KISS

Kernel de Implementación Super Simple

Hacemos un TP simple para explicar lo complejo

Cátedra de Sistemas Operativos

Trabajo práctico Cuatrimestral

-1C2022 -
Versión 1.1

Índice

Historial de Cambios

v1.0 (12/04/2022) Publicación del enunciado

v1.1 (26/04/2022) Versión 1.1:

- *Se modifica la estructura de la TLB.*
- *Se explicita que se debe tener una única cola de procesos bloqueados.*
- *Se explicita mensaje de finalización de proceso y su procedimiento en Memoria.*
- *Rewording de algunos conceptos para su mejor entendimiento.*

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: 09/04/2022
- Fecha de primera entrega: 16/07/2022
- Fecha de segunda entrega: 23/07/2022
- Fecha de tercera entrega: 30/07/2022
- Lugar de corrección: A definir según situación sanitaria.

Evaluación del Trabajo Práctico

El trabajo práctico consta de una evaluación en 2 etapas.

La primera etapa consistirá en las pruebas de los programas desarrollados en el laboratorio[1]. Las pruebas del trabajo práctico se subirán oportunamente y con suficiente tiempo para que los alumnos puedan evaluarlas con antelación. Queda aclarado que para que un trabajo práctico sea considerado evaluable, el mismo debe proporcionar registros de su funcionamiento de la forma más clara posible.

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

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

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

[1] Se definirá según la condición sanitaria si se realizaran presenciales o virtuales

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*.

Deployment y Testing del Trabajo Práctico

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

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

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

Aclaraciones

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

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 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.

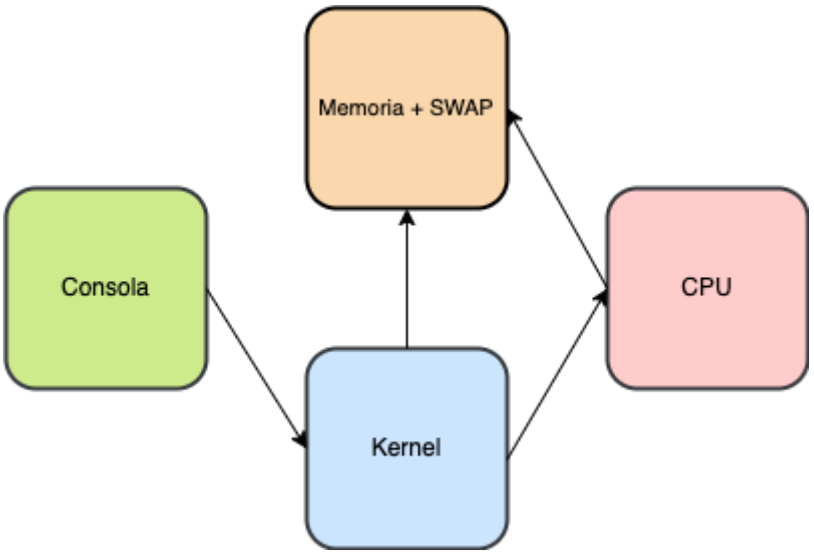
¿Qué es el trabajo práctico y cómo empezamos?

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 el esquema de memoria explicado en el módulo correspondiente.

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



El trabajo práctico consiste de 4 módulos: **Consola** (N instancias), **Kernel**, **CPU** y **Memoria** (1 instancia cada uno).
El proceso de ejecución del mismo consiste en crear procesos a través de instancias del módulo **Consola**, las cuales enviarán la información necesaria al módulo **Kernel** para que el mismo pueda crear las estructuras necesarias a fin de administrar y planificar su ejecución mediante diversos algoritmos. Estos procesos serán ejecutados en el módulo **CPU**, quien interpretará sus instrucciones y hará las peticiones a **Memoria** que fuera necesarias. Este último módulo implementará un esquema de memoria virtual (paginación a demanda), por lo que incorporará también un área de SWAP.
Una vez que un proceso finalice tras haber sido ejecutadas todas sus instrucciones, el Kernel devolverá un mensaje de finalización a su **Consola** correspondiente y cerrará la conexión.

Distribución Recomendada

Estimamos que a lo largo del cuatrimestre la carga de trabajo para cada módulo será la siguiente:

- Consola: **10%**
- Kernel: **30%**
- CPU: **30%**
- Memoria+SWAP: **30%**

Dado que el trabajo funciona bajo el concepto iterativo incremental, recomendamos modificar dichos números a lo largo del cuatrimestre para que todos los integrantes del grupo participen en varios módulos.

