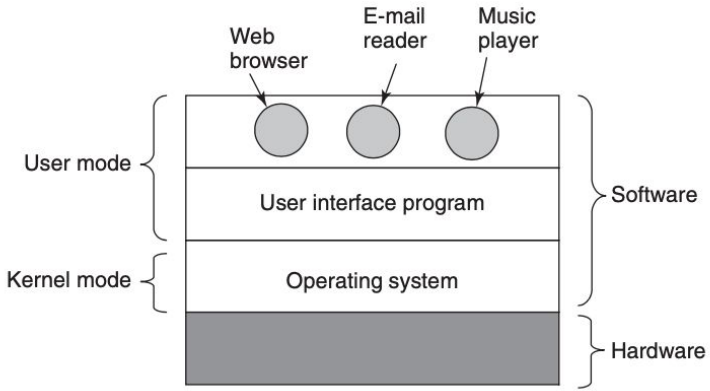
Sistemas Operativos 72.11

Introducción

ITBA
Instituto Tecnológico de Buenos Aires

¿Qué es un sistema operativo?

- Imaginemos programas sin Sistemas Operativos (SO)
- Kernel vs user mode

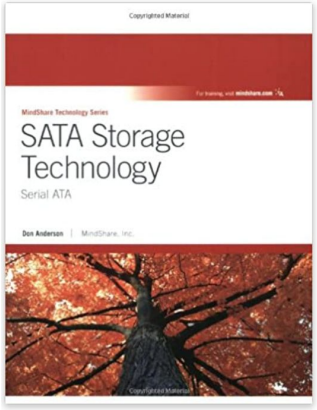


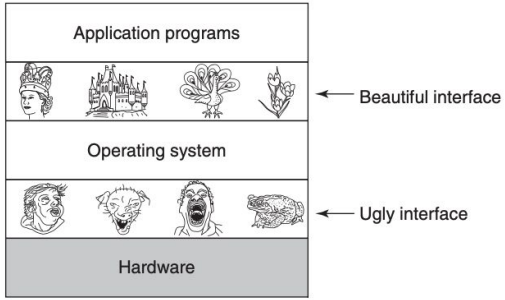
- ¿Por qué el SO tiene privilegios?
- admin vs invitado
- ¿La shell es especial?

- Proveer un conjunto abstracto y limpio de los recursos (hardware)
- Administrar estos recursos

¿Qué es un sistema operativo?

máquina extendida





- 2 abstracciones:
  Driver de disco y archivos

- Ocultar el hardware y ofrecer una interfaz limpia, elegante y consistente al programador
- Las abstracciones son una de las claves para comprender los SOs

- En este curso vamos a estudiar estas abstracciones para las aplicaciones en detalle

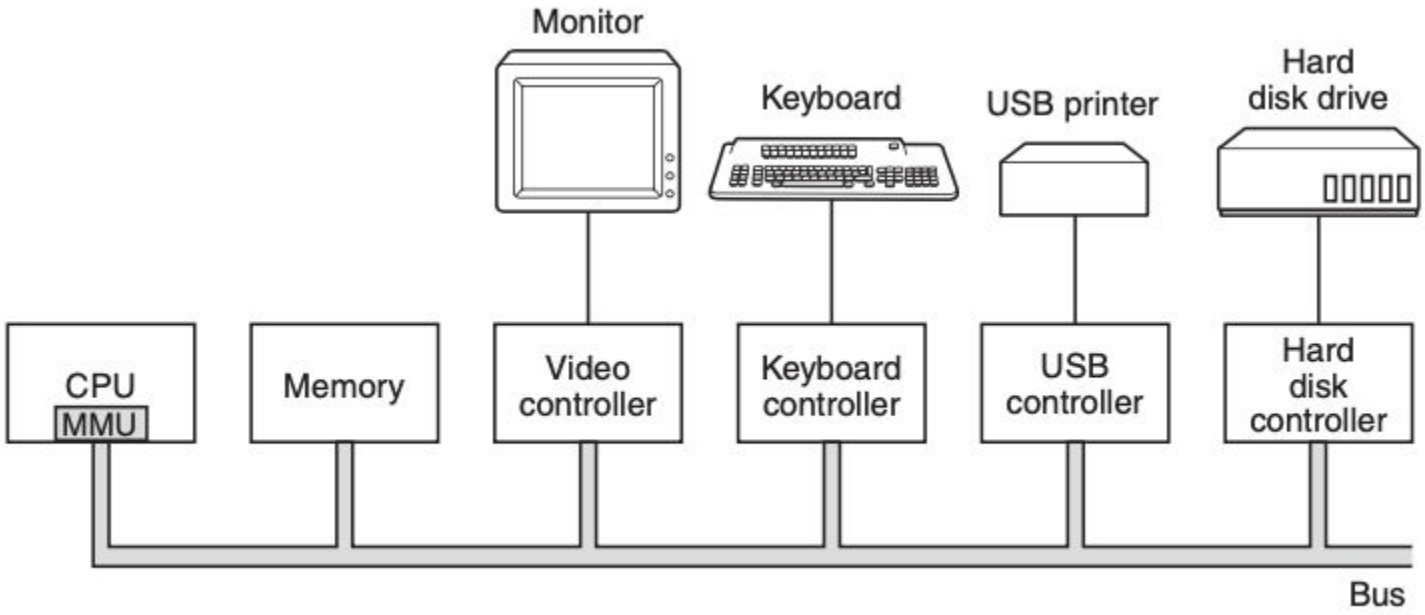
¿Qué es un sistema operativo?
Administrador de recursos

