# SISTEMA OPERATIVO

Partes principales de una computadora como CPU, RAM, BUS, DISPOSITIVOS DE E/S (disco, monitor, lectora,etc)

Diagrama de capas:

HARDWARE -> OS(modo kernel) -> API (modo usuario) -> aplicaciones

**Deinfinimos el sistema operativo según sus funcionamientos:**

1. Como interfaz en ususario/hardware (maquina extendida)
2. Gestor de recusros.

Intuitivamente se podría considerar al sistema operativo como el software que ejecuta en modo kernel, sin embargo, no todo el OS funciona en su totalidad en modo kernel.

Según la forma 1 de definir el OS, se lo puede interpretar como:

El hardware de nuestra PC cuenta con una programación absolutamente compleja y de distintos lenguajes entre los distintos dispositivos de hardware que tengamos, entonces aprender como funciona cada uno con sus respectivos lenguaje nos costaría un montón, entonces es aquí donde el OS entra en juego y funciona como una capa de abstracción de todos esos lenguajes complejos y nos brinda una interfaz “bonita” con la que podemos interactuar mediante funciones mas entendibles y accesibles para el usuario.

Respecto a la forma 2 de definir el OS:

Si consideramos que hay hardware que quiere ser usado por cada programa y cada usuario de la computadora es necesario realizarlo de cierta manera que para que esto se lleve a cabo de manera ordenada. Es por esto que el OS entra en juego a través de la técnica de multiplexación, tanto tiempo como en espacio.

**Multiplexacion en tiempo**: se realiza mediante la asignación de turnos a los distintos programas para que utilices los recursos de nuestra PC. Otorgando asi un tiempo de uso de los recursos a cada programa, generando asi un uso determinado para un programa y luego seguirá el que tenga el turno siguiente y asi.

**Multiplexacion en espacio:** se logra asignando una porción del recurso a cada programa que quiere ejecutarse, como por ejemplo, la memoria RAM la cual se puede proporcionar asignado direcciones y espacio de memoria a las distintas aplicaciones que quieren ejecutarse

# TIPOS DE SITEMAS OPERATIVOS

**Mainframe:** mas alto nivel, diseñados para procesar muchos trabajos a la vez, donde la mayoría de actividades requieren operaciones de E/S.

**Servidor:** de un nivel mas bajo, diseñador para funcionar en un hardware de PC mas grande, con capacidades de mayor velocidad, grandes capacidades de almacenamiento, entre otros. Principalmente se utilizan para almacenar bases de datos de menor tamaño que las que se almecenan en los mainframes, compartir recursos como archivos y carpetas e impresoras, también para aplicaciones web, generalmente los ISP los utilizan.

**Multiprocesador:** computadoras con mayor cantidad de cores y dedicadas al poder de computo y calculo. Hay 3 tipos, computadoras paralelas, multicomputadores y multiprocesadores. Existen desde hace años, el mas usado UNIX

**Computadoras personales:** son los mas utilizados, optimizados para ser utilizado por un solo usuario. Utilizado para procesar texto, navegar en internet, jugar videojuegos.

**Computadoras de mano:** sistema operativo orientados para que entren en la palma de la mano, el ejemplo perfecto son los smartphones, mercado liderado por Android, iOS.

# CLASIFICACION GENERAL

Se pueden clasificar en 2 grandes grupos:

* **proprosito general:** permiten instalar aplicaciones que queramos para poder realizar cualquier tarea que necesitemos.
* **Propósito especifico:** orientados para ejecutar un conjunto pequeño de tareas especificas, ejemplo, estación meteorológica donde solamente se miden presión atmosferica, temperatura, etc.

Entre los de **propósito especifico** podemos destacar los siguientes OS’s:

* **Sistemas operativos embebidos**: son aquellos que se utilizan como elemento de control para aquellos dispositivos de uso diario, los cuales no necesitan grandes prestaciones, tales como televisores no inteligentes, microondas, reproductor DVD, entre otros. No permiten la instalación de aplicaiones externas porque lo que siempre operan con un software seguro brindado generalmente por el fabricante
* **Sistemas operativos de tiempo real:** caracterizados por tener el tiempo como característica determinante en sus procesos. Basandose en el sensado de agentes externos del sistema. Controlando que determinadas acciones se realicen en un periodo acotado de tiempo, siendo esto lo correcto. Mientras que si esas acciones no se realizan en el debido tiempo, pueden surgir problemas e incluso alguna catástrofe.

LLAMADAS AL SISTEMA (SYSTEM CALLS)

Son un capa de abstracción de tipo API. Brindando al programador una librería de funciones pudiendo acceder a las funciones que tiene el OS. Es decir, por ejemplo cuando declaramos una variable en C mediante **int c;** no estamos asignando el espacio en memoria de manera explicita pero si lo realiza cuando se complica llamando a la system call correspondiente para hacerlo.

