RESUMEN PRIMER TEÓRICO INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS

Facultad de Informática, UNLP.
Fecha de Realización: 28/10/2023.

Contenido

Clase 1 (Introducción 1)

Sistema Operativo Definiciones:

- Software intermediario entre un usuario y el Hardware de una computadora.
- Necesita procesador y memoria para existir.
- Controla la ejecución de los procesos.

Perspectiva del Usuario sobre el SO (de arriba hacia abajo):

- Se genera una abstracción con respecto a la arquitectura.
- El SO "oculta el Hardware y presenta a los programas abstracciones más fáciles de manejar.
- Los programas de aplicación son los "cliente" del SO.
- **Se genera Comodidad y amigabilidad.**

Perspectiva desde la administración de recursos (de abajo hacia arriba):

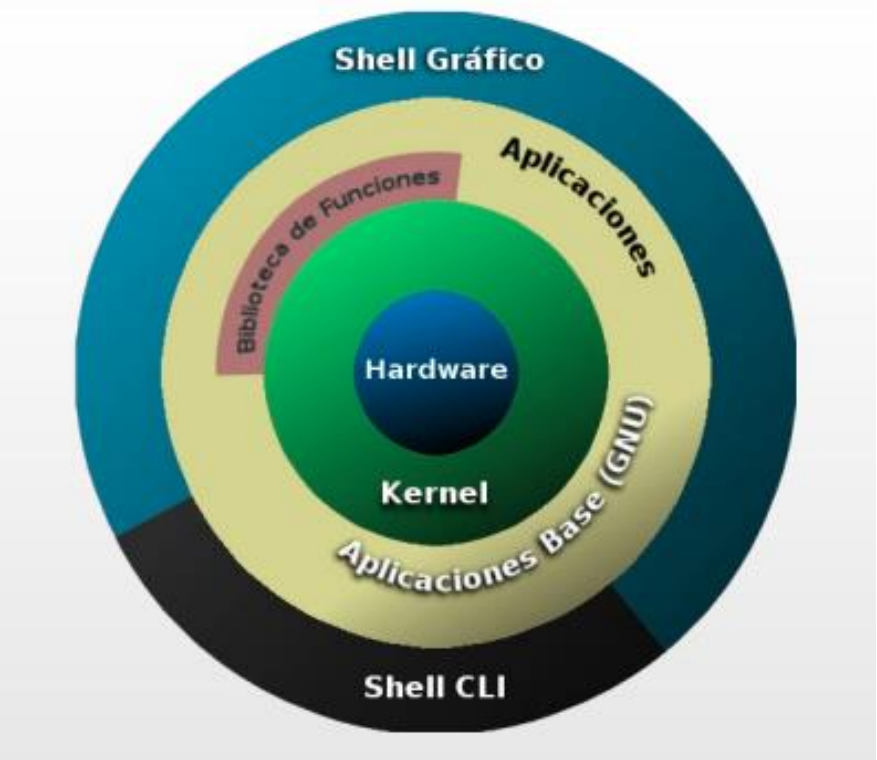
- Administra los recursos de Hardware de los procesos.
- Provee un conjunto de servicios al usuario.
- Maneja la memoria secundaria y la E/S.
- Ejecuta simultáneamente procesos.
- Multiplexación en tiempo (CPU) y espacio (memoria).

Objetivos del Sistema Operativo:

- **Comodidad:** Hacer fácil el uso del hardware.
- **Eficiencia:** Hacer un uso eficiente de los recursos del sistema.
- **Evolución:** Permitir agregar funciones sin interferir con otras anteriores.

Componentes de un Sistema Operativo:

- **Kernel:** Capa que administra el hardware.
- **Shell:** GUI/CUI o CLI.
- **Herramientas:** Editores, compiladores, librerías, etc.



Kernel:

- Porción de código que se encuentra en memoria principal.
- Se encarga de la administración de los recursos.
- **Implementa servicios esenciales:**
  - Manejo de memoria y de la CPU.
  - Administración de procesos.
  - Comunicación y Concurrencia.
  - Gestión de la E/S.

Servicios de un Sistema Operativo:

- **Administración y planificación del procesador:**
  - Multiplexación de la carga de trabajo.
  - Busca Imparcialidad.
  - Busca que no haya bloqueos.
  - Manejo de Prioridades.
- **Administración de Memoria:**
  - Busca una administración eficiente de la memoria.
  - Jerarquía de Memoria.
  - Protección de programas que compiten o se ejecutan concurrentemente.

- **Administración del File System:**
  - Acceso a medios de almacenamiento externos.
- **Administración de dispositivos:**
  - Ocultamiento de dependencias de hardware.
  - Administración de accesos simultáneos.
- **Detección de errores y respuestas:**
  - Errores de hardware internos y externos (Memoria/CPU y Dispositivos).
  - Errores de software (Excepciones).
  - Incapacidad del Sistema Operativo para conceder una solicitud de aplicación.
- **Interacción del Usuario (Shell).**
- **Contabilidad:**
  - Estadísticas de uso.
  - Monitorear parámetros de rendimiento.
  - Anticipar necesidades de mejoras futuras.
  - Dar elementos si es necesario facturar tiempo de procesamiento.