****Aclaración**
Será condición necesaria de aprobación demostrar conocimiento teórico y de trabajo en alguno de los módulos principales (Kernel, CPU y/o Memoria). Desarrollar únicamente temas de conectividad, serialización, sincronización, o el módulo Consola, es insuficiente para poder entender y aprender los distintos conceptos de la materia. Dicho caso será un motivo de desaprobación.

Módulo: Consola

El módulo **Consola** será el punto de entrada. Su objetivo es definir los procesos que habrá en nuestro sistema, instanciándose tantas veces como procesos queramos tener en el mismo.

Para poder instanciar una consola, la misma recibirá por parámetro:

- El tamaño que tendrá el proceso
- Un path a un archivo de texto con pseudocódigo, con el que creará una lista de instrucciones

El módulo tendrá también un archivo de configuración con los datos para conectarse al módulo Kernel. Esta información será enviada al mismo para que pueda crear el respectivo PCB y administrarlo como proceso.

Lineamiento e Implementación

La **Consola** deberá aceptar 2 parámetros para ser ejecutado: el **path** del archivo de pseudocódigo y el **tamaño** que tendrá el espacio de direcciones del proceso al cargarse en Memoria, el mismo es independiente del archivo de pseudocódigo. Se recomienda utilizar los parámetros argc y argv de la función main ayudándose con esta guía.

Al iniciar, el módulo leerá el archivo de pseudocódigo y parseará cada línea terminada en “\n” como una instrucción. Cada instrucción se compone de un identificador y sus parámetros, siendo los últimos, en caso que los tuviera, siempre numéricos (valores enteros no signados de 4 bytes). De esta manera generará una **lista de instrucciones** que corresponden al programa que representa.

Luego, leerá su archivo de configuración, se conectará al Kernel y le enviará toda la información del proceso (lista de instrucciones, tamaño).

Una vez enviada la información del proceso, esperará por el mensaje de finalización y terminará.

Archivo de pseudocódigo

Este archivo contendrá una serie de líneas terminadas en “\n” las cuales deben ser parseadas para ser transformadas en instrucciones que luego se enviarán al Kernel.

Cada línea tendrá el formato “IDENTIFICADOR parámetros”. Los parámetros serán siempre valores numéricos separados por un espacio. Según el identificador, cada línea puede tener entre 0 y 2 parámetros.

Para facilitar su legibilidad y posteriores pruebas del trabajo práctico, el identificador NO_OP tendrá un parámetro especial, que no debe ser tomado como parámetro de la instrucción en sí, sino que representa la cantidad de veces que se repite esa misma instrucción. Por ejemplo, “NO_OP 5” debe ser tomado como 5 instrucciones seguidas de NO_OP.

Dicho esto, los posibles identificadores a parsear serán los siguientes:

- **NO_OP**: 1 parámetro (descrito anteriormente)
- **I/O y READ**: 1 parámetro
- **COPY y WRITE**: 2 parámetros
- **EXIT**: 0 parámetros

El comportamiento de cada una de estas instrucciones está detallada en el módulo CPU que será el que finalmente deba ejecutarlas.

Ejemplo de líneas a parsear

```
NO_OP 5
I/O 3000
WRITE 4 42
COPY 0 4
READ 0
EXIT
```

Archivo de configuración

Campo	Tipo	Descripción
IP_KERNEL	String	IP del Kernel al que habrá que conectarse
PUERTO_KERNEL	Numérico	Puerto del Kernel al que habrá que conectarse

Ejemplo de Archivo de Configuración

```
IP_KERNEL=127.0.0.1
PUERTO_KERNEL=8000
```

Módulo: Kernel

El módulo **Kernel,** dentro de nuestro trabajo práctico, será el encargado de gestionar la ejecución de los diferentes procesos que se ingresen al sistema mediante las consolas, planificando su ejecución en la CPU del sistema.

