workers.

El módulo Storage al momento de iniciarse deberá poder recibir como mínimo el siguiente parámetro de ejecución:

- Archivo de configuración, el cual se encuentra detallado más abajo.

```
➜ ~ ./bin/storage [archivo_config]
```

Estructura de directorios y archivos

El FS se montará sobre un path (ruta) definido por archivo de configuración que llamaremos raíz.
Este directorio raíz deberá contener 3 archivos nativos:

- Un archivo “superblock.config” que contendrá la información administrativa del FS.
- Un archivo “bitmap.bin” que indicará el estado de los bloques del FS (libre/ocupado).
- Un archivo “blocks_hash_index.config” que contendrá un hash por cada bloque reservado del FS.

A su vez, contendrá 2 directorios nativos:

- Un directorio “physical_blocks”, donde se encontrará el espacio de bloques físicos del volumen del FS
- Un directorio nativo “files” donde se encontrará el espacio lógico del FS.

Archivo superblock.config

Debe ser un archivo compatible con las config de la biblioteca commons y contendrá mínimamente:

- Tamaño del FS en bytes
- Tamaño de Bloque en bytes

Ejemplo:

```
FS_SIZE=4096
BLOCK_SIZE=128
```

Archivo bitmap.bin

El objetivo de este archivo es identificar los bloques físicos reservados en nuestro File System.

Indicará para cada bloque físico su estado (libre/ocupado) utilizando *un bit* por cada bloque (estructura conocida como bitarray). [6][7]

Cualquier implementación que no resguarde el contenido del bitmap como array de bits es motivo de desaprobación del TP.

Archivo blocks_hash_index.config

El objetivo de este archivo es asociar cada bloque físico *ocupado* con un identificador único (hash), generado a partir del contenido del mismo. De esta manera, dos bloques lógicos con el mismo contenido podrán ser asociados al mismo bloque físico al tener el mismo hash.

Un ejemplo de este archivo es el siguiente:

```
4d186321c1a7f0f354b297e8914ab240=block0000
ea5cb8ff0abf6b4ca0080069daaeada0=block0001
67d6c28fac7541d9ce1f46ba4f84e149=block0002
749dfe7c0cd3b291ec96d0bb8924cb46=block0003
5058f1af8388633f609cadb75a75dc9d=block0004
```

Función de Hash

Se utilizará el algoritmo Message-Digest Algorithm 5 (MD5), el cual, en base a un contenido (input), genera un resultado (digest) de 128 bits, representados por una cadena de 32 caracteres hexadecimales.

Para realizar dicho cálculo, se debe utilizar la función crypto_md5() de la biblioteca commons.

Directorio 'physical_blocks'

Este directorio representará cada bloque físico del FS como un archivo de tamaño fijo definido por archivo de superbloque, por ejemplo:

```
/home/utnso/storage/
          ↳ ./physical_blocks/
             ↳ ./block0000.dat
             ↳ ./block0001.dat
             ↳ ./block0002.dat
             ↳ ./block0003.dat
             ↳ ./block0004.dat
             …
             ↳ ./block9997.dat
             ↳ ./block9998.dat
             ↳ ./block9999.dat
```

[6] Deberá ser compatible con el formato de bitarray usado por la biblioteca commons.
[7] Se recomienda investigar el uso de la función mmap() para facilitar su implementación.

Directorio ‘files’

Este directorio representará cada archivo dentro de nuestro FS, que a partir de este momento lo llamaremos **File,** como un directorio nativo, siendo el nombre del directorio el nombre que tendrá el *File* dentro del FS. Teniendo esto en cuenta, no podrán haber múltiples *Files* con el mismo nombre dentro de este FS.

Cada directorio dentro del *File* contendrá un subdirectorio nativo por cada ***Tag*** del mismo dentro del FS, siendo el nombre del directorio el nombre que tendrá el *Tag* dentro del FS. Similarmente a los *Files*, no podrán haber múltiples *Tags* con el mismo nombre para un mismo File.

```
/home/utnso/storage/
           ↳ ./physical_blocks # physical space
           ↳ ./files/ # logical space
              ↳ ./arch1/
                 ↳ ./tag_1_0_0/
                    ↳ ./metadata.config
                    ↳ ./logical_blocks/
                       ↳ ./000000.dat
                       ↳ ./000001.dat
                       ↳ ./000002.dat
                 ↳ ./tag_2_0_0/
                    ↳ ./metadata.config
                    ↳ ./logical_blocks/
                       ↳ ./000000.dat
                       ↳ ./000001.dat
                       ↳ ./000002.dat
```

