CENTRO UNIVERSITARIO
DE TECNOLOGÍA
Y ARTE DIGITAL

Sistemas Operativos

1. Introducción

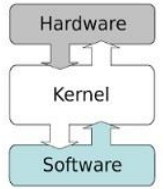
Javier García Algarra
javier.algarra@u-tad.com
Miguel Ángel Mesas
miguel.mesas@u-tad.com
2021-2022

¿Por qué estudiar SSOO?

1. Porque son una obra cumbre de la ingeniería software. Es difícil encontrar un software más complejo, que atienda tantas necesidades, que sea producto de la colaboración de miles de ingenieros y que haya atraído tanto talento. Otras obras de ingeniería (Caminos, Canales y Puertos):
   - Puente de la bahía de Jiaozhou, China (Puente más largo del mundo, 41 km)
     Puente Bahía Jiaozhou / Enlaces de la construcción - YouTube
   - Puente del agua de Magdeburg, Alemania (ayuda a cruzar el río Elba sin salirse del canal Elbe-Havel)
     Puente canal de Magdeburgo (Alemania) - YouTube

2. Porque alguna vez os habréis preguntado por qué haciendo click en un dibujito de repente se ejecuta algo tan complicado como un navegador web.

3. Porque son la plataforma sobre la que se ejecutarán todas las aplicaciones que haréis en vuestra vida. Si no sabéis cómo funcionan nunca llegareis a entender qué distingue el software profesional de un programa aficionado.



4. Porque el día que os jubiléis ahí seguirá algún derivado de Unix dando servicio. Es uno de los pocos elementos que permanecen en el tiempo. Las máquinas y los lenguajes de programación de 1980 son dinosaurios, pero la arquitectura de Linux no se diferencia en esencia de la de Unix de ese año.

Este curso NO es…

1. Una introducción al uso de comandos o programación shell script. Eso ya lo aprendisteis en primero y si no lo recordáis conviene refrescarlo.

2. Un curso de administración Unix/Linux o Windows. La figura del administrador de sistemas es muy importante y es un rol bien definido. El administrador tiene que conocer cómo funciona el Sistema Operativo para configurarlo y gestionarlo en condiciones, pero no es el objeto de este curso (aunque veremos alguna herramienta de sysadmin)

3. Un curso de programación de sistemas, para construir drivers o diseñar nuevos sistemas de ficheros. Eso no está a nuestro alcance …por ahora. Sin embargo, haremos bastante uso de la programación de sistemas.

4. Un concurso de belleza para ver si Windows es mejor que macOS o Linux que FreeBSD. Todos tienen su campo de aplicación, un ingeniero SW debe saber cuales son los puntos fuertes y débiles de cada uno.

¿Cómo aprenderemos SSOO?

1. Asimilaremos la teoría, como en los libros de introducción general a Sistemas Operativos. Esto solo no basta.

2. Para aprender de Sistemas Operativos hay que ver código. El problema es que el código de Linux o FreeBSD puede provocar convulsiones a un programador novel por su dificultad. Usaremos el sistema xv6, un “juguete” desarrollado por el MIT que, sin embargo, contiene los elementos esenciales. Lo instalaremos, lo usaremos y lo modificaremos.

3. Programando. Un virtuoso del violín o la guitarra eléctrica no aprende leyendo historia de la música. Programaremos en Linux sobre una máquina virtual y sobre el citado xv6. Aprenderemos a hacer un mínimo driver Linux.

4. Compilando e instalando un sistema operativo desde cero. Bajaremos una distribución Linux y la haremos arrancar configurando y compilando. Usaremos herramientas de traza propias de un sysadmin profesional.

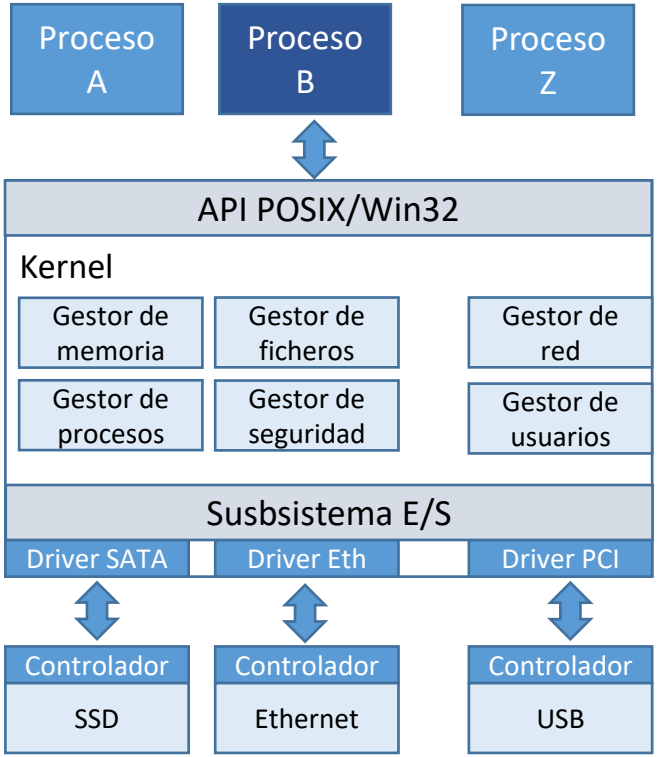
¿Qué es un Sistema Operativo?

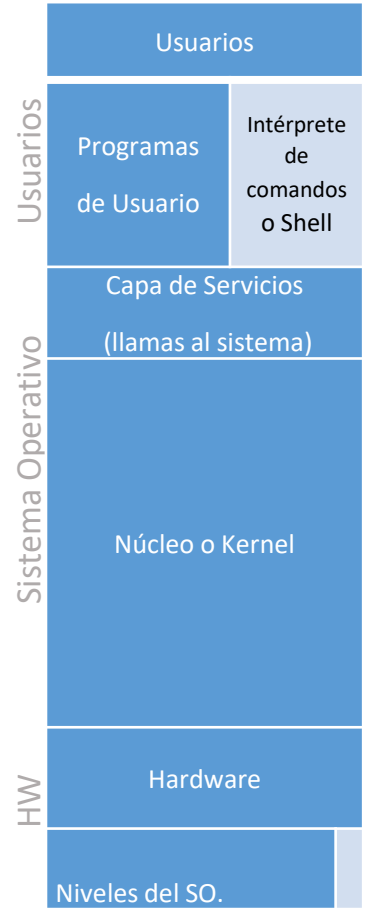
Es el programa más importante que se ejecuta en la máquina

Proporciona dos funciones básicas

1. Abstracción de los detalles físicos. Los procesos se ejecutan en un entorno virtual y acceden a los recursos usando primitivas del sistema

2. Permite que múltiples aplicaciones y usuarios compartan el mismo equipo de forma segura.





La máquina desnuda

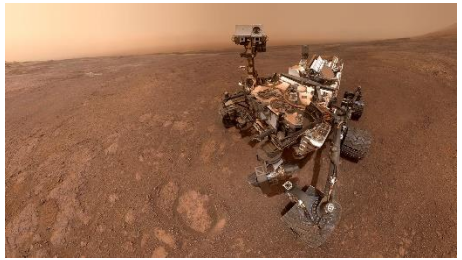
Se denomina máquina desnuda a un ordenador sin sistema operativo. Un reloj digital sencillo, una lavadora de hace dos décadas, equipos basados en microcontrolador como Arduino, los ordenadores personales de los 80, son ejemplos de máquina desnuda.