Lineamiento e Implementación

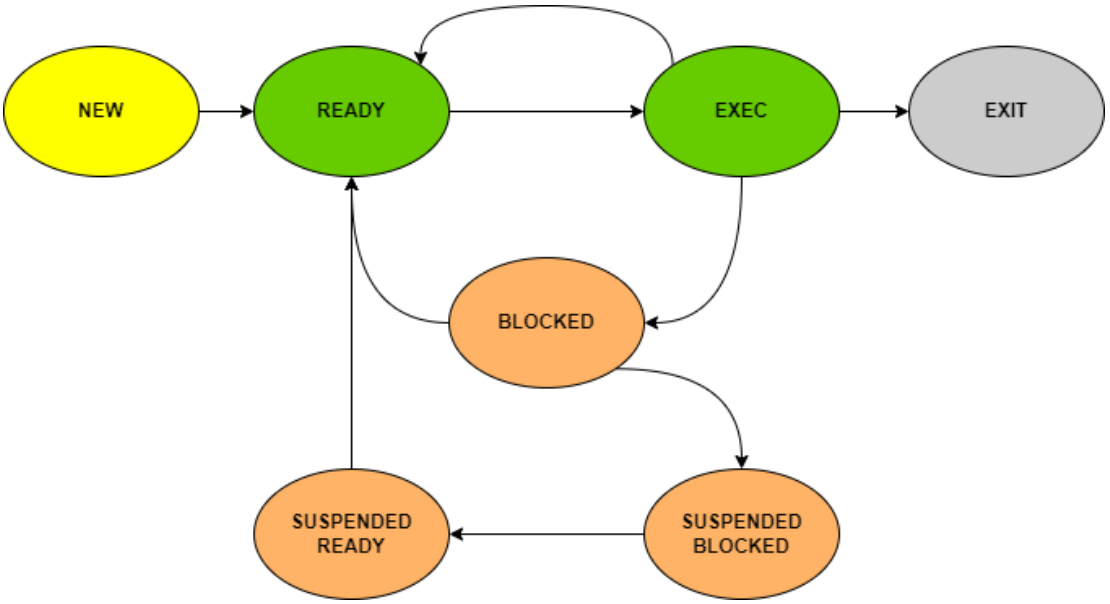
Al ser un módulo que deberá mantener conexiones con la CPU, la memoria y las diferentes consolas que conecten, deberá implementarse mediante una estrategia de múltiples hilos de ejecución.

Cualquier fallo en las comunicaciones deberá ser informado mediante un mensaje en el archivo de log y el sistema deberá continuar con su ejecución.

Este módulo gestionará la planificación de los procesos utilizando una estructura de control conocida por su sigla en inglés como **PCB** y los estados de este PCB pasarán por **tres planificadores distintos**. Uno de **Largo Plazo**, uno de **Mediano Plazo** y uno de **Corto Plazo**, los cuales se detallaran más adelante.

Diagrama de estados

El kernel utilizará el diagrama de 7 estados para la planificación de los procesos, teniendo en cuenta que se tendrá solamente una cola de procesos bloqueados, siendo sus operaciones atendidas en modo FIFO, y siguiendo los lineamientos de suspensión que se detallarán posteriormente en la explicación del algoritmo de mediano plazo.



PCB

El PCB será la estructura base que utilizaremos dentro del Kernel para administrar los procesos lanzados por medio de las consolas. El mismo deberá contener los datos definidos a continuación y se deberá enviar a la CPU a través de la conexión de *dispatch* al momento de poner a ejecutar un proceso.

- **Id**: identificador del proceso.
- **Tamaño**: tamaño en bytes del proceso, el mismo no cambiará a lo largo de la ejecución.
- **Instrucciones**: lista de instrucciones a ejecutar.
- **Program_counter**: número de la próxima instrucción a ejecutar.
- **Tabla_páginas**: tabla de páginas del proceso en memoria, esta información la tendremos recién cuando el proceso pase a estado **READY**.
- **Estimación_ráfaga**: Estimación utilizada para planificar los procesos en el algoritmo SRT, la misma tendrá un valor inicial definido por archivo de configuración y será recalculada bajo la fórmula de promedio ponderado vista en clases.

Planificador de Largo Plazo

Al conectarse una consola al kernel, deberá generarse la estructura PCB detallada anteriormente y asignarse este proceso al estado **NEW**.

En caso de que el grado de multiprogramación lo permita[2], los procesos pasarán al estado **READY**, enviando un mensaje al módulo Memoria para que inicialice sus estructuras necesarias y obtener el valor de la tabla de páginas que deberá estar almacenado en nuestro **PCB.**

Cuando se reciba un PCB con motivo de finalizar el mismo, se deberá pasar al proceso al estado **EXIT** y dar aviso al módulo Memoria para que éste libere sus estructuras. Una vez liberadas, se dará aviso a la Consola de la finalización del proceso.

