Sistemas Operativos

Dr. Luis Gerardo de la Fraga

E-mail: fraga@cs.cinvestav.mx
http://cs.cinvestav.mx/~fraga

Departamento de Computación
Cinvestav

8 de mayo, 2015

Contenido general del curso

1. Introducción
2. Arquitectura de computadoras y de sistemas operativos
3. Manejo de procesos
4. Manejo de dispositivos de entrada y salida
5. Manejo de memoria
6. Sistema de archivos
7. Virtualización de SO
8. Sistemas operativos distribuidos

- El curso se impartirá en dos partes
- Aquí veremos las primeras cuatro unidades
- Realizaremos cuatro prácticas
- Página del curso:
  `http://cs.cinvestav.mx/~fraga/Cursos/SistemasOperativos/2015`

Prácticas

1. Instalar virtualbox y ejecutar minix de forma virtual
2. Usar la máquina virtual para arrancar linux y reducir al mínimo el tamaño del software.
3. Configuración de dispositivos dentro del linux mínimo
4. Programación con hilos de los problemas clásicos sobre compartición de recursos.

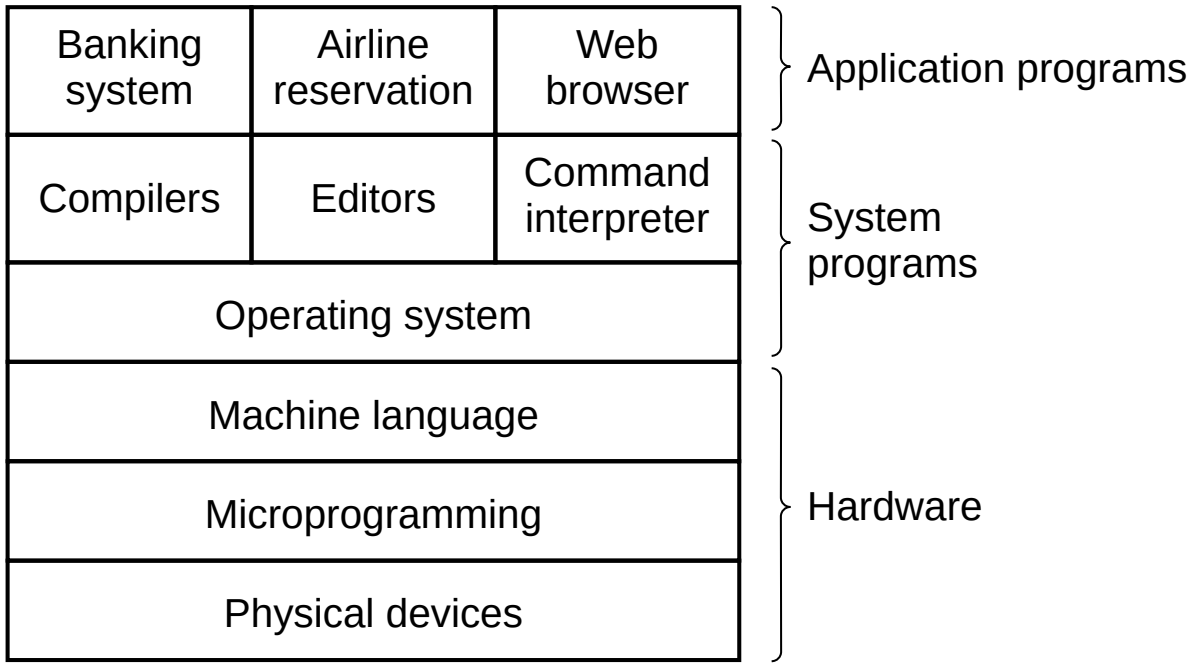
Libro de texto:

- A.S. Tanenbaum, A.S. Woodhull,
  Operating systems, design and implementation.
  2nd edition. Prentice Hall.

La figuras mostradas en estas transparencias fueron descargadas de `http://www.cs.vu.nl/~ast/books/`. El profesor A.S. Tanenbaum las puso libremente disponibles (¡Gracias!).

Introducción

- Sin software una computadora es básicamente un bulto de metal
- Con el software, una computadora puede almacenar, procesar, recuperar información, desplegar documentos, buscar en internet y realizar muchas otras actividades útiles.
- Una computadora moderna consiste de: procesadores, memoria principal (RAM), discos, periféricos, interfaces de red y algunos otros dispositivos de entrada y salida.
- Se pueden realizar programas en la computadora porque existe un software que aísla todos los detalles de cómo programar directamente el hardware de una computadora.
- Este software crea una **máquina virtual** que es más fácil de entender y programar.
- Y este software es el **sistema operativo**



Una computadora y los dos tipos de programas (software) que utiliza.

- El hardware está compuesto por circuitos integrados, alambres, fuente de alimentación, monitores de despliegue, etc.
- **microprogramación** es una capa de software primitivo que directamente controla algunos dispositivos y provee una interfaz más limpia hacia la siguiente capa.
- El microprocesador tiene un intérprete que extrae las instrucciones en lenguaje de máquina como ADD, MOVE y JUMP y las ejecuta en una serie de pasos pequeños.
- Toda la serie de instrucciones que interpreta el microprograma define lo que es el *lenguaje de máquina*.
- El lenguaje de máquina consiste de un conjunto de entre 50 y 300 instrucciones (la mayoría para mover datos, realizar operaciones aritméticas y comparar valores).

