GECK

Gran Ejemplo de Creación de Kernels

No será el jardín del Edén pero es un buen Kernel.

Cátedra de Sistemas Operativos

Trabajo práctico Cuatrimestral

-2C2022 -
Versión 1.1

Índice

Historial de Cambios

v1.0 (04/09/2022) Publicación del enunciado

v1.1 (24/09/2022) Actualización de la definición de IO para Teclado y Pantalla y ajustes de textos que no estaban claros.

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: 82 días
- Fecha de comienzo: 05/09/2022
- Fecha de primera entrega: 26/11/2022
- Fecha de segunda entrega: 03/12/2022
- Fecha de tercera entrega: 17/12/2022
- Lugar de corrección: Laboratorio de Sistemas - Medrano.

Evaluación del Trabajo Práctico

El trabajo práctico consta de una evaluación en 2 etapas.

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

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

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

La defensa del trabajo práctico (o coloquio) consta de la relación de lo visto durante la teoría con lo implementado. De esta manera, una implementación que contradiga lo visto en clase o lo escrito en

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

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.

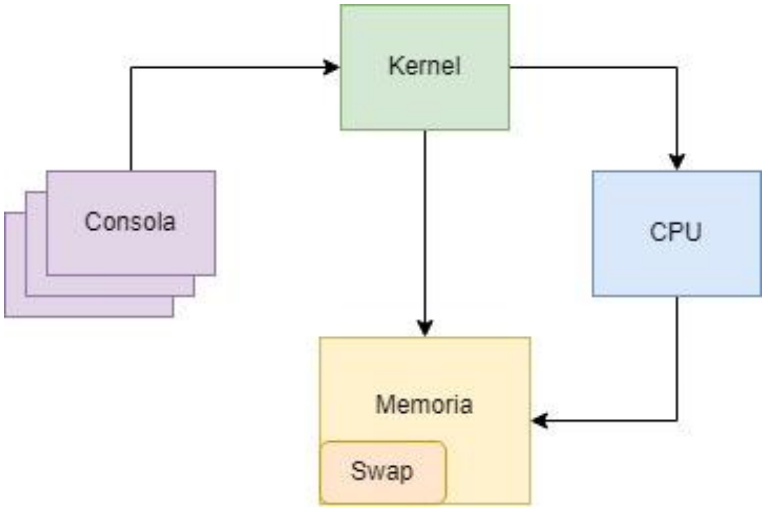
¿Qué es el trabajo práctico y cómo empezamos?

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** (múltiples instancias), **Kernel**, **CPU** y **Memoria** (1 instancia cada uno).
El proceso de ejecución del mismo consiste en crear procesos a través 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.

La **Memoria** implementará un esquema de segmentación paginada con memoria virtual (paginación bajo demanda) por lo que el módulo memoria también contará con un submódulo 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: **35%**
- CPU: **30%**
- Memoria+SWAP: **25%**

Dado que se contempla que los conocimientos se adquieran a lo largo de la cursada, se recomienda que el trabajo práctico se realice siguiendo un esquema iterativo incremental, por lo que por ejemplo la memoria no necesariamente tendrá avances hasta pasado el primer parcial.

Aclaración Importante

*Será condición necesaria de aprobación demostrar conocimiento teórico y de trabajo en alguno de los módulos principales (**Kernel, CPU** 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 directa.

Cada módulo contará con un listado de **logs mínimos y obligatorios**, pudiendo ser extendidos por necesidad del grupo en un archivo aparte.

De no cumplir con los logs mínimos, el trabajo práctico *no se considera apto para ser evaluado* y por consecuencia *desaprobado*.

Módulo: Consola

El módulo consola, será el punto de partida de los diferentes procesos que se crearán dentro de esta simulación de un kernel. Cada instancia de dicho módulo ejecutará un solo proceso.

Para poder iniciar una consola, la misma deberá poder recibir por parámetro los siguientes datos:

- Archivo de configuración
- Archivo de pseudocódigo con las instrucciones a ejecutar.

Lineamiento e Implementación

El módulo **consola** deberá recibir la ruta (path) de los 2 archivos mencionados anteriormente (configuración y pseudocódigo)[2].

Al iniciar el módulo leerá ambos archivos (configuración y pseudocódigo) y parseará cada línea terminada en `"\n"` del archivo de pseudocódigo como una instrucción para generar el listado de instrucciones.

