UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ingeniería en Sistemas de Información

Cache 13

Atrévete TP

Cátedra de Sistemas Operativos

Trabajo práctico Cuatrimestral

- 2C2015 -
Versión [1.0]

Table of Contents

Introducción

Cache 13 es un pequeño sistema que emulará el funcionamiento de algunos módulos del Sistema Operativo. El mismo permitirá ejecutar algunos programas a los que llamaremos “mCod”, compuestos de sentencias sencillas. Estos programas en ejecución serán denominados procesos “mProc”. El orden en el cual se ejecutarán estará determinado por un planificador de corto plazo que, basándose en diferentes algoritmos, seleccionará el siguiente proceso a ejecutar.

La ejecución de estas sentencias requerirá diferentes accesos a memoria, tales como lecturas, escrituras, inicialización y finalización. Estos accesos serán realizados utilizando un administrador de memoria, y requerirán la utilización de todos los conceptos aprendidos durante las clases.

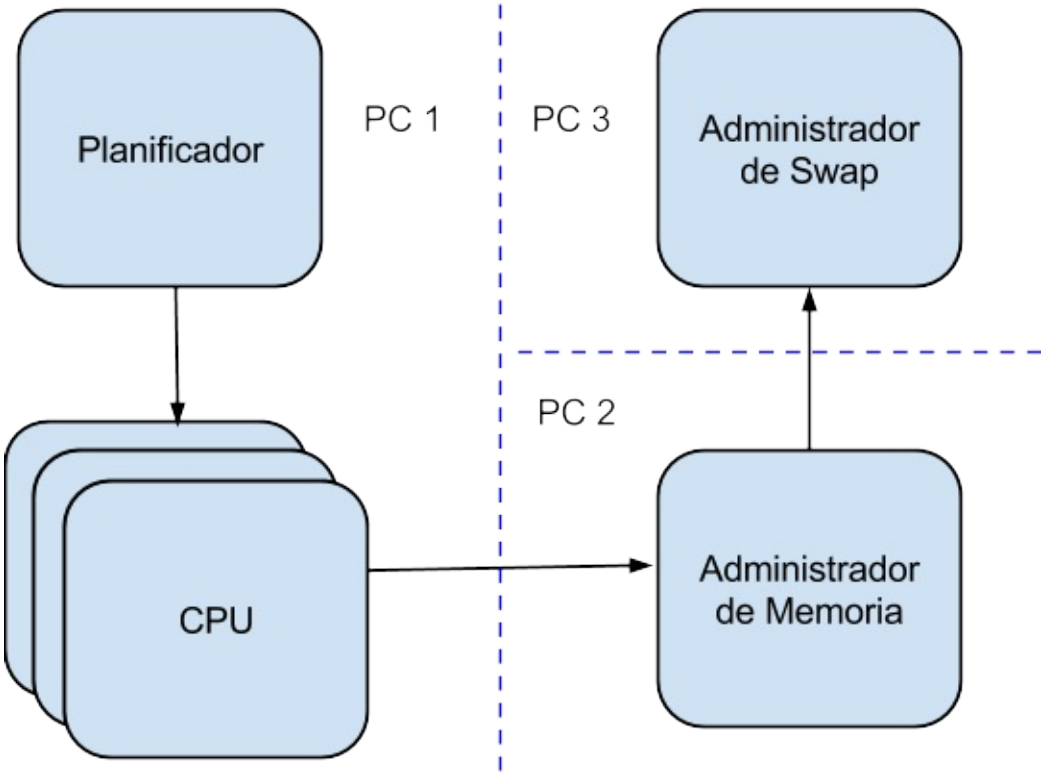
Esta simulación tiene por objetivo profundizar en los conceptos relacionados a la planificación de procesos y a la utilización de memoria virtual en un Sistema Operativo tradicional, implementado con paginación por demanda. Por ende, algunos de los accesos a memoria, requerirán también accesos a una partición de swap.

Para que el desarrollo del Trabajo Práctico coincida con el orden en que se dictan los temas de la materia, las diferentes entregas (de aquí en más, checkpoints) fueron cuidadosamente confeccionadas, evitando así retrabajo. Recomendamos fuertemente respetar este orden. Los requisitos indicados son los **mínimos** para considerar el checkpoint aprobado, pero es una excelente práctica continuar avanzando en caso de cumplirlos antes de la fecha límite.