- ¿Qué pasaría si 2 o más procesos intentan imprimir un documento sin un SO de por medio?

- Administrar recursos incluye multiplexar estos recursos
  Tiempo
  Espacio

- Ejemplos

Revisión del hardware



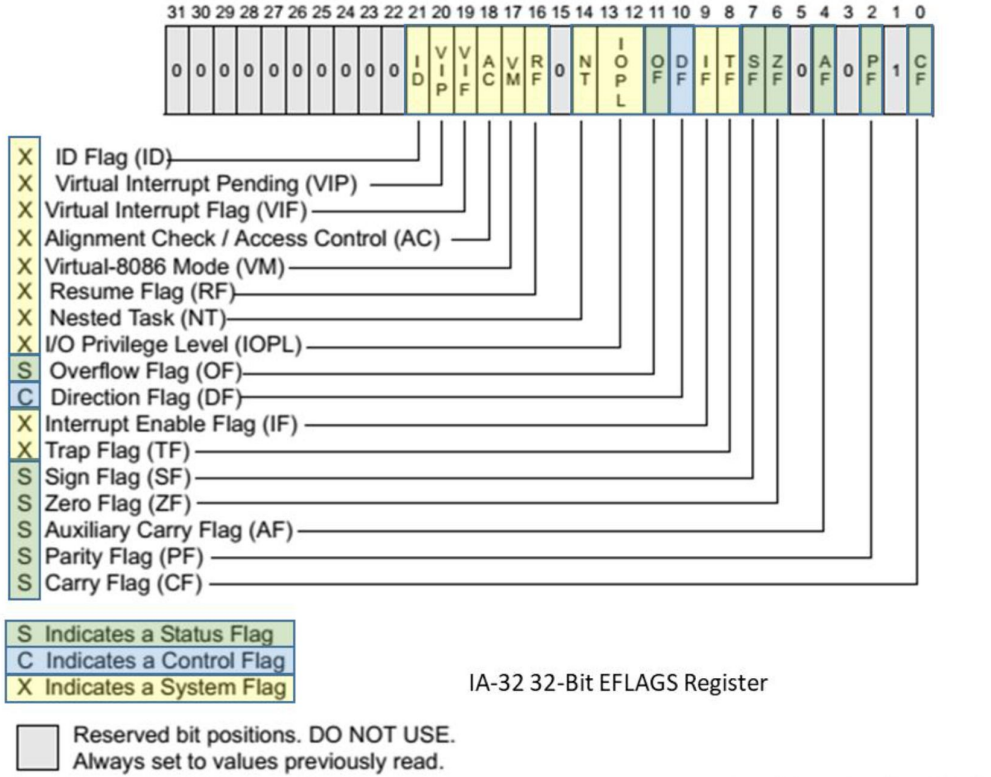
Revisión del hardware

Procesador

- El ciclo básico del CPU es obtener la siguiente instrucción de la memoria, determinar su tipo y operandos y ejecutarlas
  Repetir.
- Cada procesador tiene su conjunto de instrucciones
- Registros vs. memoria
- Registros especiales
  Instruction Pointer (IP)
  Stack Pointer (SP)
  Program Status Word (PSW / RFLAGS)
- Context switch
- Pipeline vs superscalar CPU
- Modos de ejecución

