Table of Contents

Introducción

El trabajo práctico consiste en simular ciertos aspectos de un sistema multiprocesador con la capacidad de interpretar la ejecución de scripts escritos en un lenguaje creado para esta ocasión. Este sistema planificará y ejecutará estos scripts (en adelante “Programas”) controlando sus solicitudes de memoria y administrando los accesos a recursos, como los dispositivos de entrada/salida y los semáforos compartidos.

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 (API) que brindan los sistemas operativos modernos
- Entienda aspectos del diseño de un sistema operativo moderno
- Afirme diversos conceptos teóricos de sistemas operativos 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

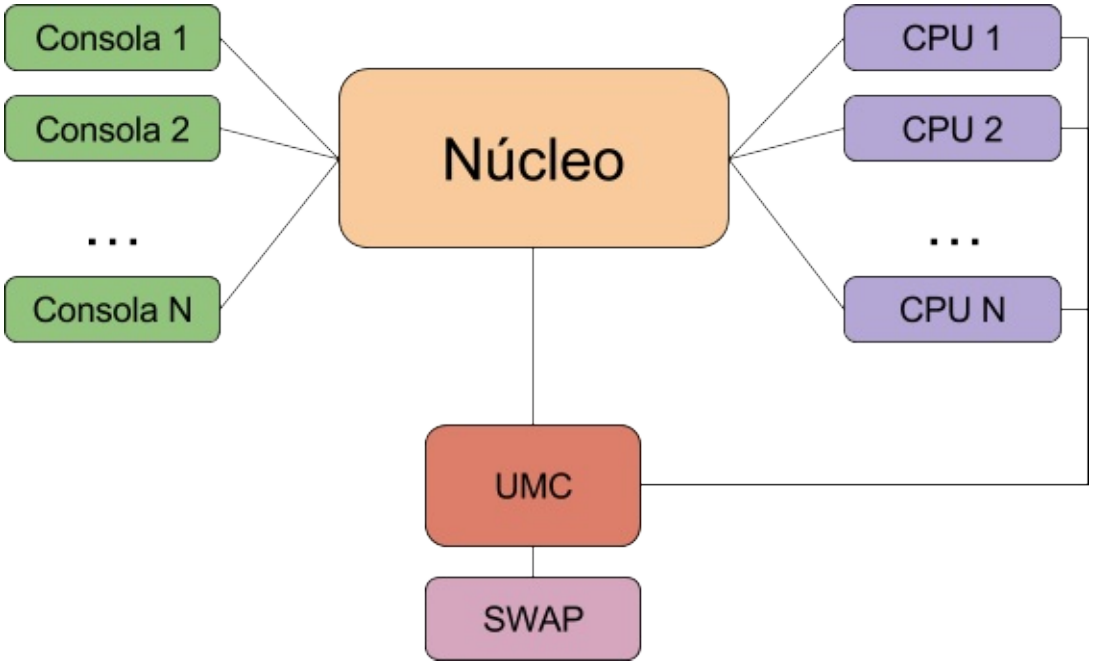
Características

- Modalidad: grupal (5 integrantes) y obligatorio
- Tiempo estimado para su desarrollo: 11 semanas
- Fecha de comienzo: sábado 16-Abril
- Fecha de entrega: sábado 2-Julio
- Fecha de primer recuperatorio: sábado 16-Julio
- Fecha de segundo recuperatorio: sábado 30-Julio
- Lugar de corrección: Laboratorio de Medrano

Aclaraciones

Debido al fin académico del trabajo práctico, los conceptos reflejados son, en general, versiones simplificadas o alteradas de los componentes reales de hardware y de sistemas operativos modernos, a fin de resaltar algún aspecto de diseño. En algunos casos los aspectos no fueron tomados de manera literal, por lo que invitamos a los alumnos a leer las notas y comentarios al respecto que haya en el enunciado, como así también a reflexionar y discutir con sus compañeros, ayudantes y docentes al respecto.

Arquitectura del sistema



Proceso Consola

El Proceso Consola es un proceso simple que permite enviar a ejecutar Programas AnSISOP al sistema distribuido, y funcionar como interfaz del mismo para recibir los resultados de su ejecución o los mensajes que el Programa AnSISOP necesite imprimir por pantalla.

Arquitectura del Proceso Consola

La Consola es un intérprete de scripts AnSISOP, cuya única responsabilidad es enviar el código del programa al Núcleo y mostrar por pantalla los mensajes que el Núcleo le indique imprimir.

Al iniciar, leerá su archivo de configuración, se conectará mediante sockets[1] al Proceso Núcleo y, luego de un intercambio de mensajes inicial (handshake) , enviará el código del Programa AnSISOP al Núcleo. A partir de ese momento, el proceso quedará a la espera de mensajes del Núcleo correspondientes a las sentencias `imprimir` e `imprimirTexto`, con los valores que deberá mostrar en pantalla.

- Existirá una instancia de este proceso por cada Programa AnSISOP a ejecutar en el sistema
- El Proceso Consola deberá poder ser utilizado como intérprete de scripts AnSISOP mediante el encabezado hashbang (`#!`)[2]
- La terminación del proceso Consola implica la finalización de la ejecución del Programa AnSISOP en el sistema y viceversa

[1] Siempre que en el enunciado se lea la palabra socket, se refiere a los sockets `STREAM` tipo `AF_INET`

[2] Ver http://mgarciaisaia.github.io/tutorial-c/blog/2014/03/20/she-bangs-she-bangs/

Proceso Núcleo

El proceso Núcleo es el proceso principal del sistema. Recibirá los Programas AnSISOP y planificará su ejecución en los distintos Procesos CPUs del sistema según el algoritmo Round Robin. Además, será el encargado de resolver las llamadas a sistema de los Programas.

Arquitectura del Proceso Núcleo

El proceso Núcleo al ser iniciado se conectará con el Proceso Unidad de Memoria Central (UMC), obtendrá el tamaño de página, y quedará a la espera de conexiones por parte de Procesos CPU o Procesos Consola.

Al contar con al menos un Proceso CPU comenzará a planificar los diversos Programas AnSISOP en función del algoritmo de planificación.

- En cualquier momento, instancias de los procesos CPU y procesos Consola pueden ingresar o desconectarse del sistema.
- La falta de procesos CPU en el sistema es posible, en ese caso los Programas AnSISOP quedarán a la espera.