Para mejor comprensión de la terminología utilizada, se colocarán en **negrita** las referencias a los diferentes procesos del TP. Los términos en *itálica* harán referencia a conceptos que fueron o serán vistos durante las clases teóricas. Por último, se enfatizarán ciertos aspectos que no deben ser pasados por alto utilizando texto subrayado.

Arquitectura de Cache 13

El sistema “Cache 13” constará de los siguientes Procesos[1]:



Los mismos deberán ser capaces de correr en tres PCs diferentes, como muestra el esquema. Al iniciarse, deberán conectarse teniendo en cuenta que los diferentes pedidos se realizarán en el sentido que muestran las flechas.

Para que la evaluación del trabajo práctico sea la adecuada, es muy importante que todos los parámetros que se mencionan como configurables estén en archivos de configuración correspondientes a cada proceso. Esta práctica se debe realizar también para parámetros de entorno tales como IPs, puertos, rutas, etc. Algunos procesos retardan la ejecución de ciertas funcionalidades, a fin de que la simulación se acerque más a un caso real, por lo cual se recomienda investigar el uso de las funciones para “dormir” procesos, conocidas como la familia de funciones `sleep()`. Este es el único caso donde estará permitido su uso.

Dado que el Trabajo Práctico se realizará de forma incremental, se recomienda prestar especial atención al diseño de las interfaces entre los diferentes procesos. De esta forma, cuando una funcionalidad crezca, la interfaz será la misma. Esto es especialmente útil cuando se requiera implementar diferentes algoritmos que realicen la misma tarea de forma distinta.

[1] *Proceso*: un programa en ejecución. Para más información referirse a la clase teórica asociada.

Proceso Planificador

Será el encargado de administrar la ejecución de los procesos “mProc”, permitiendo que estos inicien, ordenándolos para que ejecuten en las distintas **CPUs** y finalizándolos cuando sea necesario. Para ello, utilizará programas “mCod”, que serán puestos en ejecución.

A partir de que el **Planificador** inicie, deberá mostrar por pantalla una consola, que será capaz de soportar algunos comandos, definidos en el anexo respectivo. Se debe contemplar la posibilidad de ejecutar varios procesos “mProc” concurrentemente (*multiprogramación*), por lo cual la consola deberá estar siempre disponible. Además, el mismo programa “mCod” deberá poder ser ejecutado más de una vez, generando múltiples procesos “mProc” conviviendo al mismo tiempo en el sistema.

Luego de iniciado, el **Planificador** quedará a la espera de que se conecten distintas **CPUs**, sobre las cuales enviará a ejecutar estos procesos cuando la planificación lo indique y cuando éstas se encuentren libres.

Tras recibir por consola una solicitud de ejecución de un programa “mCod”, se generará su *PCB*[2] asociado, incluyendo en él la ruta relativa del archivo; una vez creada esta estructura se lo podrá considerar como listo para ejecutar. Para más detalles sobre el formato de este archivo ver el Anexo II: Formato de los Archivos.

El **Planificador** será también el responsable de decidir qué proceso debe ejecutar en cada momento, basándose en diferentes algoritmos y conociendo la disponibilidad de cada **CPU**. Una vez tomada la decisión, deberá enviar a la **CPU** su *contexto de ejecución* (en nuestro caso, la ruta al “mCod” y el puntero a la próxima instrucción), y la cantidad de sentencias a ejecutar en caso de que la planificación se realice con *desalojo*. Cuando la **CPU** finalice la ejecución de cada *ráfaga*[3] o bien cuando interrumpa la ejecución de un proceso, ésta devolverá una serie de mensajes, que deberán ser escritos en el log del proceso **Planificador**. Para simplicidad a la hora de planificar, cualquier **CPU** libre podrá ser otorgada a un proceso que la necesite.

El algoritmo de planificación a utilizar deberá ser especificado por archivo de configuración, así como todos los parámetros adicionales que cada algoritmo en particular requiera.

Archivo de configuración

Nombre de Campo	Valor de ejemplo
`PUERTO_ESCUCHA`	`4000`
`ALGORITMO_PLANIFICACION`	`FIFO` [4]
`QUANTUM`	`5`

El valor de Quantum sólo será utilizado si el algoritmo usado lo requiere.

[2] El *Process Control Block (PCB)* será una estructura de datos que tendrá toda aquella información que el grupo considere necesaria para ejecutar los procesos. La misma deberá ser validada y justificada con los ayudantes asignados, en las diferentes instancias de evaluación. [3] Se considera una *ráfaga* a un conjunto de instrucciones que ejecuta una CPU en un determinado período de tiempo hasta que finaliza, se bloquea o es interrumpido. [4] Otra alternativa es `RR`.

