

Introducción a los sistemas operativos

Rodolfo Baader


Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Sistemas Operativos, segundo cuatrimestre de 2014


(2) Algunas aclaraciones preliminares

- Material original de las teóricas preparado por F. Schapachnik.
- Ampliado y actualizado por R. Baader y D. Fernández Slezak.
- Gráficos extraídos de distintas fuentes, internet, bibliografía de la materia.

(3) Algunas aclaraciones preliminares

- Cosas importantes (tal vez no las únicas): 
- Diapos numeradas.
- Su NO pregunta SÍ molesta.
- Las siguientes cosas no son equivalentes (de a pares):
 - 1 Presenciar esta clase.
 - 2 Leer los apuntes.
 - 3 Presenciar las clases prácticas sobre el tema.
 - 4 Hacer los talleres.
 - 5 Hacer las prácticas.
- ¿Cómo sé si entendí los temas?
 - Los prácticos: si me salen los ejercicios (en un tiempo razonable).
 - Los teóricos: si soy capaz de explicarlos con mis propias palabras.
- Bibliografía
 - Silberschatz, A. and Galvin, P.B. and Gagne, G., *Operating system concepts*, Addison-Wesley.
 - Tanenbaum, A.S., *Modern operating systems*, Prentice Hall New Jersey.

(4) Qué es un SO y por qué dedicarle toda una materia

- Una forma de dividir a los sistemas informáticos:
 - Hardware (lo que se puede patear).
 - Software específico (lo que sólo se puede insultar).
- Hace falta un intermediario entre ellos 
 - ...para que el software específico no se tenga que preocupar con detalles de bajo nivel del HW (visión de usuario).
 - ...para que el usuario use correctamente el HW (visión del propietario del HW).
- Por qué toda una materia:
 - Porque esta capa es suficientemente específica e interesante como para estudiarla en detalle.
 - Porque aquí surgen problemas muy interesantes.
 - Y porque es el marco “natural” para estudiar algunos de esos problemas, que son de carácter más general.

(5) Un poco de Historia

- Década de 1950: aparecen las primeras computadoras comerciales. IBM 7090.



- 1961: Clementina: primera computadora para fines científicos de Argentina.

(6) Máquinas de la época

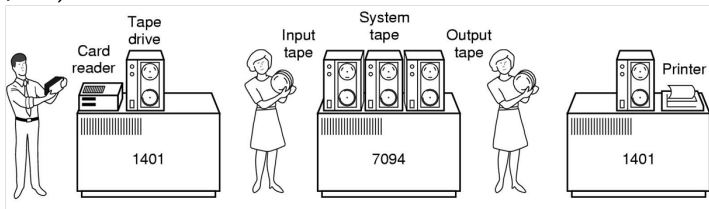
- Las computadoras costaban millones de USD y estaban en ambientes dedicados.
- Miren http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP7090.html
- El usuario llevaba sus tarjetas perforadas en assembler, o típicamente en FORTRAN.
- El operador las ponía en la entrada, las tarjetas se leían, el programa corría y luego imprimía el resultado, que el usuario pasaba a buscar después.



- Problema: mucho tiempo de procesamiento desperdiciado.

(7) Máquinas de la época (cont.)

- Solución: unas computadoras más baratas, que sólo costaban miles de dólares, eran adicionadas al sistema. Sabían hacer tarjeta → cinta, cinta → impresora. Los mainframes aprendieron a leer y escribir cintas magnéticas (mucho más rápido).





- Estos sistemas se conocieron como *sistemas batch*, porque no se los manejaba interactivamente como hoy, sino que los programas se les daban en tandas (cada cinta).
- ¿Quién hacía la intermediación usuario/HW? El operador.
- Queda planteada la primer preocupación importante para los SO: cada usuario sólo debe recibir sus impresiones.

(8) Unos años más tarde...

- La siguiente generación de computadoras, caracterizada por máquinas como la IBM/360, ya tenían un sistema operativo más formal (OS/360 en ese caso).
- Pretendían solucionar un problema importante: mientras se leían las cintas en memoria, el procesador estaba ocioso. Ídem cuando se escribía el resultado.
- Esto es muy indeseable, porque es caro.
- Idea: mientras se accede a los dispositivos, que el procesador procese otro trabajo, aunque sea un pedacito.

(9) Unos años más tarde... (cont.)

- Nace el concepto de *multiprogramación*. De esta manera, el *throughput* o *rendimiento* aumenta. El trabajo j_1 toma el mismo tiempo que antes, o incluso un poco más, pero $j_1 + j_2$ tarda menos. 
- Con él, otro concepto fundamental: la *contención*.  Varios programas pueden querer acceder a un mismo recurso a la vez.

(10) Poco después...

- Usar los sistemas batch era un bajón, especialmente para programar y debuggear. Por eso surgió la idea de conectar muchas terminales a una misma computadora, y darles un poquito de tiempo de procesador a las que están siendo usadas.
- Eso se llama *timesharing*, y es una variación de la multiprogramación.
- El pionero fue MULTICS, del que descende UNIX (Ken Thompson, Dennis Ritchie).
- UNIX es fundamental en la historia de los SO, tanto por los conceptos que introdujo como por su vigencia.
- Linux está inspirado en UNIX.
- Si les interesa su historia (muy divertida):
<http://www.bell-labs.com/history/unix/>


(11) Veníamos diciendo...

- Recordemos:
 - ...para que el software específico no se tenga que preocupar con detalles de bajo nivel del HW (visión de usuario).
 - ...para que el usuario use correctamente el HW (visión del propietario del HW).
- Vimos un poco la segunda parte. Entendamos la primera.