Dentro de cada uno de estos directorios de tags tendrá un archivo compatible con las config de la biblioteca commons llamado “metadata.config”, el cuál contendrá el tamaño del File:Tag, el estado del File:Tag (el cual podrá ser WORK_IN_PROGRESS o COMMITED) y la lista ordenada de los números de bloques físicos del FS que pertenecen al File:Tag, por ejemplo:

```
TAMAÑO=160
BLOCKS=[17,2,5,10,8]
ESTADO=WORK_IN_PROGRESS
```

Por último, con el objetivo de tener una visión más directa de los bloques lógicos del File:Tag, se deberá tener también un directorio nativo “logical_blocks” cuyo contenido será de un archivo nativo por cada bloque lógico del File:Tag. Cada uno de estos archivos nativos será un hard link[8] al bloque físico que corresponde del FS.

[8] Se recomienda investigar el uso de la función link()

Inicialización

Al momento de inicializar el FS lo primero que se debe verificar es el valor FRESH_START del archivo de configuración. Este valor nos indicará si se debe formatear el volumen (iniciando un FS nuevo) o si se quiere mantener el contenido preexistente.

Al inicializar desde cero el FS (FRESH_START=TRUE), el único archivo nativo que necesitamos tener obligatoriamente es el archivo `superblock.config`. Ese archivo nos indicará los datos necesarios para poder formatear nuestro FS, eliminando previamente los demás archivos que componen nuestro FS en caso de existir.

También se deberá crear un primer File llamado “initial_file” confirmado con un único Tag “BASE”, cuyo contenido será un (1) bloque lógico con el bloque físico nro 0 (cero) asignado, completando el bloque con el caracter 0, por ejemplo: “00000…”. Dicho File/Tag no se podrá borrar.

Estructura interna

Un ejemplo de cómo se vería la estructura completa de directorios y archivos nativos de nuestro TP sería el siguiente:

```
/home/utnso/storage/
            ↳ ./superblock.config
            ↳ ./bitmap.bin
            ↳ ./blocks_hash_index.config
            ↳ ./physical_blocks/
               ↳ ./block0000.dat
               ↳ ./block0001.dat
               ↳ ./block0002.dat
               ↳ ./block0003.dat
               ↳ ./block0004.dat
            ↳ ./files/
               ↳ ./initial_file/
                  ↳ ./BASE/
                     ↳ ./metadata.config
                     ↳ ./logical_blocks/
                        ↳ ./000000.dat # hard link a ‘block0000.dat’
               ↳ ./arch1/
                  ↳ ./tag_1_0_0/
                     ↳ ./metadata.config
                     ↳ ./logical_blocks/
                        ↳ ./000000.dat # hard link a ‘block0003.dat’
                        ↳ ./000001.dat # hard link a ...
                        ↳ ./000002.dat
                  ↳ ./tag_2_0_0/
                     ↳ ./metadata.config
                     ↳ ./logical_blocks/
                        ↳ ./000000.dat
                        ↳ ./000001.dat
                        ↳ ./000002.dat
                  ↳ ./tag_3_0_0/
                     ↳ ./metadata.config
                     ↳ ./logical_blocks/
                        ↳ ./000000.dat
                        ↳ ./000001.dat
```

```
                    ↳ ./000002.dat
                    ↳ ./000003.dat
```

Operaciones

El storage ofrecerá una serie de operaciones, susceptibles de ser solicitadas por los Workers. Las mismas no deberán ser confundidas con las instrucciones de una Query a ser interpretadas por los Workers. Las instrucciones ejecutadas en los Workers implicarán la ejecución de una o más operaciones del Storage.

Creación de File

Esta operación creará un nuevo File dentro del FS. Para ello recibirá el nombre del File y un Tag inicial para crearlo.

Deberá crear el archivo de metadata en estado WORK_IN_PROGRESS y no asignarle ningún bloque.

Truncado de archivo

Esta operación se encargará de modificar el tamaño del File:Tag especificados agrandando o achicando el tamaño del mismo para reflejar el nuevo tamaño deseado (actualizando la metadata necesaria).

Al incrementar el tamaño del File, se le asignarán tantos bloques lógicos (hard links) como sea necesario. Inicialmente, todos ellos deberán apuntar el bloque físico nro 0.

Al reducir el tamaño del File, se deberán desasignar tantos bloques lógicos como sea necesario (empezando por el final del archivo). Si el bloque físico al que apunta el bloque lógico eliminado no es referenciado por ningún otro File:Tag, deberá ser marcado como libre en el bitmap[9].

Tag de File

Esta operación creará una copia completa del directorio nativo correspondiente al Tag de origen en un nuevo directorio correspondiente al Tag destino y modificará en el archivo de metadata del Tag destino para que el mismo se encuentre en estado WORK_IN_PROGRESS.