Proceso CPU

Este proceso será el encargado de interpretar y ejecutar las instrucciones enviadas por el planificador. Dado que algunas instrucciones requerirán accesos a memoria, deberá comunicarse con el **Administrador de Memoria** a fin de realizarle diferentes pedidos. Para simular la presencia de múltiples CPUs, será necesario implementar *hilos*[5]. La cantidad de hilos **CPU** será configurable y no variará durante la ejecución del sistema. Cada uno de los hilos **CPU** deberá tener un número que lo distinga (id), y este valor será utilizado para identificar todos los eventos que ocurrieron en esta **CPU**, en los logs de todos los procesos que lo necesiten.

Al iniciar, intentará conectarse al proceso **Planificador** y se quedará a la espera de solicitudes de ejecución de programas “mCod” por parte del mismo. También, intentará conectarse al proceso **Administrador de Memoria**, que es a quien solicitará las operaciones de lectura o escritura de memoria. En caso de que no pueda conectarse con alguno de ellos, la **CPU** finalizará su ejecución informando en el archivo de log del proceso.

Cuando el **Planificador** envíe el *contexto de ejecución* a alguna **CPU**, ésta pondrá en ejecución el proceso “mProc”. Para ello abrirá y leerá el archivo ejecutable completo cuya ruta relativa fue enviada, interpretando cada línea como una instrucción[6].

Cada instrucción tendrá un valor de retorno, que será enviado al planificador al terminar la *ráfaga*. Las posibles instrucciones a implementar son:

- `iniciar`
- `leer`
- `escribir`
- `entrada-salida`
- `finalizar`

Luego de ejecutar cada instrucción, la **CPU** deberá esperar una cantidad de tiempo configurable (en segundos), simulando el tiempo de ejecución de la misma, antes de continuar con la siguiente instrucción. Tanto al ejecutar la instrucción “finalizar”, como al haber ejecutado el tiempo indicado por el **Planificador**, o bien al ejecutar una instrucción de entrada salida, la **CPU** devolverá el resultado de todas las instrucciones ejecutadas durante esa *ráfaga*. Hasta que esto ocurra, la **CPU** no enviará ninguna respuesta relacionada a este proceso.

Al ejecutar una instrucción que requiera acceso a memoria, se llamará al **Administrador de Memoria** para satisfacerla y la ejecución del proceso “mProc” deberá esperar la respuesta para continuar, aunque esta fuese prolongada. Para simplificar el trabajo práctico, no se contempla el reinicio de una instrucción ante un *fallo de página*[7].

Una vez enviada la información mencionada anteriormente al **Planificador**, volverá a quedar a la espera de solicitudes de ejecución.

Archivo de Configuración

Nombre de Campo	Valor de Ejemplo
`IP_PLANIFICADOR`	`192.168.101.10`
`PUERTO_PLANIFICADOR`	`4000`
`IP_MEMORIA`	`192.168.101.11`
`PUERTO_MEMORIA`	`5000`
`CANTIDAD_HILOS`	`4`
`RETARDO`	`2` [8]

[5] *Hilo* o *Thread*: traza de ejecución perteneciente a un proceso, susceptible de ser planificada por el sistema operativo. Para más información referirse a la clase teórica asociada. [6] En un Sistema Operativo real, la CPU utilizaría un puntero al código en memoria. A fines de simplificar el trabajo práctico, el proceso **CPU** podrá abrir y cerrar el archivo con el código, dejando en memoria únicamente la sección de datos. [7] En un Sistema Operativo real, un fallo de página devolvería una excepción que es tratada de forma especial, generando al menos una operación de entrada salida. Por dicho motivo, es habitual bloquear al proceso hasta que la página solicitada esté disponible, permitiendo que otro proceso utilice la CPU. [8] Especificado en segundos

Proceso Administrador de Memoria

Será el encargado de administrar la memoria del sistema. Soportará *paginación por demanda*, utilizando una *TLB* y una partición de *swap*. Además limitará la cantidad de *marcos* otorgados a cada proceso, siendo esta cantidad configurable, con *asignación fija* y *reemplazo local*[9]. La asignación de frames a un determinado proceso será efectiva sólo cuando este lo necesite, pudiendo nunca llegar al límite mencionado. Al iniciar, se conectará al **Administrador de Swap** y quedará a la espera de conexiones de hilos **CPU**.