Luego se conectará al kernel y enviará la información correspondiente al listado de instrucciones y los tamaños de los segmentos. Una vez recibida la confirmación de recepción, la consola quedará a la espera de nuevos mensajes del Kernel los cuales pueden ser:

- Imprimir un valor por pantalla, con un tiempo de retardo definido por archivo de configuración antes de responder al Kernel.
- Solicitar que se ingrese un valor por teclado.

Ambas operaciones recibirán únicamente valores **numéricos** y una vez finalizadas se deberá informar al Kernel para que éste desbloquee el proceso.

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 “INSTRUCCIÓN parámetros”. Los parámetros serán siempre valores separados por espacios. Según la instrucción, cada línea puede tener entre 0 o 2 parámetros.

Dicho esto, las posibles instrucciones a parsear serán los siguientes:

- **SET, ADD, MOV_IN, MOV_OUT, I/O**: 2 parámetros
- **EXIT**: 0 parámetros

[2] Se recomienda utilizar los parámetros `argc` y `argv` de la función `main` ayudándose con esta guía.

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

```
SET AX 1
ADD AX BX
MOV_IN CX 4
MOV_OUT 8 DX
I/O DISCO 10
I/O TECLADO AX
I/O PANTALLA BX
EXIT
```

Archivo de configuración

Campo	Tipo	Descripción
IP_KERNEL	String	IP del Kernel al cual debe conectarse
PUERTO_KERNEL	Numérico	Puerto del Kernel al cual debe conectarse
SEGMENTOS	Lista	Lista ordenada de los tamaños de los segmentos de Datos del proceso en los cuales podrá leer o escribir.
TIEMPO_PANTALLA	Numérico	Tiempo de retardo en milisegundos luego de imprimir el valor recibido desde Kernel.

Ejemplo de Archivo de Configuración

```
IP_KERNEL=192.168.1.1
PUERTO_KERNEL=8000
SEGMENTOS=[64, 256, 128, 32]
TIEMPO_PANTALLA=2000
```

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

Este módulo deberá mantener dos conexiones activas con la CPU, una de ellas se denominará "dispatch" y la otra "interrupt". También mantendrá una conexión activa con la memoria y las diferentes consolas que se conecten. Para ello, deberá implementarse mediante una estrategia de múltiples hilos de ejecución.

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

Diagrama de estados

El kernel utilizará el diagrama de 5 estados para ordenar la planificación de los procesos. Dentro del estado BLOCK, se tendrán **múltiples colas.**

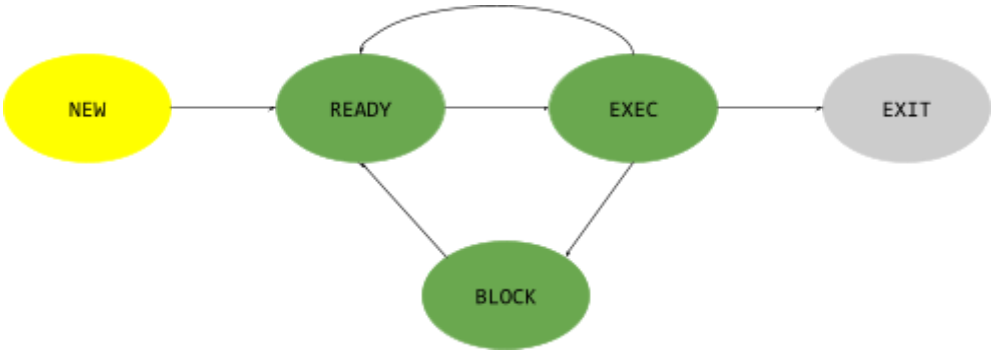
Las 2 primeras serán fijas y asociadas a cada consola que haya ingresado un proceso al sistema, ya que deberán enviar un mensaje a la misma y dejar el proceso bloqueado hasta recibir su respuesta:

- **Pantalla**: se deberá leer el registro que el módulo CPU nos informó, enviar su valor al módulo Consola para que lo imprima y esperar a que responda.
- **Teclado**: se deberá recibir un valor desde el módulo Consola y guardarlo en el registro que el módulo CPU nos informó.

Aparte se tendrá una lista de Dispositivos de Entrada-Salida, los cuales vendrán por archivo de configuración y tendrán un nombre y una duración por unidad de trabajo. La cantidad de unidades de trabajo será lo que deberá informarnos la CPU al momento de devolver un PCB por una instrucción de I/O[3]. En estos dispositivos los procesos se encolan a medida que llegan las peticiones de I/O.