Complejidad de un Sistema Operativo:

- Es extenso y complejo.
- Desarrollado por partes.
- Cada parte debe ser analizada y desarrollada entendiendo su función, cuáles son sus entradas y sus salidas.

Clase 2 (Introducción 2)

Funciones principales de un SO:

- Brindar abstracciones de alto nivel a los procesos de usuario.
- Administrar eficientemente el uso de la CPU y la Memoria.
- Brindar asistencia para la realización de Entrada-Salida (E/S) por parte de los procesos.

Problemas que un SO debe evitar:

- Que un proceso se apropie de la CPU

- Que un proceso intente acceder a una posición de memoria fuera de su espacio declarado.
- Que un proceso intente ejecutar instrucciones de E/S o privilegiadas.

El SO debe:

- Gestionar el uso de la CPU
- Detectar intentos de ejecución de instrucciones de E/S ilegales
- Detectar accesos ilegales a memoria
- Proteger el vector de interrupciones ya que mediante este se podría tomar control del Kernel.

El SO cuenta el apoyo del Hardware de las siguientes maneras:

- **Modos de Ejecución:** Define limitaciones en el conjunto de instrucciones que se puede ejecutar en cada modo (Modo Kernel y Modo Usuario).
  - ❖ **Modo Kernel:** Cualquier instrucción que se ejecute se puede llevar a cabo.
  - ❖ **Modo Usuario:** Conjunto de instrucciones reducidas.
- **Interrupción de Clock:** Se debe evitar que un proceso se apropie de la CPU, para ello se generan interrupciones cada cierto tiempo para que el SO gestione y controle que un proceso malicioso no esté queriendo tomar control de la CPU y del Kernel.
- **Protección de la Memoria:** Se deben definir límites de memoria a los que puede acceder cada proceso (registros base y límite)

Modos de Ejecución:

- Hay un bit en la CPU que indica el modo actual.
- Las instrucciones privilegiadas deben ejecutarse en **modo Supervisor o Kernel.**
- En modo Usuario, el proceso puede acceder sólo a su espacio de direcciones.
- El **kernel** del SO se ejecuta en modo supervisor y el resto del SO y los programas de usuario se ejecutan en modo usuario.
- Cuando arranca el sistema, arranca en modo Kernel.

- Cada vez que comienza a ejecutarse un proceso de usuario, este bit se debe poner en modo usuario.
- Cuando hay **un trap o una interrupción**, el bit de modo se pone en modo Kernel. Esta es la única forma de pasar a Modo Kernel, el proceso de usuario no lo puede hacer explicamente.

Cómo actúa para proteger…

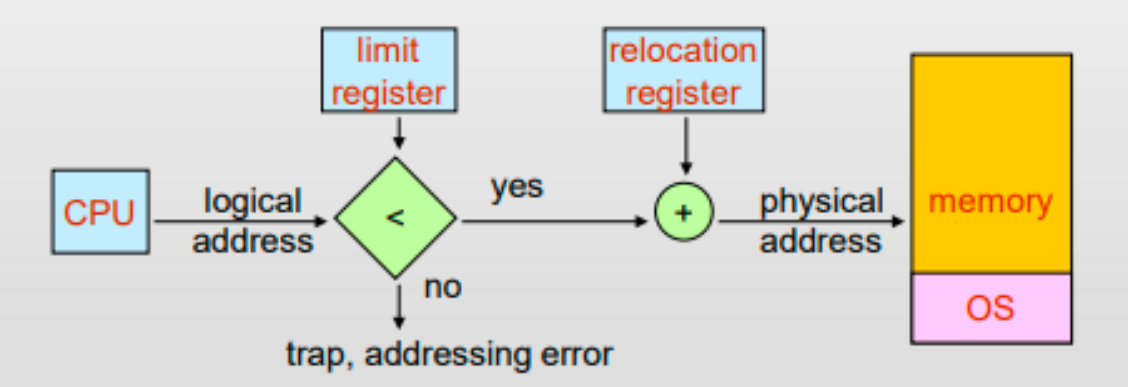
- Cuando el proceso de usuario intenta por sí mismo ejecutar instrucciones que pueden causar problemas (las llamadas instrucciones privilegiadas), el HW lo detecta como una operación ilegal y produce un trap al SO. Por lo tanto el sistema cambia a Modo Kernel asegurándose de tener el control de la CPU, sin dejar que el proceso la apropie.

Resumen de los Modos de Ejecución:

- **Modo Kernel:**
  - **Gestión de procesos:** Creación y terminación, planificación, intercambio, sincronización y soporte para la comunicación entre procesos.
  - **Gestión de memoria:** Reserva de espacio de direcciones para los procesos, Swapping, Gestión y páginas de segmentos.
  - **Gestión E/S:** Gestión de buffers, reserva de canales de E/S y de dispositivos de los procesos.
  - **Funciones de soporte:** Gestión de interrupciones, auditoría, monitoreo.
- **Modo Usuario:**
  - Debug de procesos, definición de protocolos de comunicación gestión de aplicaciones (compilador, editor, aplicaciones de usuario.
  - En este modo se llevan a cabo todas las tareas que no requieran accesos privilegiados.
  - En este modo no se puede interactuar con el hardware.
  - El proceso trabaja en su propio espacio de direcciones.