La máquina desnuda

Por el contrario, no son máquinas desnudas, porque incluyen un sistema operativo, las siguientes:

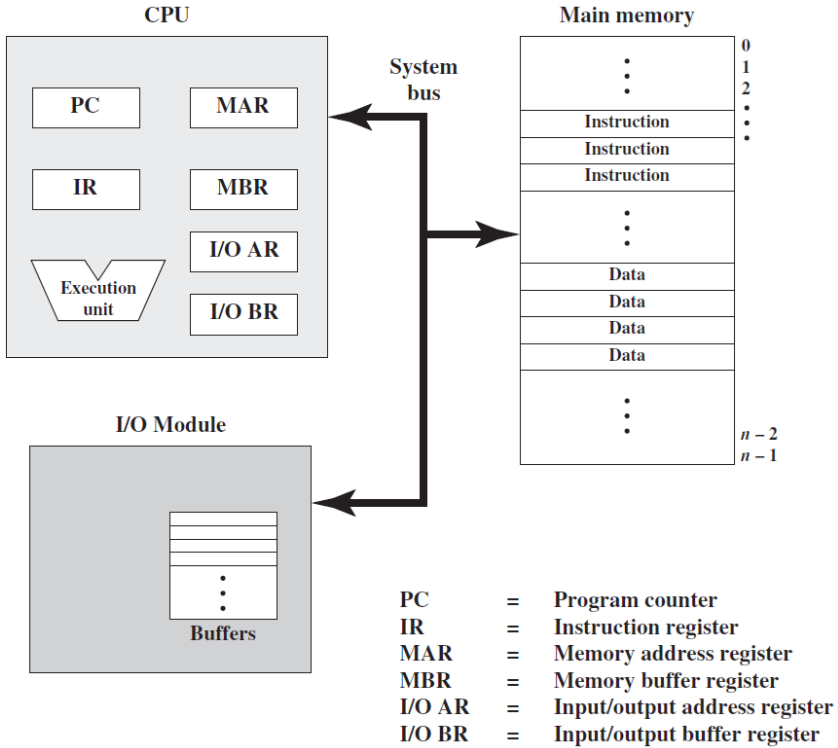
android

VxWorks



FREE RTOS

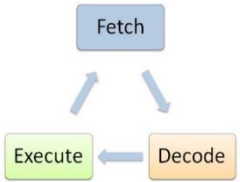
La máquina desnuda



La máquina desnuda es la arquitectura propuesta por Von Neumann.

El cerebro del ordenador es la CPU (Central Processing Unit) que consta de la unidad aritmético-lógica (ALU), la unidad de control y una serie de registros, de propósito general y de propósito especial.

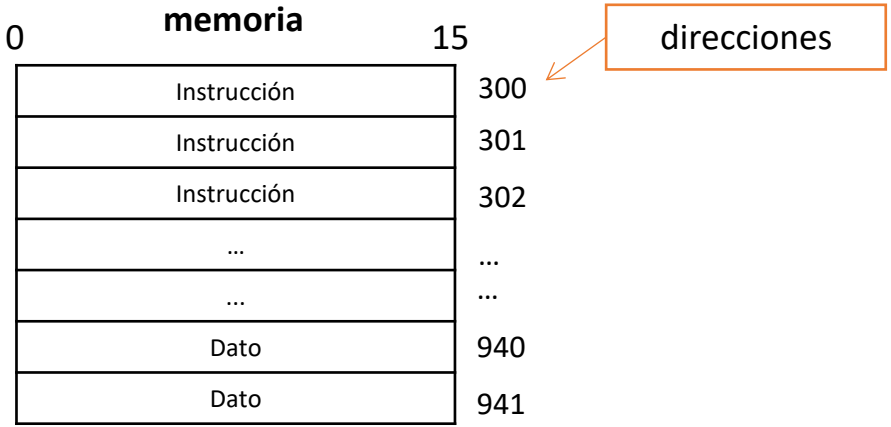
Los programas y datos residen en la memoria. La CPU funciona según un ciclo de lectura, decodificación y ejecución de cada instrucción de código máquina. El contador de programa PC apunta siempre a la siguiente instrucción a ejecutar



La máquina desnuda

Tenemos una máquina cuya memoria se organiza en posiciones de 16 bits.

Las instrucciones del lenguaje máquina son de 16 bits. Los 4 de más peso son el código de operación (Codop) los 12 restantes la dirección implicada.



Tiene un PC, un IR (Registro de Instrucción, en el que la Unidad de Control decodifica la instrucción leída de memoria), y un único registro para almacenar datos llamado AC (acumulador).
Las tres primeras operaciones realizables por el procesador son

Código de operación	Codop (hexa)	Instrucción
0001	1	Cargar AC desde memoria
0010	2	Guardar AC en memoria
0101	5	Sumar al AC un dato de la memoria