El hecho de ser una librería de funciones es debido a que debe ocultar todo tipo de implementación que exista por debajo de su nivel, que es básicamente el hardware.

Cada OS tiene su propia librería implementada, veremos las que se basan en el estándar POSIX.

**P:** portable

**O:** operative

**S:** system

**I:** interface

**X:** for X (UNIX)

que no es otra cosa que una familia de estándares que determinan como debe implementase la API, estableciendo las funciones necesaria y minimas, como asi tambin estableciendo como tiene que ser la consola de línea de comando.

En gran medida los servicios ofrecidos por las llamadas al sistemas determinan que tiene que hacer el SO. Las llamadas se pueden caracterizar por las tareas que realizan.

**Para administración de procesos:**

Fork(): es la principal, la única que permite crear procesos. El proceso que lo llama se llama proceso padre y el creado proceso hijo. El proceso creado es un duplicado exacto del proceso padre

Waitpid(pid, &statloc, option): permite al proceso padre esperar que termine determinado proceso o bien a su proceso hijo en ejecución

Execve(name, argv, envirop): reemplaza el código en el proceso hijo creado con fork() para que haga la determinada tarea, pues no se quiere que haga lo mismo que el proceso padre.

Exit(): termina un proceso y retorna un valor de control para determinar como fue su conclusión. Junto con waitpid y execve son las básicas para determinar la vida del proceso.

**Para administración de archivos:**

Open(file, how,…): abre un archivo para escritura, lectura o ambas. Si el archivo no existe se puede pedirle que lo cree.

Close(file): cierra el archivo simplemente.

Read(fd, buffer, nbyte): lee determinada cantidad de bytes del archivo, pudiendo hacerlo debido a la existencia de un puntero especial que se mueve automáticamente cuando se realiza una lectura nueva.

Write(fd, buffer, nbyte): escribe determinada cantidad de bytes que se encuentran en memoria, utilizando tambien, el puntero especial del sistema.

Lseek(fs, offset, whence): se utiliza para ubicar un puntero del SO en una ubicación especifica del archivo, estableciendo cual es el desplazamiento dentro del archivo, teniendo de referencia el principio o el final del archivo.

Stat(name, &buff): devuelve la información del estado actual del archivo, tal como tamaño, fec modificación.

**Para administración de directorios:**

Mkdir(name, mode): crea un directorio nuevo

Rmdir(name): elimina un directorio, pero si o si tiene que estar vacio para poder hacerlo.

Link(name1, name2): permite hacer referencia a un Directorio ubicado en otra parte e incluso con otro nombre, permitiendo asi, que los cambios relizado en cualquiera de las partes se vea reflejado en ambas.

Unlink(name): elimina la referencia del archivo, si es la ultima, espera a que nadie lo este utilizando. Equivalente a eliminar el archivo.

Mount(special, name, flag): permite acoplar dos sistemas de archivos en uno solo, ejemplo cuando se conecta un pendrive.

Unmount(special): desacopla un sistema de archivos

**Para usos misceláneos:**

Alguna de estas se relacionan con comandos de la consola:

Chdir(dirname): cambia el directorio de trabajo actual, para no estar escribiendo direcciones absolutas.

Chmode(name, mode): cambia los permisos de acceso a un directorio

Kill(pid, signal): envia una señal para que un proceso, por ejemplo, pare o termine.

Time(&second): devuelve la cantidad de segundos transcurridos desde 1-1-70

**API WIN 32:**

Esta es la API que tiene Windows, donde se implementan las miles de llamadas al sistema que Microsoft maneja, a comparación con el centenar de POSIX.

Un problema de estas llamadas es que no todas ejecutan en modo kernel, hay muchas que ejecutan en modo usuario.

(ver cuadro de comparación de diapositivas)

ESTRUCTURA DEL SISTEMA OPERATIVO

El núcleo o kernel del SO forma parte de esta estructura. La estructura del OS pueden clasificarse en dos grupos:

* Monolíticas
* Estructurada: capas,

**Monoliticas:**

Se caracteriza por que el SO esta ejecutándose en un solo programa. Cada procedimiento dentro del programa principal puede llamar a otro y por ende no existe ocultamiento de información. Los datos, cuando se llama a una función, se colocan en una pila (esto cumpliría el rol de estructura) y se ejecuta una trap.