Protección de la Memoria:

- Se debe delimitar el espacio de direcciones del proceso, para ello se puede limitar el espacio de direcciones mediante por ejemplo, un registro base y un registro límite que son cargados por el Kernel.
- El kernel debe proteger para que los procesos de usuario no puedan acceder donde no les corresponde. Ya sea una zona de memoria privada o protegida como también puede ser el espacio de direcciones de otro proceso.



Protección de la E/S:

- Las instrucciones de E/S se definen como privilegiadas y deben ejecutarse en Modo Kernel.
- Cada vez que un proceso de usuario quiere realizar E/S debe hacerlo mediante una llamada al sistema o call system (servicio del sistema operativo).

Protección de la CPU:

- Uso de interrupción por clock para evitar que un proceso se apropie de la CPU.
- Se implementa normalmente a través de un clock y un contador. El kernel le da valor al contador que se decrementa con cada tick de reloj y al llegar a cero puede expulsar al proceso para ejecutar otro.
- Las instrucciones que modifican el funcionamiento del reloj son privilegiadas.

System Calls:

- Es la forma en que los programas de usuario acceden a los servicios del SO.
- Se ejecutan en modo kernel o supervisor.

- Categorías de system calls:
  - ❖ Control de Procesos (fork por ejm.).
  - ❖ Manejo de archivos (open, close, read, write, etc.).
  - ❖ Manejo de dispositivos (mkdir, rmdir, link, unlink, mount, unmount, etc).
  - ❖ Mantenimiento de información del sistema (time, etc.).
  - ❖ Comunicaciones.

Procesos 1

Diferencias entre un programa y un proceso:

PROGRAMA	**PROCESO**
• Es estático. • No tiene program counter. • Existe desde que se edita hasta que se borra de nuestra computadora.	• Es dinámico. • Tiene program counter. • Su ciclo de vida comprende desde que se solicita ejecutar hasta que termina.

Componentes de un proceso:

- Componentes mínimos de un proceso para poder ejecutarse:
  - ❖ Sección de Código (texto).
  - ❖ Sección de Datos (variables globales).
  - ❖ Stack(s) (datos temporarios: parámetros, variables temporales y direcciones de retorno).

Stacks:

- Un proceso cuenta con 1 o más stacks pero en general son 2, uno para el modo Kernel y otro para el modo Usuario.
- Se crean automáticamente y su medida se ajusta en run-time.
- Está formado por stack frames que son pushed (al llamar a una rutina) y popped (cuando se retorna de ella).
- El stack frame tiene los parámetros de la rutina (variables locales), y datos necesarios para recuperar el stack frame anterior (el contador de programa y el valor del stack pointer en el momento del llamado).

Atributos de un proceso:

- Identificación del proceso (PID), y del proceso padre (PPID).
- Identificación del usuario que lo "disparó".
- Si hay estructura de grupos, grupo que lo disparó.
- En ambientes multiusuario, desde que terminal y quien lo ejecuto.

Process Control Block (PCB):

- Estructura de datos asociada al proceso (abstracción).
- Existe una por proceso.
- Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina.
- Contiene la siguiente información de un proceso:
    - ❖ PID, PPID, etc.
    - ❖ Valores de los registros de la CPU (PC, AC, etc).
    - ❖ Planificación (estado, prioridad, tiempo consumido, etc).
    - ❖ Ubicación (representación) en memoria.
    - ❖ Accounting.
    - ❖ Entrada salida (estado, pendientes, etc).

¿Qué es el espacio de direcciones de un proceso?

- Conjunto de direcciones de memoria que ocupa el proceso.
- No incluye su PCB o tablas asociadas.
- Un proceso en Modo Usuario solo puede acceder a su espacio de direcciones y uno en Modo Kernel puede acceder a estructuras internas como la PCB o a espacios de direcciones de otros procesos.

El contexto de un proceso:

- Incluye toda la información que el SO necesita para administrar el proceso, y la CPU para ejecutarlo correctamente.
- Las cosas que son parte del contexto son: los registros de CPU, inclusive el contador de programa, prioridad del proceso, si tiene E/S pendientes, etc.

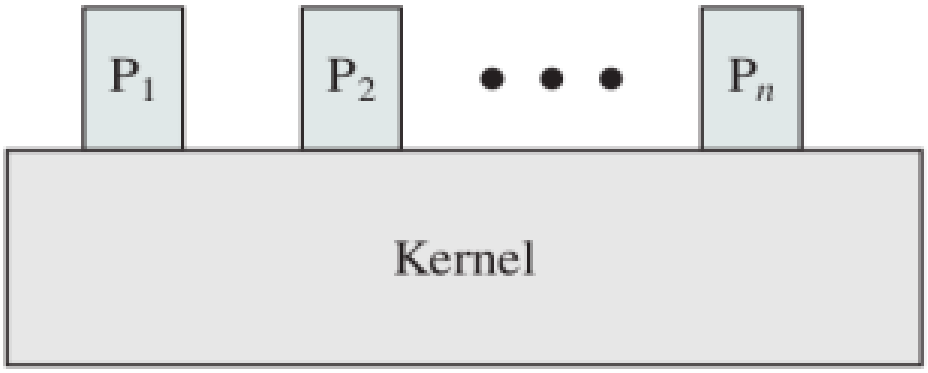
Administración de procesos	Administración de memoria	Administración de archivos
Registros	Apuntador a la información del segmento de texto	Directorio raíz
Contador del programa	Apuntador a la información del segmento de datos	Directorio de trabajo
Palabra de estado del programa	Apuntador a la información del segmento de pila	Descripciones de archivos
Apuntador de la pila		ID de usuario
Estado del proceso		ID de grupo
Prioridad		
Parámetros de planificación		
ID del proceso		
Proceso padre		
Grupo de procesos		
Señales		
Tiempo de inicio del proceso		
Tiempo utilizado de la CPU		
Tiempo de la CPU utilizado por el hijo		
Hora de la siguiente alarma		