Revisión del hardware

Procesador



PSW

https://k0deless.github.io/posts/RE-Introduction/

Revisión del hardware

Procesador

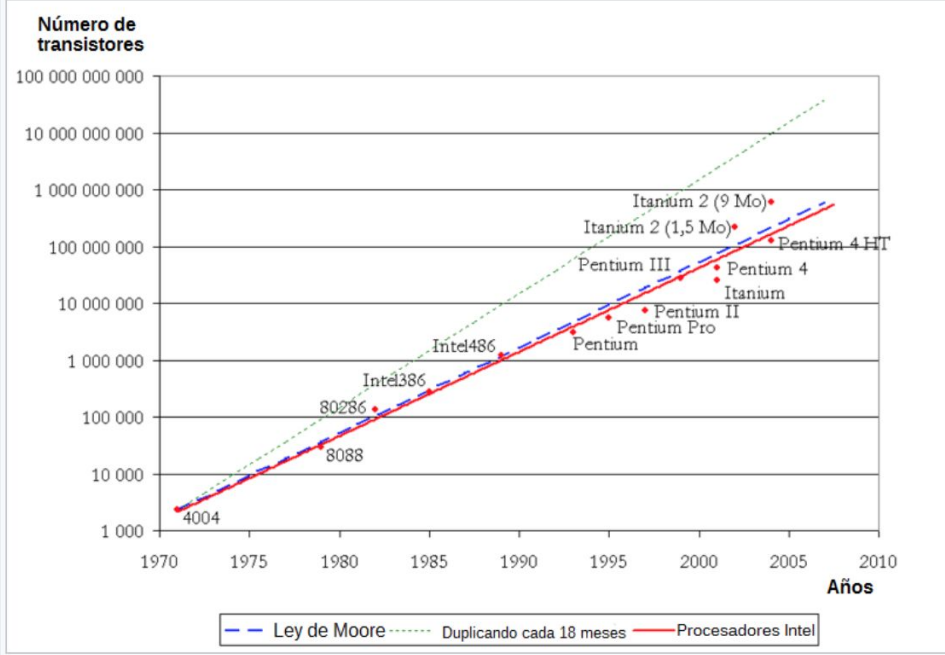
- **Multithreading**

Mantener el estado de 2 threads e intercambiar entre ellos rápidamente cuando sea necesario. No es paralelismo.

- **Multicore**

Replicar los núcleos independientes. Pueden soportar múltiples threads.

Ley de Moore



La ley de Moore comparada con los datos históricos.

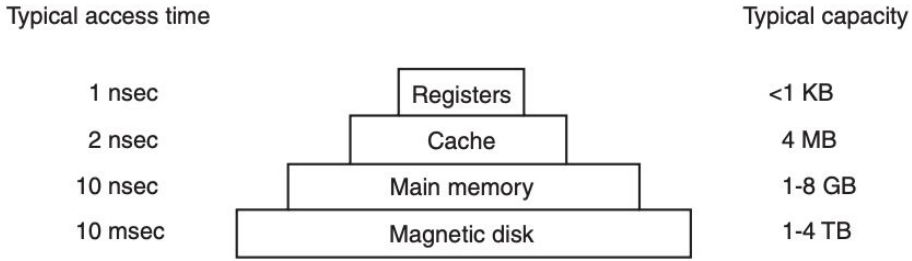
https://es.wikipedia.org/wiki/Ley_de_Moore#cite_note-IntelInterview-2

Revisión del hardware

Memoria

- Idealmente la memoria debería ser:
  Extremadamente rápida (más que la ejecución de una instrucción)
  Absurdamente grande (para albergar a todos los procesos juntos)
  Asquerosamente barata

- Jerarquía de memoria



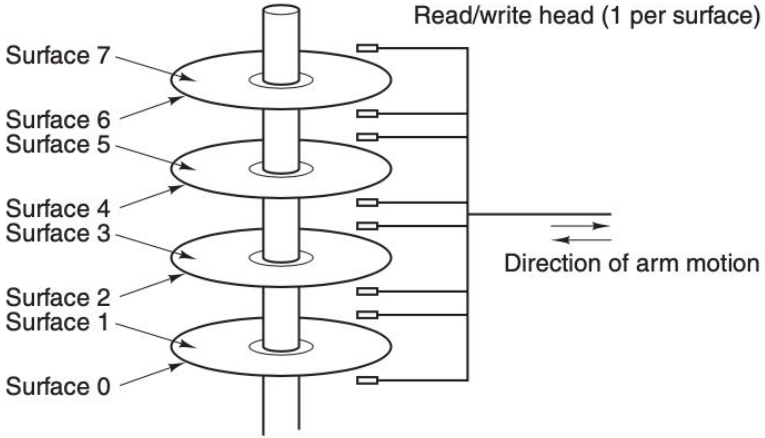
- Ejemplos de cache

- Cuándo guardar algo en la cache
- Dónde guardarlo en la cache
- Qué entrada eliminar si hace falta
- Dónde guardarlo en la memoria principal

¿Por qué **random** access memory?