- Los microprocesadores **RISC** no presentan el nivel de microprogramación.
- La máquina con un microprocesador RISC ejecuta directamente las instrucciones.
- Una de las funciones principales del sistema operativo (SO) es ocultar toda esta complejidad.
- El SO da al programador un conjunto de instrucciones más conveniente para trabajar.

- Sobre el SO se encuentran todo el resto del software
- Se tiene el intérprete de comandos (el [programa] **shell**),
- el sistema de ventanas, compiladores, editores y programas similares de aplicación independiente.
- Todos estos programas no son parte del sistema operativo

- El sistema operativo es la porción del software que se ejecuta en **modo del núcleo** o **modo supervisor**
- Este está protegido en hardware de su manipulación por parte del usuario
- Los compiladores y editores se ejecutan en **modo usuario**.
- Por ejemplo, un usuario no puede escribir su propia interrupción de disco,
- y normalmente está protegido por hardware sobre cualquier intento del usuario de modificar el comportamiento de la interrupción.

¿Qué es un sistema operativo?

1. Es una máquina extendida.
   Desde este punto de vista, el trabajo del sistema operativo es el de proveer a los usuarios de una **máquina extendida** o **máquina virtual** que abstrae todos los detalles del hardware y que es más conveniente de usar que la máquina actual.
2. Es un gestor de recursos
   Desde este punto de vista, el trabajo del sistema operativo es la gestión eficiente de las diferentes partes del sistema (memoria, disco duro, archivos, impresoras, red, etc.). El SO mantiene un registro de quién está usando que recurso, otorga las requisiciones de recursos, cuenta su uso y es el mediador en conflictos de requisiciones de diferentes programas y usuarios.

Historia de los sistemas operativos

1. Primera generación (1945-1955): tubos de vacío y tarjetas de conexión
2. Segunda generación (1955-1965): transistores y sistemas en lotes
3. Tercera generación (1965-1980): CIs y multiprogramación
4. Cuarta generación (1980-presente): computadoras personales
5. Quinta generación (2010(?)-): teléfonos inteligentes (?)

Antes de la primera generación

- La primera computadora verdadera fue diseñada por el matemático inglés Charles Babbage (1792–1871). Este era un artefacto puramente mecánico que nunca llegó a trabajar. Por supuesto, no tenía sistema operativo.
- Babbage descubrió que podría necesitar software para su máquina analítica y contrató a Ada Lovelace, quien era la hija del poeta británico Lord Byron. Ella fue la primera programadora.

Primera generación (1945-1955): tubos de vacío y tarjetas de conexión (1/2)

- Después de Babbage, poco esfuerzo se realizó en la construcción de computadoras hasta el inicio de la Segunda Guerra Mundial
- A la mitad de la década del 1940, Howard Aiken en Harvard,
- John von Neumann en el Instituto de Estudios Avanzados en Princenton,
- J. Presper Eckert y William Mauchley en la Universidad de Pensilvana, y
- Konrad Zuse en Alemania, entre otros,
- construyeron motores de cálculo con tubos de vacío.
- Estás máquinas eran enormes, ocupando un cuarto entero con decenas de miles de tubos de vacío,
- pero mucho más lentas que una calculadora de bolsillo.

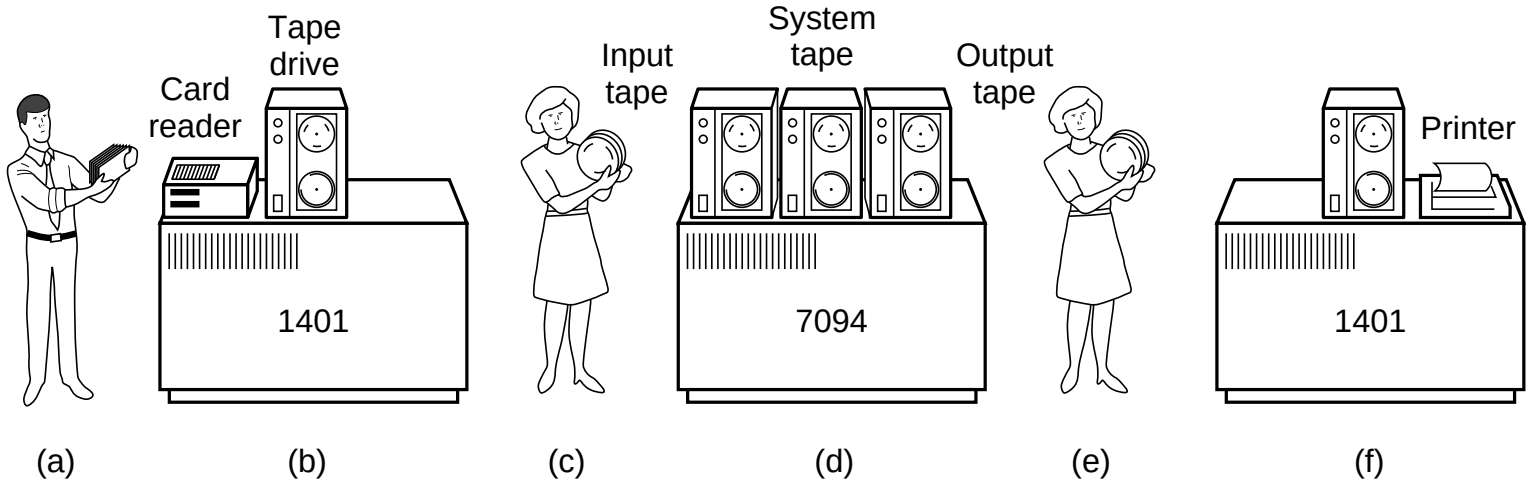
Primera generación (1945-1955): tubos de vacío y tarjetas de conexión (2/2)