Creación del PCB

Al recibir la conexión de un nuevo Programa, el Núcleo intercambiará unos mensajes iniciales con el mismo, para luego recibir la totalidad del código fuente del script que se deberá ejecutar.

El Núcleo creará la estructura PCB con al menos los siguientes campos[3]:

- Un identificador único (PID)
- Program Counter (PC)
- Posición del Stack (SP)
- Índice de Stack

A partir de esta información, el Núcleo deberá solicitarle a la UMC lás páginas necesarias para almacenar el código del programa y el stack. Para simular el comportamiento de un sistema real, también le enviará el código completo del Programa, que la UMC deberá

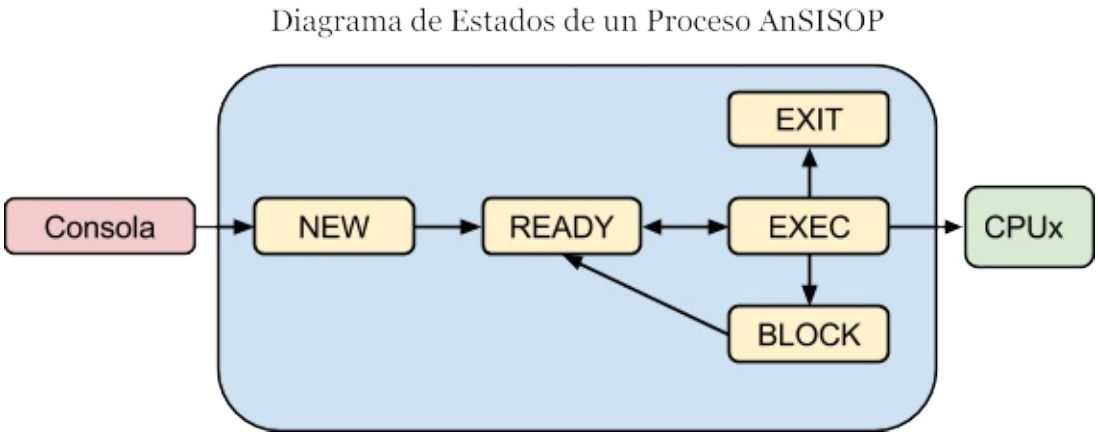
almacenar página por página en el Swap. **La UMC no almacenará las páginas de código inicialmente**: sólo las almacenará el Swap, y se cargarán en UMC cuando ocurran los fallos de página correspondientes intentando leer el código.

Si no se pudiera obtener espacio suficiente para algunas de las estructuras necesarias del proceso, entonces se le rechazará el acceso al sistema, informándose oportunamente en el proceso Consola correspondiente a ese Programa.

Planificación de procesos

Los PCB de los Programas AnSISOP serán enviados a los diversos Procesos CPU para su ejecución en el orden determinado por Round Robin, siguiendo un modelo de 5 estados[4].

- Recibirá los nuevos PCBs (`NEW`) y los encolará en el estado de listos (`READY`)
- Enviará a ejecutar los Programas (`EXEC`) en cola de listos (`READY`) a los CPU disponibles
- Moverá a la cola de listos (`READY`) todos los procesos que hayan terminado su ráfaga de CPU o concluido su entrada/salida
- Moverá a la cola de salida (`EXIT`) aquellos procesos que hayan concluido su ejecución
- Finalizará el ciclo de vida de los procesos de la cola de salida (`EXIT`) enviando el correspondiente mensaje de finalización a los procesos consolas asociados y destruyendo los PCBs
- Recibirá de cada CPU la información actual de cada PCB en ejecución y enviará a cada CPU libre al primer elemento de la cola de listos (`READY`)
- Si un proceso debiera ser bloqueado (ver razones), este pasará a la cola de bloqueados (`BLOCK`); al desbloquearse, volverá a la cola de listos (`READY`)



Adicionalmente, cualquier programa deberá poder finalizar su ejecución si se lo requiere desde su proceso Consola. En el caso en el que el programa esté ejecutando, se esperará hasta que la ráfaga de ejecución termine correctamente. Si se encuentra en cualquier otro estado, deberá ser capaz de liberar sus recursos y frenar su ejecución.

Operaciones privilegiadas

El lenguaje AnSISOP es lo suficientemente potente para permitir que los Programas trabajen de manera colaborativa para resolver una tarea específica.

En particular, permite que el Programa utilice variables compartidas al sistema, cuyo valor es accesible por cualquier Programa en ejecución. Para evitar situaciones de condiciones de carrera brinda también al programador la posibilidad de utilizar semáforos para sincronizar la ejecución de ciertas porciones de sus Programas. Por último, gestionará los pedidos de ejecución de operaciones de entrada/salida, administrando las colas de espera de cada dispositivo.

Las Operaciones Privilegiadas, por necesitar recursos que exceden al Proceso CPU, se resolverán en el Núcleo, por lo que serán enviadas por el Proceso CPU mediante el socket de comunicación que lo une con el Núcleo. Este recibirá la solicitud y responderá en consecuencia.

Existen cinco Operaciones Privilegiadas:

- `obtener_valor [identificador de variable compartida]`
- `grabar_valor [identificador de variable compartida] [valor a grabar]`
- `wait [identificador de semáforo]`
- `signal [identificador de semáforo]`
- `entrada_salida [identificador de dispositivo] [unidades de tiempo a utilizar]`

En el código AnSISOP, los identificadores de las variables compartidas, para diferenciarlas de las variables normales, comenzarán con el caracter signo de admiración (!), seguido del identificador de una cadena de texto, por ejemplo: `!mivariable`, `!var123`. Las variables compartidas existentes en el sistema son definidas por archivo de configuración y automáticamente inicializadas en cero.

Los semáforos estarán definidos por archivo de configuración con un identificador alfanumérico y un valor inicial, por ejemplo: `Semaforo1` o `cantidadImpresoras`. Se crearán al iniciar el proceso Núcleo, y se considera una excepción abortiva intentar acceder a una variable compartida o semáforo inexistente, por lo que no está dentro del alcance de la evaluación.

Dispositivos de entrada/salida y colas de espera

Por archivo de configuración se definirá qué dispositivos de entrada/salida habrá presentes en el sistema. Cada uno de estos dispositivos tendrá un identificador y un valor de retardo en milisegundos por unidad de ejecución.

