¿Qué es un Sistema Operativo?

Un Sistema Operativo (SO) es el software que actúa como intermediario entre el hardware de la computadora y el usuario. Desde una perspectiva 'de arriba hacia abajo', abstrae y administra el hardware, mientras que desde una perspectiva 'de abajo hacia arriba', organiza los procesos y provee servicios al usuario. En resumen

- Gestiona el HW.
- Controla la ejecución de los procesos.
- Interfaz entre aplicaciones y HW
- Actúa como intermediario entre un usuario de una computadora y el HW de la misma

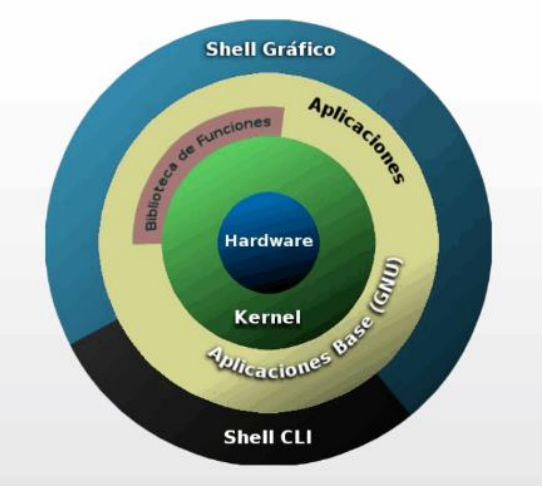
Objetivos de los SO:

- Comodidad
- Eficiencia
- Evolución

Por ejemplo, un sistema operativo facilita que el usuario interactúe de forma cómoda con las aplicaciones (Comodidad), asegura el uso óptimo de recursos como la CPU y la memoria (Eficiencia), y permite la integración de nuevas tecnologías o funcionalidades (Evolución).

Componentes de un SO:

- **Kernel**
- **Shell**
- Herramientas



Kernel (Núcleo):

Se halla en memoria principal y se encarga de la administración de los recursos.

Implementa servicios esenciales como

- Manejo de memoria
- Manejo de la CPU
- Administración de procesos
- Comunicación y concurrencia
- Gestión de la E/S

Servicios de los SO:

Servicios del Sistema Operativo incluyen:

- **Administración de Memoria**: Control del uso de la memoria física y virtual.
- **Administración del Almacenamiento**: Gestión de discos y sistemas de archivos.
- **Administración de Dispositivos**: Control y comunicación con periféricos como impresoras y discos externos.
- **Detección de errores y respuestas**: Manejo de fallos en el sistema.
- **Interacción del Usuario (Shell)**: Proveer una interfaz que permita al usuario comunicarse con el SO.
- **Contabilidad**: Registrar el uso de recursos por los procesos para auditoría.

Problemas a evitar que se ocupa el SO

- Que un proceso se apropie de la CPU.
- Que un proceso intente ejecutar instrucciones de E/S, por ejemplo.
- Que un proceso intente acceder a un área de memoria fuera de su espacio declarado

Para evitar esto el SO entre otras cosas debe

- Gestionar el uso de la CPU
- Detectar intentos de ejecución de instrucciones E/S ilegales
- Proteger el vector de interrupciones

Para hacer esto se **apoya en el HW** mediante

- Modos de ejecuciones
- Interrupción de Clock
- Protección de la memoria

Apoyo en el HW

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.

Para pasar de un modo a otro se hace de maneras distintas

De kernel a usuario se hace mediante una instrucción especial.

De usuario a kernel solo se puede pasar mediante un trap o interrupción.

Tener en cuenta que el SO arranca en modo kernel.

Los traps del modo usuario se producirán porque intenta hacer una operación indebida para ese modo, por lo tanto, le pasa el mando al modo kernel, ahora si nos sale un trap estando ya en el modo kernel, ahí en Windows por ejemplo ocurriría una **pantalla azul**.