- Un grupo de personas diseñaba, construía, programaba, operaba y mantenía cada máquina.
- La programación se hacía en lenguaje de máquina, alambrando tarjetas de conexiones para controlar las funciones básicas de la máquina.
- No se usaba ningún sistema operativo.
- El modo de operación era registrarse para tener una sesión, ir al cuarto de la máquina para insertar su tarjeta alambrada en la computadora, y esperar una horas sin que ningún tubo de vacío se quemara durante la ejecución.
- Todos los problemas que resolvía eran numéricos, como realizar tablas de senos y cosenos.
- A finales de 1950 se introdujeron tarjetas perforadas, lo cual hacía posible realizar programas y leerlos. El resto del procedimiento era igual.

Segunda generación (1955-1965): transistores y sistemas en lotes (1/2)

- A mediados de 1950 se creó el transistor
- Las computadoras se volvieron lo suficientemente confiables para ser manufacturadas y vendidas a los clientes con la expectativa que podrían funcionar lo suficiente para realizar algún trabajo interesante.
- Por primera vez, hubo una separación clara entre diseñadores, operadores, programadores y personal de mantenimiento.
- La máquina se mantenía en un cuarto especial con aire acondicionado con un grupo de operadores profesionales para hacerla funcionar.
- Solo las grandes corporaciones, las agencias de gobierno o las universidades podían pagar el precio de varios millones de dolares por esas máquinas.

Segunda generación (1955-1965): transistores y sistemas en lotes (2/2)



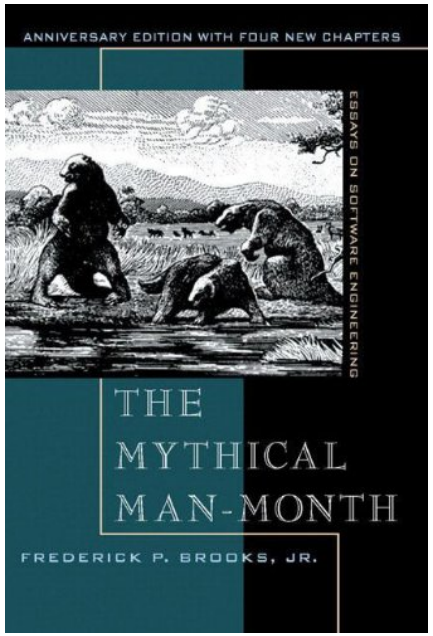
- Uno de los primeros sistemas en lotes:
- (a) los programadores acarrean las tarjetas a la 1401,
- (b) la 1401 lee el lote de trabajos en la cinta,
- (c) los operadores acarrean la cinta de entrada a la 7094,
- (d) la 7094 hace el trabajo,
- (e) los operadores acarrean la cinta de salida a la 1401,
- (f) la 1401 imprime la salida.

Tercera generación (1965-1980): CIs y multiprogramación (1/4)

- A comienzos de los 1960 existían dos lineas de productos:
- Computadoras de gran escala, como la 7094, dedicadas a cálculos científicos en ciencia y en ingeniería;
- Computadoras comerciales, como la 1401, usadas por bancos y compañías de seguros para ordenamiento en cintas e impresiones.
- IBM trató de resolver este problema con su Sistema/360.
- Las 360 fue una serie de máquinas, compatible en software, de tamaño como la 1401 hasta unas más poderosas que la 7094.
- Las máquinas variaban deferían solo en el precio y el rendimiento (tamaño de memoria, velocidad del procesador, número permitido de dispositivos de entrada/salida, etc.).
- Entonces estás máquinas realizaban tanto cómputo comercial como científico.
- IBM siguió esta línea con mejor tecnología en las series 370, 4300, 3080 y 3090.