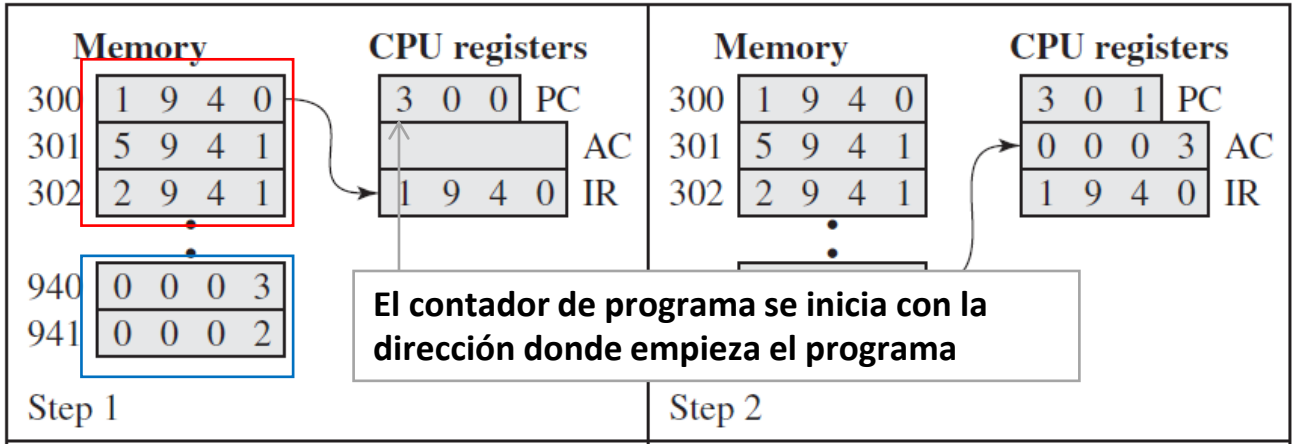
Programa

1940
5941
2941

La máquina desnuda

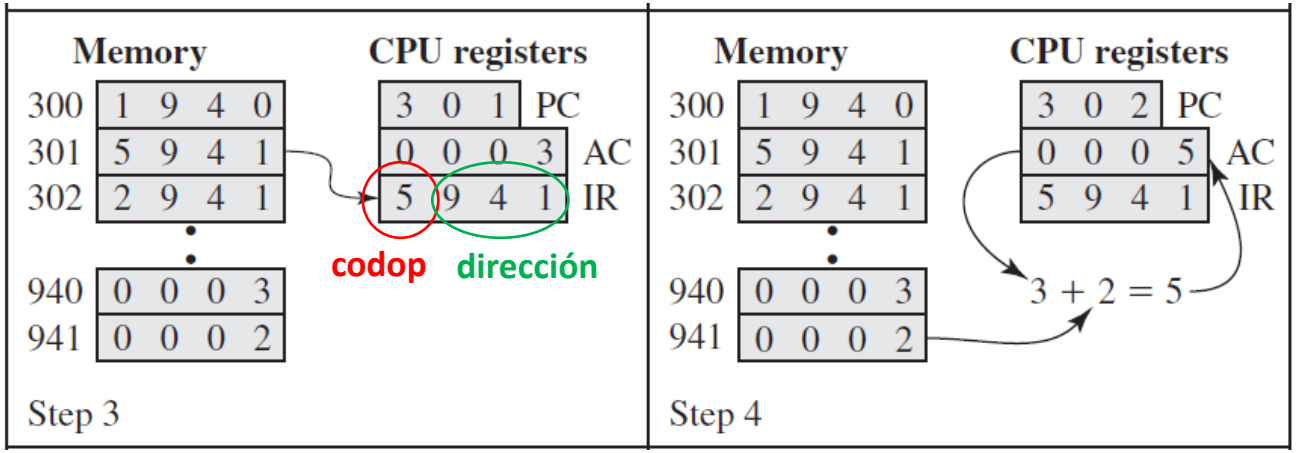
Instrucciones

Datos



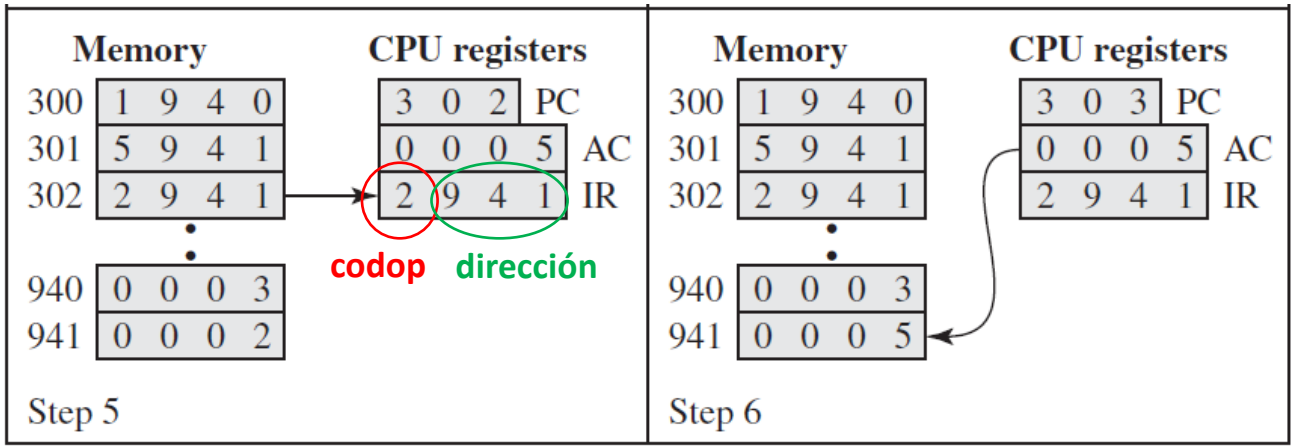
1940 = Cargar en AC el contenido de la dirección 940

La máquina desnuda



5941 = Sumar a AC el contenido de la dirección 941

La máquina desnuda



2941 = Guardar el contenido de AC en la dirección 941

La máquina desnuda

El flujo de ejecución lineal puede alterarse mediante instrucciones de salto o llamadas a subrutina, en cuyo caso se utiliza una zona especial de la memoria (pila o stack) para guardar la dirección de la instrucción de retorno y pasar los parámetros. El registro SP (stack pointer) guarda la dirección de la última palabra del stack utilizada, el stack funciona como una LIFO.

La relación con el mundo exterior se realiza mediante el módulo de entrada/salida. Los dispositivos de entrada/salida (por ejemplo, un ratón) pueden requerir atención en cualquier momento del programa. Para ello se emplea el mecanismo de interrupción, mediante una señal eléctrica que llega a la CPU. La CPU salva el contexto de la última instrucción, ejecuta la rutina de atención de la interrupción y regresa al punto en el que el programa se interrumpió.

La memoria se conecta a la CPU mediante un bus de direcciones, un bus de datos y un bus de control. La velocidad de acceso a la RAM es mucho más lenta que la del acceso a los registros internos de la CPU.

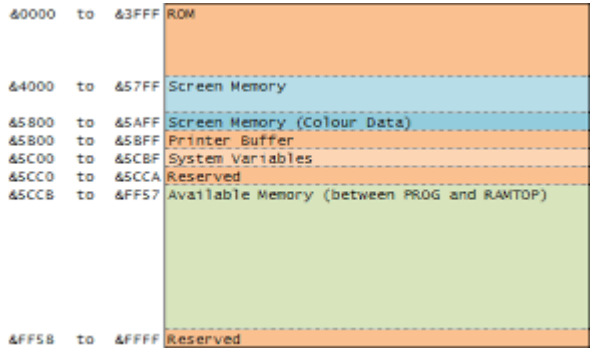
En la máquina de von Neumann **solo se puede ejecutar una instrucción en cada momento**. Esto es igual en los procesadores modernos, aunque si disponen de varios cores, cada uno tendrá su propio hilo de ejecución, pero sigue siendo único por core.

La máquina desnuda

En una máquina desnuda, se ejecuta un único programa. Al arrancarse o resetearse la CPU el PC apunta a una dirección fija, que depende el modelo de microprocesador, donde se instalaba un programa cargador. Este puede lanzar una misma aplicación siempre, como en aludido reloj digital o arrancar algún menú sencillo para que el usuario decida qué función se ejecuta a continuación

El programa iniciador está grabado en un soporte persistente, como ROM o Flash.