Nota: *Se tendrán la misma cantidad de elementos en la lista de dispositivos que de duración por unidad de trabajo.*

[3] Por ejemplo, la instrucción "I/O DISCO 10" con una duración por unidad de trabajo de 2 segundos tomará 20 segundos en total.



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 como mínimo los datos definidos a continuación que representan el *Contexto de Ejecución* del proceso y se deberá enviar a la CPU a través de la conexión de *dispatch* al momento de poner a ejecutar un proceso, pudiéndose extender esta estructura con más datos que requiera el grupo.

- **Id**: identificador del proceso.
- **Instrucciones**: lista de instrucciones a ejecutar.
- **Program_counter**: número de la próxima instrucción a ejecutar.
- **Registros de la CPU**: Estructura que contendrá los valores de los *registros de uso general* de la CPU.
- **Tabla de Segmentos**: Contendrá los tamaños de los segmentos de datos del proceso y el número o identificador de tabla de páginas asociado a cada uno.

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 máximo de multiprogramación lo permita[4], los procesos pasarán al estado **READY**, enviando un mensaje al módulo Memoria para que inicialice sus estructuras necesarias y obtenga el índice/identificador de la tabla de páginas de cada segmento que deberán estar almacenados en la **tabla de segmentos del PCB**. La salida de **NEW** será mediante el algoritmo **FIFO**.

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.

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

Planificador de Corto Plazo

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

- **FIFO**: First in First out.
- **RR**: Round Robin.
- **Feedback (Cola Multinivel Retroalimentada)**.

En el caso de los algoritmos con desalojo (Round Robin), cuando se cumpla el quantum definido por archivo de configuración se deberá enviar un mensaje de interrupción al módulo CPU a través de la conexión de *interrupt* para indicarle que deberá desalojar al proceso que se encuentra actualmente en ejecución.

Al recibir el PCB del proceso en ejecución, se seleccionará el siguiente proceso a ejecutar según indique el algoritmo[5]. Durante este periodo la CPU se quedará esperando el nuevo PCB.

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

Para el cálculo de los tiempos de Round Robin se deberá crear un hilo específico que espere el tiempo definido por archivo de configuración (en milisegundos) y pasado este lapso, envíe la interrupción a la CPU.

Implementación del Algoritmo Feedback o Colas Multinivel Retroalimentadas

El algoritmo de Feedback consta de 2 colas de ready las cuales (ordenadas por prioridad) son:

1. *Round Robin* con el mismo quantum definido por configuración.
2. *FIFO*

Todos los procesos nuevos ingresan a la cola 1 (Round Robin).

Por fin de quantum son movidos a la cola 2 (FIFO).

Al desbloquearse vuelven a la cola 1 (Round Robin).

No hay desalojo entre colas, es decir, si llega un proceso a la cola 1 no debe desalojar a un proceso de la cola 2 que se encuentre ejecutando.

Page Fault

En caso de que el módulo CPU devuelva un PCB con motivo de Page Fault, se deberá crear un hilo específico para atender esta petición.

La resolución del Page Fault consta de los siguientes pasos:

[5] Puede ser el mismo proceso que estaba ejecutando.

1.- Mover al proceso al estado Bloqueado. Este estado bloqueado será independiente de todos los demás ya que solo afecta al proceso y no compromete recursos compartidos.

2.- Solicitar al módulo memoria que se cargue en memoria principal la página correspondiente, la misma será obtenida desde el mensaje recibido de la CPU.

3.- Esperar la respuesta del módulo memoria.

4.- Al recibir la respuesta del módulo memoria, desbloquear el proceso y colocarlo en la cola de ready correspondiente.

Logs mínimos y obligatorios

Creación de Proceso: "`Se crea el proceso <PID> en NEW`"

Cambio de Estado: "`PID: <PID> - Estado Anterior: <ESTADO_ANTERIOR> - Estado Actual: <ESTADO_ACTUAL>`"

Motivo de Bloqueo: "`PID: <PID> - Bloqueado por: <DISPOSITIVO>`"

Fin de Quantum: "`PID: <PID> - Desalojado por fin de Quantum`"

Ingreso a Ready: "`Cola Ready <ALGORITMO>: [<LISTA DE PIDS>]`". En caso de Feedback loguear ambas colas de Ready.

