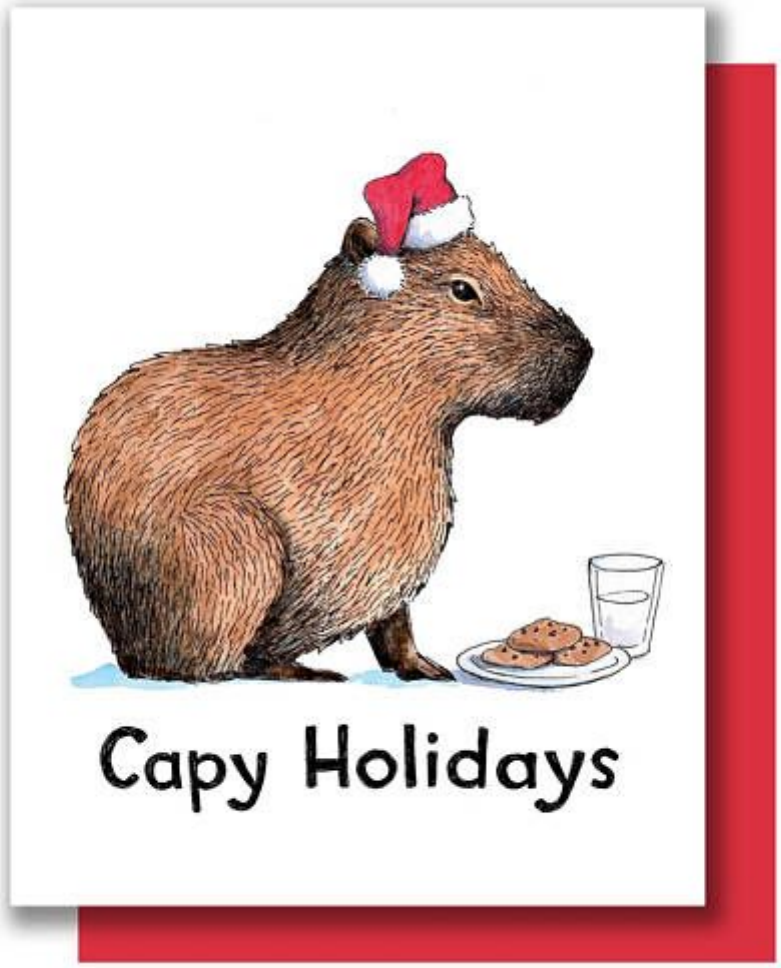
# *CarpinchOS*

*En la pelopincho ranchando en el quincho*




Cátedra de Sistemas Operativos

Trabajo práctico Cuatrimestral

-2C2021 -
Versión 1.2

# Índice

# Historial de Cambios

*v1.0 (06/09/2021) Primera Publicación*

*v1.1 (25/09/2021) Errata v1.1*

- *Se ajustó la redacción sobre el polimorfismo entre Kernel y Memoria.*
- *Se aclaró en planificación de mediano plazo que debe pasar con las páginas de los procesos suspendidos.*
- *Se agrego el tiempo de corrida del algoritmo de detección de deadlock*
- *Se eliminó el retardo de CPU en el Kernel*
- *Se agregó aclaración sobre los semáforos en la sección de bloqueos.*
- *Se agregó tamaño de página en la configuración del proceso memoria*
- *Se aclaró MARCOS_MAXIMO, MARCOS_POR_PROCESO.*
- *Se aclaró el criterio de selección de archivo de SWAP*
- *Se aclaró que el tamaño de swap corresponde a cada archivo de swap.*
- *Se aclaró cómo se debe realizar el cálculo de la próxima ráfaga*

*v1.2 (19/10/2021) Errata v1.2*

- *Se ajustó la descripción de los procesos ya que los mismos son representados por las instancias de la matelib a las cuales llamamos Carpinchos.*
- *Se especificaron los errores del proceso de memoria.*
- *Se aclaró el criterio a utilizar al encontrar 2 archivos de swap con el mismo espacio libre.*
- *Se aclaró que el tamaño máximo de un carpincho en memoria viene limitado por la cantidad de páginas que se puedan almacenar en SWAP.*
- *Se aclaró que el SWAP en el caso de asignación global, no es una extensión de la memoria.*

# 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: 06 de Septiembre
- Fecha de primera entrega: 27 de Noviembre
- Fecha de segunda entrega: 11 de Diciembre
- Fecha de tercera entrega: 18 de Diciembre
- 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 y poder ser desplegado para alcanzar su funcionamiento en no más de 15 minutos.

[1] Durante este cuatrimestre será de **modalidad virtual, utilizando instancias hosteadas en la nube de la VM server,** a menos que la facultad defina lo contrario.

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 y en caso de desaprobarse, la misma no puede ser recuperada.

## Deployment y Testing del Trabajo Práctico

Al tratarse de una plataforma distribuida, los procesos involucrados podrán ser ejecutados en diversas máquinas virtuales en distintas computadoras físicas o virtuales. 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 gestionar 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 y permitir que los mismos sean realizables dentro del marco de un cuatrimestre.