El **Administrador de Memoria** utilizará una estructura en memoria que emulará una única cache *TLB*, con las columnas que sean necesarias para su buen funcionamiento. Su cantidad de entradas será configurable, y no variará durante la ejecución de este proceso. También se deberá poder deshabilitar el uso de esta cache, con una opción en el archivo de configuración.

Además, el **Administrador de Memoria** manejará una *tabla de páginas* para cada proceso, con las columnas que sean necesarias para cada instancia del Trabajo Práctico. Por último, reservará una cantidad de *memoria principal para procesos* configurable, para ser dividida en marcos de tamaño configurable. A fines prácticos, ante cualquier acceso a la *tabla de páginas o a memoria principal*[10], se deberá esperar una cantidad de tiempo configurable -en segundos, e igual para ambos tipos de acceso-, simulando el tiempo de acceso a memoria.

Cuando un **CPU** informe el inicio de un nuevo proceso “mProc”, deberá crear las estructuras necesarias para administrarlo correctamente. Además, deberá informar tal situación al **Administrador de Swap**, junto con la cantidad de páginas de datos a utilizar, para que este asigne el espacio necesario en su partición.

Ante un pedido de lectura de página de alguno de los procesadores, el **Administrador de Memoria** realizará la traducción a *marco (frame)* y se devolverá el contenido correspondiente[11]. En caso de que la página no se encuentre en *memoria principal*, será solicitada al proceso **Administrador de Swap**, corriendo luego el algoritmo correspondiente para cargarla en *memoria principal*. A fines de simplificar el desarrollo, se asumirá que cualquier página solicitada por un proceso es válida.

Ante un pedido de escritura de página de alguno de los procesadores, se realizará la traducción a marco (frame), y se actualizará su contenido. Queda a criterio del grupo devolver o no el contenido, puesto que el proceso **CPU** ya lo conoce. En caso de que la página no se encuentre en *memoria principal*, será solicitada al proceso **Administrador de Swap**, corriendo luego el algoritmo correspondiente para cargarla en *memoria principal*. Es importante recordar que las escrituras a *Swap* se hacen únicamente cuando se reemplaza una página que fue modificada previamente. Si fuese necesario asignar un marco y no hubiese ninguno disponible, se finalizará el proceso “mProc”.

Cuando un **CPU** informe el fin de un nuevo proceso “mProc”, el **Administrador de Memoria** deberá eliminar las estructuras usadas para administrarlo. Además, deberá informar tal situación al **Administrador de Swap**, para que este libere el espacio utilizado en su partición.

Por último, el **Administrador de Memoria**, deberá ser capaz de recibir e interpretar tres *señales* diferentes[12]. La primera de ellas deberá limpiar completamente la TLB (*TLB flush*), utilizando un hilo correctamente sincronizado para esto, evitando problemas de concurrencia. La segunda señal deberá limpiar completamente la *memoria principal*, actualizando los bits que sean necesarios en las tablas de páginas de los diferentes procesos. Para evitar problemas de concurrencia, aquí también se deberá utilizar un hilo correctamente sincronizado. Por último, la tercera señal deberá realizar un volcado (dump) del contenido de la *memoria principal*, en el archivo log de **Administrador de Memoria**, creando para tal fin un proceso nuevo[13]. El volcado deberá indicar el número de *marco* y su contenido, utilizando una fila por cada *marco*.

Archivo de Configuración

Nombre de Campo	Valor de Ejemplo
`PUERTO_ESCUCHA`	`5000`
`IP_SWAP`	`192.168.101.12`
`PUERTO_SWAP`	`6000`
`MAXIMO_MARCOS_POR_PROCESO`	`3`
`CANTIDAD_MARCOS`	`128`
`TAMANIO_MARCO`	`256`
`ENTRADAS_TLB`	`4`
`TLB_HABILITADA`	`SI`
`RETARDO_MEMORIA`	`8` [14]

[9] Para más información, referirse al capítulo 9 de “Fundamentos de Sistemas Operativos”, de Abraham Silberschatz. [10] Si bien el Trabajo Práctico simplifica este aspecto, recuerde que las tablas de páginas se almacenan en Memoria Principal. [11] Recuerde los pasos del proceso de traducción: búsqueda en TLB, luego (de ser necesario) búsqueda en la tabla de páginas y por último búsqueda del frame correspondiente. [12] Se deberán usar las señales `SIGUSR1`, `SIGUSR2` y `SIGPOLL` para estos tres escenarios. [13] Recomendamos investigar el uso de `fork` y el concepto de “Copy on write”. [14] En segundos

Proceso Administrador de Swap

Será el encargado de administrar la *memoria virtual* de “Cache 13”, utilizando un esquema muy sencillo. Para ello, al iniciarse, creará un archivo de tamaño configurable (en bytes), el cual representará nuestra *partición de swap*, y quedará a la espera de la conexión del **Administrador de Memoria**. El archivo de swap deberá ser rellenado con el caracter `\0`, a fines de inicializar la partición[16]. El tamaño de las páginas escritas en swap es configurable[17], así como también el nombre de este archivo.