Revisión del hardware

Disk



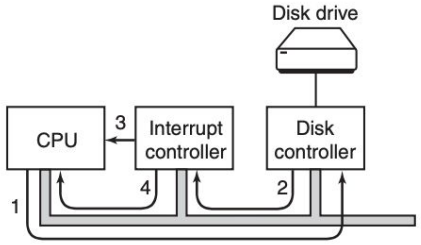
- Dada una posición de los cabezales, cada cabezal puede leer un **track**, todos los **tracks**, forman un **cylinder**.
- Cada **track** está dividido en **sectors**.
- **SSD** (Solid State Drives)
- **Memoria Virtual**
- “How do Hard Disk Drives Work?” - Branch Education - 15:16 - YouTube

Revisión del hardware

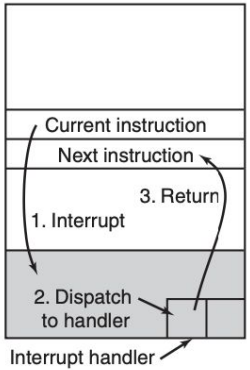
Dispositivos I/O (Input/Output)

- Un dispositivo consta de 2 partes
  Controlador (abstracción)
  Dispositivo

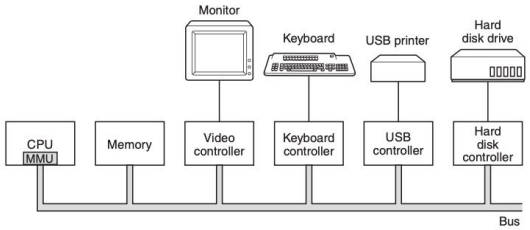
- 3 formas de IO
  Busy waiting
  Interrupción
  DMA (Direct Memory Access)



(a)



(b)



1. El driver le indica al controlador qué hacer. Este inicia la operación en el dispositivo.
2. El controlador señaliza al controlador de interrupciones (CI).
3. Cuando el CI está listo (otras interrupciones) setea el pin correspondiente en el CPU
4. El CI escribe el número de dispositivo (vector de interrupciones) en el bus.

Revisión del hardware
Booteo

- La placa madre posee un programa llamado **BIOS** (Basic I/O System)
  Flash RAM, no volátil pero actualizable por el SO
- La **BIOS** posee instrucciones para interactuar con la pantalla, teclado y disco

1. Chequear cuánta memoria tenemos y si ciertos dispositivos como el teclado están instalados y responden correctamente.
2. Escanea buses PCIe y PCI en busca de dispositivos instalados.
3. Determina el dispositivo de booteo.
4. Se carga el primer **sector** en memoria y se ejecuta
5. Este **sector** suele leer una tabla de particiones y se se carga un segundo **sector**
6. Se carga en memoria el SO y se ejecuta.
7. El SO consulta a la BIOS los dispositivos conectados

El zoológico de los Sistemas Operativos

- **Mainframe:** Mucho almacenamiento y memoria. Orientado a muchos procesos. Soporta trabajos por lotes, transacciones y timesharing. OS/390, tendencia a UNIX.
- **Server:** Computadoras personales o incluso mainframes. Impresoras, archivos o webs. Solaris, FreeBSD, Linux y Windows Server.
- **Multiproceso:** (Necesaria la separación)
- **Personal Computer:** Proveer soporte a un único usuario. Linux, FreeBSD, Windows y Apple's OS X.
- **HandHeld Computer:** Tablets y smartphones. Android y iOS
- **Embedded:** Electrodomésticos y teléfonos antiguos. ROM, no permite la instalación de aplicaciones. Embedded Linux, QNX y VxWorks.
- **Sensor-node:** Redes de nodos. Poseen CPU RAM y ROM. Sensan diferentes condiciones. Conexión inalámbrica. TinyOS
- **Real Time:** Necesitamos que una acción ocurra en un momento específico. hard vs soft. Industria, aviónica, milicia y multimedia. eCos y FreeRTOS
- **Smart Card:** Pagos electrónicos. Del tamaño de una tarjeta de crédito. Restricciones de potencia y memoria muy altas. Obtienen energía por inducción o al conectarse a un lector.