Planificador de Corto Plazo

Los procesos que estén en estado **READY** serán planificados mediante los siguientes algoritmos:

- **FIFO**: First in First out
- **SRT**: Shortest Remaining Time (también visto como SJF con desalojo)

En el caso de los algoritmos con desalojo, cuando un proceso llegue a la cola READY se deberá enviar una Interrupción al proceso CPU a través de la conexión de *interrupt* para indicarle que deberá desalojar al proceso que se encuentra actualmente en ejecución.

[2] El grado de multiprogramación siempre será un valor que permita mantener todos los procesos cargados en memoria principal.

Al recibir el PCB del proceso en ejecución, se calcularán las estimaciones correspondientes, según indique el algoritmo, para seleccionar el siguiente a ejecutar[3].

Cabe aclarar que en todos los casos el PCB será recibido a través de la conexión de *dispatch*, quedando la conexión de *interrupt* dedicada solamente a enviar mensajes de interrupción.

Planificador de Mediano Plazo

Este planificador se encargará de gestionar las transiciones **(BLOCKED -> SUSPENDED-BLOCKED)** y **(SUSPENDED-READY -> READY)**

Para que un proceso entre en suspensión se debe cumplir que el mismo esté bloqueado por un tiempo mayor al límite definido por archivo de configuración[4].

Al suspenderse un proceso se enviará un mensaje a Memoria con la información necesaria y se esperará la confirmación del mismo.

Es importante tener en cuenta que un proceso en estado **SUSPENDED-READY** tendrá más prioridad para entrar a la cola **READY** que los procesos en **NEW**, cuando el grado de multiprogramación lo permita.

La transición (**SUSPENDED-BLOCKED** -> **SUSPENDED-READY)**, al ser una transición que va a darse al finalizar una entrada/salida, no necesariamente forma parte del planificador de Mediano Plazo.

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
IP_CPU	String	IP a la cual se deberá conectar con la CPU
PUERTO_CPU_DISPATCH	Numérico	Puerto al cual se deberá conectar con la CPU para mensajes de dispatch
PUERTO_CPU_INTERRUPT	Numérico	Puerto al cual se deberá conectar con la CPU para mensajes de interrupción

[3] Puede ser el mismo proceso que estaba ejecutando.
[4] Se recomienda investigar el funcionamiento de la función usleep().

PUERTO_ESCUCHA	Numérico	Puerto en el cual se escucharán las conexiones de los procesos
ALGORITMO_PLANIFICACION	String	Define el algoritmo de planificación de corto plazo. FIFO / SRT
ESTIMACION_INICIAL	Numérico	Estimación inicial para el cálculo de la primer ráfaga de CPU en milisegundos
ALFA	Numérico	Alfa para el cálculo de las rafagas de CPU
GRADO_MULTIPROGRAMACION	Numérico	Grado de multiprogramación del módulo
TIEMPO_MAXIMO_BLOQUEADO	Numérico	Tiempo máximo en milisegundos que permanecerá un proceso bloqueado antes de ser suspendido.

Ejemplo de Archivo de Configuración

```
IP_MEMORIA=127.0.0.1
PUERTO_MEMORIA=8002
IP_CPU=127.0.0.1
PUERTO_CPU_DISPATCH=8001
PUERTO_CPU_INTERRUPT=8005
PUERTO_ESCUCHA=8000
ALGORITMO_PLANIFICACION=FIFO
ESTIMACION_INICIAL=10000
ALFA=0.5
GRADO_MULTIPROGRAMACION=4
TIEMPO_MAXIMO_BLOQUEADO=10000
```

Módulo: CPU

El módulo **CPU** es el encargado de interpretar y ejecutar las instrucciones de los PCBs recibidos por parte del **Kernel**. Para ello, simulará un ciclo de instrucción tradicional (Fetch, Decode, Fetch operands, Execute y Check Interrupt).

A la hora de hacer las peticiones a **Memoria**, tendrá que traducir las *direcciones lógicas* (propias del proceso) a *direcciones físicas* (propias de la memoria). Para tal efecto, simulará las operaciones de una MMU disponiendo de una TLB.