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 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, donde los grupos tendrán que planificar procesos externos no implementados por los grupos, que ejecuten peticiones al sistema de recursos. El sistema deberá, mediante esta interacción, habilitar recursos de memoria (bajo un esquema de paginación pura), recursos de entrada-salida y semáforos.

Para el desarrollo del TP se propone la utilización de la metodología Iterativo-Incremental, donde se solicitarán en una primera instancia la implementación de ciertos módulos para luego poder realizar una integración total con los restantes.

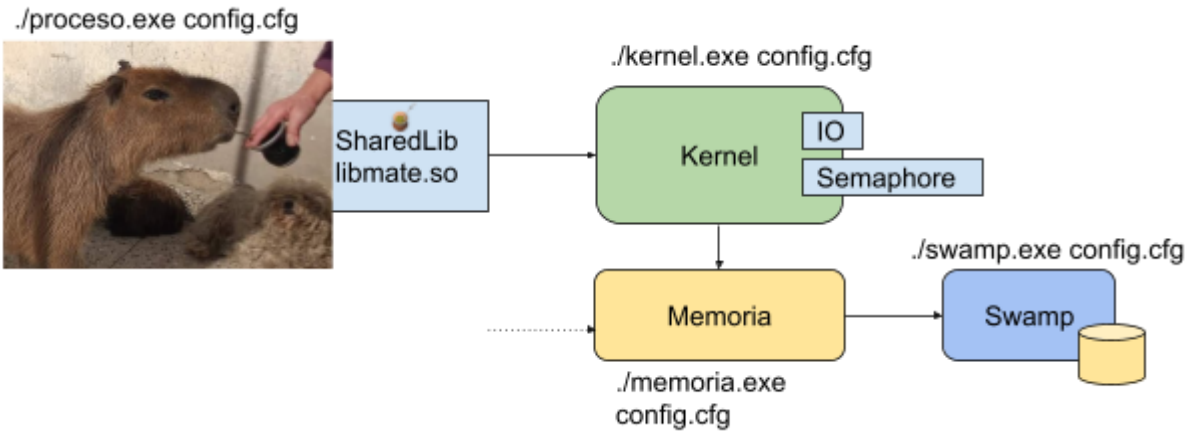
Recomendamos seguir el lineamiento de los distintos puntos de control que se detallan al final de este documento. Estos puntos están planificados y estructurados para que sean desarrollados a medida y en paralelo a los contenidos que se ven en la parte teórica de la materia. Cabe aclarar que esto es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

## Arquitectura del sistema

Como se mencionó anteriormente, el entregable del TP va a recibir peticiones de procesos que los grupos no hayan implementado. Esto se va a realizar mediante el **uso de una Biblioteca Compartida**, que sí será parte del entregable.

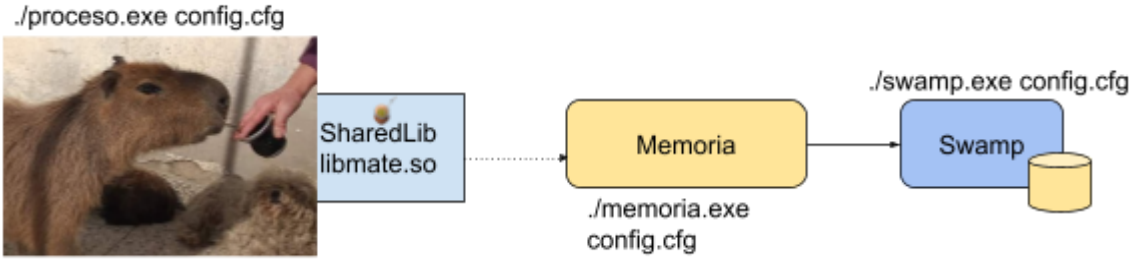
Estas instancias, de ahora en adelante denominados **Carpinchos** (para hacer más amena la dinámica y evitar la dualidad en la palabra original), agregarán a su compilación la referencia a la biblioteca ya compilada en el *LD_LIBRARY_PATH* antes de ejecutar para poder referenciar el código de cada grupo.

La biblioteca **matelib (🧉)** será el punto de entrada a cualquier otro módulo del sistema, comunicándose con el módulo Kernel mediante sockets TCP/IP. Para tener una idea gráfica:




El **Kernel** se comunicará con el módulo de **Memoria**, permitiendo este último interactuar con el módulo de **Memoria Swap**. Ante las funcionalidades de memoria, el Kernel solo hará un pasamanos de la configuración, actuando como un intermediario de la memoria.

La interfaz esperada de la Biblioteca ante el Kernel debe ser **polimórfica** contra la memoria. **De esta forma, en ausencia del Kernel, debe poder configurarse la IP y Puerto de la memoria; anulando las funcionalidades de planificación, pero permitiendo las de memoria:**



## Distribución Recomendada

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.

- **Kernel: 1,5**
- **Lib: 0,5**
- **Memoria: 2**
- **Swap: 1**

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*.

# Módulo: MateLib 🧉

Como mencionamos anteriormente, los distintos Carpinchos harán uso de esta biblioteca, que será desarrollada por cada grupo.