El zoológico de los Sistemas Operativos

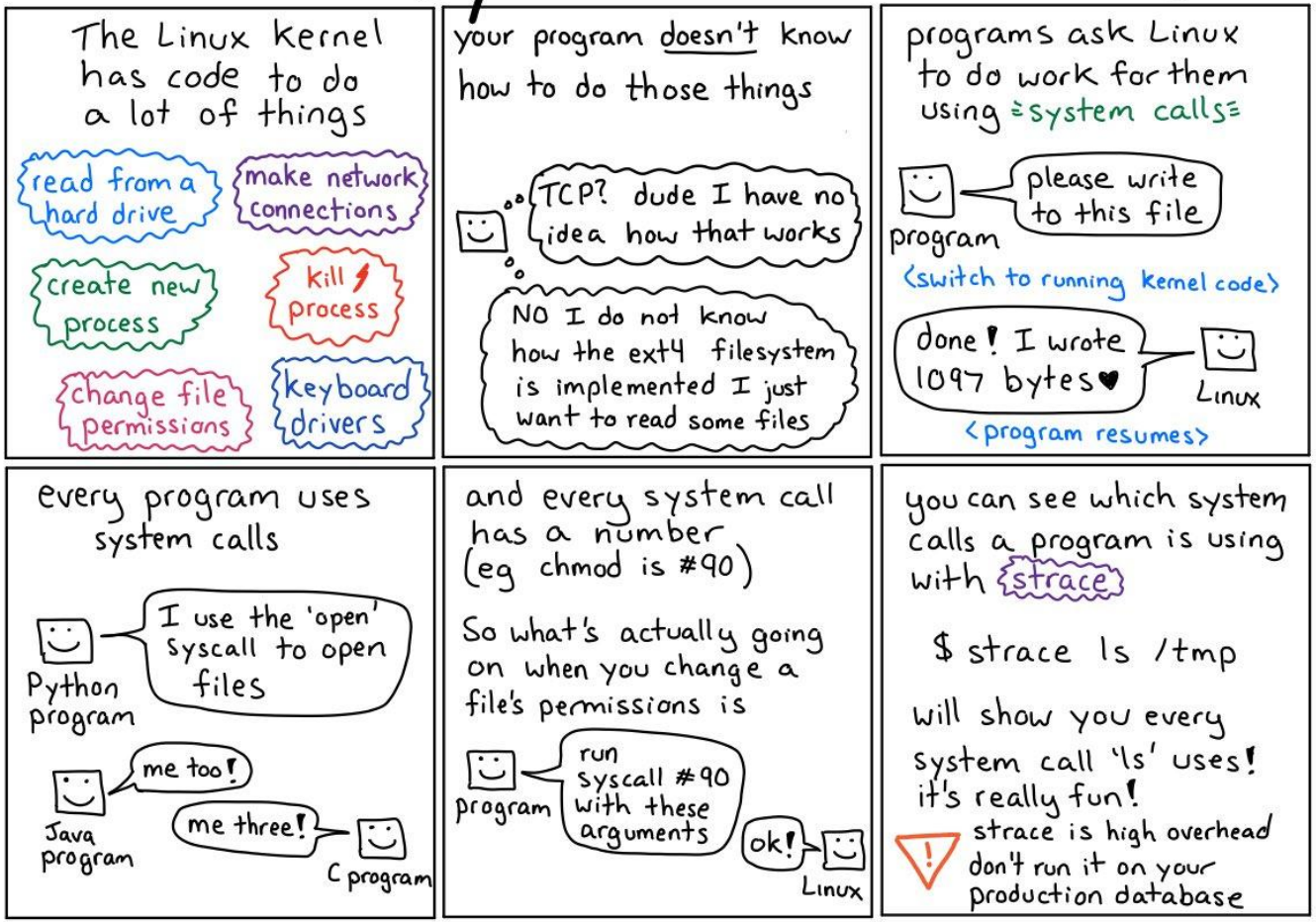
System calls

- Proveer un conjunto abstracto y limpio de los recursos (hardware) **-> entender qué hace el SO**
- Administrar estos recursos **-> transparente para el proceso**