Durante el transcurso de la ejecución de un proceso, se irá actualizando su **PCB,** que luego será devuelto al **Kernel** bajo los siguientes escenarios: finalización del mismo (instrucción **EXIT**), necesitar ser bloqueado (instrucción **I/O**), o deber ser desalojado (**interrupción**).

Lineamiento e Implementación

Al iniciarse el módulo, se conectará con la **Memoria** y realizará un *handshake* inicial para recibir la configuración relevante de la misma que le permita traducir las direcciones lógicas a físicas. Esto debería incluir al menos *cantidad de entradas por tabla de páginas* y *tamaño de página.*

Quedará a la espera de conexión a través del puerto *dispatch* por parte del **Kernel**, quien, una vez conectado, le enviará un PCB para ejecutar. Habiéndose recibido, se procederá a realizar el ciclo de instrucción tomando como punto de partida la instrucción que indique el Program Counter del PCB recibido.

Quedará a la espera también de conexión a través del puerto *interrupt* por parte del **Kernel** para recibir mensajes de **interrupción** en cualquier momento, por lo que estos módulos mantendrán dos conexiones en simultáneo.

Ciclo de Instrucción

Fetch

La primera etapa del ciclo consiste en buscar la próxima instrucción a ejecutar. En este trabajo práctico, las instrucciones estarán contenidas dentro de la estructura del PCB[5] a modo de lista. Teniendo esto en cuenta, utilizaremos el *Program Counter* (también llamado *Instruction Pointer*), que representa el número de instrucción a buscar, para buscar la próxima instrucción. Al finalizar el ciclo, este último deberá ser actualizado (sumarle 1).

[5] Esto no es lo que ocurriría en Sistemas Operativos actuales ni lo que se ve en clase, pero lo tomamos como una simplificación para la realización del trabajo en los tiempos del cuatrimestre y facilitar un desarrollo de tipo iterativo incremental.

Decode

Esta etapa consiste en interpretar qué instrucción es la que se va a ejecutar. Esto es importante para determinar si la próxima etapa (Fetch Operands) es necesaria. Siendo específicos, solamente la instrucción COPY tiene operandos que deben ser buscados en memoria antes de poder ejecutarla.

Fetch Operands

En continuación del párrafo anterior, el segundo parámetro de las instrucción COPY representa la dirección lógica del valor que queremos escribir, por lo que en esta etapa deberá buscarse dicho valor en memoria antes de seguir a la próxima etapa del ciclo de instrucción.

Execute

- **NO_OP**: No hacer nada, solamente esperar un tiempo determinado por archivo de configuración. El objetivo principal de esta instrucción es hacer que la ejecución sea más predecible para facilitar las pruebas de los algoritmos de planificación de corto plazo.
- **I/O**(*tiempo*): Esta instrucción representa una syscall de I/O bloqueante. Se deberá devolver el PCB actualizado al **Kernel** junto al tiempo de bloqueo en milisegundos.
- **READ**(*dirección_lógica*): Se deberá leer el valor de memoria correspondiente a esa dirección lógica e imprimirlo por pantalla.
- **WRITE**(*dirección_lógica*, *valor*): Se deberá escribir en memoria el valor del segundo parámetro en la dirección lógica del primer parámetro.
- **COPY**(*dirección_lógica_destino*, *dirección_lógica_origen*): Se deberá escribir en memoria el valor ubicado en la dirección lógica pasada como segundo parámetro, en la dirección lógica pasada como primer parámetro. A efectos de esta etapa, el accionar es similar a la instrucción WRITE ya que el valor a escribir ya se debería haber obtenido en la etapa anterior.
- **EXIT**: Esta instrucción representa la syscall de finalización del proceso. Se deberá devolver el PCB actualizado al Kernel para su finalización.

Cabe aclarar que **todos** los valores a leer/escribir en memoria serán numéricos enteros no signados de **4 bytes**, considerar el uso de *uint32_t*.

Check Interrupt

En este momento, se deberá chequear si el **Kernel** nos envió una *interrupción*. De ser el caso, se devuelve el PCB actualizado al Kernel. De lo contrario se reinicia el ciclo de instrucción..

Cabe aclarar que en todos los casos el PCB debe ser devuelto a través de la conexión de dispatch, quedando la conexión de interrupciones dedicada solamente a recibir mensajes de interrupción.