Esta estructura tiene grandes desventajas que afectan gravemente al funcionamiento del SO, estas pueden ser:

* Dificil de mantener o entender, pues es un solo código, lo cual también hace que un solo fallo termine por hacer que falle completamente el SO.
* Dificil de agregar una funcionalidad nueva pues hay millones de líneas de código que en cualquier cambio pueden romperse.
* El usuario tiene acceso a los procedimiento del sistema operativo dejandolo vulnerable.

**Estructura por capas:**

Primera implementación de una mejora para la estructura monolítica, con el planteamiento de armar capas encargadas de procedimientos específicos. Todo esto fue posible a la evolución de la arquitectura de los procesadores, permitiendo definir distintos niveles de ejecucion.

La capa 0 es el hardware, mientras que la capa N es la interfaz de usuario. De esta manera un proceso que esta en la capa N puede asumir que toda tarea administrativa hacia las capas inferiores están correctamente hechas. Ejemplo de este SO es el THE (Disjkstra).

Entre algunas ventajas tenemos el hecho de cada capa solamente puede acceder a su inmediata inferior. Sin embargo la desventaja estaba en que era complejo determinar que capa corria en modo kernel y cual en modo usuario, de ahí el surgimiento de la siguiente estructura.

**Estructura micro-kernel:**

La idea principal de esta estructura era “dejar lo menos posible dentro del nucleo”, logrando asi reducir drásticamente las líneas de código y la probabilidad de bugs.

Dividiendo asi el SO en modulos y dejando solamente el micro-kernel ejecutando en modo kernel, logrando asi que cuando un modulo falle solamente se vea afectado ese y no el resto de modulos del SO.

Ejemplo de esta estructura es MINIX (SO)

**Estructura cliente-servidor:**

Es una variación del micro-kernel basado en dos procesos claves: clientes y servidores.

Puede no tener núcleo pues se basa solamente en clientes y servidores.

La comunicación entre estos procesos se realizan mediante mensajes, por lo que no es necesario que los dos esten trabajando en el mismo lugar

**MODULO II: PROCESOS E HILOS**

**PROCESO:** abstracción mas antigua e importante. Brindando la capacidad de operar concurrentemente, incluso con un solo CPU.

Todo software ejecutable, incluso el SO, se gestiona organiza mediante procesos.

**Un proceso es una instancia de un programa en ejecución, incluyendo su contexto de ejecucion.**

Teoricamente cada proceso tiene su CPU virtual propio, pero en realidad es el procesador conmutando rápidamente entre procesos, esta actividad de conmuatacion de procesos se denomina **MULTIPROGRAMACION** y la cantidad de procesos que se puedan almacenar en memoria se denomina **GRADO DE MULTIPROGRAMACION.**

**CREACION DE PROCESOS**

Hay cuatro formas o situaciones que crean procesos:

* Inicio del sistema : cuando inicia el sistema se crean procesos que están en primer plano, es decir, que tienen un interacción directa con el usuario, pero también hay procesos en segundo plano que se denominan demonios
* Llamada desde un proceso para la creación de otro: esto es básicamente que un proceso puede crear a otros, ejemplo, el administrador de tareas que hace que abra el antivirus
* Petición de usuario para crear: ejecución por parte del usuario como por ejemplo hacer doble click en un icono
* Inicio de trabajo por lotes: tarea típica del mainframe. El mainframe crea un proceso para ejecutar cada tarea que llega a la cola de entrada.

**TERMINACION DE UN PROCESO**

Los procesos pueden terminar de manera voluntaria o involuntaria. Algunos motivos son:

* Salida normal (voluntaria): básicamente cuando termina la ejecución de su codigo
* Salida por error (voluntaria): cuando se produce un error durante la ejecucion, como cuando se intenta abrir un archivo que no existe
* Error fatal (involuntaria): ocurre cuando se quiere acceder a instrucciones ilegales o se referencia una dirección de memoria inexistente, entonces el SO opta por terminar con el procesos
* Eliminado por otro proceso (involuntaria): en este caso un proceso invoca una llamada al sistema para terminar con un proceso determinado, para esto el proceso que invoca la llamada debe tener los permisos. Ejemplo debug en vscode y lo terminamos cancelando la ejecución.

**ESTADOS DE UN PROCESO**

Si bien sabemos que los procesos son entidades independientes, a veces, pueden interactuar entre si puesto que los datos de salida de uno son la entrada de otro proceso.

Sin embargo esto puede ser un problema debido a que no todos los procesos se ejecutan a la misma velocidad, por lo que suele ocurrir que un proceso que esta listo para ejecutarse se bloquea porque todavía no tiene los recursos necesario (inherente). También puede ocurrir que el SO decida detenerlo puesto que le asigna el CPU a otro proceso (tecnisismo).