Cambio de Contexto (Context Switch):

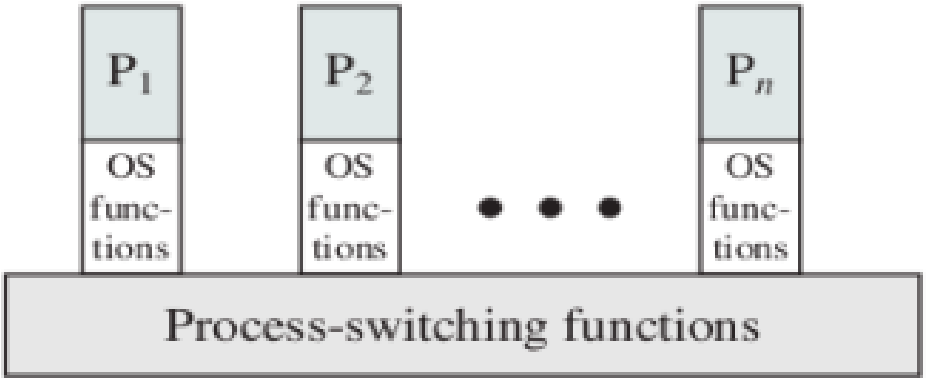
- Se produce cuando la CPU cambia de un proceso a otro.
- Se debe resguardar el contexto del proceso saliente, que pasa a espera y retornará después a la CPU y se debe cargar el contexto del nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada en dicho contexto.
- Es tiempo no productivo de CPU.
- Se realiza en modo Kernel.

Enfoques de diseño sobre el Kernel:

- **Enfoque 1 – El Kernel como entidad independiente:**
  - ❖ El Kernel se ejecuta fuera de todo proceso.
  - ❖ Cuando un proceso es “interrumpido” o realiza una System Call, el contexto del proceso se salva y el control se pasa al Kernel del sistema operativo.
  - ❖ El Kernel tiene su propia región de memoria y su propio Stack.
  - ❖ Finalizada su actividad, le devuelve el control al proceso (o a otro diferente).
  - ❖ El Kernel NO es un proceso.
  - ❖ Se ejecuta como una entidad independiente en modo privilegiado.



- **Enfoque 2 – El Kernel "dentro" del Proceso:**
  - El "Código" del Kernel se encuentra dentro del espacio de direcciones de cada proceso y es compartido por todos los procesos.
  - El Kernel se ejecuta en el MISMO contexto que algún proceso de usuario.
  - El Kernel se puede ver como una colección de rutinas que el proceso utiliza.
  - Cada proceso tiene su propio stack (uno en modo usuario y otro en modo kernel).
  - El proceso es el que se Ejecuta en Modo Usuario y el kernel del SO se ejecuta en Modo Kernel (cambio de modo).
  - Cada interrupción (incluyendo las de System Call) es atendida en el contexto del proceso que se encontraba en ejecución pero en modo Kernel.



Procesos 2

Estados de un Proceso:

- **New:** Un usuario “dispara” el proceso. Un proceso es creado por otro proceso: su proceso padre. En este estado se crean las estructuras asociadas (PCB), y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria.
- **Ready to run:** El proceso está listo para cargar su Program Counter y empezar a ejecutarse, pero no lo está haciendo, se encuentra almacenado en la cola de listos compitiendo por la CPU.
- **Running:** Se genera un cambio de contexto y el proceso empieza su ejecución teniendo la CPU a su disposición hasta que se termine su Quantum, se necesite una operación de I/O, hasta que termine o hasta que sea bloqueado. Puede tener tres salidas: **“Exit”, “Waiting” o “Ready to run”.**
- **Exit:** Se libera la CPU y se eliminan todas las estructuras del proceso en la memoria.
- **Waiting:** El proceso pasa a este estado cuando necesita esperar que se cumpla un evento para seguir ejecutándose (System calls, I/O, etc.). Queda en un estado de bloqueo hasta que la espera termina. Sigue en memoria, pero no tiene la CPU. Al cumplirse el evento, pasará al estado de listo.

Colas en la planificación de procesos:

- Para realizar la planificación, el SO utiliza la PCB de cada proceso como una abstracción del mismo. Las PCB se enlazan en Colas siguiendo un orden determinado.
- Ejemplos de colas de Planificación:
  - **Cola de trabajos o procesos:**
    - Contiene todas las PCB de procesos en el sistema.
  - **Cola de procesos listos:**
    - PCB de procesos residentes en memoria principal esperando para ejecutarse.
  - **Cola de dispositivos:**
    - PCB de procesos esperando por un dispositivo de I/O.