MMU

A la hora de **traducir direcciones lógicas a físicas**, la CPU debe tomar en cuenta que el esquema de memoria del sistema es de **Paginación de 2 niveles**. Por lo tanto, las direcciones lógicas se compondrán de la siguiente manera:

[entrada_tabla_1er_nivel | entrada_tabla_2do_nivel | desplazamiento]

Estas traducciones, en los ejercicios prácticos que se ven en clases y se toman en los parciales, normalmente se hacen en binario. Como en la realización del trabajo práctico es más cómodo utilizar números enteros en sistema decimal, y tomando en cuenta que las tablas tanto de 1er nivel como de 2do nivel tendrán la misma cantidad de entradas, la operatoria sería más parecida a la siguiente:

número_página = *floor*(dirección_lógica / tamaño_página)
entrada_tabla_1er_nivel = *floor*(**número_página** / cant_entradas_por_tabla)
entrada_tabla_2do_nivel = **número_página** *mod* (cant_entradas_por_tabla)
desplazamiento = dirección_lógica - **número_página** * tamaño_página

Como el único dato de la memoria que tenemos en el PCB es la tabla de páginas de 1er nivel del proceso, en total deberemos realizar 3 accesos a memoria para realizar la traducción:

1. Un primer acceso para conocer en qué tabla de páginas de 2do nivel está direccionado el marco en que se encuentra la página a la que queremos acceder
2. Un segundo acceso para conocer en qué marco está la misma
3. Finalmente acceder a la porción de memoria correspondiente (la dirección física).

TLB

Para agilizar el proceso de traducción, el módulo CPU contará también con una caché TLB. La misma deberá tener **obligatoriamente** la estructura [página | marco] y se definirá por archivo de configuración:

- La cantidad de entradas
- El algoritmo de reemplazo, que puede ser FIFO o LRU.

Archivo de configuración

Campo	Tipo	Descripción
ENTRADAS_TLB	Numérico	Cantidad de entradas de la TLB
REEMPLAZO_TLB	String	Algoritmo de reemplazo para las entradas de la TLB. FIFO o LRU.

RETARDO_NOOP	Numérico	Tiempo en milisegundos que se deberá esperar al ejecutar una instrucción NoOp
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_DISPATCH	Numérico	Puerto en el cual se escuchará la conexión del Kernel para mensajes de dispatch
PUERTO_ESCUCHA_INTERRUPT	Numérico	Puerto en el cual se escuchará la conexión del Kernel para mensajes de interrupciones

Ejemplo de Archivo de Configuración

```
ENTRADAS_TLB=4
REEMPLAZO_TLB=LRU
RETARDO_NOOP=1000
IP_MEMORIA=127.0.0.1
PUERTO_MEMORIA=8002
PUERTO_ESCUCHA_DISPATCH=8001
PUERTO_ESCUCHA_INTERRUPT=8005
```

Módulo: Memoria + SWAP

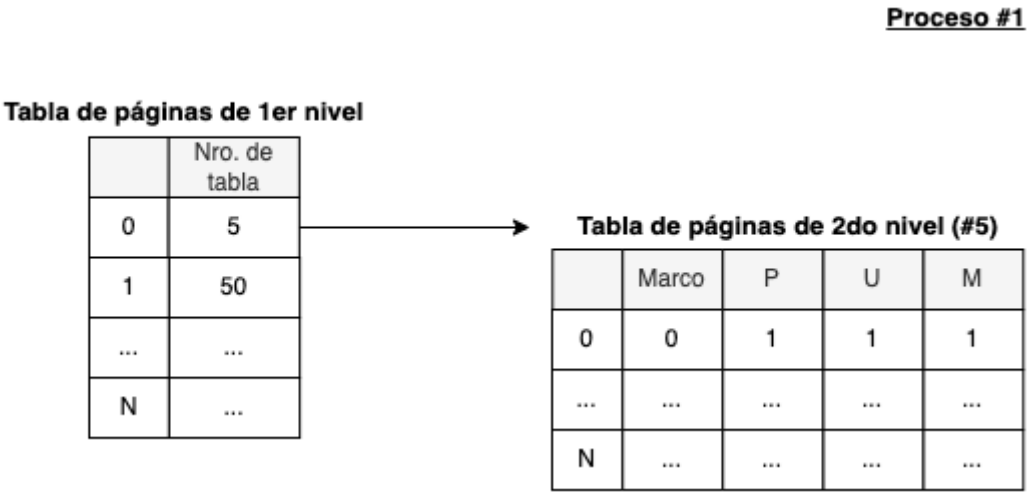
Este módulo será el encargado de responder los pedidos realizados por la CPU para leer y/o escribir en una tabla de páginas, o bien, una porción del espacio de usuario de la memoria.