Como se observa en el Anexo, desde el lenguaje AnSISOP se puede solicitar que un programa realice entrada/salida en un dispositivo, indicando su identificador y la cantidad de unidades a ejecutar, por ejemplo: `io Disco 10`.

El primer parámetro (`Disco`) corresponde al identificador del dispositivo de entrada/salida a utilizar. Los dispositivos del sistema son independientes entre sí, y cada uno puede ser utilizado por una única solicitud de entrada/salida a la vez, por lo que el programa de ejemplo deberá esperar a que terminen todas las peticiones previas de entrada/salida sobre el `Disco` antes de ejecutar la suya, pero podrá ejecutarse en paralelo con una petición a la `Impresora`.

El segundo parámetro (`10`) es la cantidad de operaciones de entrada/salida que deben realizarse. A efectos del Trabajo Práctico, “realizar una entrada/salida” sólo significará esperar la cantidad de tiempo indicada por la cantidad de operaciones a realizar y el tiempo por unidad del dispositivo correspondiente. Por ejemplo, si el `Disco` tiene configurado un retardo de 5ms, para completar la operación de 10 unidades se deberá esperar[5] 50ms, durante los que el dispositivo no podrá realizar ninguna otra operación de entrada/salida.

Archivo de Configuración

Al iniciar, el Proceso Núcleo deberá leer los siguientes parámetros de un archivo de configuración, cuya ruta se indicará como argumento de la línea de comandos del programa.

Parámetro	Tipo de dato	Descripción
`PUERTO_PROG`	[numérico]	Puerto TCP utilizado para recibir las conexiones de los Programas
`PUERTO_CPU`	[numérico]	Puerto TCP utilizado para recibir las conexiones de los CPUs
`QUANTUM`	[numérico]	Valor del Quantum (en instrucciones a ejecutar) del algoritmo Round Robin. Este valor puede modificarse en tiempo de ejecución[6]
`QUANTUM_SLEEP`	[numérico]	Valor de retardo en milisegundos que el CPU deberá esperar luego de ejecutar cada sentencia. Este valor puede modificarse en tiempo de ejecución
`SEM_IDS`	[array: alfanumérico]	Identificador de cada semáforo del sistema. Cada posición del array representa un semáforo
`SEM_INIT`	[array: numérico]	Valor inicial de cada semáforo definido en `SEM_IDS` , según su posición
`IO_IDS`	[array: alfanumérico]	Identificador de cada dispositivo de entrada/salida
`IO_SLEEP`	[array: numérico]	Retardo en milisegundos de cada unidad de operación de entrada/salida de cada dispositivo definido en IO_IDS, según su posición
`SHARED_VARS`	[array: alfanumérico]	Identificador de cada variable compartida
`STACK_SIZE`	[numérico]	Tamaño en páginas del Stack

Ejemplo de Archivo de Configuración

```
PUERTO_PROG=5000
PUERTO_CPU=5001
QUANTUM=3
QUANTUM_SLEEP=500
IO_IDS=[Disco, Impresora, Scanner]
IO_SLEEP=[1000, 2000, 1000]
SEM_IDS=[SEM1, SEM2, SEM3]
SEM_INIT=[0, 0, 5]
SHARED_VARS=[!Global, !UnaVar, !tiempo3]
STACK_SIZE=2
```

Como se observa en el ejemplo el `SEM3` es inicializado con valor 5 y la `Impresora` tiene un retardo de 2000ms. Las variables compartidas `!Global` , `!Unavar` , y `!tiempo3` se inicializan en cero.

3

[3] Para más información, referirse al Anexo I - Bloque de Control del Programa

[4] Recuerde que el estado "suspendido" tiene sentido sólo cuando es posible suspender (enviar a swap) al proceso completo

[5] Investigar la llamada al sistema `usleep`

[6] Para tener un ejemplo de cómo detectar cambios en los archivos, ver https://github.com/sisoputnfrba/so-inotify-example

Proceso Unidad de Memoria Central (UMC)

El Proceso UMC[7] es el responsable en el sistema de brindar a los Programas espacio en memoria para que estos realicen sus operaciones. Simulará un mecanismo de paginación por demanda, utilizando una TLB y un espacio de swap. Para esto utilizará una serie de estructuras administrativas internas que deberá crear y mantener.

Controlará la cantidad de marcos otorgados a cada proceso, siendo este un valor configurable, con asignación fija y reemplazo local. La asignación de frames a un determinado Programa se hará efectiva sólo cuando este lo necesite, pudiendo nunca llegar al límite mencionado.

Arquitectura de la UMC

Al iniciar, solicitará **un bloque de memoria contigua**[8] de tamaño configurable por archivo de configuración, para simular la memoria principal del sistema. Luego creará las estructuras administrativas necesarias para poder gestionar dicho espacio, permitiendo a cada Programa AnSISOP en funcionamiento utilizar un conjunto de páginas de tamaño fijo.

A la UMC se conectarán, mediante sockets, el proceso Núcleo y los diversos CPUs. Por cada conexión, la UMC creará un hilo dedicado a atenderlo, que quedará a la espera de solicitudes de operaciones. La UMC deberá validar cada pedido recibido, y responder en consecuencia.

- Ante cualquier solicitud de acceso a la tabla de páginas o a memoria principal, el proceso UMC deberá esperar una cantidad de tiempo configurable -en milisegundos, e igual para ambos tipos de acceso-, simulando el tiempo de acceso a memoria.
- Es importante destacar la naturaleza multi-hilo del proceso, por lo que será parte del desarrollo del trabajo atacar los problemas de concurrencia que surgieran.
- Para simplificar el desarrollo de este proceso, a diferencia de la realidad, las estructuras administrativas no deben ser almacenadas en el mismo espacio de memoria utilizado por los Programas.

Cache TLB

La UMC 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 caché con una opción en el archivo de configuración. El algoritmo de reemplazo a utilizar aquí será LRU. Ante un cambio de proceso, se realizará una limpieza (`flush`) en las entradas que correspondan.

Operaciones de la UMC

El Proceso UMC, simulando aspectos de un controlador de memoria real, maneja una interfaz reducida, que no puede ser ampliada.