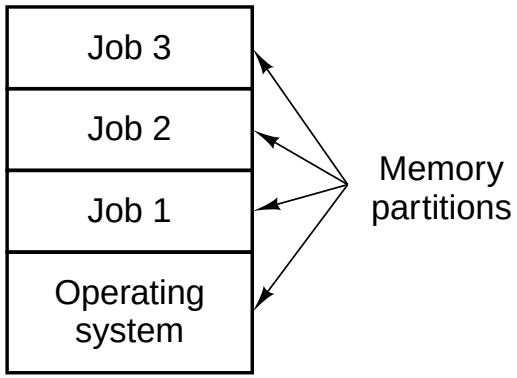
Tercera generación (1965-1980): CIs y multiprogramación (2/4)

- La 360 fue la primera línea de computadoras que usó circuitos integrados, lo que conllevó una mejor ventaja precio/rendimiento sobre la 2a generación.
- Todo el software, incluyendo el sistema operativo, tenía que ser compatible en todos los modelos.
- Fred Brooks, uno de los diseñadores del OS/360 escribió este libro en 1975:



Tercera generación (1965-1980): CIs y multiprogramación (3/4)

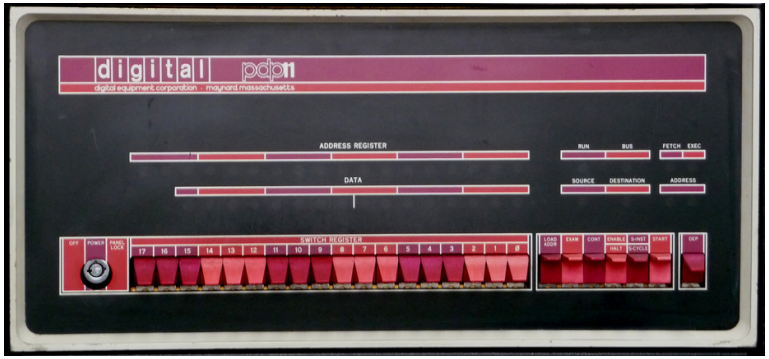
- La más importante contribución del OS/360 fue la **multiprogramación**.



- También se creo el **spooling** (De Simultaneos Peripheral Operation On Line)
- Y la **compartición de tiempo**, una variante de la multiprogramación.

Tercera generación (1965-1980): CIs y multiprogramación (4/4)

- MIT, Bell Labs y General Electric se embarcaron en la construcción de una máquina utilitaria, copiando el modelo del servicio de la energía eléctrica.
- El sistema se llamaría **MULTICS** (De MULTIplexed Information and Computing Service)
- Esta máquina enorme proveería poder de cómputo a toda la ciudad de Boston
- Multics introdujo muchas ideas seminales en la literatura sobre computación
- Otro desarrollo en esta generación fueron las **minicomputadoras**, comenzando con la DEC PDP-1 en 1961
- Tenía 4K de palabras de 18 bits pero costaba 120,000 USD (un 5 % del precio de una 7094)
- El diseño culminó en la PDP-11



- Ken Thompson, quién trabajó en MULTICS, desarrolló UNIX para una PDP-7.
- La IEEE desarrolló un estándar para UNIX que se llama **POSIX**. Este define una interfaz mínima de llamadas al sistema que un sistema UNIX debe soportar.

Cuarta generación (1980-presente): computadoras personales

- Los circuitos integrados permitieron contener miles de transistores sobre un centímetro cuadrado de silicio.
- Las PC no son diferentes, en términos de arquitectura, de las minicomputadoras como la PDP-11,
- su diferencia es el precio.
- La difusión del poder de cómputo, de forma interactiva con gráficos excelentes, conllevó el crecimiento de la industria del software para PCs
- Los dos SO iniciales fueron MSDOS de Microsoft y UNIX
- La primera versión de MSDOS fue primitiva, subsecuentes versiones incluyeron características más avanzadas, muchas tomadas de UNIX.
- Windows al principio fue más un shell que un SO real.
- Windows 95 ya no necesitaba MSDOS
- UNIX dominó en estaciones de trabajo y computadoras de alto desempeño

La historia de GNU/Linux

1. El proyecto GNU lo comenzó Richard Stallman en 1984 en el MIT[1]
2. GNU significa *GNU is Not Unix*
3. El primer proyecto fue crear el compilador gcc
4. El segundo fue el editor emacs
5. "Computer users should be free to modify programs to fit their needs, and free to share software, because helping other people is the basis of society."
6. En 1991, Linus Tovalds desarrolló un núcleo que trabajaba con las bibliotecas de GNU.
7. En 1992 lo liberó generando el sistema GNU/Linux

[1] `http://www.gnu.org/gnu/the-gnu-project.html`

La historia de MINIX

1. La versión 6 del UNIX de AT&T era libre
2. La versión 7 venía con una licencia que prohibía el estudio de su código fuente en cursos
3. Tanenbaum decidió escribir un SO desde cero, compatible con UNIX
4. De esta forma los estudiantes podrían diseccionar un SO real para verlo desde adentro.
5. MINIX significa mini-UNIX
6. Se escribió en C para la IBM/PC
7. Luego se hizo compatible con el estándar POSIX
8. Tienes los comandos básicos de UNIX: cat, grep, ls, make y un shell
9. Tanenbaum nunca quiso agrandarlo,
10. hasta que Linus Tovalds decidió hacer un clon de MINIX para realizar un sistema realmente de producción (no educativo) con muchas más características.

"Los tópicos teóricos que se cubren normalmente en gran detalle en cursos y libros de sistemas operativos, como los algoritmos de despacho, en la práctica no son tan importantes."

"Puntos importante como la E/S y los sistemas de archivos, generalmente se descuidan porque hay poca teoría sobre de ellos."