Interrupción por Clock:

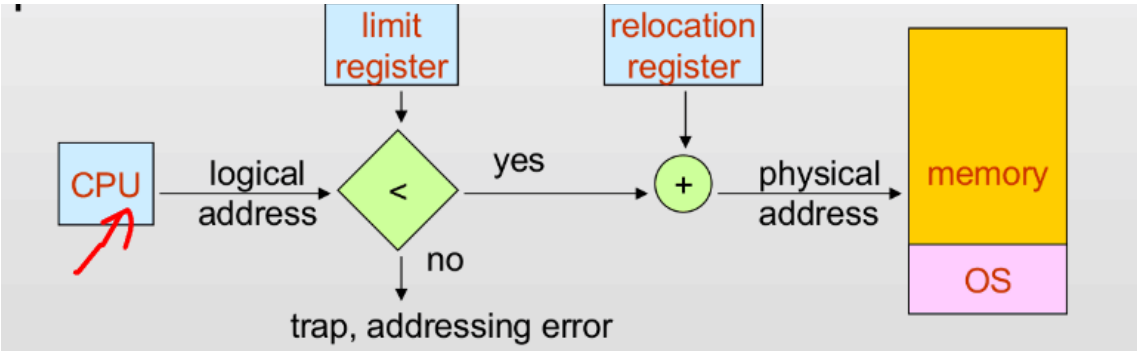
Cada tanto tiempo si el proceso no ha finalizado, directamente se finaliza el proceso para pasar a otro, de manera que un proceso no se apropie de la CPU con un while infinito, por ejemplo.

Protección de la memoria:

Delimitar el espacio de direcciones del proceso

Poner límites a las direcciones que puede utilizar un proceso

La CPU (El kernel) se encarga de que el proceso no intente pasar sus límites APOYANDOSE en HW



System calls:

Cuando un proceso necesita servicio del SO, los solicitan a través de **System Calls**, que se ejecutan en modo kernel y se invocan a través de interrupciones. (Para más detalle ver video tema 1 Clase 2 filmina 58)

Tipos de kernel:

Monolítico: toda funcionalidad que debe implementar el SO se ejecuta en modo kernel.

Este modo implica menos tiempo en la resolución de las cosas

Se apunta a la perfomance.

Microkernel: se intenta hacer lo más chico posible el kernel haciendo que se encargue lo que sí o sí debe ser hecho por él, implementando que el modo usuario de apoyo a este.

Este modo implica una mayor seguridad a que no haya errores que se producen en el modo kernel (lo de la pantalla azul, por ejemplo)

Se apunta a la seguridad.

La mayoría de procesadores utiliza microkernel, Linux y Windows son uno de ellos.

Procesos

Un proceso es un programa en ejecución, que cambia con el tiempo y requiere recursos del sistema (como CPU, memoria y dispositivos de E/S) para funcionar. Es una entidad dinámica que nace cuando se inicia y muere al completarse. Va llevando un PC (program counter). Se lo considera como dinámico a diferencia de un programa que se lo considera estático ya que no cambia.

Los SO modernos permiten múltiples procesos al mismo tiempo, pero solo uno puede ejecutarse a la vez en un solo procesador.

Los procesos están compuestos por: sección de código, sección de datos, pilas o stacks (para pasar parámetros, guardar datos temporales, etc). También usa la CPU.

Las pilas o stacks se crean automáticamente y están compuestas por stack frames. Hay una pila para cada modo (kernel y usuario).

Atributos de un proceso: identificador (para distinguirlo del resto), estado, prioridad, contador de programa, punteros a memoria, datos de contexto, información de estado de E/S, información de auditoría (tiempo de procesador y reloj utilizados).

Todos los atributos se almacenan en la **PCB (Process Control Block)**, que es un gran registro. Existe una por proceso y es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina.

Espacio de direcciones: Conjunto de direcciones de memoria que ocupa el proceso

No se incluye el PCB ni tablas asociadas

Un proceso en modo usuario solo puede acceder a su espacio de direcciones mientras que en modo kernel puede acceder a estructuras internas como el PCB o a espacios de direcciones de otros procesos.