Para que el manejo del espacio libre y ocupado en esta partición sea sencillo, se utilizará un esquema de asignación contigua. La *partición de swap* será considerada inicialmente como un hueco del total de su tamaño, medido en cantidad de páginas que puede alojar. Ante la llegada de un proceso “mProc”, asignará el tamaño necesario para que este sea guardado, dejando el espacio restante como libre. Esto mismo sucederá con los siguientes procesos “mProc” puestos en ejecución. Al finalizar un proceso “mProc”, el espacio que tenía asignado será marcado como libre. En caso de que esto genere dos huecos contiguos, estos se unirán formando un hueco mayor. Para administrar el espacio utilizado se usará una lista enlazada con el PID, el comienzo del proceso “mProc” en la partición, y la cantidad de páginas que ocupa. Para administrar el espacio libre, se utilizará una lista enlazada con el comienzo del hueco y la cantidad de páginas que representa. La unidad mínima de asignación será una página[18].

Cuando le sea informada la creación de un nuevo proceso “mProc”, procederá a buscar un hueco donde quepa. En caso de que el espacio total disponible no sea suficiente, deberá rechazar el proceso, siendo su inicialización cancelada. En cambio, cuando la *fragmentación externa*[19] sea la que no permite crear nuevos procesos, se deberá compactar la partición. Todos los pedidos que lleguen mientras dure este procedimiento deberán esperar a que este finalice. Este procedimiento deberá esperar una cantidad de tiempo configurable (en segundos), simulando el tiempo de compactación.

Ante un pedido de lectura de página realizado por el **Administrador de 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 proceso, deberá borrarlo de la *partición de swap*.

Archivo de Configuración

Nombre de Campo	Valor de Ejemplo
`PUERTO_ESCUCHA`	`6000`
`NOMBRE_SWAP`	`swap.data`
`CANTIDAD_PAGINAS`	`512`
`TAMANIO_PAGINA`	`256`
`RETARDO_COMPACTACION`	`60`[20]

[16] Se recomienda al alumno investigar sobre la utilización del comando `dd` para crear los archivos. [17] Recuerde que el tamaño de las **páginas** es el mismo que el de los **marcos**. [18] Para más información sobre asignación contigua, remitirse al capítulo 11 de “Fundamentos de Sistemas Operativos” de Abraham Silberschatz. [19] Es muy importante no confundir la *fragmentación externa* con la falta de espacio disponible. En el primer caso el espacio está, pero no puede ser asignado por estar fragmentado. [20] En segundos.

Eventos de Logueo Obligatorio

Existen eventos que ocurren en cada proceso desarrollado que son relevantes para la verificación de su correcto funcionamiento. Es por eso que, más abajo, se definen una serie de eventos de logueo obligatorios.

Planificador

- Comienzo y fin de ejecución de procesos, indicando al menos: PID asignado y nombre del archivo “mCod”.
- Conexión y desconexión de **CPUs**.
- Ejecución del algoritmo de planificación, indicando el proceso “mProc” seleccionado para ejecutar y el contenido ordenado de todas las colas de planificación.
- Ráfaga de **CPU** completada, indicando el proceso mProc y todos los mensajes de la CPU.

CPU

Todos los hilos compartirán el mismo archivo log, prefijando en cada evento el número de cpu que está realizando el logueo

- Instancia de **CPU** creada/conectada al **Administrador de Memoria**, indicando el identificador de hilo **CPU** (id)
- *Contexto de ejecución* recibido, indicando todos sus datos, y el quantum de ejecución.
- Instrucción ejecutada, indicando PID, el valor de sus parámetros (si hubiese) y el resultado de la instrucción.
- Ejecución de rafaga concluida, indicando PID.