A su vez, será el encargado de manejar un espacio de SWAP que se encontrará abstraído en un archivo para cada proceso, de forma tal que se le permite al mismo manejar un mayor nivel de direccionamiento.

Lineamiento e Implementación

Esquema de memoria

La asignación de memoria de este trabajo práctico utilizará un esquema de **paginación jerárquica de dos niveles**, donde las tablas de ambos niveles contarán con la misma cantidad de entradas, definida por archivo de configuración. La tabla de primer nivel estará compuesta porreferencias a las tablas de páginas de segundo nivel, como se muestra en el siguiente gráfico.



Además debe implementar **memoria virtual**, más específicamente, **paginación bajo demanda**. En este contexto, se definirá una asignación de cantidad de marcos fija definida por archivo de configuración y un scope de reemplazo local. El algoritmo de reemplazo será definido por archivo de configuración, pudiendo ser Clock o Clock Modificado.

Cabe aclarar que para la realización de este trabajo práctico se considera que, a diferencia de las páginas de los procesos, **sus tablas de páginas no necesitan ser swapeadas**. Este punto difiere a lo visto en las clases, donde normalmente las tablas tienen la misma estructura en todos sus niveles y las mismas pueden ser swapeadas.

Las tablas de páginas son estructuras administrativas que **no** deben guardarse en el espacio de memoria reservado para los procesos.

Estructuras

La memoria contará principalmente con 3 estructuras:

- Un espacio contiguo de memoria (representado por un void*). Este representará el espacio de usuario de la misma, donde los procesos podrán leer y/o escribir.
- Las tablas de páginas, que representarán el espacio de Kernel.
- Un archivo por proceso, que representarán el espacio de SWAP de cada uno.

Comunicación con Kernel y CPU

Inicialización del proceso

El módulo deberá crear las estructuras administrativas necesarias y enviar como *respuesta* el número de tabla de página de primer nivel de ese proceso.

Suspensión de proceso

Al ser suspendido un proceso, se debe liberar su espacio en memoria, escribiendo en SWAP solamente la información necesaria. Se debe tener en cuenta que, para la realización de este trabajo práctico, no se requiere el swapeo de tablas de páginas.

Finalización de proceso

Al ser finalizado un proceso, se debe liberar su espacio de memoria y eliminar su archivo de SWAP correspondiente. Se debe tener en cuenta que, para la realización de este trabajo, no es requerido eliminar las tablas de páginas del proceso[6].

Acceso a tabla de páginas

El módulo deberá responder lo siguiente:

- Si se trata de una tabla de primer nivel, el número de la tabla de segundo nivel.
- Si se trata de una tabla de segundo nivel, el número de marco correspondiente. Para este caso, si la página correspondiente no se encuentra en memoria (bit de presencia en 0) se deberá cargar la misma en algún marco para poder finalmente responderle a la CPU. Téngase en cuenta que esto podría incluir reemplazos de páginas y escrituras en swap.

[6] Esto difiere con lo que ocurriría normalmente en un sistema real, los invitamos a pensar en las implicancias que conlleva, así como en las ventajas que tendría poder swapear las tablas de páginas.

Acceso a espacio de usuario

El módulo deberá realizar lo siguiente:

- Ante un pedido de lectura, devolver el valor que se encuentra en la posición pedida
- Ante un pedido de escritura, escribir lo indicado en la posición pedida. En caso satisfactorio se responderá un mensaje de 'OK' o, de lo contrario, un mensaje de error.

Manejo de SWAP

Como se mencionó anteriormente el submódulo de SWAP va a ser el encargado dentro de la memoria de manejar las peticiones de lectura y escritura en SWAP.

Los archivos de SWAP se van a ir generando a medida que se carguen los diferentes procesos en la memoria y cuyo nombre va a constar del **PID del proceso** y una extensión la cual vamos a llamar *.swap*, quedando de esta forma por ejemplo los archivos *1.swap, 2.swap, 3.swap* correspondiendo a los procesos con PID 1, 2 y 3. Al finalizar un proceso, se deberá eliminar su archivo de swap asignado.