Inicializar programa

Parámetros: [Identificador del Programa] [Cantidad de Páginas totales requeridas] [Código del Programa]

Cuando el proceso Núcleo comunique el inicio de un nuevo Programa AnSISOP, se crearán las estructuras necesarias para administrarlo correctamente. Para ello, UMC recibirá de este último un Identificador de programa, la cantidad de páginas totales requeridas, y el código del Programa. Luego, deberá informar de esta situación al Proceso Swap, reservando la cantidad de páginas a usar, y escribiendo las páginas de código. Notar que las páginas recibidas por la UMC no serán cargadas en memoria principal hasta que sea requerido, respetando el principio de paginación bajo demanda.

Solicitar bytes de una página

Parámetros: [#página], [offset] y [tamaño]

Ante un pedido de lectura de página de alguno de los procesos CPU, se realizará la traducción a marco (frame) y se devolverá el contenido correspondiente. En caso de que la página no se encuentre en memoria principal, será solicitada al proceso Swap, corriendo luego el algoritmo correspondiente para cargarla en memoria principal.

Almacenar bytes en una página

Parámetros: [#página], [offset], [tamaño] y [buffer]

Ante un pedido de escritura de página de alguno de los procesadores, se realizará la traducción a marco (frame), y se actualizará su contenido.

En caso de que la página no se encuentre en memoria principal, será solicitada al proceso 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.

Finalizar programa

Parámetros: [Identificador del Programa]

Cuando el proceso Núcleo informe el fin de un Programa AnSISOP, se deberá eliminar las estructuras usadas para administrarlo. Además, deberá informar tal situación al proceso Swap, para que este libere el espacio utilizado en su partición.

Otras operaciones

También soporta algunas operaciones que simplifican el desarrollo del trabajo práctico y su operatoria.

- Handshake: [Tipo: Núcleo/CPU]
- Cambio de proceso activo: [Identificador del Programa]

Algoritmos de reemplazo

Cuando un Programa AnSISOP necesite acceder a una página, es posible que esta no se encuentre en memoria principal. Para solucionar esta situación, se deberán implementar los algoritmos Clock y Clock Modificado, siendo ambos configurables.

Consola

El Proceso UMC, al iniciar, quedará a la espera de comandos enviados por teclado permitiendo al menos las siguientes funcionalidades. El diseño de la misma y la sintaxis de los comandos queda a criterio del equipo.

- `retardo` : Este comando permitirá modificar la cantidad de milisegundos que debe esperar el proceso UMC antes de responder una solicitud. Este parámetro será de ayuda para evaluar el funcionamiento del sistema.
- `dump` : Este comando generará un reporte en pantalla y en un archivo en disco del estado actual de:
  - Estructuras de memoria: Tablas de páginas de todos los procesos o de un proceso en particular.
  - Contenido de memoria: Datos almacenados en la memoria de todos los procesos o de un proceso en particular.
- `flush`
  - `tlb` : Este comando deberá limpiar completamente el contenido de la TLB.
  - `memory` : Este comando marcará todas las páginas del proceso como modificadas.

Archivo de Configuración

Al iniciar, el Proceso UMC deberá leer los siguientes parámetros de un archivo de configuración, el cual podrá ser parametrizable como primer argumento del programa.

Parámetro	Tipo de dato	Descripción
`PUERTO`	[numérico]	Puerto TCP utilizado para recibir las conexiones del Núcleo y los CPUs
`IP_SWAP`	[ip]	Dirección IP del proceso Swap
`PUERTO_SWAP`	[numérico]	Puerto TCP donde se encuentra escuchando el proceso Swap
`MARCOS`	[numérico]	Cantidad de marcos disponibles en el sistema
`MARCO_SIZE`	[numérico]	Valor en bytes del tamaño de marco del sistema
`MARCO_X_PROC`	[numérico]	Cantidad máxima de marcos asignable a cada Programa AnSISOP
`ALGORITMO`	[alfanumérico]	Algoritmo de reemplazo de páginas
`ENTRADAS_TLB`	[numérico]	Cantidad disponible de entradas en la TLB. 0 = Deshabilitada
`RETARDO`	[numérico]	Cantidad de milisegundos que el sistema debe esperar antes de responder una solicitud

[7] A pesar de que la implementación de la memoria virtual en un sistema real es una combinación entre el sistema operativo y el CPU (MMU), se implementa en este caso como un solo proceso aparte, para facilitar su comprensión e integración con el sistema

[8] Utilizando alguna función de la familia de `malloc`

Proceso Swap

Este proceso simulará el administrador de memoria virtual de un sistema operativo. Será el encargado de administrar la memoria virtual del sistema, para ello, al iniciarse, creará un archivo de tamaño configurable (en páginas), el cual representará nuestra *partición de swap*, y quedará a la espera de la conexión del **proceso UMC**.

Inicialmente el archivo de swap deberá ser rellenado con el caracter `\0`, a fines de inicializar la partición[9]. El tamaño de las páginas escritas en swap es configurable[10], 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 Programa AnSISOP, asignará el tamaño necesario para que este sea guardado, dejando el espacio restante como libre. Esto mismo sucederá con los siguientes Programas AnSISOP puestos en ejecución. Al finalizar un Programa AnSISOP, el espacio que tenía asignado será marcado como libre.

Para administrar el espacio utilizado, se utilizará un BitMap para saber si un espacio está disponible o no, y además como complemento se empleará una estructura de control que deberá contar con (mínimamente) los siguientes campos: PID, número de página y la posición de la misma dentro del swap. La unidad mínima de asignación será una página[11].

Cuando le sea informada la creación de un nuevo Programa AnSISOP, 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*[12] 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 el mismo finalice. El procedimiento de compactación deberá esperar una cantidad de tiempo configurable (en milisegundos), 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 Programa AnSISOP, deberá borrarlo de la *partición de swap* marcando las páginas ocupadas como disponibles.

Archivo de Configuración

Nombre de Campo	Valor de Ejemplo
`PUERTO_ESCUCHA`	6000
`NOMBRE_SWAP`	`swap.data`
`CANTIDAD_PAGINAS`	512
`TAMANIO_PAGINA`	256[13]
`RETARDO_ACCESO`	600
`RETARDO_COMPACTACION`	60000

El `RETARDO_ACCESO` es el retardo a aplicar a cada operación de lectura o escritura de página antes de confirmarla. El `RETARDO_COMPACTACION` es el tiempo a esperar para simular la complejidad relativa del proceso de compactación.