Cada uno de estos Carpinchos compilarán con una **referencia estandarizada de los prototipos de las funciones de la lib**, que pueden encontrar definidos en el repositorio como matelib.h: https://github.com/sisoputnfrba/matelib.git . Al momento de correr, referenciarán el binario compilado por los grupos, con la variable de entorno *LD_LIBRARY_PATH* (Ver ejemplo).

En dicho repositorio encontrarán, aparte de la referencia de los prototipos, una explicación de su uso, ejemplos de implementación de una lib y ejemplos de cómo los Carpinchos harán uso de la lib compilada.

La comunicación de este módulo con los demás será mediante sockets TCP/IP, sin hacer distinciones de cual otro módulo del sistema esté referenciando.

Recomendamos fuertemente la lectura del **README.md** del repositorio, dado que se amplía un poco los lineamientos técnicos y requerimientos no funcionales de la misma.

Finalmente, el **archivo de configuración a utilizar por la lib será de especificación libre**, cada grupo deberá especificar el formato y la información necesaria para la operación de la biblioteca.

# Módulo: Kernel

Este módulo será el encargado de gestionar y planificar todas las solicitudes que los carpinchos harán mediante la lib para solicitar recursos de nuestro sistema.

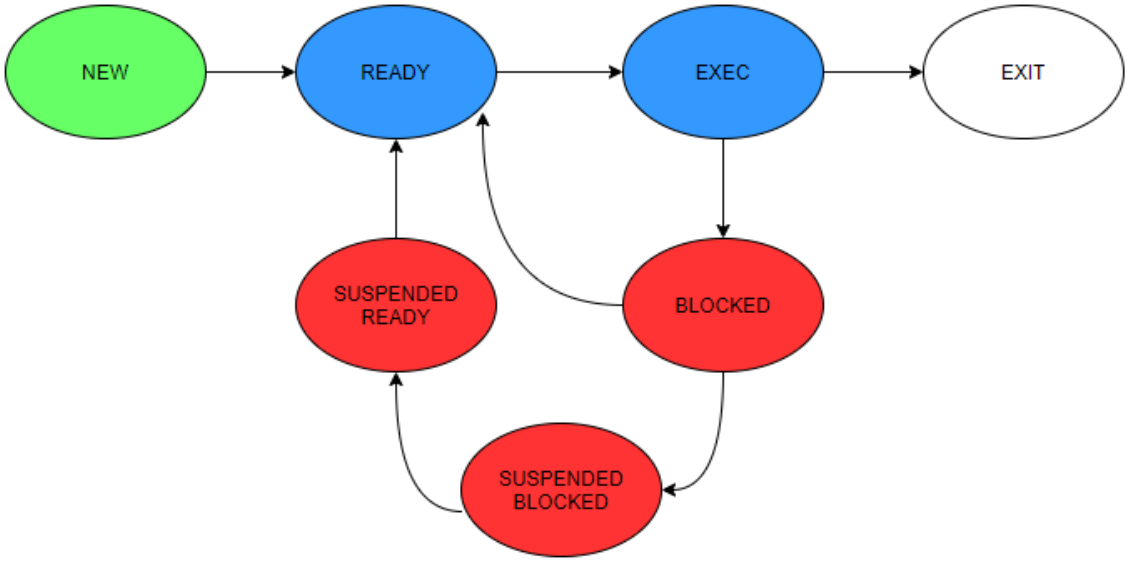
## Lineamiento e Implementación

Este módulo deberá atender las peticiones de manera concurrente y deberá implementarse utilizando múltiples hilos de ejecución.

Todas las solicitudes de la biblioteca deberán pasar por este módulo primero para que los distintos planificadores las planifiquen cuando corresponda.

Este módulo **tendrá tres planificadores distintos**. Uno de largo plazo, uno de corto y uno de mediano plazo. Cada planificador correrá de manera independiente a los otros con su propio algoritmo.

## Diagrama de estados




## Inicialización - Largo plazo

Cuando la lib haga su inicialización con *mate_init*, deberá conectarse a este módulo para que genere las estructuras administrativas que se necesiten.

Todos los carpinchos al conectarse se iniciarán en NEW, y una vez que todas sus estructuras estén creadas exitosamente se pasarán a READY **únicamente si el grado de multiprogramación del sistema**

**lo permite**, grado que será ingresado por archivo de configuración. Este planificador usará el algoritmo **FIFO.**

## Ejecución - Corto plazo

La transición de READY a EXEC se hará mediante el uso de este planificador, el cual funcionará con los algoritmos **SJF sin desalojo o HRRN,** según lo ingresado por archivo de configuración**.** Este módulo es multi CPU, por lo que la cantidad de carpinchos que puedan estar en EXEC al mismo tiempo estará medida por el **grado de multiprocesamiento** ingresado por archivo de configuración. Una vez el carpincho esté en EXEC, debido a la naturaleza sin desalojo de los algoritmos propuestos, este no saldrá hasta que no termine o se bloquee.

Para el cálculo de las ráfagas estimadas, se deberá utilizar la fórmula de la media exponencial. El alfa y la estimación inicial para el cálculo de la primera ráfaga entrarán por archivo de configuración, y la ráfaga real anterior será cero. Para el cálculo de posteriores ráfagas, se deberá calcular cuánto tiempo en milisegundos estuvo el carpincho en exec tomando la diferencia entre el *timestamp*[2] al ingresar y al salir de exec.