El SWAP, al imitar los accesos a disco, deberá implementarse como si fuera un único hilo de ejecución. Esto significa que **no se podrán tener 2 o más lecturas/escrituras en paralelo**. Y, para que la simulación tenga más sentido con la realidad y la diferencia de velocidades existente entre la memoria RAM y el disco, **cada acceso** a SWAP deberá tener una espera en milisegundos la cual estará definida por archivo de configuración a modo de retardo de swap.

Archivo de configuración

Campo	Tipo	Descripción
PUERTO_ESCUCHA	Numérico	Puerto en el cual se escuchará la conexión de módulo.
TAM_MEMORIA	Numérico	Tamaño expresado en bytes del tamaño de la memoria.
TAM_PAGINA	Numérico	Tamaño de las páginas en bytes.
ENTRADAS_POR_TABLA	Numérico	Cantidad de entradas de cada tabla de páginas.
RETARDO_MEMORIA	Numérico	Tiempo en milisegundos que se deberá esperar para dar una respuesta al CPU.

ALGORITMO_REEMPLAZO	String	Algoritmo de reemplazo de páginas (CLOCK/CLOCK-M).
MARCOS_POR_PROCESO	Numérico	Cantidad de marcos permitidos por proceso en asignación fija.
RETARDO_SWAP	Numérico	Tiempo en milisegundos que se deberá esperar para cada operación del SWAP (leer/escribir).
PATH_SWAP	String	Carpeta donde se van a encontrar los archivos de SWAP de cada proceso.

Ejemplo de Archivo de Configuración

```
PUERTO_ESCUCHA=8002
TAM_MEMORIA=4096
TAM_PAGINA=64
ENTRADAS_POR_TABLA=4
RETARDO_MEMORIA=1000
ALGORITMO_REEMPLAZO=CLOCK-M
MARCOS_POR_PROCESO=4
RETARDO_SWAP=2000
PATH_SWAP=/home/utnso/swap
```

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. No es obligatorio que se cumplan estos puntos necesariamente en el mismo orden.

Checkpoint 1: Conexión Inicial

Fecha: 23/04/2022

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 configuración y logs.
- Definir el Protocolo de Comunicación.
- Comenzar el desarrollo del módulo consola como cliente de kernel

Lectura recomendada:

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

Checkpoint 2: Avance del Grupo

Fecha: 14/05/2022

Objetivos:

- **Proceso Consola (completo):**
  - Genera la conexión con el proceso Kernel.
  - Obtiene las instrucciones del archivo de pseudocódigo pasado por path y le envía la información necesaria al Kernel.
- **Proceso Kernel:**
  - Crea las conexiones con la Memoria, la CPU y atención de las consolas.
  - Generar estructuras base para administrar los PCB con la información recibida por el módulo Consola.
- **Proceso CPU:**
  - Generar las estructuras de conexión con el proceso Kernel y Memoria.
  - Recibe PCB del módulo Kernel.
- **Proceso Memoria:**
  - Generar las estructuras de conexión con los procesos kernel y CPU.

Lectura recomendada:

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

Checkpoint 3: Obligatorio

Fecha: 04/06/2022 - Forma de entrega a definir según situación sanitaria.

Objetivos:

- **Proceso Kernel (completo):**
  - Planificador de largo plazo funcionando tomando en cuenta el grado de multiprogramación.
  - Planificador de mediano plazo funcionando sin funcionalidad en la Memoria.
  - Planificador de corto plazo incluyendo FIFO y SRT.
- **Proceso CPU:**
  - Implementa el ciclo de instrucción completo.
  - Ejecuta instrucciones NoOp, IO y Exit.
- **Proceso Memoria:**
  - Responde de manera genérica a los mensajes de Kernel y CPU.

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)

Checkpoint 4: Avance del Grupo

Fechas: 25/06/2022

Objetivos:

- **Proceso CPU (completo):**
  - Implementa TLB
  - Ejecuta instrucciones Read, Write y Copy.
- **Proceso Memoria:**
  - Implementa tablas de páginas de 2 niveles.
  - Responde los mensajes de CPU con datos.
  - Implementa lógica de swap para suspensión de procesos.

Lectura recomendada:

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

Checkpoint 5: Entregas Finales

Fechas y lugar: 16/07/2022 - 23/07/2022 - 30/07/2022 (Forma de entrega a definir según situación sanitaria)

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.

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