En los ordenadores primitivos, el programa se cableaba en memoria, el operador introducía manualmente en el contador de programa la dirección de inicio y así se lanzaba la ejecución.



Mapa de memoria del Z80

Limitaciones de la máquina desnuda

Ineficiente gestión de los recursos

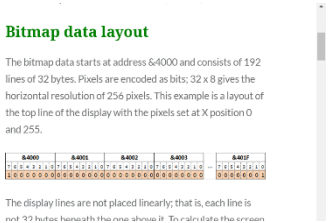
Cuando se lanza una instrucción bloqueante, por ejemplo `scanf()`, la CPU queda ociosa cuando podría atender a otros programas

Seguridad

El programa de usuario puede acceder a cualquier recurso de la máquina y la posibilidad de *crash* es muy alta. No hay protección

Nula abstracción

Obliga al usuario a conocer los detalles del hardware y acopla el SW de forma extraordinaria al HW de la máquina

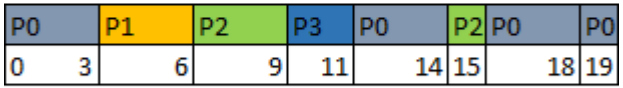


Recursos del ordenador

El Sistema Operativo consigue que cada proceso (programa) funcione como si tuviese una máquina entera a su disposición (Lección 3)

A. Repartiendo el uso de la CPU (*scheduling*)

Processos	Duração da fase de uso da CPU
P0	10 ms
P1	3 ms
P2	4 ms
P3	2 ms

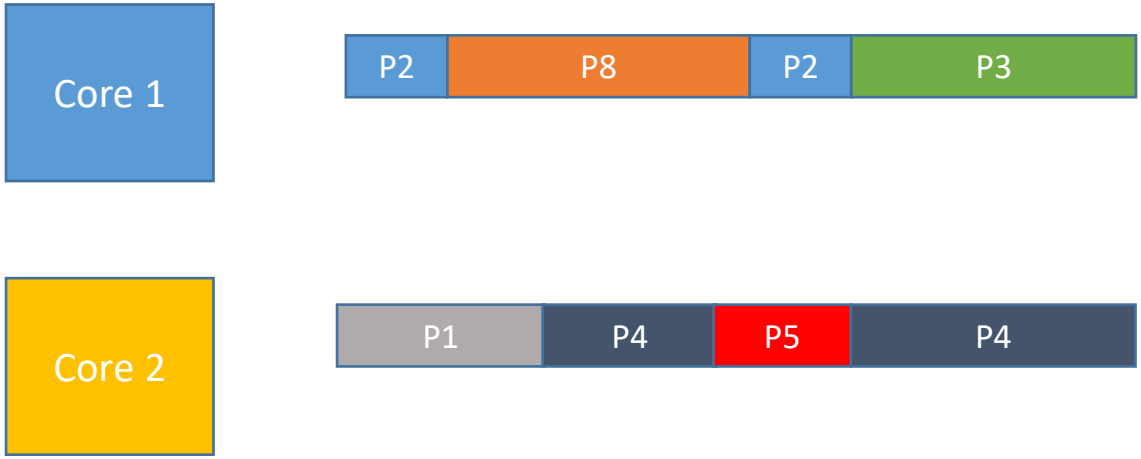


Tempo Total: 19 ms

Tempo Média de Espera: P0(0+11+15) + P1(3) +
+ P2(6+14) + P3(9) / 4 = 14,5 ms

Recursos del ordenador

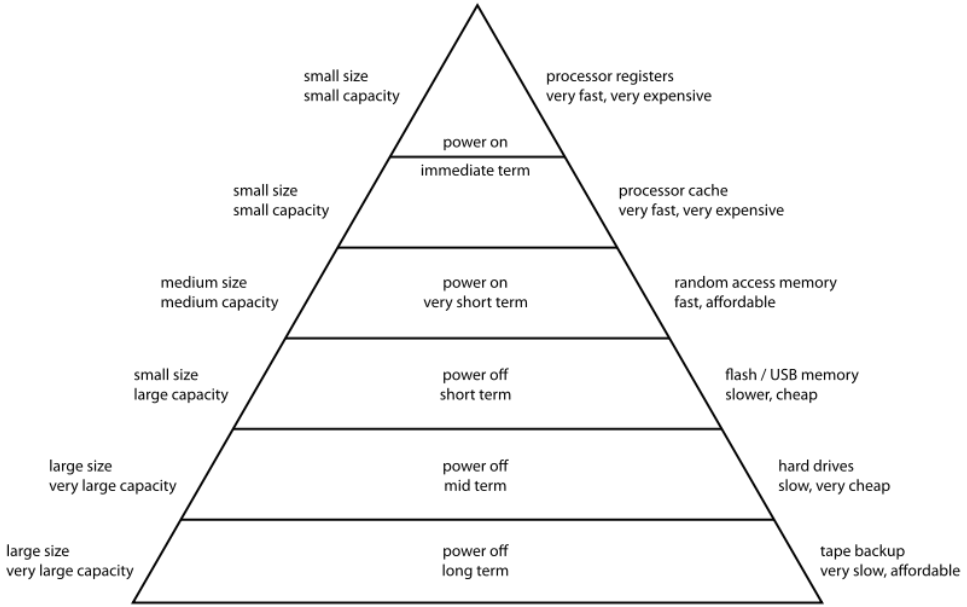
Cuando el sistema tiene múltiples cores cada uno puede atender una secuencia independiente de programas, pero solo uno por core en cada instante



La planificación puede ser independiente por core o conjunta. La planificación multicore conjunta excede de los límites de este curso.

Recursos del ordenador

B. Manejando una compleja jerarquía de memoria (Lección 4)

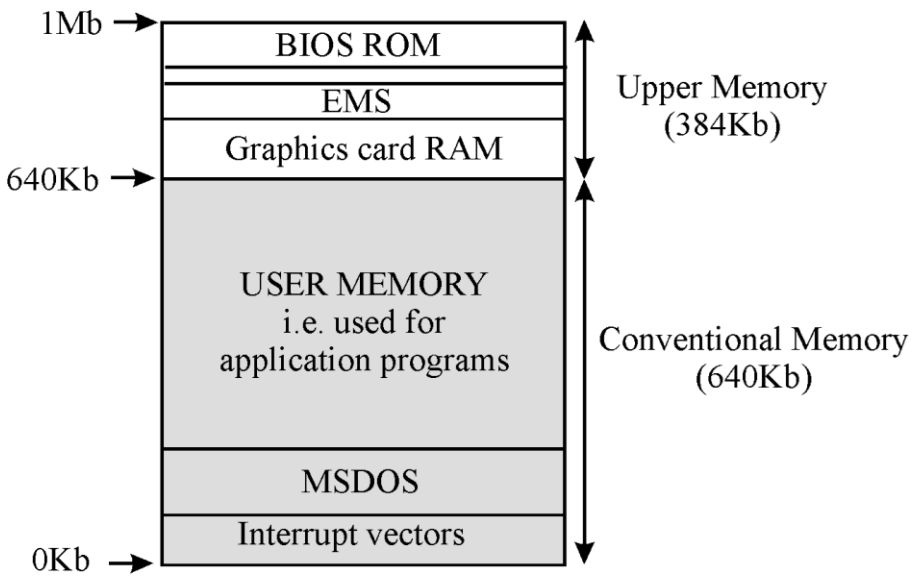


Un Sistema Operativo sin VM

En un PC original, la memora máxima accesible eran 2^{20} bits = 1 MB porque el 8086 tenía 20 bits de direcciones.