Conceptos de sistemas operativos

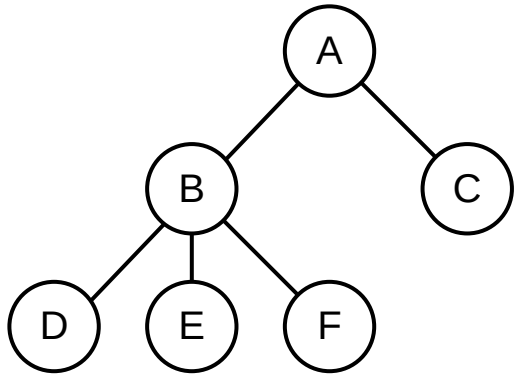
- La interfaz entre el sistema operativo y los programas de usuario está definida en un conjunto de instrucciones llamado **llamadas al sistema**.
- De forma general se pueden dividir en dos categorías:
  - Procesos
  - Archivos
- El shell es el intérprete de comandos de MINIX

Procesos (1/3)

- Un proceso es un programa en ejecución
- Un proceso contiene:
  - un espacio de direcciones (desde 0 hasta un máximo en el que el proceso puede leer y escribir)
  - un conjunto de registros (que incluye el contador de programa, el apuntador a la pila y otros más).
  - Y toda la información necesaria para ejecutar el programa.
- Toda esta información se almacena en una **tabla de procesos**

Procesos (2/3)

- Un proceso (suspendido) consiste de su espacio de direcciones y su entrada en la tabla de procesos (que contiene sus registros, entre otras cosas).
- Un proceso crea otros procesos llamados **procesos hijos**
- Procesos que cooperan para realizar algún trabajo se comunican usando **comunicación interprocesos** (IPC en inglés).



Un árbol de procesos

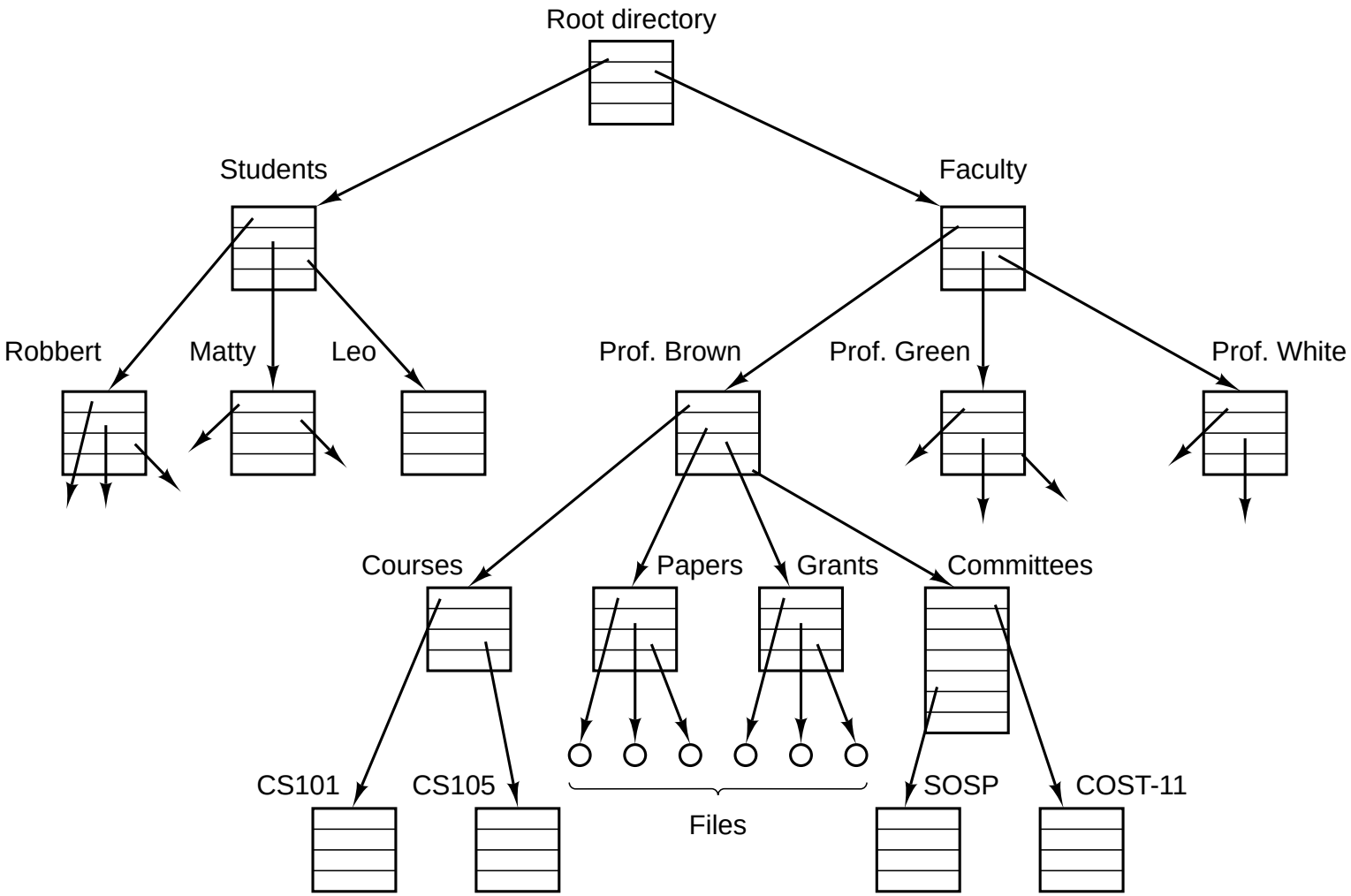
Procesos (3/3)