Entonces teniendo en cuenta esto podemos distinguir los siguiente estados de un proceso:

* **En ejecución:** esta listo y usando el procesado en ese momento
* **Listo:** ya es ejecutable pero no se le asigno el cpu todavía
* **Bloqueado:** no puede continuar debido a que esta esperando los recursos necesarios.
* **Nuevo:** cuando a sido creado pero no se le asigno espacio en la memoria
* **Saliente:** cuando termino de ejecutarse y finalizo, aunque el procesado lo retiene para recopilar estadísticas
* **Bloqueado/suspendido:** bloqueado almacenado en disco esperando los recursos
* **Listo/bloqueado:** listo para ser ejecutado, pero almacenado en disco.

**Teniendo en cuenta el grado de multiprogramación del SO define el rendimiento del mismo.**

**ESTRUCTURAS DE CONTROL DE LOS PROCESOS**

Las estructuras de control principalmente utilizadas por el SO son tablas. Hay distintas tablas, tabla de memoria, tabla de dispositivos E/S, tabla de archivos, tabla de procesos (la importante).

* Tabla de memoria: utilizadas para la memoria y disco. Parte de la memoria es reservada para el SO y el resto para los procesos.   
  Tiene que tener almacenada en la tabla la siguiente informacion
  + Las reservas de memoria por los procesos, tanto para el programa como las variables dinamicas
  + Reservas de disco para los procesos
  + Atributos de protección que restringen el uso de la memoria o disco
  + Las reservas de memoria virtual,
* Tabla de E/S: utilzada para gestionar los dispositivos de E/S, como los discos, placas de red, etc. Contiene información sobre el estado y qué procesos los están utilizando, permitiéndole saber al SO que bloque de memoria están usando
* Tabla de archivos: contiene detalle e información sobre la implementación de los mismos
* Tabla de procesos: cada entrada en esta tabla contiene referencia a un proceso

IMPLEMENTACION DE UN PROCESO

Los procesos constan de componentes como:

* Programa: que es básicamente el código a ejecutar por el mismo.
* Datos variables: locales o globales
* Stack: estructura que utiliza para realizar la comunicación con otros procesos y con las llamadas al sistema
* Atributos: es la información que utiliza el SO para control del proceso, el conjunto de estos es denominado **PCB.**

Al conjunto de componentes de un proceso se lo llama **IMAGEN DEL PROCESO**. Generalmente esta almacenada en disco, sin embargo, para que el SO pueda gestionar el proceso una porción o en su totalidad debe estar en memoria.

Mientras que la tabla de procesos DEBE estar en su totalidad en memoria principal. Donde cada entrada de la tabla contiene un puntero a la imagen del proceso.

La PCB se dijo que es conjuntos de atributos de control el proceso, pudiéndose agrupar en 3 categorias:

* **Identificación del proceso:** PID, PPID, UID
* **Información del estado del procesador:** valores que van tomando los registros del procesador que son visibles por el ususario
* **Información de control del proceso:** es de suma importancia debido a que basado en esta información se puede detminar el estado del proceso y su nivel de prioridad, estructuras de datos para determinar si esta en una cola de espera o como asi también información para determinar si el proceso es hijo y/o padre de otro proceso, banderas o señales relativas a comunicaciones con otros procesos, privilegios como la capacidad de poder terminar con otros procesos, como asi también poder acceder a determinados bloques de memoria.

**CAMBIO DE PROCESO**

Se produce cuando el SO decide sacar de ejecución al proceso que se esta ejecutando y asigna otro nuevo al CPU. Esto ocurre principalmente por un **interrupción**, la cual puede ser de reloj, de E/S, es decir ya tiene los recursos que necesitaba y puede pasar a listo, y por fallo de memoria (mod iv). Tambien se da un cambio cuando ocurre una trap por un ejecución de una instrucción que provoca que el proceso falle y el SO determina si puede recuperarlo o directamente lo saca e incorpora otro proceso nuevo a la CPU. Y por ultimo cuando se realiza una llamada al sistema.

Resumiendo los pasos para un cambio de proceso son:

* Salvar el estado del proecesador
* Acutalizar pcb del proceso
* Mover el proceso a la cola apropiada (listo o bloqueado)
* Sleccion del nuevo proceso
* Actualizar el pcb del nuevo proceso
* Actualizar las tablas que tengan referencia al proceso
* Restaurar el estado del procesador para el determinado proceso

Ocurre una particularidad cuando ocurre un interrupción y el SO decide pasar a modo kernel para interpretar que hacer con determinada interrupción, a este cambio se le denomina **cambio de modo** y no es un cambio de estado. Si el sistema decide no hacer nada con la interrupción entonces el SO vuelve a ejecutarse en modo usuario y retoma la ejecución del proceso sin realizar ningún cambio de estado