La BIOS se instalaba en las direcciones más altas. En las más bajas cargaban el MS-DOS residente y los vectores de interrupción.

Los programas del usuario se cargaban hasta los 640 kB.

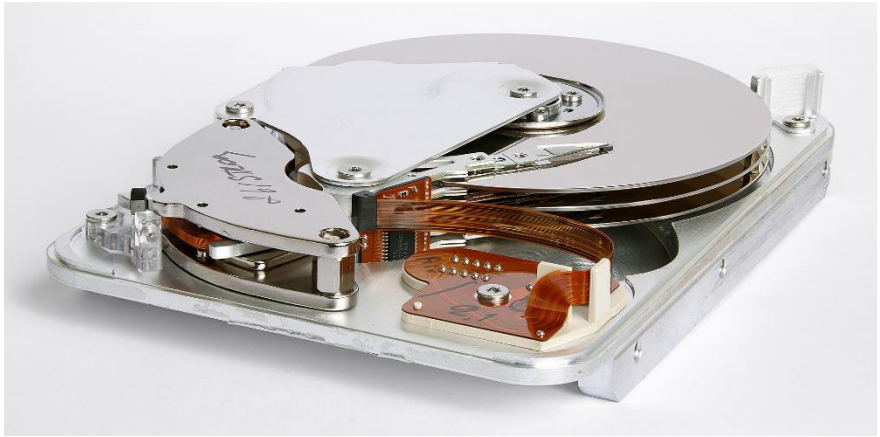


Mapa de memoria de un PC

Recursos del ordenador

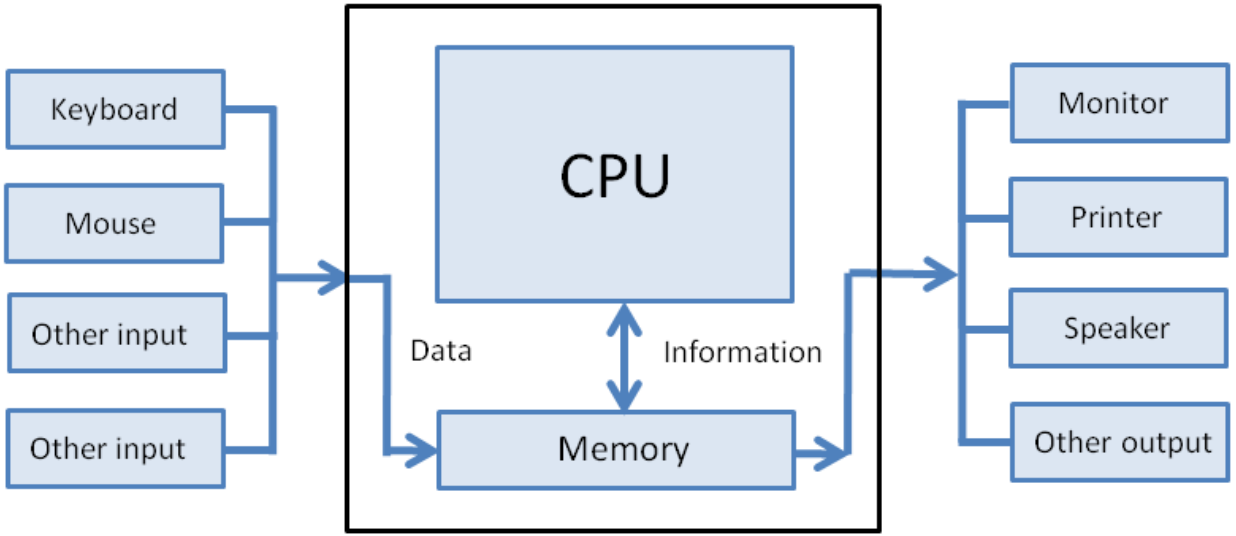
C. Gestionando el acceso al almacenamiento secundario (Lección 5)

```
pfile = fopen("mifichero.txt","r");
```



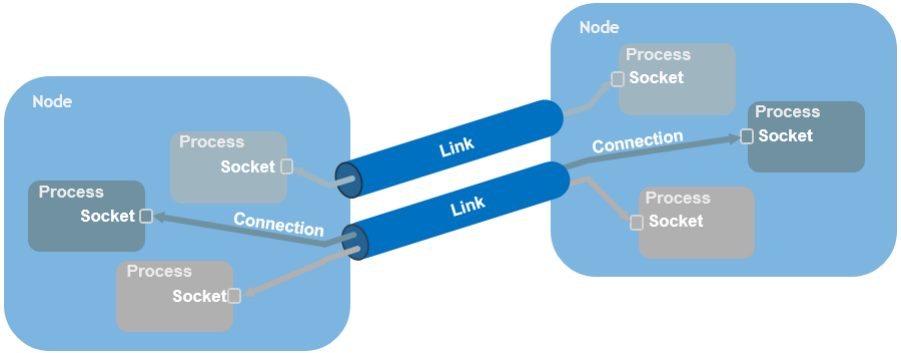
Recursos del ordenador

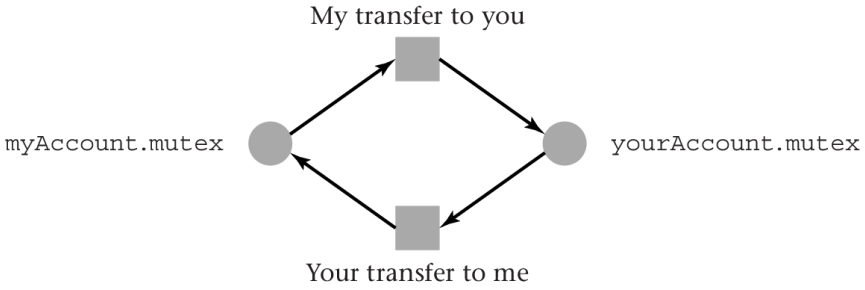
D. Manejando los dispositivos de entrada salida (Lección 6)



Recursos del ordenador

E. Permitiendo la comunicación entre procesos y su sincronización (Lección 7)





Modos usuario/supervisor

La ejecución secuencial simultánea de varios programas plantea problemas serios de seguridad.