[9] Se recomienda al alumno investigar sobre la utilización del comando `dd` para crear los archivos

[10] Recuerde que el tamaño de las **páginas** es el mismo que el de los **marcos**

[11] Para más información sobre asignación contigua, remitirse al capítulo 11 de "Fundamentos de Sistemas Operativos" de Abraham Silberschatz

[12] 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.

[13] Recuerde que el tamaño de página debe ser el mismo en todo el sistema. Este valor puede hacerlo parametrizable por archivo de configuración o consultarlo al proceso UMC al establecer conexión.

Proceso CPU

Este proceso es uno de los más importantes del trabajo ya que es el encargado de interpretar y ejecutar las operaciones escritas en código AnSISOP de un Programa.

Estará en permanente contacto con el Proceso UMC, tanto para obtener información del Programa en ejecución, como para actualizar las estructuras requeridas luego de ejecutar una operación.

Al iniciar, se conectará a la UMC y obtendrá el tamaño de página a utilizar. Además se conectará al proceso Núcleo y quedará a la espera de que este le envíe el PCB de un Programa AnSISOP para ejecutarlo.

Incrementará el valor del registro Program Counter del PCB y utilizará el índice de código para solicitar a la UMC la porción de memoria en que se encuentre la próxima sentencia a ejecutar. Al recibirla, parseará la instrucción, ejecutará las operaciones requeridas, actualizará los valores del Programa en la UMC, actualizará el Program Counter en el PCB y notificará al Núcleo que concluyó un **quantum**[14].

Ejemplo

Luego de recibir de la UMC la instrucción `a = b + 3`, el parser la interpretará y ejecutará las siguientes **primitivas**:

1. `obtener_direccion('b')`: utilizando el índice del stack, devolverá la posición de la variable `b`. Por ejemplo: Pag: 5, Offset: 8, Size: 4.
2. `obtener_valor(Pag: 5, Offset: 8, Size: 4)`: pedirá a la UMC el valor de la variable `b`, en este caso los 4 bytes a partir del offset 8 de la página 5. Supongamos b = 9.
3. `obtener_direccion('a')`: como 1), pero para saber dónde guardar el resultado. Por ejemplo, Pag: 4, Offset: 0, Size: 4.
4. `almacenar(Pag: 4, Offset: 0, Size: 4, 11)`: almacenará en la UMC el resultado de la suma `(9 + 3 = 11)`.

El ingreso a funciones o procedimientos requiere que se asienten datos como el punto de retorno o las variables locales en el índice de Stack - Ver detalle en el Anexo I - Bloque de Control del Proceso: Índice del Stack

Hot plug

En cualquier momento de la ejecución del sistema pueden conectarse nuevas instancias del proceso CPU. Será responsabilidad del Núcleo aceptar esas nuevas conexiones, y tener en cuenta a las nuevas CPUs a la hora de planificar.

Mediante la señal SIGUSR1, se le podrá notificar a un CPU que deberá desconectarse una vez concluida la ejecución del Programa actual, dejando de dar servicio al sistema.

Fin de la ejecución

Al ejecutar la última sentencia, el CPU deberá notificar al Núcleo que el proceso finalizó para que este se ocupe de solicitar la eliminación de las estructuras utilizadas por el sistema.

Excepciones

Existe la posibilidad que el Proceso CPU reciba un mensaje de excepción como resultado de una solicitud a la UMC. En este caso deberá notificarla por pantalla y concluir la ejecución del Programa actual.

[14] En este punto el alumno ya puede notar que si el proceso fuera desalojado y su PCB enviado a otro CPU, este tendría ahí y en la UMC toda la información necesaria para continuar su normal ejecución

Anexo I - Bloque de Control del Programa (PCB)

Al igual que en un sistema operativo convencional, todo Programa estará identificado por una estructura denominada PCB (Process Control Block - Bloque de Control del Programa) la cual contendrá el identificador único del proceso y el ***program counter (PC)*** del programa. Con esta información, cada proceso CPU puede retomar la ejecución desde el punto que se encontraba la última vez que el proceso fue expropiado por el Núcleo.

Junto con los campos comunes del PCB, se almacenarán algunas estructuras auxiliares para el proceso (como los tres índices) con el fin de simplificar la comprensión del trabajo práctico y su posterior desarrollo.

Estructura	Información
Identificador Único	Número identificador del proceso único en el sistema
Program Counter	Número de la próxima instrucción del Programa que se debe ejecutar
Páginas de código	Cantidad de páginas utilizadas por el código del Programa AnSISOP, empezando por la página cero
Índice de código	Estructura auxiliar que contiene el offset del inicio y del fin de cada sentencia del Programa
Índice de etiquetas	Estructura auxiliar utilizada para conocer las líneas de código correspondientes al inicio de los procedimientos y a las etiquetas
Índice del Stack	Estructura auxiliar encargada de ordenar los valores almacenados en el Stack

Índice de Código

AnSISOP es un lenguaje interpretado, por lo que las líneas de código pueden tener distintas longitudes, además de poder existir líneas en blanco o comentarios.

Dado que la UMC atiende únicamente solicitudes relacionadas con posiciones de memoria, el índice de código es una estructura que almacena el desplazamiento respecto al inicio del código del programa y la longitud de cada línea ejecutable.

De esta manera es posible saber la ubicación de cada línea *útil* en el código para poder solicitarla a la UMC.

Es vital recordar que el índice **no es de líneas sino de instrucciones**: el código para generar este índice recorrerá el código de un programa ignorando las primeras líneas vacías o con comentarios hasta encontrar la primer instrucción válida, registrará su inicio y su longitud y, luego, buscará la siguiente ignorando saltos y comentarios. Repetirá el procedimiento hasta el fin del archivo.