Módulos de la planificación:

- Son módulos (SW) del Kernel que realizan distintas tareas asociadas a la planificación.
- Se ejecutan ante determinados eventos que así lo requieren:
  - ❖ Creación/Terminación de procesos.
  - ❖ Eventos de Sincronización o de E/S.
  - ❖ Finalización de lapso de tiempo.
- Son 3: **Short term Scheduler, Medium term Scheduler y Long term Scheduler**.
- El nombre de cada Scheduler proviene de la frecuencia de ejecución de cada uno, siendo Short el que mayor fre y Long el menos ejecutado.

Dispatcher:

- Hace cambio de contexto, cambio de modo de ejecución...”despacha” el proceso elegido por el Short Term.

Loader:

- Carga en memoria el proceso elegido por el long term.

Long term Scheduler:

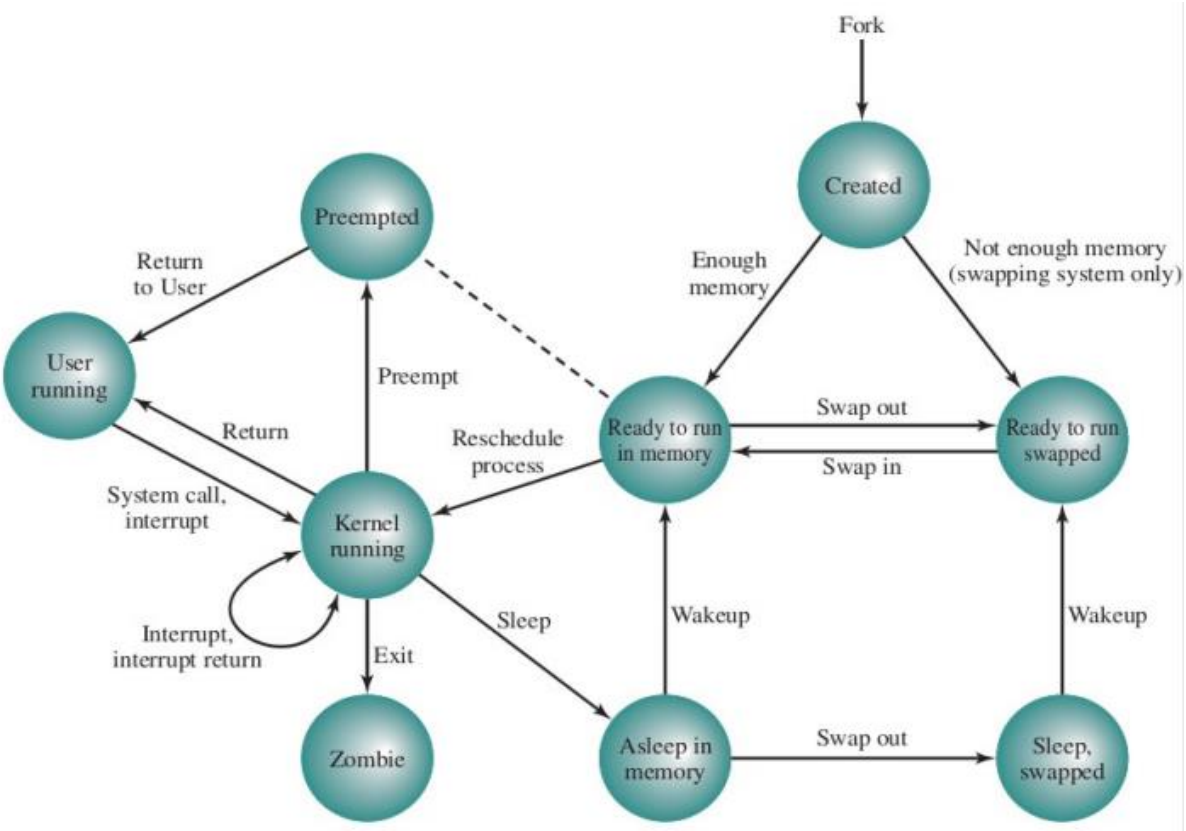
- Controla el grado de multiprogramación, es decir, la cantidad de procesos en memoria.
- Puede no existir este scheduler y absorber esta tarea el de short term.
- **Pasa de New -> Ready y de New -> Ready/Suspend**.

Medium Term Scheduler (swapping):

- Si es necesario, reduce el grado de multiprogramación (grado máximo de procesos que el Administrador de Memoria puede ejecutar).
- Saca temporalmente de memoria los procesos que sea necesario para mantener el equilibrio del sistema.
- Sacar a memoria (swap out), volver a memoria (swap in).
- **Pasa de Ready/Suspend -> Ready y de Ready/Suspend -> Blocked.**

Short Term Scheduler:

- Decide a cuál de los procesos en la cola de listos se elige para que use la CPU.
- **Pasa de Ready -> Running y de Running -> Waiting/Sleep.**



Procesos 3

Procesos CPU-Bound:

- Mayor parte del tiempo utilizando la CPU.

Procesos I/O-Bound:

- Mayor parte del tiempo esperando por I/O.

Comportamiento de los procesos:

- Surge la necesidad de atender rápidamente procesos I/O-bound para mantener el dispositivo ocupado y aprovechar la CPU para procesos CPU-bound.