Page Fault: "`Page Fault PID: <PID> - Segmento: <Segmento> - Pagina: <Página>`"

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

Campo	Tipo	Descripción
PUERTO_CPU_INTERRUPT	Numérico	Puerto al cual se deberá conectar con la CPU para mensajes de interrupción
PUERTO_ESCUCHA	Numérico	Puerto en el cual se escucharán las conexiones de los módulos Consola
ALGORITMO_PLANIFICACION	String	Define el algoritmo de planificación de corto plazo. FIFO / RR / FEEDBACK
GRADO_MAX_MULTIPROGRAMACION	Numérico	Grado máximo de multiprogramación del módulo
DISPOSITIVOS_IO	Lista	Lista ordenada de los dispositivos de Entrada Salida
TIEMPOS_IO	Lista	Lista ordenada de los tiempos en milisegundos por unidad de trabajo de los dispositivos de Entrada Salida
QUANTUM_RR	Numérico	Tiempo en milisegundos que se debe esperar antes de desalojar a un proceso que se encuentra en EXEC.

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=8002
PUERTO_ESCUCHA=8000
ALGORITMO_PLANIFICACION=RR
GRADO_MAX_MULTIPROGRAMACION=4
DISPOSITIVOS_IO=[DISCO, IMPRESORA]
TIEMPOS_IO=[100, 50000]
QUANTUM_RR=2000
```

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 simplificado (Fetch[6], Decode, 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 **Contexto de Ejecución,** que luego será devuelto al **Kernel** bajo los siguientes escenarios: finalización del mismo (instrucción **EXIT o Error**), necesitar ser bloqueado (instrucción **I/O** o **Page Fault**), 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 la cantidad *de entradas por tabla de páginas* y *tamaño de página.*

Quedará a la espera de las conexiones por parte del **Kernel**, tanto por el puerto *dispatch* como por el puerto *interrupt*. Una vez conectado, el **Kernel** le enviará el **Contexto de Ejecución** para ejecutar. Habiéndose recibido, se procederá a realizar el ciclo de instrucción tomando como punto de partida la instrucción que indique el *Program Counter* recibido.

La CPU contará con una serie de **registros de propósito general**, los cuales tendrán los siguientes nombres: AX, BX, CX, DX. Estos registros almacenarán valores enteros no signados de 4 bytes (*uint32_t*). Estos valores deberán devolverse al Kernel como parte del **Contexto de Ejecución** del proceso.

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[7] a modo de lista. Teniendo esto en cuenta, utilizaremos el *Program Counter* (también llamado *Instruction Pointer*),

[6] A diferencia de la realidad donde el Fetch busca la instrucción en memoria, en esta simplificación, buscaremos las instrucciones dentro del PCB.

[7] 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.

que representa el número de instrucción a buscar, para buscar la próxima instrucción. Al finalizar el ciclo, este último deberá ser actualizado (sumarle 1).

Decode

Esta etapa consiste en interpretar qué instrucción es la que se va a ejecutar y si la misma requiere de un acceso a memoria o no.

En el caso de las instrucciones que no accedan a memoria, las mismas deberán esperar un tiempo definido por archivo de configuración (RETARDO_INSTRUCCION), a modo de simular el tiempo que transcurre en la CPU.

Las instrucciones de I/O y EXIT al representar syscalls, no tendrán retardo de instrucción.

Execute

En este paso se deberá ejecutar lo correspondiente a cada instrucción:

- **SET** (Registro, Valor)**:** Asigna al registro el valor pasado como parámetro.
- **ADD** (Registro Destino, Registro Origen): Suma ambos registros y deja el resultado en el Registro Destino.
- **MOV_IN** (Registro, Dirección Lógica): Lee el valor de memoria del segmento de Datos correspondiente a la Dirección Lógica y lo almacena en el Registro.
- **MOV_OUT** (Dirección Lógica, Registro)**:** Lee el valor del Registro y lo escribe en la dirección física de memoria del segmento de Datos obtenida a partir de la Dirección Lógica.
- **I/O** (Dispositivo, Registro / Unidades de trabajo): Esta instrucción representa una syscall de I/O bloqueante. Se deberá devolver el **Contexto de Ejecución** actualizado al **Kernel** junto el dispositivo y la cantidad de unidades de trabajo del dispositivo que desea utilizar el proceso (o el Registro a completar o leer en caso de que el dispositivo sea Pantalla o Teclado).