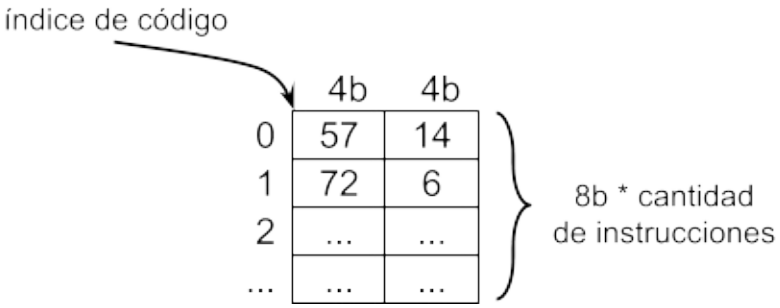
Por ejemplo, para un script como el siguiente:

```
#!/usr/bin/ansisop
begin
# primero declaro las variables
variables a, b
a = 20

print a

end
```

La instrucción 0 (`variables a, b`) comienza en el byte 57, con una longitud de 14 bytes, mientras que la instrucción 1 (`a = 20`) comienza en el byte 72 y mide 6 bytes. Se recomienda utilizar un array de dos valores enteros como el descrito a continuación:



Al momento de ejecutar una instrucción, la CPU solicitará a la UMC la entrada del Índice correspondiente a la instrucción a ejecutar, determinada por el Program Counter. Teniendo el byte de inicio y longitud de la instrucción, realizará la petición a la UMC en el correspondiente valor de `Pagina` , `Offset` y `Longitud` .

Índice de Etiquetas

El Índice de Etiquetas consta de la serialización de un diccionario que asocia el identificador de cada función y etiqueta del programa con la primer instrucción ejecutable de la misma (es decir, el valor que el Program Counter deberá tomar cuando el programa deba pasar a ejecutar esa función o saltar a esa etiqueta).

Índice del Stack

Esta estructura, también parte del PCB, es una manera auxiliar de simplificar la operatoria del segmento de Stack de un proceso.

Cada vez que un Programa AnSISOP ingrese a una función, se deberá agregar un elemento a este índice con los siguientes atributos:

Nombre	Tipo	Descripción
Argumentos (args)	Lista de Argumento	Posiciones de memoria donde se almacenan las copias de los argumentos de la función
Variables (vars)	Lista de Variable	Identificadores y posiciones de memoria donde se almacenan las variables locales de la función
Dirección de Retorno (retPos)	Numérico	Posición del índice de código donde se debe retornar al finalizar la ejecución de la función
Posición de la variable de retorno (retVar)	Posición de memoria (Pag#, Offset, Size)	Posición de memoria donde se debe almacenar el resultado de la función provisto por la sentencia RETURN

Ejemplo de Índice de Stack

Para ejemplificar el Índice del Stack, utilizaremos el siguiente código AnSISOP:

```
#!/usr/bin/ansisop
begin
variables a,g
    a = 1
    g <- doble a
    print g
end

function doble
variables f
    f = $0 + $0
    return f
end
```

Como se puede observar en el código, existen dos funciones: la principal y la función doble.

Al iniciar el programa, se crea la el siguiente registro en la estructura de stack.

pos	args	vars	retPos	retVar

0		ID: (Pag, Off, Size)
		`a: (0, 0, 4)`
		`g: (0, 4, 4)`		

Como se puede observar, la función principal no tiene argumentos, ni debe retornar por lo que esos campos están vacíos.

El programa ejecuta y, en un punto, llega a la línea de código útil número 3: `g <- doble a`

Aquí se debe dejar registrado en una nueva posición de esta estructura que, al concluir con la función `doble`, debe regresar a la posición 3 del código y debe almacenar el resultado en la variable `g`.

pos	args	vars	retPos	retVar

0		ID: (Pag, Off, Size)
		`a: (0, 0, 4)`
		`g: (0, 4, 4)`		

1	(Pag, Off, Size) `(0, 8, 4)`	ID: (Pag, Off, Size) `f: (0, 12, 4)`	3	(Pag, Off, Size) `(0, 4, 4)`

La dirección de retorno es la línea 3, y la dirección de la variable de retorno coincide con la de la variable `g`.

La función `doble` recibe un argumento (en el ejemplo el valor de `a`, que es 1), el cual debe ser copiado a una nueva posición de memoria, y tiene una variable (`f`) la cual debe ser creada en otro espacio de memoria.

Al concluir la función `doble` y la variable `g` asignada con el resultado, este nuevo registro del índice de Stack debe ser eliminado.

Anexo II – Especificación del Lenguaje AnSISOP

AnSISOP es un lenguaje de programación interpretado de propósito general y bastante bajo nivel. Su sintaxis es simple y, en general, no es muy recomendable para utilizar en tareas productivas. En cambio, su objetivo es principalmente académico: ayudar a entender los conceptos y mecanismos que un sistema operativo debe tener en cuenta para manejar la ejecución de los programas.

El lenguaje se divide en dos capas: la sintaxis de alto nivel, utilizada para escribir los scripts, y las operaciones Primitivas que la CPU deberá ejecutar para llevar a cabo las instrucciones de los primeros. El alumno deberá desarrollar el intérprete del programa y las diversas primitivas que serán invocadas.

Para la evaluación del trabajo práctico no se proveerán programas con errores de sintaxis ni de semántica.

Sintaxis

- El lenguaje AnSISOP es case-sensitive; es decir, `hola` y `Hola` son diferentes
- El código principal del programa estará comprendido entre las palabras reservadas `begin` y `end`. `begin` solo indica por donde comenzará a ejecutar el programa.
- Las sentencias finalizan con un salto de línea. Los saltos adicionales son ignorados.
- Toda línea comenzada por un caracter numeral (`#`) es un comentario y debe ser ignorado.
- Todo programa deberá terminar con una línea en blanco.
- Un `:` seguido de una palabra es una etiqueta que será utilizada para permitir saltos dentro del código con `jump`, `jz` y `jnz`, explicados más adelante.
- Todas las variables dentro de una función son locales.
- Una función puede llamar a otra función.

Variables

Las variables locales se declaran luego de la sentencia `variables`. Son solamente de tipo entero con signo y su identificador es un caracter alfabético [a-zA-Z]. Su valor no debe ser inicializado. Se las indica en el código solo con su nombre.

Las variables dadas como parámetros de funciones se nombran con un único dígito [0-9] y se accederá a ellas en el código anteponiendo el signo `$` (`$0` refiere al primer parámetro, y así).

Las variables compartidas[15] se declaran e inicializan en la configuración del Núcleo, se nombran como cadenas sin restricción de nombre, y se las indica en el código como `!identificador`.