Administrador de Memoria

- Proceso mProc creado, indicando PID y cantidad de páginas asignadas
- Solicitud de escritura/lectura recibida, indicando PID y N° de página TLB hit/miss, indicando N° de página ingresado, N ° de marco resultante, y en caso de miss resultado del algoritmo de reemplazo (si aplica)
- Acceso a memoria realizado, indicando PID, N° de página y N° de marco
- Acceso a swap (fallo de página), indicando PID, resultado del algoritmo de sustitución de páginas, estado inicial y final de las colas y punteros correspondientes (si aplica)
- Señal recibida/tratamiento de señal terminado, indicando tipo y acción a ejecutar.

Administrador de Swap

- Proceso mProc asignado, indicando PID, N° de byte inicial, y tamaño en bytes
- Proceso mProc liberado, indicando PID, N° de byte inicial, y tamaño en bytes liberado
- Proceso mProc rechazado por falta de espacio
- Compactación iniciada/finalizada por fragmentación externa
- Escritura/lectura solicitada, indicando PID, N° de byte inicial, tamaño y contenido

En general, no se debería imprimir por pantalla ninguno de estos eventos, sino que para ello debe usarse el archivo de log. Se reservará la impresión por pantalla para indicar terminaciones anormales, escribiendo los detalles en el archivo de log. Para poder seguir las entradas que va logueando un proceso del Sistema Operativo, se recomienda usar el comando: `tail -f <archivo de log>`, que va imprimiendo nuevas líneas conforme va creciendo dicho archivo. Por ejemplo:

```
> tail -f cpu_instancia_01.log
```

Cada vez que un proceso del Sistema Operativo inicie su ejecución, deberá imprimir en el log una línea especial indicando que comenzó la ejecución del mismo. Además, es posible ampliar la cantidad de eventos a loguear por encima de los mínimos especificados arriba, siempre que contribuya a entender el funcionamiento del programa y el mismo siga siendo legible y fácil de seguir. Se recomienda utilizar la funcionalidad de Logging de la Commons Library que facilita la generación y escritura de archivos de log.

Anexo I: Especificación del lenguaje mAnsisOp

El lenguaje deberá soportar las siguientes operaciones:

`iniciar N`

- Deberá informar al **Administrador de Memoria** que se ha iniciado un proceso “mProc” de N páginas de datos.
- Deberá retornar la expresión `mProc X - Iniciado` o bien `mProc X - Fallo`, dependiendo de su correcta iniciación.

`leer N`

- Deberá leer la página N del proceso “mProc” en ejecución.
- Deberá retornar la expresión `mProc X - Pagina N leida:` junto al contenido de esa página concatenado. Ejemplo: `mProc 10 - Pagina 2 leida: contenido`

`escribir N “texto”`

- Deberá escribir el texto en la página N del proceso “mProc” en ejecución, sobreescribiendo todo su contenido. De ser necesario, se rellenará con el caracter `\0`.
- Deberá retornar la expresión `mProc X - Pagina N escrita:` junto al nuevo contenido de esa página concatenado. Ejemplo: `mProc 1 - Pagina 2 escrita: otro contenido`.

`entrada-salida T`

- Realizará una *entrada-salida* de tiempo T. La ejecución de esta instrucción provocará que la **CPU** informe al **Planificador** que debe bloquear al proceso “mProc” en ejecución durante T segundos.
- Deberá retornar la expresión `mProc X en entrada-salida de tiempo T`.
- Además, esta instrucción libera la **CPU**.

`finalizar`

- Concluye la ejecución del proceso “mProc”. Aquí se da aviso al **Administrador de Memoria** para que elimine la tabla de páginas asociada, limpie si es necesario la TLB y dé aviso al **Administrador de Swap**.
- Deberá retornar la expresión `mProc X finalizado`.

Anexo II: Formato de los archivos

Los archivos ejecutables tendrán el siguiente formato:

- Una cantidad de instrucciones variable
- Terminan con la instrucción `finalizar`
- Se coloca una instrucción por línea, finalizada con un punto y coma

Ejemplo de programa:

```
escribir "Hola" 1
leer 1
entrada-salida 5000
finalizar
```

Anexo III: Comandos del Planificador

El proceso **Planificador** deberá ser capaz de soportar los siguientes comandos:

`correr PATH`

- PATH es la ruta relativa al programa “mCod” a ejecutar.
- Este comando deberá ejecutar el programa en cuestión.
- Ejemplo: `correr prueba.cod` ejecutará el programa “mCod” llamado `prueba.cod`.

`finalizar PID`

- PID es el ID del proceso “mProc” en ejecución.
- Este comando deberá mover el *puntero de instrucciones* haciéndolo apuntar a la última del programa “mCod”. De esta forma, cuando este proceso ejecute nuevamente, finalizará.
- Ejemplo: `finalizar 13`, colocará el puntero en la última línea del proceso “mProc” de PID 13.