Planificación:

- Necesidad de determinar cuál de todos los procesos que están listos para ejecutarse, se ejecutará a continuación en un ambiente multiprogramado.

Algoritmo de Planificación:

- Algoritmo utilizado para realizar la planificación del sistema.

Algoritmos Apropiativos:

- En los algoritmos Apropiativos (preemtive) existen situaciones que hacen que el proceso en ejecución sea expulsado, por el **Short term Scheduler**, de la CPU.
- La apropiación esta relaciona al recurso CPU.
- Ejemplos de algoritmos Apropiativos:
  - ❖ Round Robin.
  - ❖ STRF.
  - ❖ Prioridades Apropiativo.
- El proceso puede ser expulsado de la CPU según la planificación implementada:
  - ❖ Se le termina su quantum (Algoritmo round robin).
  - ❖ Llega a la cola de listos un proceso de mayor prioridad (Algoritmo prioridades apropiativo).
  - ❖ Llega a la cola de listos un proceso con menor tiempo restante (Algoritmo SRTF).

Algoritmos No Apropiativos:

- En los algoritmos No Apropiativos (nonpreemptive) los procesos se ejecutan hasta que el mismo (por su propia cuenta) abandone la CPU.
- La apropiación esta relaciona al recurso CPU.
- Ejemplos de algoritmos No Apropiativos:
  - ❖ FCFS.
  - ❖ SJF.
  - ❖ Prioridades No Apropiativos.
- El proceso deja el estado de ejecución solo cuando:
  - ❖ Termina (Syscall Exit).
  - ❖ Se bloquea voluntariamente (SysCall wait, sleep, etc).

- ❖ Solicita una operación de E/S bloqueante (Syscall Read, Write, etc).

Categorías de los Algoritmos de Planificación:

- Según el ambiente es posible requerir algoritmos de planificación diferentes, con diferentes metas:
  - ❖ **Equidad:** Otorgar una parte justa de la CPU a cada proceso.
  - ❖ **Balance:** Mantener ocupadas todas las partes del sistema.
- Ejemplos:
  - ❖ Procesos por lotes (batch).
  - ❖ Procesos Interactivos.
  - ❖ Procesos en Tiempo Real.

Procesos Batch:

- No existen usuarios que esperen una respuesta en una terminal.
- Se pueden utilizar algoritmos no apropiativos.
- Metas propias de este tipo de algoritmos:
  - ❖ **Rendimiento**: Maximizar el número de trabajos por hora.
  - ❖ **Tiempo de Retorno**: Minimizar los tiempos entre el comienzo y la finalización.
  - ❖ El Tiempo de espera se puede ver afectado.
  - ❖ **Uso de la CPU**: Mantener la CPU ocupada la mayor cantidad de tiempo posible.

Procesos Interactivos:

- No solo interacción con los usuarios, un servidor por ejemplo necesita de varios procesos para dar respuesta a diferentes requerimientos.
- Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU.
- Metas propias de este tipo de algoritmos:
  - ❖ **Tiempo de Respuesta**: Responder a peticiones con rapidez.
  - ❖ **Proporcionalidad**: Cumplir con expectativas de los usuarios.

Política Versus Mecanismo:

- El algoritmo de planificación debe estar parametrizado, de manera que los procesos/usuarios pueden indicar los parámetros para modificar la planificación.
- El Kernel implementa el mecanismo.
- El usuario/proceso/administrador utiliza los parámetros para determinar la Política.

Procesos 4

Creación de procesos:

- Un proceso es creado por otro proceso.
- Un proceso padre tiene uno o más procesos hijos.
- Se forma un árbol de procesos.

Actividades en la creación:

- Crear la PCB.
- Asignar PID único.
- Asignar memoria para regiones (Stack, Text y Datos).
- Crear estructuras de datos asociadas (Fork -> copiar el contexto, regiones de datos, text y stack).

Relación entre procesos Padre e Hijo:

- **Con respecto a la Ejecución:**
  - ❖ El padre puede continuar ejecutándose concurrentemente con su hijo.
  - ❖ El padre puede esperar a que el proceso hijo (o los procesos hijos) terminen para continuar la ejecución.
- **Con respecto al Espacio de Direcciones:**
  - ❖ El hijo es un duplicado del proceso padre (caso Unix).
  - ❖ Se crea un **nuevo** espacio de direcciones copiando el del padre (caso Unix).
  - ❖ Se crea el proceso y se le carga adentro el programa (caso Windows).
  - ❖ Se crea un nuevo espacio de direcciones vacío (caso Windows).

Creación de Procesos:

- **Para Unix:**
  - **system call fork()** crea nuevo proceso igual al llamador.
  - **system call execve(),** generalmente usada después del **fork**, carga un nuevo programa en el espacio de direcciones.
- **Para Windows:**
  - **system call CreateProcess()** crea un nuevo proceso y carga el programa para ejecución.

System call fork():

- La system call fork() además de crear un nuevo proceso idéntico al proceso padre retorna 2 valores:
  - **Retorna 0 en el proceso hijo**.
  - **Retorna un valor > 0 o un valor < 0 en el proceso padre**, si el valor retornado es mayor que 0, entonces el padre recibe el PID del proceso hijo, pero si el valor retornado es menor que 0, entonces ocurrió un error en la creación, por lo tanto, el proceso hijo no se crea.