Asignación

Con el nombre de la variable a la izquierda de un signo igual, se le podrá asignar a una variable como valor:

- Un número entero u otra variable (local, parámetro o compartida; no semaforos)
- El resultado de una operación aritmética la cual podrá ser suma o resta

Salto condicional

Las instrucciones de salto condicional *saltar-si-no-es-cero* (`jnz` - *jump on not-zero*) y *saltar-si-es-cero* (`jz` - *jump on zero*) recibirán como parámetro una variable que evaluarán y una etiqueta a la que deberán saltar en caso de que se cumpla la condición.

Estas instrucciones, por definición del enunciado, tendrán como origen y destino el procedimiento actual. En otras palabras, el código de una función o procedimiento no podrá saltar dentro del código de otro.

> Este código de ejemplo incrementa la variable `i` de uno a diez e imprime dichos valores en pantalla
>
> ```
> #!/usr/bin/ansisop
> begin
> variables i,b
>     i = 1
>     :inicio_for
>     i = i + 1
>     print i
>     b = i - 10
>     jnz b inicio_for
>     #fuera del for
> end
> ```
>
> Observe que si la variable `i` no es igual a 10 entonces `b = i - 10` no es cero, entonces la instrucción de salto condicional iría a la etiqueta `inicio_for` hasta llegar a la 10ma iteración, donde no saltará más.

Entrada/Salida: `io`

La llamada al sistema `io` recibirá *dos* parámetros.

1. Una cadena como identificador del sistema del dispositivo de entrada/salida
2. La cantidad de operaciones de entrada/salida que se realizarán en éste dispositivo

Impresión en pantalla

Existen dos formas de impresión. `textPrint` seguido de una cadena imprimirá la cadena tal cual aparece en el código fuente. La palabra reservada `print` será utilizada para mostrar el valor de la variable que reciba como parámetro.

La información deberá ser mostrada en la terminal del programa y registrada en el log del sistema.

Ejemplo:

```
a = 0
textPrint La variable a vale
print a
```

Resultado:

```
La variable a vale
0
```

Funciones

La definición de las funciones estará dada por la palabra reservada `function` seguida del nombre de la misma. Todas las funciones retornan un valor y no existe en el lenguaje el concepto de procedimiento.

Código de ejemplo

Este código de ejemplo imprime variables.

```
#!/usr/bin/ansisop
function prueba
    variables a,b
    a = 2
    b = 16
    print b
    print a
    a = a + b
    return a
end

begin
variables a, b
    a = 20
    print a
    b <- prueba
    print b
    print a
end
```

Lo que se ve por pantalla sería (Nótese la localidad/scope de las variables):

```
20
16
2
18
20
```

[15] Llamamos "variables compartidas" a aquellas manejadas por el Núcleo, de las que todos los CPUs tiene acceso. No son "variables globales" ya que "global" refiere al scope/contexto de cada programa. No existen en AnSISOP las variables globales.

Anexo III - Primitivas de AnSISOP

Para evitar la complejidad que presenta realizar un analizador de sintaxis y dado que estas tareas no son inherentes al contenido de la materia, se le facilita al alumno un Parser que se encargará de interpretar cada línea de código y de ejecutar las Primitivas correspondientes, cuyo código será desarrollado por el alumno. Tanto su implementación directa o parcial es aconsejable.

El parser podrá obtenerse desde https://github.com/sisoputnfrba/ansisop-parser

Primitivas de AnSISOP

1. `definirVariable`
2. `obtenerPosicionVariable`
3. `dereferenciar`
4. `asignar`
5. `obtenerValorCompartida`
6. `asignarValorCompartida`
7. `irAlLabel`
8. `llamarConRetorno`
9. `retornar`
10. `imprimir`
11. `imprimirTexto`
12. `entradaSalida`
13. `wait`
14. `signal`

1. `definirVariable`

Reserva en el Contexto de Ejecución Actual el espacio necesario para una variable llamada `identificador_variable` y la registra en el Stack, retornando la posición del valor de esta nueva variable del stack.

El valor de la variable queda indefinido: no deberá inicializarlo con ningún valor default.

Esta función se invoca una vez por variable, a pesar de que este varias veces en una línea. Por ejemplo, evaluar `variables a, b, c` llamará tres veces a esta función con los parámetros `a`, `b` y `c`.

```
t_puntero definirVariable(t_nombre_variable identificador_variable);
```

2. `obtenerPosicionVariable`

Devuelve el desplazamiento respecto al inicio del segmento Stack en que se encuentra el valor de la variable `identificador_variable` del contexto actual. En caso de error, retorna -1.

```
t_puntero obtenerPosicionVariable(t_nombre_variable identificador_variable);
```

3. `dereferenciar`

Obtiene el valor resultante de leer a partir de `direccion_variable`, sin importar cual fuera el contexto actual.

```
t_valor_variable dereferenciar(t_puntero direccion_variable);
```

4. `asignar`

Copia un valor en la variable ubicada en `direccion_variable`.

```
void asignar(t_puntero direccion_variable, t_valor_variable valor)
```

5. `obtenerValorCompartida`

Solicita al Núcleo el valor de una variable compartida.

```
t_valor_variable obtenerValorCompartida(t_nombre_compartida variable)
```

6. `asignarValorCompartida`

Solicita al Núcleo asignar el valor a la variable compartida. Devuelve el valor asignado.

```
t_valor_variable asignarValorCompartida(t_nombre_compartida variable, t_valor_variable
 valor)
```

7. `irAlLabel`

Cambia la linea de ejecucion a la correspondiente de la etiqueta buscada.

```
void irAlLabel(t_nombre_etiqueta etiqueta)
```

8. `llamarConRetorno`

Preserva el contexto de ejecución actual para poder retornar luego al mismo, junto con la posición de la variable entregada por `donde_retornar`. Modifica las estructuras correspondientes para mostrar un nuevo contexto vacío, actualizando el Program Counter según corresponda.

Los parámetros serán definidos luego de esta instrucción de la misma manera que una variable local, con identificadores numéricos empezando por el 0.