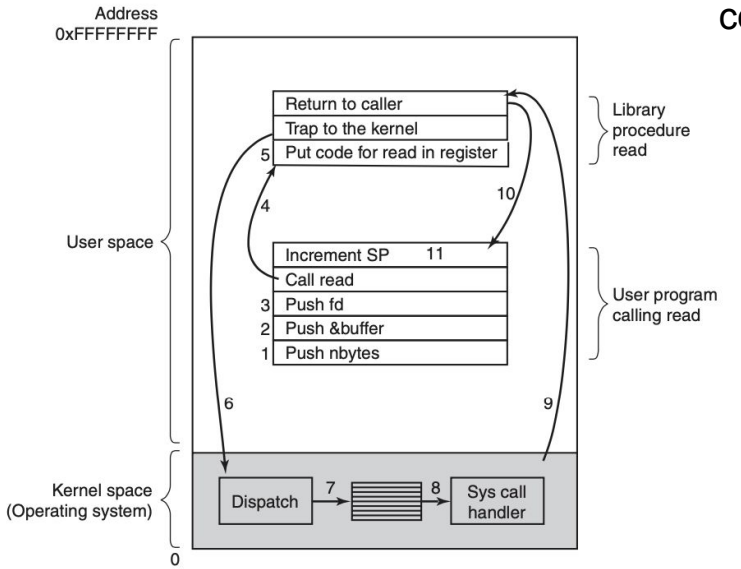
- Se puede ver como una simple llamada a función que además cambia a modo kernel
- El mecanismo para realizar una system call depende de la arquitectura y debe expresarse en assembler.
- Se provee una librería para poder realizarlas desde un lenguaje de alto nivel

system calls

Julia Evans
@b0rk



System calls



count = read(fd, buffer, nbytes);

- El caller pushea los parámetros (1, 2 y 3)
- Se llama a **lib_read** (4)
- **lib_read** setea un registro (*RAX*) con el # de syscall (5)
- **lib_read** ejecuta una instrucción TRAP (int 0x80 / syscall) (6)
- El dispatcher selecciona el handler correspondiente en función del # de syscall (7)
- El handler de read **A** leerá *nbytes* del file descriptor *fd* y lo guardará en *buffer* (8) o **B** bloqueará el proceso.
- El retorno vuelve a user-mode y continúa la ejecución de **lib_read** (9)
- **lib_read** retorna al caller (10)
- El caller restaura el SP (11)

System calls
POSIX

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

System calls
POSIX

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

System calls

Usos desde la shell - pseudocódigo

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, parameters);      /* read input from terminal */

        if (fork( ) != 0) {                     /* fork off child process */
             /* Parent code. */
             waitpid(–1, &status, 0);           /* wait for child to exit */
        } else {
             /* Child code. */
             execve(command, parameters, 0);    /* execute command */
        }
}
```

System calls

Usos desde la shell - real (xv6)

```
int main(void) {
  static char buf[100];

  // ...