- Otras llamadas al sistema para procesos localizan memoria (o la liberan si ya no la usan),
- para esperar la terminación de un proceso hijo, y
- para superponer su programa con otro distinto.
- Las **señales** son interrupciones de software que se usan para suspender su ejecución, salvar sus registros en la pila y comenzar a ejecutar un procedimiento especial para el manejo de la señal.
- Muchas trampas de hardware (como ejecutar una instrucción ilegal) también se convierten en señales.
- Cada persona autorizada para una MINIX se le asigna un **uid** (identificador de usuario)
- Cada proceso tiene el identificador de la persona que lo ejecutó
- Un proceso hijo tiene el mismo identificador que su padre
- Un uid, el super usuario, tiene poderes especiales y puede violar muchas reglas de protección.

Archivos (1/4)

- Llamadas al sistema son necesarias para crear, borrar, leer y escribir archivos.
- Antes de que puede leerse un archivo, hay que abrirlo;
- después de leerlo, hay que cerrarlo.
- Los archivos se mantienen en **directorios**
- Cada archivo puede especificarse con su **nombre de camino**
- El directorio más arriba de la jerarquía es el **directorio raíz**
- Cada proceso tiene su **directorio de trabajo**

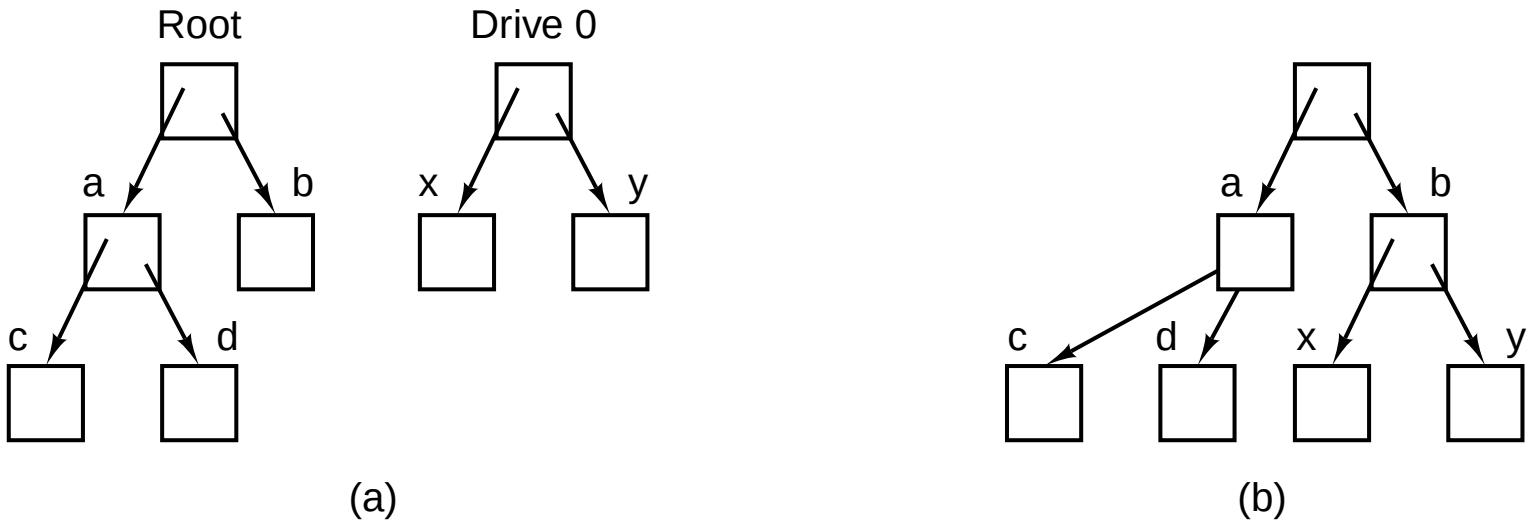
Archivos (2/4)



Un ejemplo de un sistema de archivos

Archivos (3/4)

- Archivos y directorios están protegidos con un código de 9 bits,
- **rwx** bits para leer, escribir y acceso para ejecución,
- para el propietario, el grupo y el todo el mundo.
- Al abrir un archivo se checan sus permisos
- Si el acceso es permitido, el sistema retorna un número entero llamado un **descriptor de archivo** que se usa en subsecuente operaciones.
- Si el acceso es denegado, entonces se regresa un código de error.
- Se pueden montar sistemas de archivos.



(a) Antes de montarse, los archivos en el manejador 0 no pueden accesarse. (b) Después de montarse, se vuelven parte de la jerarquía de archivos.

