Tema 1.1: Introducción a los Sistemas Operativos.

- La función del sistema operativo es facilitar al usuario el uso del computador, antes solo se utilizaba para controlar y comunicarse con el hardware. Todas las funcionalidades y aplicaciones están sobre el sistema operativo.
- **Porque saber sobre SS. OO**:
  - Influye en el funcionamiento, seguridad y rendimiento del computador. Por ello es importante conocer cómo funciona.
  - Elegir el que mejor se adapte a nuestras necesidades o de la empresa es fundamental.
    - Lo mas importante, que sea rentable. Hay que proteger la inversión.
    - Encontrar administradores que sepan sobre el sistema.
    - Que ofrezca soporte y se actualice.
  - Al desarrollar aplicaciones saber como se comportará y comprender los problemas.
    - Conocer los servicios que ofrece y como invocarlos. Y el desarrollo multi-hilo.
- No tener ideas preconcebidas de los SS. OO, "ser agnóstico", utilizar el que mejor se adapte a lo que necesito, hacer una checklist y ver cual cumple más.

- LECTURA CAPÍTULO 1, libro CARRETERO 2007. Conceptos arquitectónicos del computador.

- **Sistema Operativo:** Programa intermediario entre el usuario y el hardware. Oculta el hardware, pero lo controla para ofrecer servicios al usuario, el usuario y hardware son muy distintos.
    - Ejecuta programas.
    - Hace uso eficiente de recursos.
    - Proporciona una visión de máquina virtual.
  - **Funciones del Sistema Operativo:**
    - **Gestor de recursos**:
      - Asigna y recupera recursos, como es la memoria.
      - Protege al usuario, controla el acceso y los permisos de los usuarios.
      - Contabiliza y monitoriza los recursos del sistema para ver como esta y que le pasa.
    - **Máquina extendida (servicios):**
      - Ejecución de programas.
      - Órdenes de E/S, leer/escribir, abrir/cerrar.
      - Operación sobre archivos.
      - Detección y tratamiento de errores, para que se reconozca y resuelvan.
    - **Interfaz de usuario:**
      - Shell, es una interfaz gráfica externa al sistema operativo.
  - **Niveles del Sistema Operativo**, los 3 principales:
    - Usuario.
    - Shell.
    - Llamadas al sistema o Servicios.
    - Núcleo/Kernel. Se ha ido reduciendo el Kernel y ampliando los Servicios
    - Firmware/HardwareAbstractLogic, entre Kernel y Hardware.
    - Hardware.
  - **Modos de ejecución**:
    - Modo **usuario**: Esta restringido, no puede acceder a ciertas aplicaciones, E/S, etc. Para proporcionar más seguridad.
    - Modo **núcleo**: Puede ejecutar cualquier aplicación, sin restricciones. Más "peligro".
  - Los procesos y el sistema operativo, tienen espacios memoria reservada separados, y el del sistema operativo no es accesible para el usuario.
  - Cuando un proceso necesita un servicio, se hace con una llamada al sistema, y el sistema lleva a cabo la función de la llamada.

    - **Modificar el sistema operativo:** Principalmente para añadir funcionalidades o nuevo **hardware**. Pero se deben seguir unos estándares para asegurar que las aplicaciones funcionen en sistemas que sigan el mismos estándar.

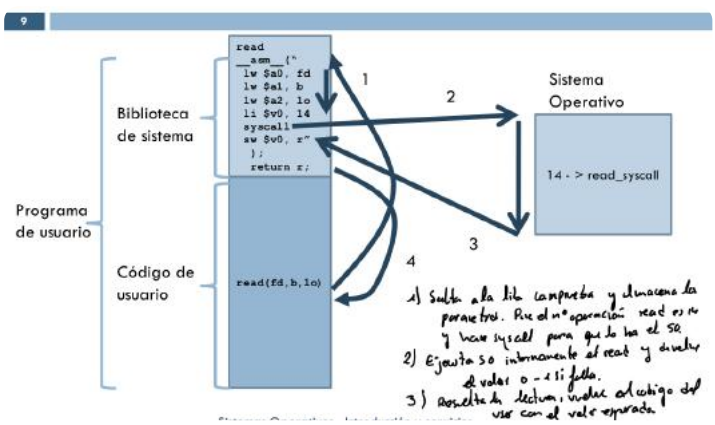
- **Sistemas operativos monolíticos**: No hay estructura clara y bien definida, pero es más eficiente, aunque su mantenimiento es más difícil. Todos los módulos se pueden llamar entre ellos. Siempre modo núcleo y todo enlazado en un ejecutable.
- **Sistemas operativos estructurados por capas:** Organizado por capas superpuestas, con una interfaz clara y bien definida. Una capa se apoya en la inferior, resulta menos eficiente, pero todo está mas restringido y es más fácil el desarrollo y depuración.
- **Sistemas operativos estructurados cliente/servidor:** Consiste en implementar la mayor parte del sistema operativo como procesos de usuario, y una pequeña parte corre en modo núcleo llamada microkernel. Muy flexible, ya que funciona como pequeños programas.