  // Read and run input commands.
  while(getcmd(buf, sizeof(buf)) >= 0){
    if(buf[0] == 'c' && buf[1] == 'd' && buf[2] == ' '){
      // Chdir must be called by the parent, not the child.
      buf[strlen(buf)-1] = 0;  // chop \n
      if(chdir(buf+3) < 0)
        printf(2, "cannot cd %s\n", buf+3);
      continue;
    }
    if(fork1() == 0)
      runcmd(parsecmd(buf));
    wait();
  }
  exit();
}
```

https://github.com/mit-pdos/xv6-public/blob/master/sh.c

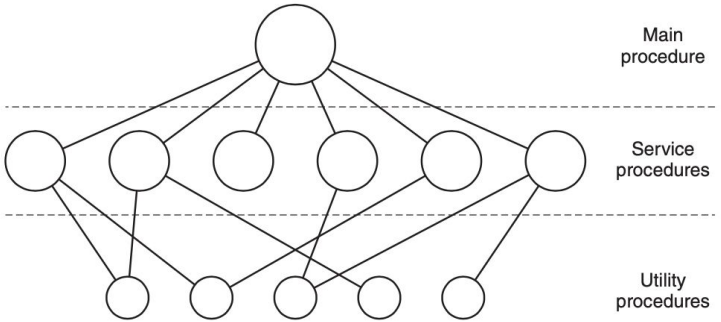
Estructura de un sistema operativo

- Monolithic systems
- Microkernels
- Virtual machines

Estructura de un sistema operativo

Monolithic systems

- Todo el sistema operativo se compila como un único binario.
- Toda función tiene visibilidad del resto de las funciones. Eficiente pero complejo.
- Un error en cualquier función hace fallar el sistema operativo completo
- Soportan **shared libraries** o **DLLs**
- De todos modos, existe una estructura

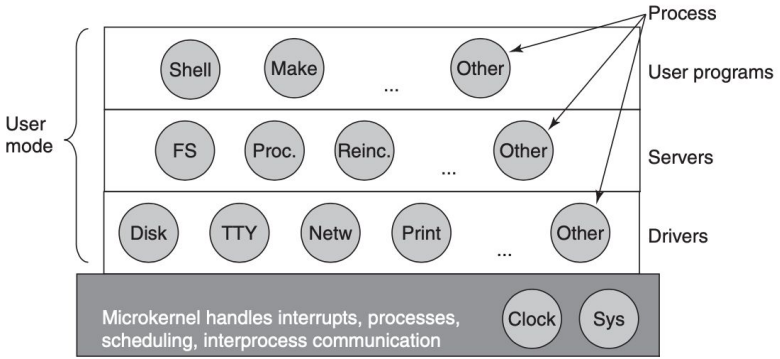


- Linux, FreeBSD, OpenBSD, NetBSD, Microsoft Windows (95, 98, Me), Solaries, HP-UX, DOS, OpenVMS, XTS-400.

Estructura de un sistema operativo

Microkernels

- El objetivo es dejar en el kernel tan poco código como se pueda.
- Un error fuera del kernel no hace fallar al sistema operativo completo.
- El resto de componentes corre en user-mode.
- Alta confiabilidad (Real time).

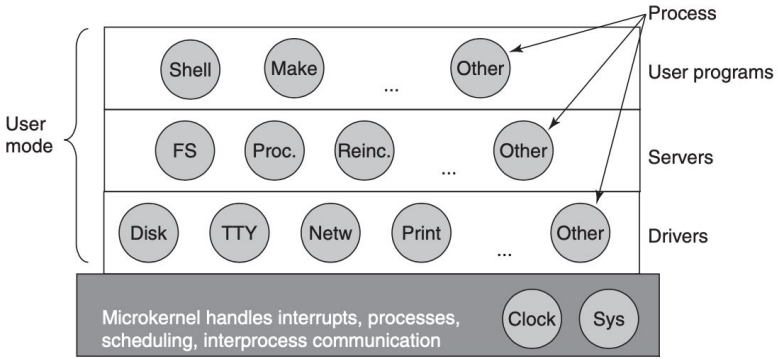


- Integrity, K42, L4, PikeOS, QNX, Symbian y MINIX

Estructura de un sistema operativo

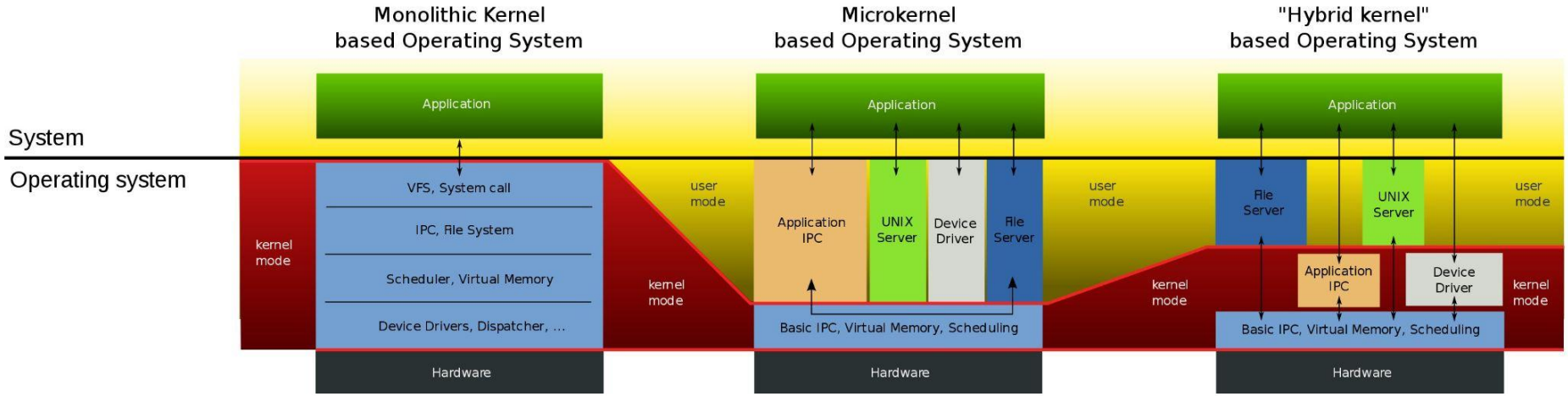
Microkernels - MINIX

- **Reincarnation server**
  Padre de todos los drivers y servers
  Chequea constantemente si los drivers y servers están vivos
  Limpia los drivers y servers muertos
  Chequea en una tabla la acción por defecto. Por ejemplo, reiniciarlo



Estructura de un sistema operativo

Monolithic vs Microkernels



https://en.wikipedia.org/wiki/Monolithic_kernel

Estructura de un sistema operativo

Microkernels - Mecanismo vs política

- La separación entre mecanismo y política es una estrategia que permite reducir el tamaño del kernel

- Colocar el mecanismo en el kernel y la política por fuera, por ejemplo, asignar prioridades a los procesos y ejecutarlos según estas prioridades:
  Elegir al proceso de mayor prioridad (mecanismo) puede estar en el kernel
  Asignar las prioridades (política) puede estar por fuera del kernel.