Archivos (4/4)

- Existen también **archivos especiales**
- Se proveen estos archivos para que los dispositivos de E/S se manejen como archivos
- Existen dos clases:
  - **archivos especiales de bloque**
  - **archivos especiales de caracter**. Estos modelan impresoras, modems y otros dispositivos que aceptan o producen un flujo de caracteres.
- Las **tuberías** es un pseudoarchivo que se usa para conector dos procesos.

En abril de 1991, Linus Torvalds, un estudiante de 21 años de la Universidad de Hersinki, en Finlandia, empezó a trabajo en algunas ideas pora un sistema operativo. Empezó con un interruptor de tareas para un Intel 80386 en lenguaje ensamblador y un manejador de terminal. El 25 de agosto de 1991, Torvalds puso lo suguiente en `comp.os.minix`, un grupo de noticias en Usenet:

> *I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since April, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).*
>
> *I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months [...] Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.*

`http://en.wikipedia.org/wiki/Linux_kernel`

Llamadas del sistema

- 53 llamadas al sistema existen en MINIX
- 37 (?) en MINIX 3
- `http://wiki.minix3.org/doku.php?id=developersguide:kernelapi`

Administración de procesos

`pid=fork()`	Crea un proceso hijo idéntico al padre
`pid=waitpid(pid,&statloc,opts)`	Espera que el hijo termine
`s = wait( &status )`	Versión antigua de waitpid
`s = execve(name, argv, envp )`	Reemplaza la imagen central de un proceso
`exit( status )`	Termina el proceso en ejecución y regresa `status`
`size = brk( addr )`	Pone el tamaño del segmento de datos
`pid = getpid()`	Regresa el id del proceso
`pid = getgrp()`	Regresa el id del grupo de procesos
`pid = setsid()`	Crea una nueva sesión y regresa su id del grupo de procesos
`l=ptrace(req,pid,addr,data)`	Se usa para debugear

Para manejo de señales

`s = sigaction(sig, &act, &oldact)`	Define una acción a tomar en una señal
`s = sigreturn(&context)`	Regreso de una señal
`s = sigprocmask(how, &set, &old)`	Examina o cambia la máscara de una señal
`s = sigpending(set)`	Obtiene el conjunto de señales bloqueadas
`s = sigsuspend(sigmask)`	Reemplaza la máscara de señal y suspende el proceso
`s = kill(pid, sig)`	Envía una señal a un proceso
`residual = alarm(seconds)`	Pone la alarma del reloj
`s = pause()`	Suspende al que llama hasta la siguiente señal

Para manejo de archivos (1/2)

`fd = creat(name,mode)`	Forma obsoleta de crear un archivo nuevo
`fd = mknod(name, mode, addr)`	Crea un nodo-i regular, especial o de directorio
`fd = open(file, how, ...)`	Abre un archivo para lectura, escritura o ambos
`s = close(fd)`	Cierra un archivo abierto
`n = read(fd, buffer, nbytes)`	Lee datos de un archivo a un bufer
`n = write(fd, buffer, nbytes)`	Escribe datos de un bufer a un archivo
`pos = lseek(fd, offset, whence)`	Mueve el puntero a un archivo
`s = stat(name, &buf)`	Obtiene el información de estado de un archivo
`s = fstat( fd, &buf )`	Obtiene el información de estado de un archivo
`fd = dup( fd )`	Localiza un nuevo descriptor de archivo para un archivo abierto
`s = pipe(&fd[0])`	Crea una tubería

Para manejo de archivos (2/2)

`s = ioctl(fd, request, argp)`	Realiza una operación especial sobre un archivo
`s = access(name, amode)`	Checa la accesibilidad de un archivo
`s = rename(old, new)`	Da a un archivo un nombre nuevo
`s = fcntl(fd, cmd, ...)`	Pone un candado a un archivo y otras operaciones

Para manejo del sistema de directorios y archivos

`s = mkdir(name,mode)`	Crea un directorio nuevo
`s = rmdir(name,mode)`	Borra un directorio
`s = link(name1, name2)`	Crea una nuevo entrada, name2, apuntando a name1
`s = unlink(name)`	Borra la entrada a un directorio
`s = mount(special, name flag)`	Monta un sistema de archivos
`s = umount(special)`	Desmonta un sistema de archivos
`s = sync()`	Pasa todos los bloques en caché a disco
`s = chdir(dirname)`	Cambia el directorio de trabajo
`s = chroot(dirname)`	Cambia al directoria raíz