- **Clasificación de Sistemas Operativos:**
    - Número de procesos simultáneos:
        - Monotarea: Un solo proceso a la vez.
        - Multitarea: Varios procesos a la vez, necesita planificación.
    - Modo de interacción:
        - Interactivo: Permite ejecutar procesos y obtenemos la solución en el momento.
        - Por lotes: Tu pides al sistema y no sabes cuando se hará.
    - Número de usuarios simultáneos:
        - Monousuario.
        - Multiusuario.
    - Numero de procesadores:
        - Monoprocesador.
        - Multiprocesador: Son actualmente casi todos. Multicore.
    - Número de hilos:
        - Monothread.
        - Multithread.
    - Tipo de uso:
        - Cliente: Pedir procesos.
        - Servidor: Encargarse de los procesos.
        - Empotrados: Dentro de un sistema cerrado, con unas funciones determinadas.
        - Tiempo real: Se usan otros sistemas operativos para gestionarlos, otra planificación.

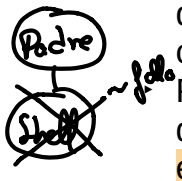
- **Arranque del sistema operativo**:
    - Comienza con el envío de la interrupción de RESET(la 0), que carga el cargador (booter) y este carga la BIOS (Basic Input Output System) en memoria, la BIOS hace una comprobación del hardware (los periféricos, los discos, la memoria...) y cuando termina comienza a cargar en memoria el sistema operativo del disco que sabe que lo almacenaba. Una vez cargado el sistema operativo, se empieza a ejecutar, lo instala todo, y cuando ha termina el sistema ya esta ejecutado y se puede liberar la memoria que ocupaba para el usuario.
        - Reset, pone los valores predefinidos a los registros y carga el cargador/booter.
        - Se carga la BIOS, hace comprobaciones y cargar el sistema operativo en memoria.
        - Se ejecuta el cargador del sistema operativo, que empieza a instalar el sistema y cuando termina el sistema ya está ejecutado, y se libera la memoria del cargador.
- **Parada del computador (Shutdown):** Procesos inverso al arranque.
    - Al apagar, se vuelca la información a disco y se terminan todos los procesos.
    - Si hay apagado brusco, se pierde información y se puede dañar, para evitarlo hay una pequeña batería que permite que se apague correctamente. Para sistemas grandes SAI.
    - Otras alternativas son: Hibernación y Apagado en espera(Standby).

Tema 1.2: Servicios del SO.

- **Ejecución del sistema operativo:** Una vez finalizado el arranque del sistema, el sistema solo se ejecuta como respuesta a interrupciones. Como son:
  - Petición de servicio, están sobre el sistema operativo y no afectan a la maquina. lpd d=daemon=demonio
  - Interrupción de periféricos o reloj.
  - Excepción de hardware.
- **Fases en la activación del Sistema Operativo:** Lo primero que se ejecuta es el proceso init y a raíz de este van apareciendo mas procesos que cuelgan de el, como si fuera un árbol del que van saliendo ramas. Al cambiar de proceso debe encontrarse tal y como estaba antes del cambio, eso se llama contexto. Antes de cambiar se almacena todo del proceso actual e inmediatamente después se cargan los datos del otro. Todo proceso tiene su contexto. Se aprovechan los tiempos de espera(de cualquier proceso) para cambiar de proceso y ejecutar mientras tanto otras cosas, estos cambios los controla el planificador y es concurrencia.
- **Activación de servicios:** Lo primero es pasar de modo usuario a modo núcleo, en la llamada el sistema tiene control total de la maquina. Ese cambio de privilegios es gracias a las bibliotecas que tienen acceso a súperusuario. Y esos privilegios pueden ocasionar problemas de seguridad. Con la interrupción software trap se activa el modo seguro del sistema operativo.
  - La rutina de bibliotecas es: Preparar la llamada al SO, instrucción trap y procesar los datos recibidos de la llamada.
- **Llamadas al sistema**:
  - Interfaz entre aplicaciones y SO.
  - Servicios típicos
    - Gestión de procesos.
    - Gestión de procesos ligeros: Si se crea, que se pueda controlar y matar.
    - Gestión de señales y temporizador: Avisos y automatizaciones.
    - Gestión de memoria.
    - Gestión de ficheros y directorios.
- **Bibliotecas**: Agrupación de funciones completas
- **Invocación de la llamada:** Cada función de la interfaz de programación(API) se corresponde con algún servicio del sistema operativo. Incluye la ejecución de trap que transfiere el control al sistema operativo con una interrupción, el sistema operativo trata la interrupción y devuelve el control al programa de usuario.
  - Salta a la librería, comprueba y almacena los parámetros necesarios para la operación, se pueden pasar en registros, tabla de memoria o en pila. Pone el numero de operación, el identificador numérico correspondiente y hace syscall (trap), para llamar al sistema a ejecutar esa operación.
  - El SO ejecuta internamente la operación, recibe el identificador que le indica la dirección de la rutina de tratamiento y la ejecuta, y devuelve 0 si ha funcionado o -1 si ha fallado.
  - Resuelta la operación, vuelve al código del usuario con el valor esperado.