- **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 **Contexto de Ejecución** actualizado al Kernel. De lo contrario se reinicia el ciclo de instrucción.

Cabe aclarar que en todos los casos el **Contexto de Ejecución** 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 **Segmentación Paginada**. Por lo tanto, las direcciones lógicas se compondrán de la siguiente manera:

[N° Segmento | N° Página | desplazamiento página]

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 la operatoria sería más parecida a la siguiente:

tam_max_segmento = cant_entradas_por_tabla * tam_pagina
num_segmento = *floor*(dir_logica / **tam_max_segmento**)
desplazamiento_segmento = dir_logica *%* **tam_max_segmento**
num_pagina = floor(**desplazamiento_segmento** / tam_pagina)
desplazamiento_pagina = **desplazamiento_segmento** % tam_pagina

Como el único dato de la memoria que tenemos en el **Contexto de Ejecución** es el identificador de la tabla de páginas de cada segmento, en total deberemos realizar 2 accesos a memoria para realizar la traducción:

1. Un primer acceso para conocer en qué marco está la misma
2. Finalmente acceder a la porción de memoria correspondiente (la dirección física).

En caso de que el desplazamiento dentro del segmento (**desplazamiento_segmento**) sea mayor al tamaño del mismo, deberá devolverse el proceso al **Kernel** para que este lo finalice con motivo de **Error: Segmentation Fault (SIGSEGV)**.

Tabla de Segmentos

Esta estructura que se encuentra en el **Contexto de Ejecución**, nos permite saber los tamaños de los diferentes segmentos y sus tablas de páginas asociadas. La estructura que siguen es similar a la siguiente:

[N° Segmento | Tamaño Segmento | Número / Índice Tabla Páginas]

TLB

Para agilizar el proceso de traducción, el módulo CPU contará también con una caché TLB. La misma deberá tener **obligatoriamente** la estructura [pid | segmento | página | marco][8] y se definirá por archivo de configuración:

- La cantidad de entradas

[8] Solamente se permiten agregar campos que faciliten la implementación de los algoritmos de reemplazo como "instante de carga" o "instante de última referencia".

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

Esta caché deberá limpiar los registros asociados a los procesos que finalicen.

Page Fault

Durante la traducción de dirección lógica a física, al momento de solicitar el marco asociado a una página de un segmento, el módulo **Memoria**, puede devolvernos 2 resultados posibles:

- Número de marco: En este caso finalizamos la traducción, actualizamos la TLB y continuamos con la ejecución.
- Page Fault: En este caso deberemos devolver el **contexto de ejecución** al Kernel *sin actualizar el valor del program counter*. Deberemos indicarle al Kernel qué segmento y número de página fueron los que generaron el page fault para que éste resuelva el mismo.

Logs mínimos y obligatorios

Instrucción Ejecutada: "`PID: <PID> - Ejecutando: <INSTRUCCION> - <PARAMETRO_1> - <PARAMETRO_2>`"

TLB Hit: "`PID: <PID> - TLB HIT - Segmento: <NUMERO_SEGMENTO> - Pagina: <NUMERO_PAGINA>`"

TLB Miss: "`PID: <PID> - TLB MISS - Segmento: <NUMERO_SEGMENTO> - Pagina: <NUMERO_PAGINA>`"

Modificación de la TLB: Imprimir todas las entradas de la TLB ordenadas por número de entrada y siguiendo el formato:

`<Nro_Entrada>|PID:<PID>|SEGMENTO:<Segmento>|PAGINA:<Pagina>|MARCO:<Marco>`

Acceso Memoria: "`PID: <PID> - Acción: <LEER / ESCRIBIR> - Segmento: <NUMERO_SEGMENTO> - Pagina: <NUMERO_PAGINA> - Dirección Fisica: <DIRECCION_FISICA>`"

Page Fault: "`Page Fault PID: <PID> - Segmento: <Segmento> - Pagina: <Página>`"

Archivo de configuración

Campo	Tipo	Descripción
ENTRADAS_TLB	Numérico	Cantidad de entradas de la TLB

Campo	Tipo	Descripción
REEMPLAZO_TLB	String	Algoritmo de reemplazo para las entradas de la TLB. FIFO o LRU.
RETARDO_INSTRUCCION	Numérico	Tiempo en milisegundos que se deberá esperar al ejecutar las instrucciones SET y ADD.
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_INSTRUCCION=1000
IP_MEMORIA=127.0.0.1
PUERTO_MEMORIA=8002
PUERTO_ESCUCHA_DISPATCH=8001
PUERTO_ESCUCHA_INTERRUPT=8005
```

Módulo: Memoria

Este módulo será el encargado de responder los pedidos realizados por la CPU para leer y/o escribir en los segmentos de Datos del proceso en ejecución, referenciados por diversas tablas de páginas.