A fines didácticos del TP, este multiprocesamiento deberá ser representado mediante un hilo por cada CPU[3] que tenga el sistema, cantidad que será ingresada por archivo de configuración. Estos hilos CPU serán los que ejecuten las solicitudes a recursos del sistema por parte de los carpinchos que se conecten mediante la lib. **Cualquier implementación que no cumpla con este requerimiento será motivo de desaprobación.**

Por un lado, existen dos maneras que un carpincho se bloquee, ambas mediante la lib:

- Que solicite un semáforo que tenga valor menor a 1.
- Que solicite una entrada/salida a un recurso definido.

## Suspensión - Mediano plazo

Por último, este planificador se encargará de gestionar las transiciones de **suspendido-bloqueado** y **suspendido-listo**.

Para que un carpincho entre en suspensión se deben cumplir simultáneamente las siguientes condiciones:

- Que haya carpinchos en BLOCKED
- Que no haya carpinchos en READY
- Que haya carpinchos en NEW

[2] investigar sys/time.h o temporal.h de las commons
[3] No es una CPU en el sentido estricto de la palabra, si no que simula las interacciones del carpincho con las funciones del kernel.

Esto significa que todo el grado de multiprogramación esté copado por carpinchos que no están haciendo uso de las CPUs. Cuando esto suceda, el planificador deberá tomar **el último elemento en haber ingresado a la cola de bloqueado y pasarlo a suspendido,** para dar entrada a nuevos carpinchos esperando el grado de multiprogramación en NEW y no generar overhead de CPU en EXEC**.** Una vez que haya terminado el bloqueo de este carpincho, este pasará a **suspendido-listo**, y volverá a READY siguiendo el algoritmo FIFO **cuando el grado de multiprogramación lo permita**, teniendo mayor precedencia los carpinchos esperando en suspendido que los carpinchos esperando en NEW.

Cuando un carpincho sea suspendido, todas sus páginas que se encuentren en memoria principal deben ser bajadas a swap. **En el caso de la asignación fija, los marcos reservados serán liberados para dar entrada a nuevos carpinchos.**

## Bloqueos

Dentro del sistema los carpinchos pueden ser desalojados de EXEC **únicamente** para bloquearse o finalizar. Existirán dos situaciones de bloqueos posibles, ambas mediante operaciones de la lib:

- **Pedir un semáforo**: Si un carpincho hace, mediante uso de la lib, un *mate_sem_wait* a un semáforo con valor menor a 1, este deberá pasar a bloqueado hasta que el semáforo se libere con *mate_sem_post*. Los carpinchos en espera se irán pasando a READY en el orden en el que fueron bloqueados uno a la vez. Es decir, por cada post se desbloqueará un único carpincho en BLOCKED, respetando el orden de la cola de espera.
  Los semáforos que pertenecen a la implementación del TP se encuentran definidos de forma global dentro del kernel, por lo que serán compartidos por todos los carpinchos indistintamente de donde sea que estén ejecutando. No deberán confundirse con los semáforos KLT tradicionales definidos dentro de las bibliotecas pthread y semaphore, ya que los de la biblioteca matelib corresponden a la implementación de semáforos a nivel de ULTs.
  Al momento de ejecutar un *mate_sem_init()*, si el semáforo ya se encuentra inicializado, el valor del mismo **no debe modificarse**.
- **Pedir I/O**: Dentro del sistema existen dispositivos de entrada-salida, cada uno con su propia duración. Un carpincho puede pedir realizar una entrada salida a un determinado dispositivo con *mate_call_io*, generando que se bloquee por el tiempo que dure esa ráfaga I/O. **Los dispositivos no pueden ser usados por dos carpinchos al mismo tiempo**, los carpinchos bloqueados pueden estar a la espera de que se libere un dispositivo si este está en uso.

## Deadlocks

Dentro de nuestro sistema se puede dar el caso que al dos o más carpinchos solicitar recursos (más específicamente, semáforos) se produzca un **interbloqueo** entre ellos. Es por eso que se deberá tomar una política de **detección y recuperación**, donde se terminarán siempre uno a uno los carpinchos iniciando por el de **mayor ID** hasta que se haya solucionado el interbloqueo.

Este algoritmo correrá cada lapso de tiempo definido por archivo de configuración.

## Archivo de Configuración

| Campo | Tipo | Descripción |
|---|---|---|
| IP_MEMORIA | [String] | IP a la cual se deberá conectar con la memoria |
| PUERTO_MEMORIA | [Numérico] | Puerto al cual se deberá conectar con la memoria |
| PUERTO_ESCUCHA | [Numérico] | Puerto en el cual se escucharán las conexiones de los carpinchos |
| ALGORITMO_PLANIFICACION | [String] | SFJ/HRRN |
| ESTIMACION_INICIAL | [Numérico] | Estimación inicial para el cálculo de la primera ráfaga, expresado en milisegundos |
| ALFA | [Numérico] | Alfa para el cálculo de la siguiente ráfaga |
| DISPOSITIVOS_IO | [List] | Nombres de los dispositivos |
| DURACIONES_IO | [List] | Duraciones de las ráfagas de los dispositivos IO en ms |
| GRADO_MULTIPROGRAMACION | [Numérico] | Grado de multiprogramación del módulo |
| GRADO_MULTIPROCESAMIENTO | [Numérico] | Grado de multiprocesamiento del módulo |
| TIEMPO_DEADLOCK | [Numérico] | Tiempo expresado en milisegundos de cada cuanto debe correr el algoritmo de detección de deadlock |