No se pretende que se pueda retornar a una variable compartida. Sí a un parámetro o variable local.

```
void llamarConRetorno(t_nombre_etiqueta etiqueta, t_puntero donde_retornar)
```

9. `retornar`

Modifica el Contexto de Ejecución Actual por el Contexto anterior al que se está ejecutando, recuperando el Cursor de Contexto Actual, el Program Counter y la dirección donde retornar, asignando el valor de retorno en esta, previamente apilados en el Stack.

```
void retornar(t_valor_variable retorno)
```

10. `imprimir`

Envía al Núcleo el contenido de `valor_mostrar`, para que este le reenvíe a la correspondiente consola del Programa en ejecución. Devuelve la cantidad de caracteres impresos.

```
void imprimir(t_valor_variable valor_mostrar)
```

11. `imprimirTexto`

Envía al Núcleo una cadena de `texto` para que este la reenvíe a la correspondiente consola del Programa en ejecución. No admite parámetros adicionales, secuencias de escape o variables. Devuelve la cantidad de caracteres impresos.

```
void imprimirTexto(char* texto)
```

12. `entradaSalida`

Informa al Núcleo que el Programa actual pretende utilizar el `dispositivo` durante `tiempo` unidades de tiempo.

```
void entradaSalida(t_nombre_dispositivo dispositivo, int tiempo)
```

13. `wait`

Informa al Núcleo que ejecute la función wait para el semáforo con el nombre `identificador_semaforo`. El Núcleo deberá decidir si bloquearlo o no.

```
void wait(t_nombre_semaforo identificador_semaforo)
```

14. `signal`

Comunica al Núcleo que ejecute la función signal para el semáforo con el nombre `identificador_semaforo`. El Núcleo decidirá si esto conlleva desbloquear a otros procesos.

```
void signal(t_nombre_semaforo identificador_semaforo)
```

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.

Checkpoint 1 - Obligatorio

Fecha: 30 de Abril

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 configuración y logs
- Implementar el Parser de AnSISOP en un programa de ejemplo
- Desarrollar un modelo de intérprete de comandos para la consola de la UMC
- Desarrollar un binario que sirva como intérprete de un script en Linux
- Realizar la estructura básica de comunicación entre los programas

Requisitos mínimos:

- Entorno instalado y funcionando. Todos los integrantes deben haber subido commits a Github.
- Tener todos los procesos del sistema creados, conectándose entre sí
- Poder conectar una consola al Núcleo y mediante un comando de prueba solicitarle que inicie un programa y que este mensaje sea enviado a la CPU, luego de la CPU al UMC y luego aquí al Administrador de Swap. Será suficiente con que cada proceso imprima el mensaje recibido por pantalla.

Distribución recomendada:

Consola: 0.5 personas **Núcleo**: 1.5 persona. **UMC**: 1 persona. **Swap**: 1 persona. **CPU**: 1 persona.

Lectura recomendada:

- http://faq.utn.so/arrancar
- Beej Guide to Network Programming
- Linux POSIX Threads
- SisopUTNFRBA Commons Libraries

Checkpoint 2 - Obligatorio

Fecha: 21 de Mayo

Objetivos:

- Modificar el código de la UMC para que permita recibir múltiples conexiones por sockets y gestionarlas utilizando hilos independientes. Utilizar semáforos para sincronizar hilos que acceden las listas compartidas de la UMC
- Diseñar el diagrama de estados de un PCB en el sistema y crear la estructura del PCB junto con sus cambios en cada estado.
- Permitir que el proceso Swap pueda operar sobre un archivo binario y desarrollar las operaciones y estructuras requeridas para administrar el espacio.
- Iniciar la implementación de las primitivas del CPU

Requisitos mínimos:

- El proceso Núcleo debe poder conocer una serie de PCBs y gestionarlos en sus distintos estados en función del algoritmo Round Robin. En todo momento debe indicar sus estados y transiciones.
- El proceso CPU debe, dado un código en AnSISOP, invocar las correspondientes primitivas y las primeras 4 deben intercambiar un mensaje de ejemplo con la UMC.
- El proceso UMC debe ser capaz de recibir pedidos de lecturas, y directamente reenviarlos al proceso Swap. También debe interpretar los pedidos para iniciar y finalizar Programas AnSISOP.
- El proceso Swap debe ser capaz de recibir nuevos programas AnSISOP. 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:

Consola: 0.25 persona. **Núcleo**: 0.75 persona. **UMC**: 1 persona. **Swap**: 1.5 personas. **CPU**: 1.5 personas.

Lectura recomendada:

- Sistemas Operativos, Silberschatz, Galvin - Capítulo 4: Hilos
- Sistemas Operativos, Silberschatz, Galvin - Capítulo 5: Planificación del Procesador

Checkpoint 3 - Obligatorio - En laboratorio

Fecha: 11 de Junio

Objetivos:

- Implementar la interfaz de mensajes de la UMC y su correspondiente validación.
- Implementar el algoritmo Clock en la UMC y LRU para su TLB.
- Las operaciones AnSISOP deben impactar en las estructuras de la UMC y del proceso Swap cuando corresponda.
- Desarrollar los hilos de entrada/salida
- El núcleo debe poder recibir nuevos Programas AnSISOP desde diversas consolas y enviar los PCB READY a ejecutar a los distintos CPUs disponibles.
- El proceso CPU deberá ejecutar programas simples impactando las modificaciones en las estructuras del PCB.

Requisitos mínimos:

- Varias instancias de un programa simple ejecutado debería poder ser ejecutado en el sistema
- Estructuras como el PCB o las estructuras administrativas de procesos como el UMC o el SWAP deben ser consistentes con la ejecución

Distribución recomendada:

Núcleo: 1 persona. **UMC**: 2 personas. **Swap**: 0.5 persona. **CPU**: 1.5 personas.

Entrega Final

Fecha: 2 de Julio

Objetivos:

- Finalizar las primitivas del CPU y permitir la ejecución de programas AnSISOP en su especificación completa
- Implementar el algoritmo Clock Modificado en la UMC
- Dar soporte a las instrucciones de semáforo y a las variables de memoria compartida
- Realizar pruebas de stress sobre el sistema, ejecutando varios Programas AnSISOP de manera simultánea e interrumpiendo su ejecución

Distribución recomendada:

CPU: 1.5 persona. **Núcleo**: 1 persona. **UMC**: 0.5 persona **Stress Test y debugging**: 2 personas.

Lectura recomendada:

- Test del trabajo práctico provistos por la cátedra