Terminación de procesos:

- Ante un (**exit**), se retorna el control al sistema operativo.
  - El proceso padre puede esperar recibir un código de retorno (via **wait**). Generalmente se lo usa cuando se requier que el padre espere a los hijos.
- Proceso padre puede terminar la ejecución de sus hijos (**kill**)
  - La tarea asignada al hijo se terminó.
  - Cuando el padre termina su ejecución habitualmente no se permite a los hijos continuar pero puede ser posible que sí. Normalmente ocurre una terminación en cascada.

Procesos Cooperativos:

- Afecta o es afectado por la ejecución de otros procesos en el sistema.
- Sirven para:
  - Compartir información (por ejemplo, un archivo).
  - Acelerar el cómputo (separar una tarea en sub-tareas que cooperan ejecutándose paralelamente).

- ❖ Planificar tareas de manera tal que se puedan ejecutar en paralelo.

Procesos Independientes:

- El proceso no afecta ni puede ser afectado por la ejecución de otros procesos. No comparte ningún tipo de dato.

Memoria 1

Memoria

- La organización y administración de la "memoria principal es uno de los factores más importantes en el diseño de los S. O.
- Los programas y datos deben estar en el almacenamiento principal para:
  - ❖ Poderlos ejecutar.
  - ❖ Referenciarlos directamente.
- El SO debe:
  - ❖ Llevar un registro de las partes de memoria que se están utilizando y de aquellas que no.
  - ❖ Asignar espacio en memoria principal a los procesos cuando estos la necesitan.
  - ❖ Liberar espacio de memoria asignada a procesos que han terminado.
  - ❖ Generar un uso eficiente de la memoria con el fin de alojar el mayor número de procesos.
  - ❖ Lograr que el programador se abstraiga de la alocación de los programas.
  - ❖ Brindar seguridad entre los procesos para que unos no accedan a secciones privadas de otros.
  - ❖ Brindar la posibilidad de acceso compartido a determinadas secciones de la memoria (librerías, código en común, etc.
  - ❖ Garantizar la performance del sistema.

Administración de Memoria:

- División Lógica de la Memoria Física para alojar múltiples procesos garantizando protección.

- Depende del mecanismo provisto por el Hardware.
- Busca una asignación eficiente, es decir, contener el mayor número de procesos para garantizar el mayor uso de la CPU.

Requisitos:

- **Reubicación**:
  - ❖ El programador no debe ocuparse de conocer donde será colocado en la Memoria RAM.
  - ❖ Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones.
  - ❖ Las referencias a la memoria se deben "traducir" según ubicación actual del proceso.
- **Protección**:
  - ❖ Los procesos NO deben referenciar/acceder a direcciones de memoria de otros procesos a menos que tengan permisos.
  - ❖ El chequeo se realiza durante la ejecución ya que NO es posible anticipar todas las referencias a memoria que un proceso puede realizar.
- **Compartición**:
  - ❖ Permitir que varios procesos accedan a la misma porción de memoria.
  - ❖ Permite un mejor uso/aprovechamiento de la memoria RAM, evitando copias innecesarias (repetidas) de instrucciones.

Abstracción - Espacio de Direcciones:

- Rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos.
- El tamaño depende de la Arquitectura del Procesador.
- Es independiente de la ubicación "real" del proceso en la Memoria RAM.

Direcciones Lógicas:

- Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria.
- Representa una dirección en el "Espacio de Direcciones del Proceso".

- Si la CPU trabaja con direcciones lógicas, para acceder a memoria principal, se deben transformar en direcciones físicas.

Direcciones Físicas:

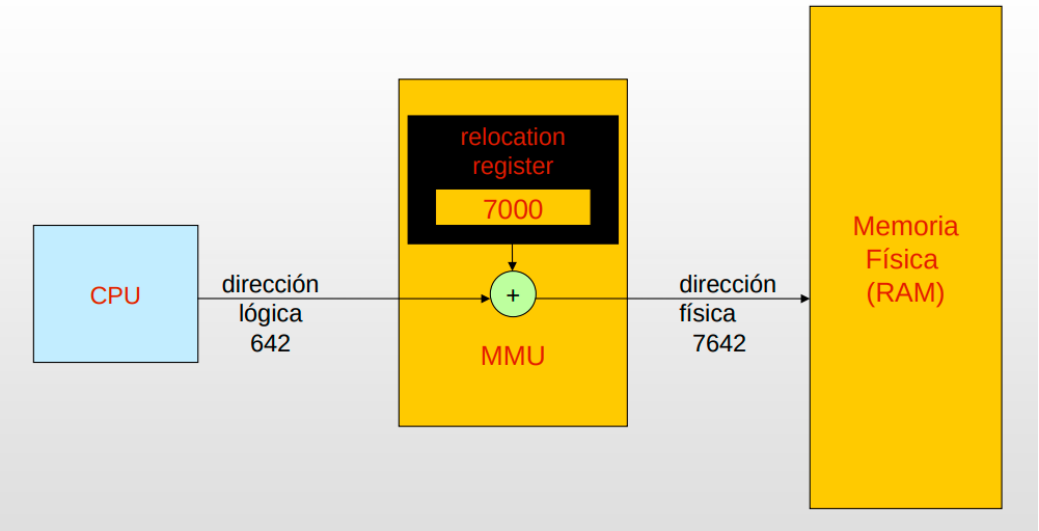
- Referencia una localidad en la Memoria Física (RAM).