### Ejemplo de Archivo de Configuración

```
IP_MEMORIA=127.0.0.1
PUERTO_MEMORIA=5001
PUERTO_ESCUCHA=6005
ALGORITMO_PLANIFICACION=SJF
DISPOSITIVOS_IO=[laguna, hierbitas]
DURACIONES_IO=[25000, 1000]
GRADO_MULTIPROGRAMACION=20
```

```
GRADO_MULTIPROCESAMIENTO=8
TIEMPO_DEADLOCK=1000
```

# Módulo: Memoria

Este módulo será el encargado de administrar la memoria de nuestro sistema y la que se le otorgará a los diferentes carpinchos que nos invoquen por medio de la biblioteca.

En niveles generales, el funcionamiento del módulo se basará en el de una MMU que trabaja con paginación bajo demanda, una TLB y swap (con este último comunicándose por sockets).

## Lineamiento e Implementación

Este módulo deberá atender las peticiones de manera concurrente y deberá implementarse utilizando múltiples hilos.

## Manejo de TLB

Este componente del módulo funcionará como una TLB real, siendo una "caché" de las tablas de páginas. La misma deberá contener la información del número de marco (parte de la dirección física) en el cual se aloja la página buscada.

Ante un acierto de la TLB (**TLB Hit**) simplemente se procederá a devolver el número de marco, con el cual se podrá obtener la información solicitada en memoria.

Ante un error de la TLB (**TLB Miss**) se deberá proceder a consultar las tablas de páginas para poder obtener la dirección física desde la memoria, y actualizar la información de la TLB mediante el algoritmo definido por archivo de configuración.

## Asignación de Memoria

La asignación de memoria de este trabajo práctico utilizará el método de **paginación pura**. Dentro de estas páginas se almacenará la memoria que los carpinchos irán pidiendo a lo largo de su ciclo de vida mediante la función `mate_memalloc()` mencionada en la lib.

Como los carpinchos (carpinchos) pueden solicitar memoria varias veces y a su vez es una cantidad variable, nos encontramos con la necesidad de manejar estructuras administrativas internas para poder distinguir los distintos allocs de cada carpincho. Estas estructuras van a formar parte de la **memoria principal** de nuestro módulo (representada por un malloc del tamaño de la memoria, obtenido por archivo de configuración) y deberán ser implementadas precisamente como se detalla a continuación.

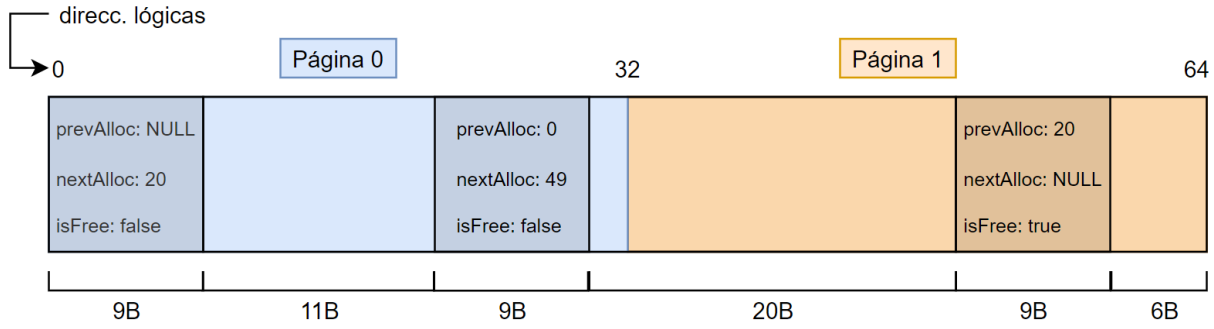
Programáticamente, la metadata de cada una de las secciones alocadas deberá ser del siguiente tipo:

```
typedef struct HeapMetadata {
      uint32_t prevAlloc,
```

```
        uint32_t nextAlloc,
        uint8_t isFree,
}
```

Donde el `prevAlloc` hace referencia a la dirección lógica del alloc anterior, `nextAlloc` es la dirección lógica del alloc siguiente y `isFree` nos dice si ese alloc está libre.

Por otro lado, sobre a las estructuras administrativas externas a la memoria principal, como ya fue mencionado, vamos a tener un esquema de paginación, donde la memoria va a estar dividida lógicamente en páginas de igual tamaño. Por lo tanto, cada carpincho va a tener su propia tabla de páginas, donde vamos a tener las referencias a los marcos donde se encuentra la información de cada página.