A su vez, será el encargado de manejar un espacio de SWAP que se encontrará abstraído en **un único archivo para todo el sistema**, de forma tal que se le permita al sistema 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 bajo demanda**, donde cada segmento creado contará con su propia tabla de páginas cuya cantidad de entradas será definida por archivo de configuración.

Marco	P	U	M	Pos en SWAP
5	1	1	1	2048
…				
5	0	0	0	4096
…				

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**.

Las tablas de páginas forman parte del espacio de Kernel del sistema, por lo 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 (segmentos de Datos del mismo).
- Las tablas de páginas, que representarán el espacio de kernel.
- Un *único* archivo para el sistema que representará el espacio de SWAP.

Comunicación con Kernel y CPU

Inicialización del proceso

El módulo deberá crear las estructuras administrativas necesarias y enviar como *respuesta* un identificador de las tablas de páginas de cada segmento.

Finalización de proceso

Al ser finalizado un proceso, se debe liberar su espacio de memoria y su espacio ocupado en SWAP. Se debe tener en cuenta que, para la realización de este trabajo, no es requerido eliminar las tablas de páginas del proceso[9].

Page Fault

El módulo deberá obtener la página asociada del espacio de SWAP y escribirla en la memoria principal.

En caso de que la memoria principal se encuentre llena o no se le puedan asignar más marcos al proceso, se deberá seleccionar una página víctima utilizando el algoritmo de reemplazo. Si la víctima se encuentra modificada, se deberá previamente escribir en SWAP.

Acceso a tabla de páginas

El módulo deberá responder el número de marco correspondiente, en caso de no encontrarse, se deberá retornar **Page Fault**.

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 y responder un mensaje de ‘OK’.

Para simular la diferencia de velocidades entre los accesos a TLB y Memoria, cada acceso a tablas o a espacio de usuario tendrá un tiempo de espera en milisegundos definido por archivo de configuración. Este tiempo **no** debe ser tenido en cuenta para las operaciones con Kernel (Creación, Finalización de proceso o Page Fault).

[9] 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.

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 el área de SWAP.

Esta área de SWAP deberá implementarse mediante un único archivo con un tamaño definido por archivo de configuración.[10]

Al iniciar la memoria, el archivo de SWAP se deberá generar desde 0 y redimensionarlo[11] al tamaño definido por config, en caso de que exista se deberá eliminar y crear nuevamente.

Todos los accesos a SWAP, dado que se simula el acceso a disco, deberán contar con un retardo definido por archivo de configuración. Este retardo aplica a todas las operaciones de lectura y escritura de páginas.

Logs mínimos y obligatorios

Creación / destrucción de Tabla de Páginas: "`PID: <PID> - Segmento: <SEGMENTO> - TAMAÑO: <CANTIDAD> paginas`"

Acceso a Tabla de Páginas: "`PID: <PID> - Página: <PÁGINA> - Marco: <MARCO>`"

Acceso a espacio de usuario: "`PID: <PID> - Acción: <LEER / ESCRIBIR> - Dirección física: <DIRECCIÓN_FÍSICA>`"

Reemplazo Página: "`REEMPLAZO - PID: <PID> - Marco: <MARCO> - Page Out: <SEGMENTO>|<PÁGINA> - Page In: <SEGMENTO>|<PAGINA>`"

Lectura de Página de SWAP: "`SWAP IN - PID: <PID> - Marco: <MARCO> - Page In: <SEGMENTO>|<PÁGINA>`"

Escritura de Página en SWAP: "`SWAP OUT - PID: <PID> - Marco: <MARCO> - Page Out: <SEGMENTO>|<PÁGINA>`"

Archivo de configuración

Campo	Tipo	Descripción
PUERTO_ESCUCHA	Numérico	Puerto en el cual se escuchará la conexión de módulo.