Estructura de un sistema operativo

Virtual machines

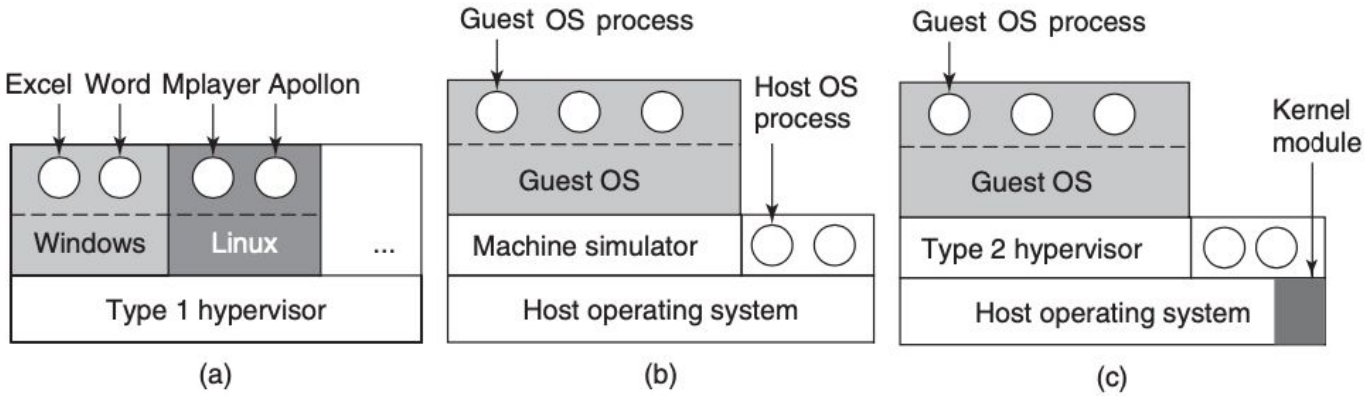
- Tradicionalmente Mail server, FTP server, Web server, etc corren en computadoras separadas
- Popular para Web hosting: servidor dedicado vs. compartido - costo y flexibilidad
- Uso personal

- El CPU debe ser virtualizable: SO corriendo en user mode ejecuta instrucción privilegiada, es necesario que haga un TRAP al virtualizador para emularla en software
- Pentium ignoraba estas instrucciones aunque existían intérpretes como Bochs (bajo rendimiento)

Estructura de un sistema operativo

Virtual machines

- **Virtual machine monitor** o **type 1 hypervisor**: Se ocupa de proveer los servicios necesarios.
- **Type 2 hypervisor**: Tiene un SO por debajo que se encarga de abstraer los procesos, file system, etc. **QEMU** y **BOCHS**.
- Sistemas híbridos.



- Virtual Machines vs Containers - PowerCert Animated Videos - 8:56 - YouTube

Práctica

1. ¿Cuáles son las 2 principales funciones de un sistema operativo?
2. Al usar cache, la memoria principal se divide en **cache lines** de 32 o 64 bytes generalmente. Siempre se almacena una **cache line** completa. ¿Qué ventajas hay respecto a almacenar de a 1 byte?
3. Las instrucciones relacionadas a I/O suelen ser privilegiadas ¿Por qué?
4. ¿Cómo facilita el diseño de un SO disponer de 2 modos de ejecución (kernel / user)?
5. ¿Cuáles de las siguientes instrucciones debería estar permitida solo en kernel mode?
   a. Deshabilitar interrupciones
   b. Leer la hora
   c. Setear la hora
   d. Cambiar el mapeo de memoria
6. Considere un sistema con 2 CPU y 2 threads por CPU (hyperthreading). Suponga que se inician 3 programas con tiempos de ejecución de 5, 10 y 20 ms respectivamente. ¿Cuánto demora la ejecución de estos programas? Los programas son 100% CPU bound, no se bloquean y no cambian de CPU.
7. ¿Qué es una instrucción TRAP? Explique su uso en sistemas operativos
8. Desde el punto de vista del programador, una syscall es como una simple función. ¿Es relevante para el programador conocer si una función resulta en una syscall?

Glosario

- Multiplexar
- PSW - Program Status Word
- Pipeline
- Superscalar
- Multithreading
- Multicore
- Ley de Moore
- HDD - Hard Disk Drive
- SSD - Solid State Drive
- POSIX - Portable Operating System Interface
- Monolithic
- Microkernel
- Principio de localidad o cercanía de referencias