Commit de un Tag

Confirmará un File:Tag pasado por parámetro. En caso de que un Tag ya se encuentre confirmado, esta operación no realizará nada. Para esto se deberá actualizar el archivo metadata del Tag pasando su estado a “COMMITED”.

Se deberá, por cada bloque lógico, buscar si existe algún bloque físico que tenga el mismo contenido (utilizando el hash y archivo blocks_hash_index.config). En caso de encontrar uno, se deberá liberar el bloque físico actual y reapuntar el bloque lógico al bloque físico pre-existente. En caso contrario, simplemente se agregará el hash del nuevo contenido al archivo blocks_hash_index.config.

[9] Para la implementación de esta parte se recomienda consultar la documentación de la syscall stat(2).

Escritura de Bloque

Esta operación recibirá el contenido de un bloque lógico de un File:Tag y guardará los cambios en el bloque físico correspondiente, siempre y cuando el File:Tag no se encuentre en estado COMMITED y el bloque lógico se encuentre asignado.

Si el bloque lógico a escribir fuera el único referenciando a su bloque físico asignado, se escribirá dicho bloque físico directamente. En caso contrario, se deberá buscar un nuevo bloque físico, escribir en el mismo y asignarlo al bloque lógico en cuestión.

Lectura de Bloque

Dado un File:Tag y número de bloque lógico, la operación de lectura obtendrá y devolverá el contenido del mismo.

Eliminar un tag

Esta operación eliminará el directorio correspondiente al File:Tag indicado. Al realizar esta operación, si el bloque físico al que apunta cada bloque lógico eliminado no es referenciado por ningún otro File:Tag, deberá ser marcado como libre en el bitmap[10].

Errores en las operaciones

A fin de unificar los posibles errores que se puedan dar a lo largo de la ejecución del módulo storage, vamos a detallar los mismos a continuación. Es posible que algunos errores no apliquen para ciertas operaciones.

Cualquier error que se de en la ejecución de una query, será motivo de finalización de la misma y se informará al Worker correspondiente el motivo del error.

File / Tag inexistente

Una operación quiere realizar una acción sobre un File:Tag que no existe (salvo la operación de CREATE que crea un nuevo File:Tag).

Una operación quiere realizar una acción sobre un tag que no existe, salvo la operación de TAG que crea un nuevo Tag.

File / Tag preexistente

La operación CREATE o la operación TAG intentan crear un un File:Tag que ya existen.

Espacio Insuficiente

Al intentar asignar un nuevo bloque físico, no se encuentra ninguno disponible.

Escritura no permitida

Una query intenta escribir o truncar un File:Tag que se encuentre en estado COMMITED.

[10] Para la implementación de esta parte se recomienda consultar la documentación de la syscall stat(2).

Lectura o escritura fuera de limite

Una query intenta leer o escribir por fuera del tamaño del File:Tag.

Retardos en las operaciones

Para todas las operaciones se deberá esperar un tiempo determinado por el valor del archivo de configuración RETARDO_OPERACION.

Adicionalmente, por cada bloque que se quiera leer y/o escribir se deberá esperar el tiempo determinado por el valor del archivo de configuración RETARDO_ACCESO_BLOQUE.

Logs mínimos y obligatorios

Conexión de Worker: "##Se conecta el Worker <WORKER_ID> - Cantidad de Workers: <CANTIDAD>"

Desconexión de Worker: "##Se desconecta el Worker <WORKER_ID> - Cantidad de Workers: <CANTIDAD>"

File Creado: "##<QUERY_ID> - File Creado <NOMBRE_FILE>:<TAG>"

File Truncado: "##<QUERY_ID> - File Truncado <NOMBRE_FILE>:<TAG> - Tamaño: <TAMAÑO>"

Creación de Tag: "##<QUERY_ID> - Tag creado <NOMBRE_FILE>:<TAG>"

Commit de Tag: "##<QUERY_ID> - Commit de File:Tag <NOMBRE_FILE>:<TAG>"

Eliminación de Tag: "##<QUERY_ID> - Tag Eliminado <NOMBRE_FILE>:<TAG>"

Bloque Lógico Leído: "##<QUERY_ID> - Bloque Lógico Leído <NOMBRE_FILE>:<TAG> - Número de Bloque: <BLOQUE>"

Bloque Lógico Escrito: "##<QUERY_ID> - Bloque Lógico Escrito <NOMBRE_FILE>:<TAG> - Número de Bloque: <BLOQUE>"

Bloque Físico Reservado: "##<QUERY_ID> - Bloque Físico Reservado - Número de Bloque: <BLOQUE>"