`ps`

- Este comando deberá escribir en la pantalla del **Planificador** el PID, el nombre del programa y el estado de cada proceso “mProc”.
- El formato deberá ser: `mProc PID: nombre -> estado`, escribiendo en una línea diferente cada proceso “mProc”. Por ejemplo:

```
mProc 1: prueba.cod -> Listo
mProc 2: otro.cod -> Ejecutando
mProc 3: entrada.cod -> Bloqueado
```

`cpu`

- Este comando deberá escribir en la pantalla del **Planificador** un listado de las CPUs actuales del sistema, indicando para cada una el porcentaje de uso del último minuto. Por ejemplo:

```
cpu 1: 50%
cpu 2: 80%
cpu 3:  0%
```

Descripción de las entregas

Para permitir una mejor distribución de las tareas y orientar al alumno en el proceso de desarrollo de su trabajo, se definieron una serie de puntos de control y fechas que el alumno podrá utilizar para comparar su grado de avance respecto del esperado. En cada una de estas entregas se otorgará feedback al grupo, para que sepan fehacientemente si su presentación cumple o no con el mínimo estipulado.

En cada checkpoint se explicita una serie de requisitos mínimos que debe tener cada entrega. Dado que la descripción de los componentes del TP enuncia generalidades, es importante que cada grupo esté atento a estos requisitos, que detallan algoritmos y funcionalidades a tener en cuenta, entre otras cosas..

Checkpoint 1 - Obligatorio

Fecha: 12 de septiembre

Objetivos:

- Familiarizarse con Linux y su consola, el entorno de desarrollo y el repositorio
- Aplicar las Commons Libraries, principalmente las funciones para listas, archivos de conf y logs
- Dar los primeros pasos en el Trabajo Práctico, creando la base de los procesos y su esquema de comunicación

Requisitos mínimos:

- Entorno instalado y funcionando. Todos los integrantes deben haber subido commits a GitHub.
- Tener los 4 procesos del sistema creados, conectándose entre sí.
- Poder enviar el comando `correr programa` al **Planificador**, y que este mensaje sea enviado a la **CPU**, luego de la **CPU** al **Administrador de Memoria** y luego del **Administrador de Memoria** al **Administrador de Swap**. Será suficiente con que cada proceso imprima el mensaje recibido por pantalla.

Distribución recomendada: Todos los miembros trabajando juntos para crear los procesos y poder subir el código a GitHub.

Lectura recomendada:

- http://faq.utn.so/arrancar
- Beej Guide to Network Programming
- Linux POSIX Threads
- SisOpUTNFRBA Commons Libraries
- Fundamentos de los Sistemas Operativos - Silberschatz, Galvin - Capítulo 3: Procesos
- Fundamentos de los Sistemas Operativos - Silberschatz, Galvin - Capítulo 4: Hilos

Checkpoint 2 - Obligatorio

Fecha: 3 de octubre

Objetivos:

- Construir incrementalmente un **Planificador** de procesos "mProc" básico
- Comunicar procesos del sistema correctamente
- Sincronizar correctamente la ejecución de los procesos del sistema
- Modelar la primera versión de las estructuras necesarias para el desarrollo

- Implementar la primera versión del **Administrador de Memoria** y del **Administrador de Swap**

Requisitos mínimos:

- El **Planificador** debe poder correr N procesos "mProc", eligiéndolos mediante un algoritmo FIFO
- La **CPU** debe ser capaz de correr 1 *hilo*. Este *hilo* debe poder interpretar los comandos leer y finalizar. Debe poder comunicarse con el Administrador de Memoria tanto para iniciar y finalizar procesos "mProc" como para realizar las lecturas necesarias.
- El **Administrador de Memoria** debe ser capaz de recibir pedidos de lecturas, y directamente reenviarlos **Administrador de Swap**. También debe interpretar los pedidos para iniciar y finalizar procesos "mProc".
- El **Administrador de Swap** debe ser capaz de recibir nuevos procesos “mProc”. También debe poder eliminar estos procesos. Ante un pedido de lectura, debe poder buscar en su partición, para devolver el contenido de página adecuado.