Contexto de un proceso: Incluye toda la información que el SO necesita para administrar el proceso, y la CPU para ejecutarlo correctamente.

Son parte del contexto, los registros de cpu, inclusive el contador de programa, prioridad del proceso, si tiene E/S pendientes, etc.

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. 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. El tiempo que consume depende del soporte de HW

<u>Estados de un proceso:</u>

Nuevo: se inicializa y se carga en memoria

Listo: Está en memoria, está listo para ejecutarse. Compite por obtener tiempo de CPU.

Ejecutando: Ya tiene CPU, puede pasar a saliente si termina, a listo si se le termina el quantum o es interrumpido por otro o a bloqueado si realiza E/S.

Saliente: Se empiezan a eliminar las estructuras que se cargaron en memoria. Al final, se borra la PCB y el proceso deja de existir.

Bloqueado: acá esperan mientras realizan E/S, y cuando termina vuelven a listo.

Cada estado posee una o varias colas de planificación, que enlazan PCBs de los procesos.

Los cambios entre estados son realizados por los módulos de planificación:

Short Term Scheduler (STS): selecciona entre los procesos listos cuál es el que se ejecuta. Tiene asociado el Dispatcher que realiza el cambio de contexto para que se ejecute el proceso.

Medium Term Scheduler (MTS): reduce el grado de multiprogramación (cantidad de procesos que tienen memoria con capacidad de ejecutarse), saca temporalmente de memoria los procesos. Tiene asociado el Swapper que realiza el swap out y swap in (sacar y llevar a memoria).

Long Term Scheduler (LTS): admite los procesos al estado de listo, determina qué procesos se cargan en memoria. Tiene asociado al Loader que se encarga de cargar el espacio de direcciones del proceso.

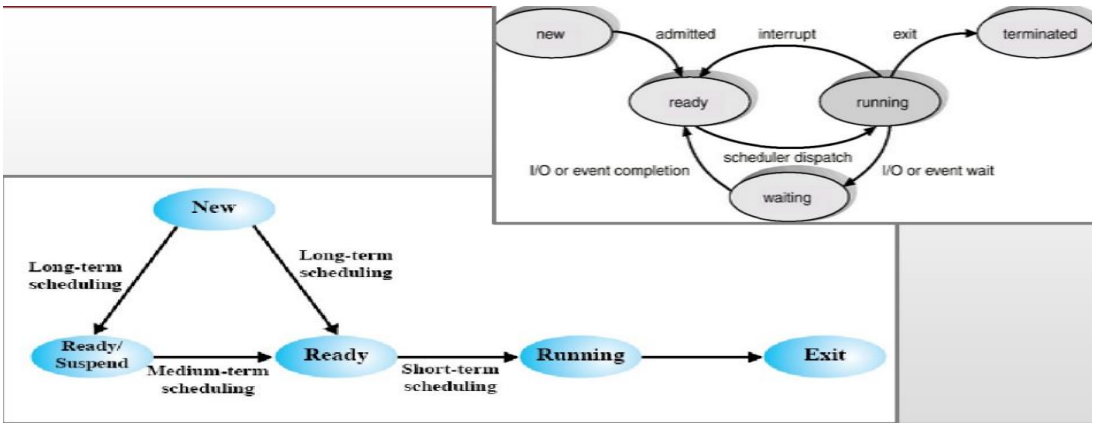
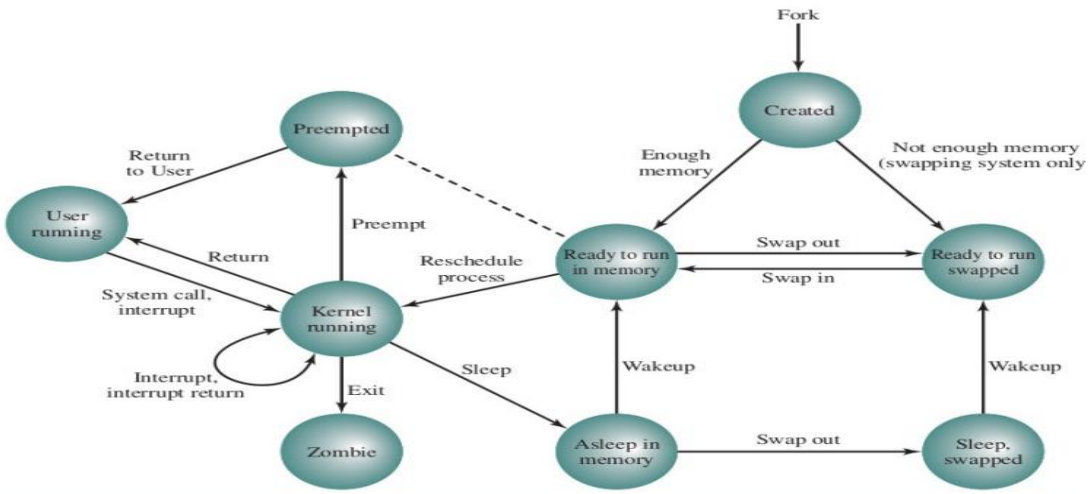