[10] Para la realización de este trabajo, no será tenido en cuenta el caso en el que este tamaño fijo sea menor a la suma del tamaño de todos los procesos en un determinado momento.
[11] Considerar el uso de la función truncate() o ftruncate()

Campo	Tipo	Descripción
TAM_MEMORIA	Numérico	Tamaño expresado en bytes del espacio de usuario 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.
TAMANIO_SWAP	Numérico	Tamaño del archivo de SWAP.

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.bin
TAMANIO_SWAP=10240
```

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: 17/09/2022

Objetivos:

- Familiarizarse con Linux y su consola, el entorno de desarrollo y el repositorio.
- Aprender a utilizar las Commons, principalmente las funciones para listas, archivos de configuración y logs.
- Definir el Protocolo de Comunicación.
- Comenzar el desarrollo de los módulos y sus conexiones.

Lectura recomendada:

- Tutoriales de “Cómo arrancar” de la materia: https://docs.utnso.com.ar/primeros-pasos
- SO Commons Library - https://github.com/sisoputnfrba/so-commons-library
- Git para el Trabajo Práctico - https://docs.utnso.com.ar/guias/consola/git
- Guía de Punteros en C - https://docs.utnso.com.ar/guias/programacion/punteros
- Guía de Sockets - https://docs.utnso.com.ar/guias/linux/sockets

Checkpoint 2: Avance del Grupo

Fecha: 01/10/2022

Objetivos:

- **Módulo Consola:**
  - Obtener el contenido de los archivos de config y de pseudocódigo.
  - Parsear el archivo de pseudocódigo y enviar la información al Kernel.
- **Módulo Kernel:**
  - Crea las conexiones con la Memoria, la CPU y atención de las consolas
  - Generar estructuras base para administrar los PCB y sus estados con la información recibida por el módulo Consola.
  - Planificador de Largo Plazo funcionando, respetando el grado de multiprogramación.
  - Planificador de Corto Plazo funcionando con algoritmo FIFO.
- **Módulo CPU:**
  - Generar las estructuras de conexión con el proceso Kernel y Memoria.
  - Envía y recibe el contexto de ejecución del módulo Kernel.
- **Módulo 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
- Guía de Buenas Prácticas de C - https://docs.utnso.com.ar/guias/programacion/buenas-practicas
- Guía de Serialización - https://docs.utnso.com.ar/guias/linux/serializacion
- Charla de Threads y Sincronización - https://docs.utnso.com.ar/guias/linux/threads

Checkpoint 3: Obligatorio - Presencial

Fecha: 05/11/2022

Objetivos:

- **Realizar pruebas mínimas en un entorno distribuido.**
- **Módulo Consola (completo):**
  - Atiende las peticiones del Kernel para imprimir por pantalla e ingresar input por teclado.
- **Módulo Kernel:**
  - Planificador de Largo Plazo funcionando, respetando el grado de multiprogramación.
  - Planificador de Corto plazo funcionando para los algoritmos RR y Colas Multinivel.
  - Manejo de estado de bloqueado sin Page Fault.
- **Módulo CPU:**
  - Implementa ciclo de instrucción completo.
  - Ejecuta instrucciones SET, ADD, IO y EXIT.
- **Módulo 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)
- Guía de Debugging - https://docs.utnso.com.ar/guias/herramientas/debugger
- Guía de Despliegue de TP - https://docs.utnso.com.ar/guias/herramientas/deploy
- Guía de uso de Bash - https://docs.utnso.com.ar/guias/consola/bash

Checkpoint 4: Avance del Grupo

Fechas: 19/11/2022

Objetivos:

- **Módulo Kernel (completo):**
  - Implementa circuito de Page Fault
- **Módulo CPU (completo):**
  - Implementa TLB
  - Implementa instrucciones MOV_IN, MOV_OUT
- **Módulo Memoria:**
  - Implementa tablas de Páginas
  - Responde los mensajes de CPU y Kernel con datos reales.
  - Respuesta Page Fault de manera genérica sin acciones.

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
- Tutorial de Valgrind - https://docs.utnso.com.ar/guias/herramientas/valgrind

Checkpoint 5: Entregas Finales

Fechas: 26/11/2022 - 03/12/2022 - 17/12/2022

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:

- Guía de Despliegue de TP - https://docs.utnso.com.ar/guias/herramientas/deploy
- Guía de uso de Bash - https://docs.utnso.com.ar/guias/consola/bash