Habiendo definido las estructuras, vamos a repasar a grandes rasgos los pasos a seguir cuando se reciba cada operación:

### memalloc

Consideremos que un **carpincho p** quiere reservar el espacio size. Lo primero que debemos hacer es buscar si ya hay algún alloc libre en el espacio de direcciones de p donde quepa el size solicitado. El algoritmo a utilizar para esta búsqueda será **First Fit**.
En caso de que se encuentre, se deberá asignar a p, considerando que si el espacio es más grande que el necesario, se deberá dividir en 2 allocs (el nuevo, ocupado, y uno libre con el resto).

En caso de que no se encuentre, se deberá generar un nuevo alloc al final del espacio de direcciones, teniendo en cuenta que si no cabe en las páginas ya reservadas se deberá solicitar más.

Recordar que el método debe **retornar la dirección lógica** al inicio de la reserva de espacio; y en caso de que no haya más espacio (tanto memoria como en swap), se **denegará** la operación haciendo que la biblioteca devuelva la dirección **NULL**.

### memfree

Cuando queramos liberar un alloc, primero debemos marcar el flag de `isFree` en true. Luego debemos usar el `prevAlloc` y el `nextAlloc` para revisar si hay allocs a derecha o izquierda que ya estén libres, en caso de que los haya debemos consolidarlos. También hay que chequear si hay páginas que quedaron totalmente libres y liberarlas.

En caso de que se quiera liberar una posición de memoria errónea, se deberá devolver el error MATE_FREE_FAULT definido en el enum mate_errors de la matelib correspondiente con el valor -5.

### memread

Se usa para obtener información de la memoria. Con la dirección lógica que nos proveen y la tabla de páginas hacemos la traducción a direcciones físicas y retornamos el contenido.
En caso de que se quiera leer una posición de memoria errónea, se deberá devolver el error MATE_READ_FAULT definido en el enum mate_errors de la matelib correspondiente con el valor -6.

### memwrite

Se usa para escribir en la memoria. Con la dirección lógica que nos proveen y la tabla de páginas hacemos la traducción a direcciones físicas y guardamos el contenido en el alloc correspondiente.
En caso de que se quiera escribir en una posición de memoria errónea, se deberá devolver el error MATE_WRITE_FAULT definido en el enum mate_errors de la matelib correspondiente con el valor -7.

Es menester aclarar que es posible escribir los datos en **cualquier parte de la memoria**, por lo tanto **la información (incluido el HeapMetadata) puede quedar dividida en múltiples páginas**, dado que el tamaño máximo no está limitado por el tamaño de las páginas. Este comportamiento es esperado y es responsabilidad del grupo manejarlo.

Finalmente, como un detalle, **el valor a retornar ante una reserva de memoria es la primera posición del bloque de datos reservado disponible (no la posición del HeapMetadata)**.

## Memoria Virtual

Dado que trabajamos con direcciones lógicas, un carpincho o hilo desconoce que comparte memoria con otros. Por lo tanto, cree que tiene **todo el espacio de direccionamiento** para él. Por obvias razones, cuando estén varios carpinchos compitiendo por la memoria, alguno llegará al punto donde **puede direccionar más memoria, pero no existen más espacios disponibles**.

Para solucionar este inconveniente, este módulo del trabajo práctico contará con **memoria virtual**, que permitirá a cada carpincho acceder a un espacio de direccionamiento mayor al provisto por la memoria principal.

El módulo debe utilizar tanto un **algoritmo de reemplazo de asignación dinámica con reemplazo global, como de asignación fija con reemplazo local**, elegidos por el archivo de configuración.

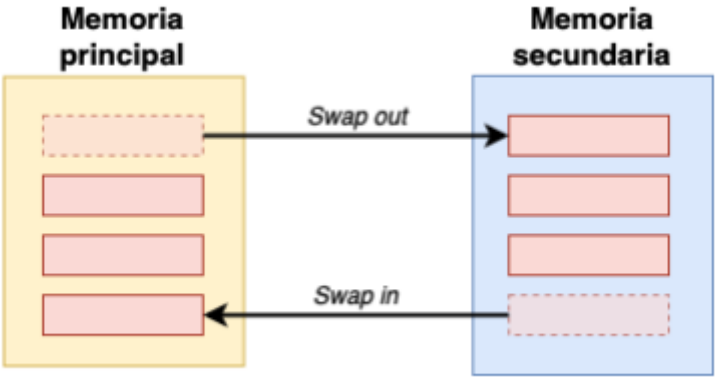
En ambos algoritmos de reemplazo, los frames que se encuentren en memoria serán una copia de las páginas almacenadas en SWAP, por lo tanto, el tamaño máximo del proceso estará definido por lo que se pueda almacenar en SWAP.

Así mismo, los algoritmos a utilizar serán definidos de forma análoga, eligiendo según config entre **Clock-Modificado y LRU**.

Para la asignación de marcos se deberán recorrer los marcos disponibles siempre de **menor a mayor**, en función del número de marco.

Análogamente, eso implica que cada vez que se realice una operación sobre una página, se debe validar si la misma está cargada en la memoria para poder ser utilizada. Si no se encuentra en memoria, se ejecuta una excepción conocida como ***page fault***, es decir, el faltante de una página.

En ese caso se debe ir a buscar la página faltante al área de swap, y se la cargará en la memoria principal, ejecutando el algoritmo de reemplazo de páginas en caso de ser necesario.