Diagrama de transiciones UNIX



Algoritmos de Planificacion

La planificación de la CPU también necesita un criterio para determinar a qué proceso se le asigna CPU. Para esto existen algoritmos de planificación.

Estos algoritmos pueden ser:

Apropiativos si el proceso en ejecución puede ser interrumpido y llevado a la cola de listos (se desperdicia CPU en los cambios de contexto).

No apropiativos si el proceso está en ejecución hasta que termine o se bloquee por algún evento.

Algunos de los algoritmos son:

- FIFO (primero en llegar, primero en ejecutarse).
- SJF (el proceso más chico primero).
- RR (los procesos se ejecutan por un quantum, que puede ser fijo o variable. cuando termina el quantum, se expulsa al proceso de la CPU y se coloca al final de la cola de listos).
- PRIORIDADES (los procesos tienen prioridades, se ejecutan en base a ellas).
- SRTF (versión apropiativa de SJF).
- VRR (igual que RR, pero cuando un proceso vuelve de E/S, tiene prioridad).

Los algoritmos pueden estar parametrizados para modificar la planificación.

Los procesos pueden ser de 2 tipos:

Batch (no hay usuarios esperando una respuesta, se pueden usar algoritmos no apropiativos)

Interactivos (tienen interacción con el usuario, conviene apropiativos).

También pueden ser **independientes** (si no afecta ni puede ser afectado por otros) o **cooperativos** (si afecta o es afectado por otros).

Los procesos siempre son creados por otro proceso. El proceso padre de todos nace ya creado ya que, al iniciar el SO, el loader lo carga. Cuando se crea un proceso, se crea la PCB y se le asigna un PID único. Hay 2 modelos para la relación padre-hijo: compiten por igual por CPU o el padre espera a que termine el hijo para ejecutarse. En Unix el espacio de direcciones del proceso hijo es un duplicado del proceso padre. Se crean procesos usando la SC fork() y se carga el programa en el espacio de direcciones con execve(). El fork copia el código para el hijo y dentro hay una condición para saber de quién es el código, si retorna 0, se ejecutó el hijo y si se retorna algo mayor a 0, el hijo se creó correctamente. Si retorna algo negativo, hubo un error.

Para finalizar los procesos se utiliza exit y el control del proceso se devuelve al SO.

Memoria

Los programas y datos deben estar en memoria principal para poder poderlos ejecutar y 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

Ademas debe

- Lograr que el programador se abstraiga de la alocacion de los programas
- Brindar seguridad entre procesos para que no accedan a donde no deben
- Brindar acceso compartido a determinadas secciones como librerías, código común, etc.
- Garantizar la perfomance del sistema.