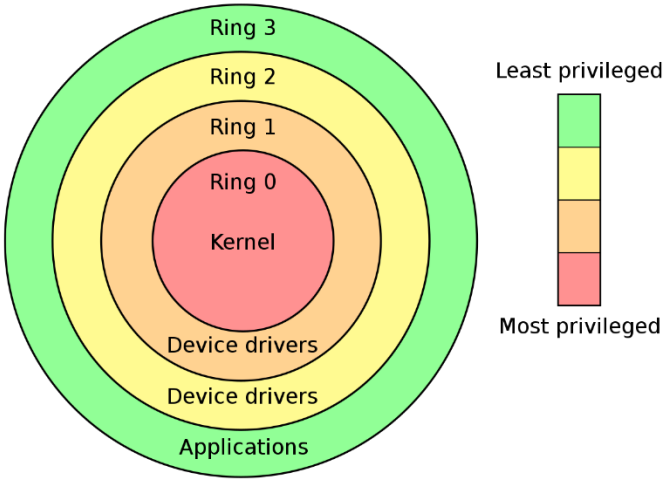
- No queremos que un programa acceda a la memoria de otro, maliciosamente o por error.
- No queremos que un *crash* en un programa produzca la caída de todo el sistemas.
- No queremos que un programa cualquiera atienda las interrupciones.
- No queremos que los dispositivos de entrada/salida y los ficheros se corrompan por accesos erróneos o indebidos.

Todas las operaciones críticas se delegan al Sistema Operativo y la imposibilidad de que el usuario las ejecute se fuerza por hardware. Hay instrucciones de código máquina que solo pueden ejecutarse en modo supervisor, controlado por un bit del registro de flags. Solo el Sistema Operativo tiene privilegio para cambiar ese flag (modo supervisor/modo usuario).

La parte crítica del Sistema Operativo (kernel) se ejecuta siempre que se atiende una interrupción y devuelve el control al usuario al terminar la atención

Modos usuario/supervisor

Intel x86



ARM

Mode	Description	
Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most Applications / OS tasks run	Unprivileged mode

Exception modes

Proporciona 4 niveles, pero Windows o Linux solo usan Kernel/Usuario (0 y 3)

Maneja un modo usuario y seis modos privilegiados

Modos usuario/supervisor

Si la aplicación de usuario necesita acceder a un recurso HW, tiene que hacerlo forzosamente a través del Sistema Operativo. Al abrir un fichero usamos la función fopen() de la librería libc que invoca a su vez la primtiva open() del Sistema Operativo. La primitiva no es una función normal, sino que coloca la información necesaria en su stack e inmediatamente después lanza la instrucción en ensamblador para provocar una interrupción SW (INT en Intel x86). Esta despierta al kernel, que la atiende, coloca el valor de retorno en el stack y devuelve el control al programa con la instrucción de retorno de interrupción.

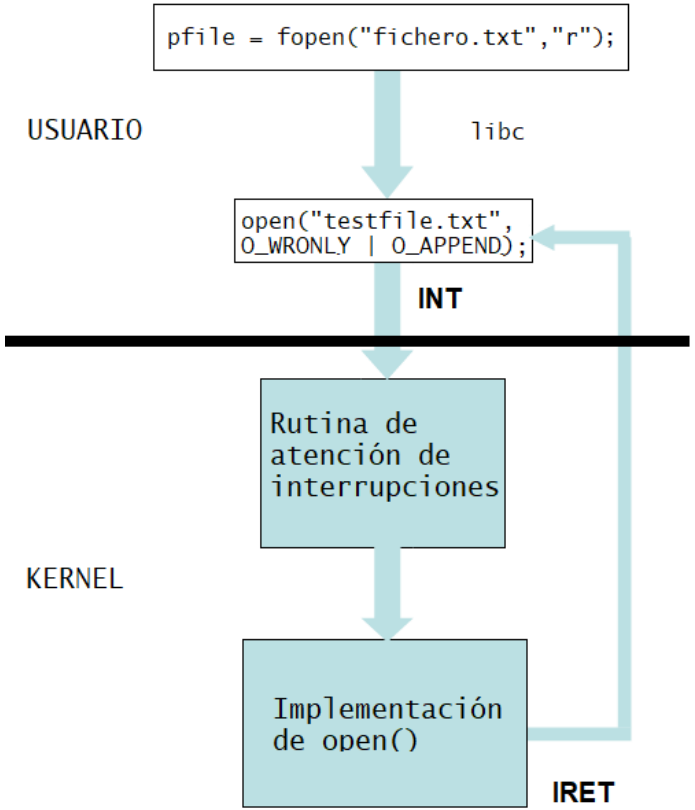
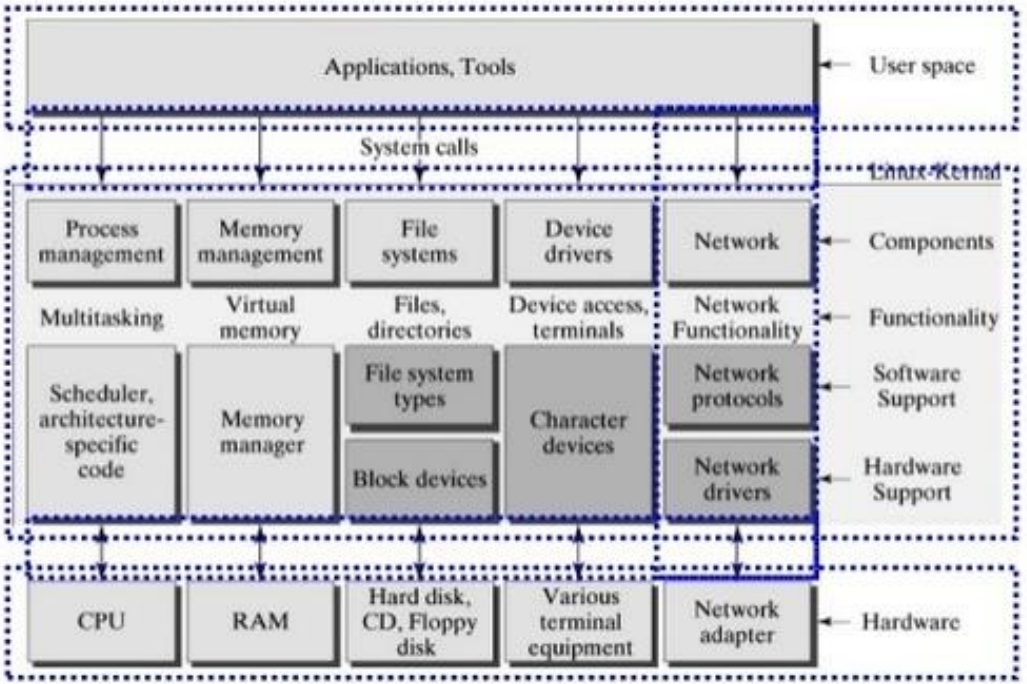


Diagrama del kernel de Linux



Tipos de Sistemas Operativos

De propósito general

- Uso personal: Windows, Linux, Mac OS
- Servidores: Unix (BSD), Linux (Familia Red Hat),
- Windows Server

Móviles

- Android, iOS, Ubuntu Touch

Empotrados

- RTOS, Windows IoT, Raspbian, Contiki

De tiempo real

- QNX, VxWorks, RTLinux

Historia de los Sistemas Operativos

Los Sistemas Operativos propiamente dichos empiezan con los ordenadores de **segunda generación**, los mainframe de los años 50. Inicialmente, se limitaron a sustituir a los operadores humanos.