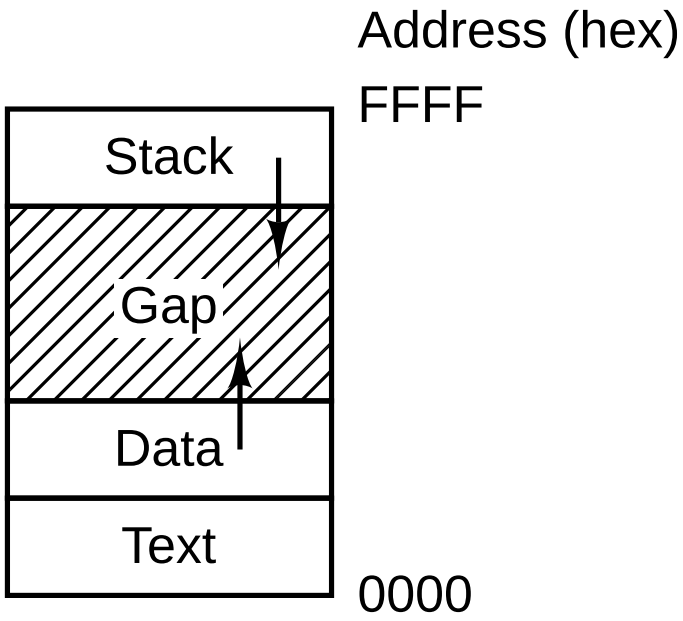
Para protección

`s = chmod(name,mode)`	Cambia los bits de protección de un archivo
`uid = getuid()`	Obtiene el uid del que llama
`gid = getgid()`	Obtiene rl gid del que llama
`s = setuid( uid )`	Pone el uid del que llama
`s = chown(name, owner, group)`	Cambia del propietario y grupo a un archivo
`oldmask = umask(complmode)`	Cambia la máscara del modo

Para manejo del tiempo

`seconds = time(&seconds)`	Obtiene el tiempo transcurrido desde 1 de enero de 1970
`s = stime(tp)`	Pone el tiempo transcurrido desde el 1 de enero de 1970
`s = utime(file, timep)`	Pone el time del último acceso a un archivo
`s = times(buffer)`	Obtiene los tiempos del usuario y del sistema

- Con **fork** se crea un proceso nuevo,
- se crea exactamente una copia del proceco original, incluyendo todos los descriptores de archivos y registros.
- Después del fork, el proceso original y la copia van por caminos separados.
- Los dos tienen una copia exacta de las variables cuando se hace el fork, pero después se pueden modificar y el cambio en un una no afecta a la otra copia.



Los segmentos de un proceso: texto, datos y la pila

- El texto lo comparten el padre y el hijo.
- El fork regresa un valor: cero en el hijo, y en el padre es el **pid**.

Ejemplo del trabajo que hace el shell

```
while( TRUE ) {
    lee_comando( comando, parametros );

    if( fork( ) != 0 ) {
        /** Código del padre.
            Espera que termine el hijo **/
        waitpid( -1, &estado, 0 );
    }
    else {
        /** Código del hijo.
            Ejecuta el comando **/
        execve( comando, parametros, 0 );
    }
}
```

```
void evalua_circuito( char *miargv[] )
{
    pid_t pid;
    int ret;

    switch ( pid = fork() ) {
        case -1:
            perror("fork()");
            exit(EXIT_FAILURE);

        case 0:  /* in the child */
            ret = execve("/opt/local/bin/ngspice", miargv, NULL );
            fprintf( stderr, "Cir %d\n", ret );
            /* exit( ret ); ** only happens if execve(2) fails **/

        default:
            if ( waitpid(pid, &ret, 0) < 0 ) {
                perror("wait pid()");
                exit(EXIT_FAILURE);
            }
    }
}
```

Llamadas al sistema para señalizar (1/3)

- Las señalas manejan la comunicación inesperada para los procesos:
- como interrumpir un comando (con ctl-C o DEL),
- para atrapar las detecciones en hardware como una instrucción ilegal o sobreflujo de punto flotante.

Llamadas al sistema para señalizar (2/3)

- Cuando una señal se envía a un proceso, el proceso simplemente se mata.
- Puede realizar otra acción usando la llamada del sistema **SIGACTION**, que pone la dirección del procedimiento que manejará la señal.
- Cuando se genera la señal el estado del proceso se guarda en la pila y entonces se llama al manejador de señales
- Cuando termina su trabajo en manejador de señales llama a **SIGRETURN**
- Las señales puedes bloquearse con SIGPROCMASK
- Un proceso se mata enviando la señal 9 (SIGKILL), que no puede cacharse o ignorarse.

Llamadas al sistema para señalizar (3/3)

- Para ignorar las señales DEL y quit (ctrl-\ )

```
sigaction( SIGINT, SIG_ING, NULL );
sigaction( SIGQUIT, SIG_ING, NULL );
```

- SIGALARM suspende un proceso cierta cantidad de tiempo.

Estructura de un sistema operativo

1. Sistemas monolíticos
2. Sistemas en capas
3. Máquinas virtuales
4. Modelo cliente-servidor

Sumario (1/2)

- Un **sistema operativo** tiene cuatro componente principales:
  1. el gestos de procesos,
  2. el gestor de dispositivos de entrada-salida,
  3. el gestor de memoria, y
  4. el gestor de archivos.
- El corazón de cualquier sistema operativo es el **conjunto de llamadas del sistema** que este puede manejar.
- Estas llamadas al sistema es lo que el sistema operativo realmente hace.

Sumario (2/2)

- El conjunto de llamadas del sistema para MINIX se puede dividir en seis grupos:
  1. Creación y terminación de procesos
  2. El manejador de señales
  3. Para la escritura y lectura de archivos
  4. Para la gestión de directorios
  5. Protección de información
  6. El mantenimiento del rastreo en el tiempo