Conversión de Direcciones:

- Es necesaria en caso de usar direcciones Lógicas.
- Una forma simple de hacer esto es utilizando registros auxiliares:
  - ❖ Registro Base:
    - ➢ Dirección de comienzo del Espacio de Direcciones del proceso en la RAM.
  - ❖ Registro Limite:
    - ➢ Dirección final del proceso o medida del proceso. Es el tamaño de su espacio de direcciones.
- Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado a memoria.
- Varían entre procesos.

Memory Management Unit (MMU)

- Dispositivo de Hardware que mapea direcciones virtuales a físicas. Es parte del Procesador y la reprogramación de la MMU solo la puede hacer el Kernel.
- El valor en el "registro de realocación" es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria y los procesos nunca usan direcciones físicas.



Mecanismos de asignación de memoria:

- **Particiones Fijas: El primer esquema implementado:**
  - ❖ La memoria se divide en particiones o regiones de tamaño Fijo (pueden ser todas del mismo tamaño o no).
  - ❖ Alojan un proceso cada una.
  - ❖ Cada proceso se coloca de acuerdo a algún criterio (First Fit, Best Fit, Worst Fit, Next Fit) en alguna partición.
  - ❖ El mejor criterio es el Best Fit.
  - ❖ Genera Fragmentación Interna.
- **Particiones dinámicas: La evolución del esquema anterior:**
  - ❖ Las particiones varían en tamaño y en número.
  - ❖ Alojan un proceso cada una.
  - ❖ Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso.
  - ❖ Genera Fragmentación Externa.
  - ❖ El mejor criterio es el Worst Fit.

Fragmentación:

- La fragmentación se produce cuando una localidad de memoria no puede ser utilizada por no encontrarse en forma contigua.

Fragmentación Interna:

- Es la porción de la partición que queda sin utilizar.

Fragmentación Externa:

- Son huecos que van quedando en la memoria a medida que los procesos finalizan.
- Al no encontrarse en forma contigua puede darse el caso de que tengamos memoria libre para alocar un proceso, pero que no la podamos utilizar.
- Para solucionar el problema se puede acudir a la compactación, pero es muy costosa.

Problemas del esquema Registro Base + Registro Límite:

- Necesidad de almacenar el Espacio de Direcciones de forma continua en la Memoria Física.

- Los primeros SO definían particiones fijas de memoria, luego evolucionaron a particiones dinámicas.
- Fragmentación.
- Mantener "partes" del proceso que no son necesarias.
- Los esquemas de particiones fijas y dinámicas no se usan hoy en día.
- **Las soluciones posibles son la Paginación y la Segmentación.**

Paginación:

- Memoria Física es dividida lógicamente en pequeños trozos de igual tamaño llamados **Marcos.**
- Memoria Lógica (espacio de direcciones) es dividida en trozos de igual tamaño que los **Marcos** llamados **Páginas.**
- El SO debe mantener una **tabla de páginas** por cada proceso, donde cada entrada contiene (entre otras) el Marco en la que se coloca cada página.
- La dirección lógica se interpreta como: Un número de página y un desplazamiento dentro de la misma.
- Puede producir **fragmentación interna**.
- Tiene como ventaja que el tiempo de búsqueda de espacio en RAM es nulo.

Segmentación:

- Esquema que se asemeja a la "visión del usuario". El programa se divide en **partes/secciones**.
- Un programa es una colección de **segmentos**. Un **segmento** es una unidad lógica como:
    - ❖ Programa Principal, Procedimientos y Funciones, variables locales y globales, stack, etc.
- Todos los segmentos de un programa pueden no tener el mismo tamaño (código, datos, rutinas).
- Las direcciones Lógicas consisten en 2 partes:
    - ❖ Selector de Segmento.
    - ❖ Desplazamiento dentro del segmento.
- Utiliza una **Tabla de Segmentos** que:
    - ❖ Permite mapear la dirección lógica en física. Cada entrada contiene:
        - ➢ **Base**: Dirección física de comienzo del segmento.

- **Limit**: Longitud del Segmento.

- El **Segment-table base register (STBR):** apunta a la ubicación de la tabla de segmentos.
- **Segment-table length register (STLR):** cantidad de segmentos de un programa.
- Puede causar **fragmentación externa**.
- Más de un proceso puede utilizar un mismo **segmento**, por lo tanto, comparten **direcciones físicas**. Con esta misma lógica e genera **protección en la memoria**.

Segmentación Paginada:

- **La paginación**:
  - Es transparente al programador.
  - Elimina Fragmentación externa.
- **La segmentación**:
  - Es visible al programador.
  - Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección.
- **Segmentación Paginada**:
  - Cada segmento es dividido en páginas de tamaño fijo.
  - Cada segmento tiene una tabla de páginas que le corresponde.
  - Toma las ventajas de ambas: compartición, protección (de parte de la segmentación), evitar fragmentaciones (de parte de la paginación).
  - Se sigue guardando en paginas
- La unidad de trabajo para subir o bajar de la RAM es la página.