El operador leía el programa y los datos desde las tarjetas perforadas, grababa el resultado en una cinta y lo cargaba y ejecutaba en el mainframe.

Los **sistemas de procesamiento por lote (batch),** eran capaces de realizar estos trabajos de forma automática y generar una única cinta con la ejecución secuencial de los programas de varios usuarios.

Un programa llamado batch monitor era el encargado de secuenciar y mantener esta ejecución y puede considerarse el embrión de los sistemas operativos. No había interactividad, el usuario solo conocía el resultado de la ejecución a posteriori y no podía introducir datos durante la ejecución.

Historia de los Sistemas Operativos

Concepto de multiplexaje (**compartir**)

Multiplexaje hace referencia a la gestión de los recursos en dos formas distintas:

- En el tiempo
- En el espacio

Cuando un **recurso se multiplexa en el tiempo**, los distintos programas toman turnos para utilizarlo: uno de ellos obtiene acceso al recurso, después otro y así en lo sucesivo. Por ejemplo, con sólo una CPU y varios programas que desean ejecutarse en ella, el sistema operativo primero asigna la CPU a un programa y después, una vez que se ha ejecutado el tiempo suficiente, otro programa obtiene acceso a la CPU, después otro, y un momento determinado el primer programa vuelve a obtener acceso el recurso.

El otro tipo de **multiplexaje es en el espacio**. En vez de que los usuarios tomen turnos, cada uno obtiene una parte del recurso. Por ejemplo, normalmente la memoria principal se divide entre varios programas en ejecución para que cada uno pueda estar residente al mismo tiempo. Otro recurso que se multiplexa en espacio es el disco duro, un solo disco puede contener archivos de muchos usuarios al mismo tiempo. Asignar espacio en disco y llevar el registro de quién está utilizando cuáles bloques de disco es una tarea de administración de recursos común del sistema operativo.

Historia de los Sistemas Operativos

En los años 60 (**tercera generación**), se desarrollan los primeros sistemas operativos modernos, con multiprogramación y multiusuario. IBM desarrolla su línea para mainframes DOS/360 y VM. El MIT comienza un proyecto experimental llamado CTSS (Compatible **Time Sharing** System). Este sistema fue el antecedente de Multics, a su vez, inspiración de UNIX de *Bell Labs*.

1963 Timesharing: A Solution to Computer Bottlenecks

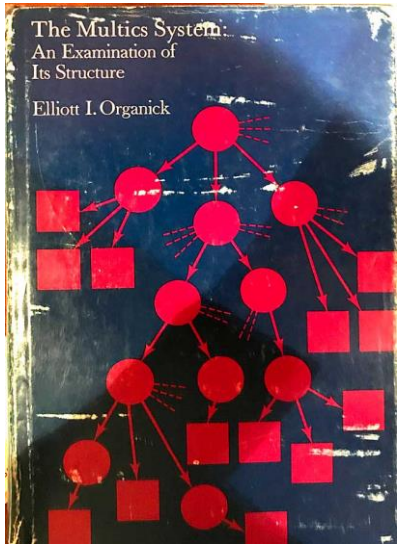
https://www.youtube.com/watch?v=Q07PhW5sCEk&t=278s

Historia de los Sistemas Operativos

Multics (Multiplexed Information and Computing Service) 1964
MIT, Bell Labs y General Electric

Proyecto muy innovador que desarrolló muchos elementos de los Sistemas Operativos modernos pero que resultó un fracaso comercial, posiblemente porque estaba muy adelantado a su tiempo.

- Memoria virtual y segmentada
- Jerarquías de ficheros
- Bases de datos relacionales
- Multicore con memoria compartida
- Seguridad



https://multicians.org/history.html

Historia de los Sistemas Operativos

Unix

En 1969 Bell Labs se retira de Multics. Ken Thompson y Dennis Ritchie empiezan a trabajar en un Sistema Operativo propio con las ideas aprendidas en Multics, una versión reducida a la que llaman Unics, que finalmente se convierte en Unix.

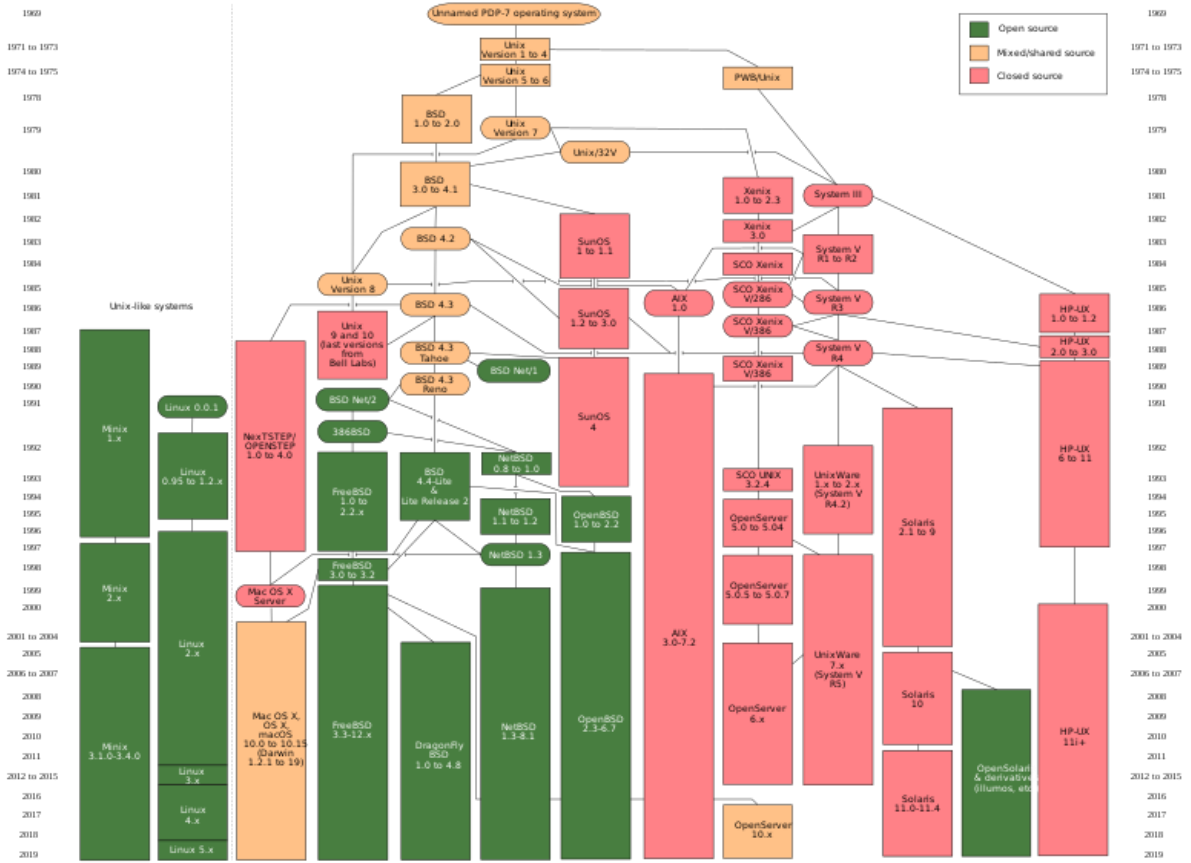
- Es el primer Sistema Operativo escrito en un lenguaje de alto nivel, C, que se desarrolla con este propósito. Esto favoreció su portabilidad a distintas plataformas HW.
- Unix establece un estándar de facto que perdura hasta hoy.
- Bell Labs no podía legalmente vender software, por lo que empezó a ceder licencias gratuitas a universidades, incluyendo el código. En 1978 en Berkeley se desarrolla la BSD (Berkeley Software Distribution), a partir de la V6 de Unix.
- Bell Labs cede licencias comerciales a fabricantes como HP, Sun y Silicon Graphics que desarrollan sus propias versiones.
- Linux no está basado en el código de Unix sino en una versión educativa reducida, MINIX, desarrollada desde cero por Andrew Tanenbaum.