Distribución recomendada: 1 persona: **Planificador**. 1 persona: **CPU**. 0.5 persona: **Adm de Memoria**. 1.5 personas **Adm de Swap**

Lectura recomendada:

- Fundamentos de los Sistemas Operativos - Silberschatz, Galvin - Capítulo 5: Planificación de la CPU
- Fundamentos de los Sistemas Operativos - Silberschatz, Galvin - Capítulo 6: Sincronización de Procesos

Checkpoint 3 - Obligatorio - Laboratorio

Fecha: 7 de noviembre - Laboratorio Azul de Medrano (3er piso)

Objetivos:

- Construir un **Planificador** de procesos de mayor complejidad, que sea capaz de soportar una variedad de algoritmos.
- Crear procesos del sistema, con múltiples hilos, correctamente sincronizados para soportar concurrencia.
- Desarrollar un **Administrador de memoria** capaz de manejar correctamente el flujo que ocurre ante cada acceso a memoria.
- Desarrollar una partición de file system de fácil manejo.

Requisitos mínimos:

- El **Planificador** debe poder correr N procesos “mProc”, eligiendolos mediante un algoritmo Round Robin (con quantum configurable) o FIFO. El algoritmo a ejecutar será configurable, y no cambiará mientras se ejecuta el **Planificador**.
- La **CPU** debe ser capaz de correr como N hilos. Estos hilos deberán poder interpretar todos los comandos previamente creados y además de permitir la escritura de páginas.
- El **Administrador de Memoria** debe ser capaz de recibir pedidos de inicio, fin, lecturas y escrituras e interpretarlos correctamente, buscando en TLB, tabla de páginas o bien en swap, según corresponda. La TLB utilizará el algoritmo FIFO para sus reemplazos. Los procesos “mProc” tendrán una asignación a demanda de marcos con un máximo configurable, igual para todos. La sustitución será local. El algoritmo de reemplazo de marcos será FIFO. Por último, deberá implementar las tres señales previamente mencionadas.
- El **Administrador de Swap** debe ser capaz de recibir pedidos de inicio, fin, lecturas y escrituras.

Distribución recomendada: 1 persona: **Planificador**. 1 persona: **CPU**. 1.5 personas: **Adm de Memoria**. 0.5 persona **Adm de Swap**.

Lectura recomendada:

- Fundamentos de los Sistemas Operativos - Silberschatz, Galvin - Capítulo 8: Memoria Principal

- Fundamentos de los Sistemas Operativos - Silberschatz, Galvin - Capítulo 10: Interfaz del sistema de archivos

Entrega Final

Fecha: 28 de noviembre

Objetivos:

- Implementar una mayor variedad de algoritmos de reemplazo de marcos.
- Fomentar la investigación de los alumnos, acerca de diferentes métricas y criterios.
- Investigar el uso de señales.
- Finalizar el desarrollo del Trabajo Práctico.
- Empaquetar el desarrollo para un fácil deployment.
- Subir y taggear en GIT la última versión estable.
- Validar los tests provistos por la cátedra.
- Evaluar el Trabajo Práctico en el laboratorio.
- Obtener conclusiones en base a la ejecución de diferentes lotes de procesos “mProc”.

Requisitos mínimos:

- El **Planificador** debe escribir en su log las siguientes métricas, cuando un proceso finaliza: tiempo de respuesta del proceso, tiempo de ejecución, tiempo de espera.
- Cada **CPU** debe calcular y mostrar su porcentaje de uso.
- El **Administrador de Memoria** debe incluir los algoritmos de reemplazo de marcos FIFO, LRU y Clock Modificado. Cuando finaliza un proceso “mProc”, debe mostrar la cantidad de fallos de página vs la cantidad total de páginas accedidas para este. Cada un minuto debe mostrar la tasa de aciertos de la TLB histórica (es decir, incluyendo todos los valores generados desde que el **Administrador** entró en funcionamiento).
- El **Administrador de Swap** debe, cada vez que un proceso “mProc” finalice, indicar la cantidad de páginas leídas y escritas por él. Además, debe ser capaz de compactar la partición cuando corresponda.

Distribución recomendada: 1 persona: **Planificador**. 0.5 persona: **CPU**. 2 personas: **Adm de Memoria**. 0.5 persona **Adm de Swap**

Lectura recomendada:

- Fundamentos de los Sistemas Operativos - Silberschatz, Galvin - Capítulo 9: Memoria Virtual

Recuperatorios

- Primer recuperatorio: 5 de diciembre
- Segundo recuperatorio: 19 de diciembre

Normas del Trabajo Práctico

El trabajo práctico se rige por las reglas detalladas en el documento Normas del Trabajo Práctico.