Los detalles de la implementación del almacenado de este swap se encuentran definidos en la sección del módulo correspondiente.

## Métricas & Señales

El módulo deberá poder reaccionar ante la llegada de las señales `SIGINT`, `SIGUSR1` y `SIGUSR2`, lo que nos permitirá realizar acciones en consecuencia.

**`SIGINT`**: Esta señal es la que recibe el proceso cuando se le envía Ctrl + C en la terminal que está ejecutando. Al momento de recibir esta señal el proceso deberá imprimir las métricas que se detallan a continuación y luego cerrarse normalmente.

- Cantidad de TLB Hit totales
- Cantidad de TLB Hit por carpincho indicando carpincho y cantidad.
- Cantidad de TLB Miss totales.
- Cantidad de TLB Miss por carpincho indicando carpincho y cantidad.

**`SIGUSR1`**: Esta señal será la encargada de generar un dump del contenido de la TLB en un archivo de texto, el cual deberá encontrarse dentro del path indicado por archivo de configuración y deberá llamarse Dump_<Timestamp>.tlb

El archivo de dump, además de la fecha y hora de cuando se realizó, deberá tener por cada entrada la siguiente información:

- Número de entrada
- Estado
- PID
- Número de página
- Marco asignado en la memoria.

*Ejemplo:*

```
------------------------------------------------------------------------
Dump: 09/07/21 10:11:12
Entrada:0   Estado:Ocupado    Carpincho: 1      Pagina: 0   Marco: 2
Entrada:1   Estado:Ocupado    Carpincho: 2      Pagina: 3   Marco: 9
Entrada:2   Estado:Ocupado    Carpincho: 3      Pagina: 3   Marco: 26
Entrada:3   Estado:Libre      Carpincho: -      Pagina: -   Marco: -
------------------------------------------------------------------------
```

**SIGUSR2**: Ante la llegada de esta señal el proceso memoria, deberá limpiar todas las entradas de la TLB.

### Eventos de log obligatorio

Para poder conocer lo que ocurre dentro del módulo y poder llevar adelante un seguimiento claro, listamos a continuación los eventos mínimos de log que esperamos que tenga el módulo, esto no quita que los alumnos puedan extender el log agregando más información si lo ven necesario.

- Conexión y desconexión de un cliente - Identificando qué carpincho se conectó.
- TLB Hit - Indicando PID, Número de página y marco.
- TLB Miss - Indicando PID, Número de página.
- Reemplazo de una entrada indicando víctima y la nueva entrada, ambos con PID, Número de página y Marco en ambos casos.

## Archivo de Configuración

| Campo | Tipo | Descripción |
|---|---|---|
| PUERTO_ESCUCHA | [Numérico] | Puerto donde escucha las conexiones la memoria |
| TAMANIO | [Numérico] | Tamaño expresado en bytes del tamaño de la memoria. |

| | | |
|---|---|---|
| TAMANIO_PAGINA | [Numérico] | Tamaño de las páginas y marcos en bytes. Siempre van a ser potencia de 2 y divisor del TAMANIO |
| ALGORITMO_REEMPLAZO_MMU | [Cadena] | Algoritmo de reemplazo de páginas |
| TIPO_ASIGNACION | [Cadena] | Modo de asignación de marcos (FIJA / DINAMICA) |
| MARCOS_POR_CARPINCHO | [Numérico] | Cantidad máxima de marcos a asignarle a un carpincho (Asignación Fija). |
| CANTIDAD_ENTRADAS_TLB | [Numérico] | Cantidad de entradas que almacenará la TLB |
| ALGORITMO_REEMPLAZO_TLB | [String] | Algoritmo de reemplazo de la TLB (FIFO / LRU) |
| RETARDO_ACIERTO_TLB | [Numérico] | Tiempo en milisegundos que deberá esperar una petición ante un acierto en la TLB<br>Puede ser 0 |
| RETARDO_FALLO_TLB | [Numérico] | Tiempo en milisegundos que deberá esperar una petición ante un fallo en la TLB<br>Puede ser 0 |
| PATH_DUMP_TLB | [String] | Path de donde deberán guardarse los Dumps de la TLB |

### Ejemplo de Archivo de Configuración

```
PUERTO=5001
TAMANIO=4096
TAMANIO_PAGINA=32
TIPO_ASIGNACION=FIJA
MARCOS_POR_CARPINCHO=4
ALGORITMO_REEMPLAZO_MMU=CLOCK-M
CANTIDAD_ENTRADAS_TLB=4096
ALGORITMO_REEMPLAZO_TLB=FIFO
RETARDO_ACIERTO_TLB=100
RETARDO_FALLO_TLB=1000
PATH_DUMP_TLB=/home/utnso/dumps/tlb
```

# Módulo: SWAmP

Este módulo será el encargado de administrar el área de intercambio o SWAP del sistema.