Administración de memoria

Es de los factores más importantes del diseño de los SO. El objetivo del diseño de la administración de la memoria es abstraer al programador, brindar seguridad, brindar acceso compartido, garantizar la performance del sistema, etc.

Se deben tener en cuenta algunos requisitos:

Reubicación: Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones.

Protección: los procesos NO deben referenciar (acceder) a direcciones de memoria de otros procesos (salvo que tengan permiso).

Compartición: es para que 2 procesos puedan compartir determinada parte de memoria (como por ejemplo rutinas comunes, librerías, espacios explícitamente compartidos, etc).

Las direcciones de memoria pueden ser: Lógicas (son las que utilizan los procesos, es independiente al lugar de la RAM donde se almacena) o Físicas (espacio físico de la memoria, es la dirección absoluta). Si se usan direcciones lógicas, se necesita una conversión, llamada resolución de direcciones (address-binding), que generalmente se hace en tiempo de ejecución. Esta resolución es realizada por el **MMU** que es un dispositivo HW que mapea direcciones virtuales a físicas a través de un registro base y límite.

Mecanismos de asignación de memoria

Con el fin de que varios procesos usen la memoria, se realizan particiones a las mismas. Estas pueden ser Fijas si se realizan de antemano o Dinámicas si se realizan a medida que llegan los procesos. La desventaja de las fijas es que puede ocurrir FI (Fragmentación Interna) y en las dinámicas puede ocurrir FE (Fragmentación Externa). FI es que quede memoria libre dentro de la partición, y FE es que quede memoria libre entre particiones.

Existen diferentes **técnicas de administración de memoria**:

Paginación: el espacio de los procesos se divide en partes iguales llamadas páginas. La memoria RAM se divide en partes iguales llamadas marcos. Cualquiera de las páginas puede ocupar cualquiera de los marcos. Existe una tabla de páginas para organizar, que indica en qué marco está cada página y sirve para que el MMU pueda traducir. Cada proceso tiene su tabla de páginas y sirve para proteger la memoria. Provoca FI ya que como el tamaño de las páginas es fijo y, en general, el último bloque de datos o instrucciones de un proceso no coincide exactamente con el tamaño de la página.

La principal ventaja de la paginación es evitar los problemas de la fragmentación.

Segmentación: cada parte de un espacio de direcciones (librerías, código, datos, pilas) se le llama segmento y estos se pueden cargar en la memoria de manera no continua. Se tiene una tabla de segmentos, que indica dirección base y límite del mismo. Puede provocar FE, pero es más sencillo respetar los principios de compartir y proteger la memoria.

En ambas no existe continuidad. (refiere a que el espacio de direcciones lógicas (el que usa el proceso) no tiene que corresponderse de manera directa y continua con el espacio de direcciones físicas (la memoria RAM) en el que se almacenan los datos o instrucciones).

Segmentación paginada: Se combinan ambas, primero se segmenta y después se página. Para cada segmento hay una tabla de páginas. Si se comparte un segmento, se comparten todas sus páginas.

Conjunto Residente o Working Set: porción del espacio de direcciones del proceso que se encuentra en memoria.

Memoria virtual: mecanismo que permite la ejecución de procesos que no se encuentren completamente en memoria. Sirve para liberar a los programadores de las limitaciones relativas al espacio de la memoria, el cual ahora será impuesto por el HW y el bus de direcciones. Para aplicar esta técnica se necesita: apoyo del hardware para ir metiendo páginas a medida que se necesite, un dispositivo de almacenamiento secundario para resguardar todo lo que no está cargado en la memoria principal (como un disco) y programar al SO para que cargue y descargue las páginas.

Para usar esta técnica se utiliza **paginación por demanda**, que consiste en cargar a memoria principal las páginas que se necesitan en un momento dado. Se usa el **bit V** para saber si la página está o no en memoria y el **bit M** para saber si fue modificada y hay que reflejar cambios. El bit V lo cambia el SW y el bit M lo cambia el HW. Cuando el bit V es 0, ocurre fallo de página o page fault, se genera una trap/excepción/interrupción por software. Si los page faults son excesivos, la performance del sistema decae.