- **Interfaz del programador:** Ofrece una visión que como maquina extendida tiene el usuario del sistema, cada sistema operativo puede ofrecer uno o varias.
- **Estándar POSIX (Portable Operative System Interface for X)**: Interfaz estándar de sistema operativos de IEEE. El objetivo es que todas la maquinas que compartan este estándar puedan ejecutar los mismo programas, portabilidad.
  - Nombres de funciones cortos y en letras minúsculas, representativos y en ingles.
  - Normalmente el éxito devuelve 0 y en caso de error -1. errno=error number y perror(errno) nos dice que error es y nos proporciona algo de información .
- **Ejecución de un mandato en Shell**: Lo que hace cada vez, hay una relación jerárquica de procesos shell, que permite que en caso de error el proceso shell padre no muera.
  - Lo primero el padre ejecuta fork(), que crea un duplicado del padre que funcionara a la vez que el padre y ese hijo se identifica por un pid(process id) de 0 y el padre tendrá otro cualquiera. Cuando se ejecuta fork el hijo siempre es pid=0.



Para que sea el hijo el que ejecute el mandato y corra el peligro de error, y no el padre. Lo que se hace es que el proceso con id 0, el hijo, ejecute el mandato con execvp(,) y si da error ese proceso hijo muere y queda vivo el padre, pero si no falla ambos seguirán vivos hasta el final del programa. Se hace un switch, si pid del hijo es -1 ha habido un problema al crearlo y termina, si es 0 ejecuta el mandato y break y si no es ninguno de los anteriores es el padre.

- **Servicio fork()**: Devuelve un pid_t que es el id del proceso, en el padre el valor es el identificador del hijo y en el hijo el valor es 0, y -1 en caso de fallo. Su función es duplicar el proceso que invoca la llamada, tanto el proceso hijo como el padre siguen ejecutando el mismo programa. El proceso hijo conservara una copia exactamente igual de todo lo que tenia el padre.

Servicio wait(&status): Bloquea la ejecución del padre hasta que el proceso hijo muere, devuelve el identificador del proceso que ha finalizado, y status es el valor que devuelve el hijo cuando llama al exit().

- **Servicio exec:** Ejecuta un programa en ese proceso, el proceso recibe toda la información y cambia su imagen, desaparece el código que tiene y se sustituye por el del programa si ha ido bien, sino retorna -1. Cuando se hace exec la tabla de ficheros abiertos no se borra. Servicio único para múltiples funciones de biblioteca. Recibe como primer argumento el nombre de la función a ejecutar y el resto son los parametros del main. execlp('ls','ls','-l')


```
int execl(const char *path, const char *arg, ...);
int execv(const char* path, char* const argv[]);
int execve(const char* path, char* const argv[], char* const envp[]);
int execvp(const char *file, char *const argv[])
```

- Cambia la imagen del proceso actual.
  - path: Ruta al archivo ejecutable.
  - file: Busca el archivo ejecutable en todos los directorios especificados por PATH.
- Descripción:
  - Devuelve -1 en caso de error, en caso contrario no retorna.
  - El mismo proceso ejecuta otro programa.
  - Los ficheros abiertos permanecen abiertos.
  - Las señales con la acción por defecto seguirán por defecto, las señales con manejador tomarán la acción por defecto.

- **Servicio exit(void exit(status)):** Finaliza la ejecución del proceso, se escribe al final de un programa o al detectar un error. Se cierran todos los descriptores de ficheros abiertos y se liberan todos lo recursos del proceso. Devolver negativo es indicativo de error y 0 de que la ejecución ha ido correctamente, pero también puede tomar otros valores. Todos lo valores que toma status sirven para dar información adicional de como ha ido la ejecución. return 0=exit(0)
- **Fichero**, es un manera de almacenar información de una manera mas persistente, para reutilizarlos o simplemente para tenerlos.

- **Operaciones genéricas sobre ficheros**: AMPLIAR
  - Crear: creat(nombre, O_WRONLY|O_CREAT|O_TRUNC, mode);
  - Borrar: unlink(nombre);
  - Abrir: open(descriptor, permisos); O_RDWR/WRONLY/RDONLY/APPEND sigue al final del fichero.
  - Cerrar: close(descriptor);
  - Leer: read(descriptorLectura, buffer, tamañoALeer);
  - Escribir: write(descriptorEscritura, buffer, longitudAEscribir);
  - Posicionar: lseek(descriptor, desplazamiento, dondeComenzar); SEEK_SET/END/CUR
  - Control: fnctl para modificación de atributos, dup para duplicar descriptores de fichero, ftruncade asignación de espacio a un fichero, stat información sobre un fichero, utime para alterar los atributos de fecha.