## Lineamiento e Implementación

Este módulo deberá implementarse con el objetivo de simular la concurrencia y el encolamiento de las peticiones a disco, por lo que independientemente de cómo se implemente, deberá atender las peticiones de a 1 a la vez. Esto quiere decir que toda la comunicación entre la memoria y swap **debe realizarse mediante el uso de un único socket, el cual no debe ser multiplexado.**

Al iniciarse, crearán archivos de tamaño (en bytes) y cantidad configurable, los cuales representarán nuestras particiones de swap, y quedarán a la espera de la o las conexiones del módulo de memoria. Estos archivos de swap deberán ser rellenados con el caracter '\0', a fines de inicializar la partición. [4]

El tamaño de las páginas escritas en swap, como también el nombre de este archivo y el tamaño del mismo deben ser tomados por archivo de configuración. Es importante tener en cuenta que el tamaño de página siempre va a ser el mismo que el de la memoria y que el tamaño del swap va a ser siempre un número múltiplo del tamaño de página.

Habrá dos esquemas de gestión para la gestión de las páginas en swap, los cuales **irán en función del esquema de asignación de páginas que haya en la memoria.**

- **Caso asignación fija:** Para que el manejo del espacio libre y ocupado en esta partición sea sencillo, se utilizará un esquema de asignación contigua, donde la cantidad máxima de marcos en swap por carpincho estará definida por archivo de configuración.
- **Caso asignación global:** No existe cantidad máxima de marcos por carpincho. El tamaño máximo de un carpincho será *como máximo* el tamaño de un archivo de swap.

Indistintamente del caso, ante la llegada a swap de nuevos carpinchos, el módulo siempre debe guardar las páginas de ese carpincho **en un único archivo de swap.** Esto quiere decir que no puede haber dos páginas de A, una en el archivo 1 y otra en el archivo 2, ambas páginas deben estar en un único archivo. A la hora de asignar el carpincho a un archivo de swap, se debe optar por el archivo **con mayor espacio disponible**, en caso de empate se deberá elegir el primer archivo que se encuentre en el listado**.**

Ante un pedido de lectura de página realizado la memoria, este módulo devolverá el contenido de esta página. Ante un pedido de escritura de página, sobreescribirá el contenido de esta página. Por último, cuando se le informe la finalización de un carpincho, deberá borrarlo de la partición de swap, devolviendo el marco al estado inicial cuando estaba vacío.

[4] investigar las funciones open, stat, truncate y mmap

## Archivo de Configuración

| Campo | Tipo | Descripción |
|---|---|---|
| IP | [String] | IP a la cual se deberá conectar con la memoria |
| PUERTO | [Numérico] | Puerto al cual se deberá conectar con la memoria |
| TAMANIO_SWAP | [Numérico] | Tamaño expresado en bytes de cada archivo de SWAP. |
| TAMANIO_PAGINA | [Numérico] | Tamaño expresado en bytes de una página. |
| ARCHIVOS_SWAP | [List] | Paths de los archivos de swap. |
| MARCOS_POR_CARPINCHO | [Numérico] | Cantidad de marcos a asignarle a un carpincho en swap (Asignación Fija) |
| RETARDO_SWAP | [Numérico] | Tiempo expresado en milisegundos[5] que deberá tener de retardo ante cada operación<br>Puede ser 0 |

### Ejemplo de Archivo de Configuración

```
IP=127.0.0.1
PUERTO=5001
TAMANIO_SWAP=4096
TAMANIO_PAGINA=64
ARCHIVOS_SWAP=[/home/utnso/swap1.bin, /home/utnso/swap2.bin]
MARCOS_POR_CARPINCHO=10
RETARDO_SWAP=500
```

[5] Investigar el funcionamiento de la función usleep();

# Descripción de las entregas

## Hito 1: Checkpoint Obligatorio Virtual - TP0

**Fecha**: 18/09/2021

**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.
- TP0 completo - https://github.com/sisoputnfrba/tp0

**Lectura recomendada:**

- Guía de sockets - http://faq.utnso.com.ar/guia-sockets
- Guía de serialización - http://faq.utnso.com.ar/guia-serializacion
- 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**: 9/10/2021

**Objetivos**:

- **Proceso Kernel:** Generar estructuras para administrar las colas de planificación. Manejar grado de multiprogramación y tener completas las conexiones de la lib.
- **Proceso Memoria:** Generar las estructuras base de la memoria y responder a los mensajes del kernel con valores fijos. Iniciar conexión con swap
- **Proceso Swap:** Generar las estructuras en memoria necesarias para poder contestar las peticiones de memoria.

**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**: 30/10/2021

**Objetivos**:

- **Proceso Kernel:** Multiprocesamiento y semáforos de la lib funcionando.
- **Proceso Memoria:** mem_alloc y swapeo funcionando sin paginación.
- **Proceso Swap:** Poder crear, leer y escribir un archivo de swap sin paginación.

**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**: 13/11/2021

**Objetivos**:

- **Proceso Kernel:** Suspensión y dispositivos I/O completos. Deadlock comenzado
- **Proceso Memoria:** Paginación completa con un esquema de asignación. TLB comenzada
- **Proceso Swap:** Múltiples swap files con un esquema de asignación implementado.

**Lectura recomendada:**

- Sistemas Operativos, Stallings, William 5ta Ed. - Parte VII: Memoria virtual (Cap. 8)
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 9: Memoria virtual

## Hito 5: Entregas Finales

**Fechas**: 27/11/2021 - 11/12/2021 - 18/12/2021

**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 estrés.

**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