The Great History of UNIX (1969-1999) | 30 Years of UNIX History

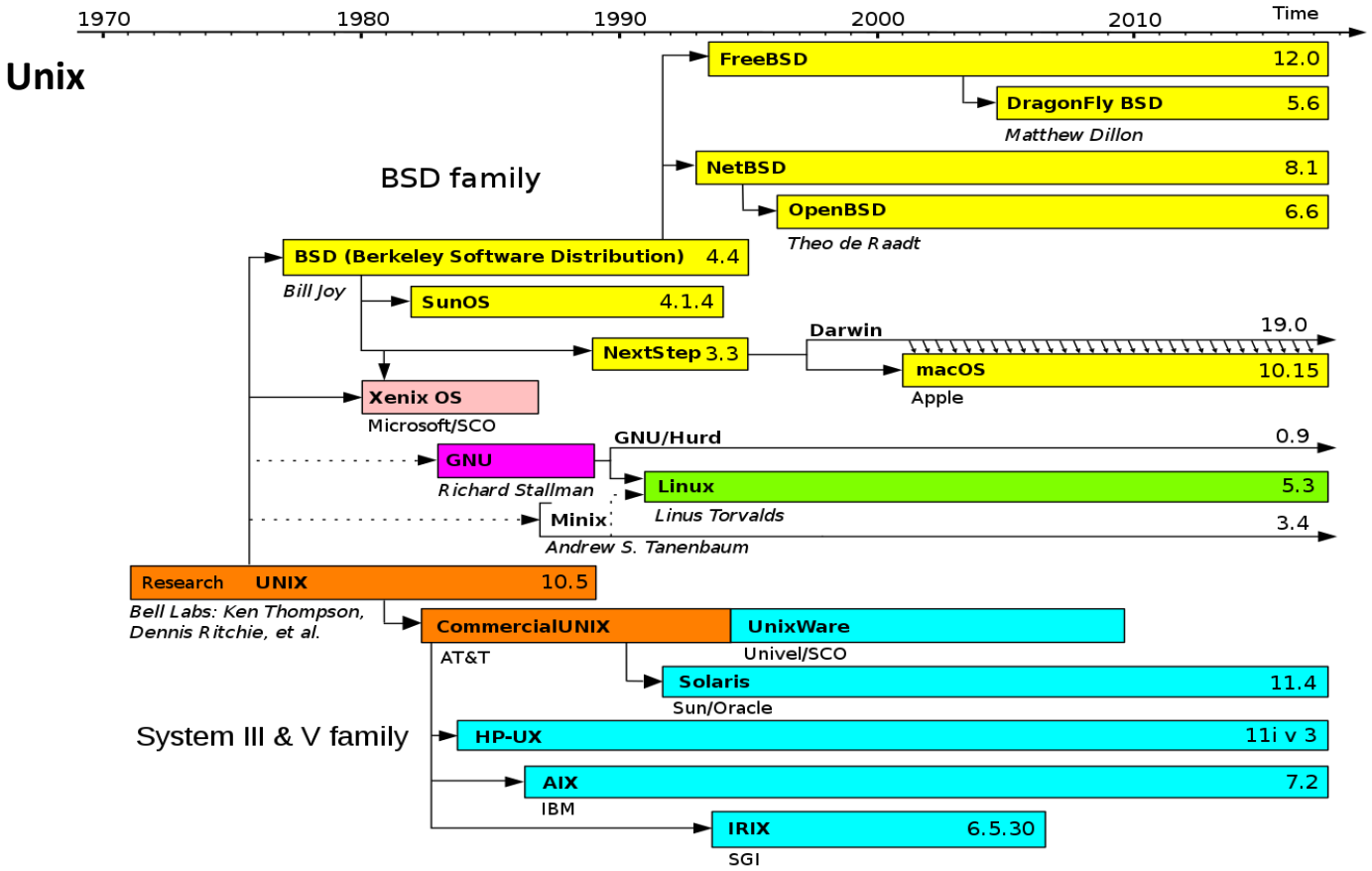
https://www.youtube.com/watch?v=DXh2_CTJW9w

Historia de los Sistemas Operativos

Unix



Historia de los Sistemas Operativos



Historia de los Sistemas Operativos

MS-DOS

IBM encargó a una empresa desconocida, Microsoft, el desarrollo de un Sistema Operativo para su Personal Computer. Microsoft compró 86-DOS un clon de CP/M, un sistema operativo monoproceso de Digital Research para máquinas Z80/8080. La versión de IBM se llamaba PC-DOS

MS-DOS era monousuario, monoproceso y dominó el mercado de ordenadores personales durante la década de los 80. A finales de esa década se lanza Windows, que originalmente solo fue un recubrimiento gráfico de MS-DOS.

IBM cometió el error de no firmar exclusividad, con lo que Microsoft podía vender su Sistema Operativo a los fabricantes asiáticos de clones.

MS-DOS fue la base de los sistemas operativos Windows 9x. A partir de Windows NT, se desarrolla un kernel desde 0.

Birth of the Microsoft DOS https://www.youtube.com/watch?v=9iwIN8E94VQ

Historia de los Sistemas Operativos

La cuarta generación (desde 1980). Los ordenadores personales.

Windows timeline