Bloque Físico Liberado: "##<QUERY_ID> - Bloque Físico Liberado - Número de Bloque: <BLOQUE>"

Hard Link Agregado: "##<QUERY_ID> - <NOMBRE_FILE>:<TAG> Se agregó el hard link del bloque lógico <BLOQUE_LOGICO> al bloque físico <BLOQUE_FISICO>"

Hard Link Eliminado: "##<QUERY_ID> - <NOMBRE_FILE>:<TAG> Se eliminó el hard link del bloque lógico <BLOQUE_LOGICO> al bloque físico <BLOQUE_FISICO>"

Deduplicación de Bloque: "##<QUERY_ID> - <NOMBRE_FILE>:<TAG> Bloque Lógico <BLOQUE> se reasigna de <BLOQUE_FISICO_ACTUAL> a <BLOQUE_FISICO_CONFIRMADO>"

Archivo de Configuración

Campo	Tipo	Descripción
PUERTO_ESCUCHA	Numérico	Puerto al cual se deberán conectar los procesos worker
FRESH_START	Booleano	Declara si se debe realizar una limpieza e inicialización de cero del FS.
PUNTO_MONTAJE	String	PATH donde se encontrará el punto de montaje de nuestra Storage con todas sus estructuras adentro
RETARDO_OPERACION	Numérico	Tiempo en milisegundos que se deberá esperar ante cada petición.
RETARDO_ACCESO_BLOQUE	Numérico	Tiempo en milisegundos que se deberá esperar luego de cada acceso a bloque.
LOG_LEVEL	String	Nivel de detalle máximo a mostrar. Compatible con log_level_from_string()

Ejemplo de Archivo de Configuración

```
PUERTO_ESCUCHA=9002
FRESH_START=TRUE
PUNTO_MONTAJE=/home/utnso/storage
RETARDO_OPERACION=8000
RETARDO_ACCESO_BLOQUE=4000
LOG_LEVEL=INFO
```

Descripción de las entregas

Debido al orden en que se enseñan los temas de la materia en clase, los checkpoints están diseñados para que se pueda realizar el trabajo práctico de manera iterativa incremental tomando en cuenta los conceptos aprendidos hasta el momento de cada checkpoint.

Check de Control Obligatorio 1: Conexión inicial

Fecha: 13/09/2025

Objetivos:

- Familiarizarse con Linux y su consola, el entorno de desarrollo y el repositorio.
- Aprender a utilizar las Commons, principalmente las funciones para listas, archivos de configuración y logs.
- Definir el Protocolo de Comunicación.
- Todos los módulos están creados y son capaces de establecer conexiones entre sí y compartir información. Por ejemplo:
    - El path de un archivo de Query se envía desde el Query Control es capaz de ser recibido por el Master.
    - El Master es capaz de enviarle el path de la query al Worker.
    - El Worker es capaz de solicitarle a Storage el tamaño de bloque.

Check de Control Obligatorio 2: Ejecución Básica

Fecha: 11/10/2025

Objetivos:

- **Módulo Query Control: Completo**

- **Módulo Master:**
    - Permitir la conexión de múltiples Workers y múltiples Query Control
    - Planificación bajo algoritmo FIFO

- **Módulo Worker:**
    - Lectura de archivos de Queries
    - Es capaz de interpretar y ejecutar todas las instrucciones
    - Creación de tablas de páginas

- **Módulo Storage:**
    - Creación de estructura de archivos y directorios bajo FRESH_START
    - Todas las operaciones se responden con un valor fijo definido por el grupo[11]

Carga de trabajo estimada:

- **Módulo Query Control:** 10%
- **Módulo Master:** 35%
- **Módulo Worker:** 35%
- **Módulo Storage**: 20%

[11] Se recomienda investigar el concepto de Mock Object

Check de Control Obligatorio 3: Comenzando a Persistir

Fecha: 08/11/2025

Objetivos:

- **Módulo Master:**
  - Completo
- **Módulo Worker:**
  - Completo
- **Módulo Storage:**
  - Implementada la creación de File:Tag
  - Las operaciones de lectura y escritura devuelven valores reales y son persistidas en el FS.
  - Manejo de espacio libre sin deduplicación (Chequeo de MD5)

Carga de trabajo estimada:

- **Módulo Master:** 20%
- **Módulo Worker:** 40%
- **Módulo Storage:** 40%

Entregas Finales

Fechas: 29/11/2025, 06/12/2025, 20/12/2025

Objetivos:

- Finalizar el desarrollo de todos los procesos.
- Probar de manera intensiva el TP en un entorno distribuido.
- Todos los componentes del TP ejecutan los requerimientos de forma integral.