Cada proceso tiene su tabla de páginas, que debe estar en memoria principal para que el MMU pueda traducir. Existen varias maneras de organizarla:

- **Tabla única lineal**
- **Tabla de 2 niveles**: se puede paginar la tabla de páginas (ósea las de segundo nivel sacarlas de memoria principal). El primer nivel posee las direcciones bases de la tabla de 2do nivel. La desventaja es que hay que acceder 2 veces para traducir direcciones
- **Tabla invertida (Hashing)**: la tabla solo posee entradas de tabla de páginas que se encuentran en memoria principal. Si la entrada de la página que se busca no está en la tabla, ocurre el fallo de página.

Las tablas de páginas multinivel:

Tiene como objetivo tener múltiples tablas de páginas, pero de menor tamaño,

La resolución de una dirección, podría causar varios accesos a la memoria.

Busca que la tabla de páginas ocupe menor cantidad de memoria RAM cuando corresponda.

La tabla de páginas puede no residir completa en memoria y se sea cargada bajo demanda.

Tamaños de página: pequeños (menos FI, tablas de páginas más grandes, más páginas en memoria), grandes (mayor FI, más eficiente mover las páginas a memoria principal). El tamaño de página lo provee el hardware.

TLB: es una memoria caché que está en la CPU o ligada al MMU que guarda entrada de tablas de páginas. Si la página está es un TLB hit y se realiza la resolución de la dirección en ese momento. Si la página no está es un TLB miss y la traducción se hace desde la tabla de páginas y se actualiza la TLB. La política de reemplazo de la TLB es LRU.

Si todas las entradas buscadas generan un TLB MISS, su uso perjudica al tiempo de resolución, Los cambios de contexto invalidan la TLB

Para poder ejecutarse, a un proceso se le deben asignar marcos de memoria. Esta asignación puede ser fija (equitativa o proporcional) o dinámica.

Al momento de seleccionar una página víctima, el reemplazo puede ser global o local (generalmente se usa la local).

Los algoritmos de reemplazo son: OPTIMO, FIFO (puede ser con segunda chance), LRU y NRU.

Hiperpaginación o Thrashing: Ocurre cuando el SO se pasa más tiempo cargando páginas o atendiendo PF que los procesos ejecutándose lo que provoca una baja importante de performance en el sistema.

Hay algunas técnicas para solucionar esto:

-**Working Set**: se basa en el modelo de localidad, que plantea que las direcciones que utiliza un proceso tienden a agruparse. Entonces, se define un delta para tomar una cierta cantidad de páginas hacia atrás o adelante y establecer la cantidad de marcos que se consideren óptimos para cubrir la ejecución de esa localidad. Si se elige un delta chico, no cubrirá la localidad y si se elige uno grande puede cubrir más de una localidad. Esta técnica es muy compleja y costosa.

-**Frecuencia de fallo de página (PFF)**: establece una cota superior e inferior de fallos de página, para asignarle marcos al proceso o sacarle marcos en base a la cantidad de fallos que tenga. Para usar esta técnica es necesario usar reemplazo local.

Algunos otros servicios del SO:

Demonio de Paginación: Proceso del sistema operativo que se ejecuta en segundo plano y realiza actividades de administración de memoria (ej: barrer páginas modificadas)

Memoria Compartida: Servicio que permite a los procesos compartir memoria para optimizar su uso. Puede ser por eficiencia o que los procesos quieran explícitamente compartir memoria

Mapeo de Archivo en Memoria: Técnica que vincula el contenido de un archivo a una región del espacio de direcciones virtuales de un proceso.

Copia en Escritura: Estrategia que permite a los procesos compartir páginas de memoria, duplicándolas sólo cuando se modifican.