(12) Leer un sector de un disquette...

- Forma de hablar directamente con el HW, para hacer una lectura de una disquettera:
- Empaquetar 13 parámetros en 9 bytes (dirección del bloque, sectores por track, modo de acceso físico, separación entre sectores, etc.).
- Esperar un rato (hay que saber cuánto).
- El controlador del floppy devuelve 23 campos de status y error empaquetados en 7 bytes.
- Además, hay que prender y apagar explícitamente el motor, porque si se lo deja prendido los disquettes se desgastan.
- Y así.
- El resto del HW no es más amigable.
- Tarea para el hogar: tomar una computadora sin SO y hacer un programa que lea nombres del disco, los ordene y los vuelva a escribir ordenados.
- Con SO: 200 LOC.
- Sin SO: para cortar... cualquier tipo de inspiración.

(13) Pasando en limpio...

- Un SO es una pieza de software que hace de intermediario entre el HW y los programas de usuario.
- Tiene que manejar la contención y la concurrencia de manera tal de lograr:
 - Hacerlo con buen rendimiento.
 - Hacerlo correctamente.
- Esto es un problema central. 
- Para lograr todo esto, corre en *nivel de privilegio 0*, es decir, *máximo privilegio*.

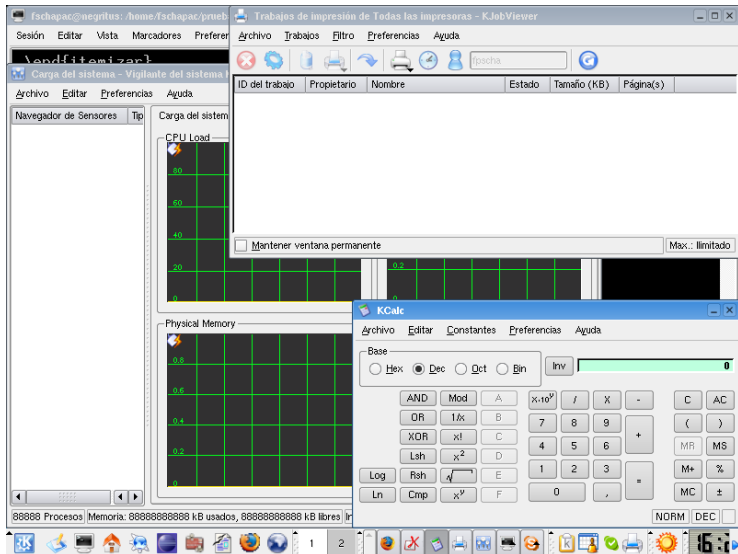
(14) Qué es y qué no es un SO

- Veamos cuánto ocupan algunos sistemas operativos:
- Ubuntu Linux 9.10: “At least 4 GB of disk space”
- FreeBSD 7.2: 5 CDs de instalación.
- Windows Seven: 16 GB para la edición “home basic”, 40 GB para las otras.
- ¡¿Todo eso es necesario?!

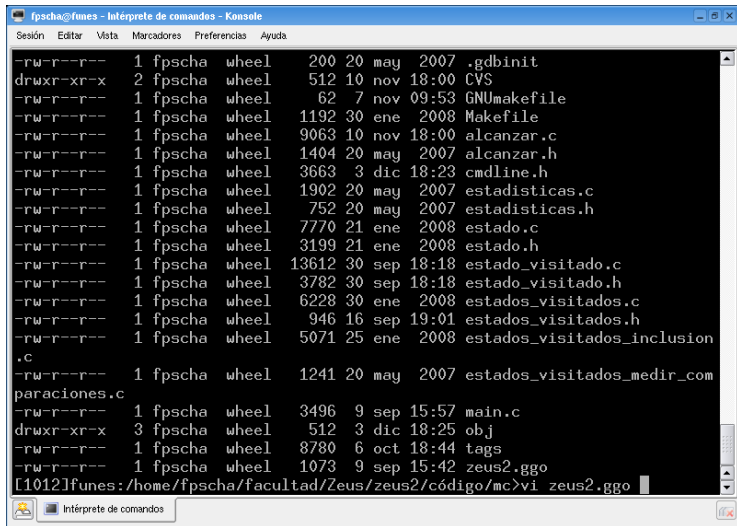
(15) Qué es y qué no es un SO (cont.)



(16) Qué es y qué no es un SO (cont.)



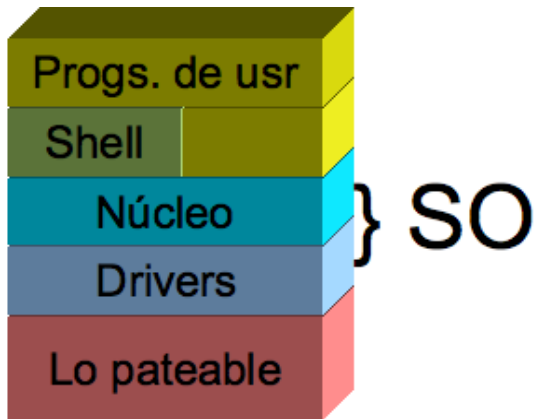
(17) Qué es y qué no es un SO (cont.)



```
fpscha@funes - Intérprete de comandos - Konsole
Sesión  Editar  Vista  Marcadores  Preferencias  Ayuda

-rw-r--r--  1 fpscha  wheel   200 20 may  2007 .gdbinit
drwxr-xr-x  2 fpscha  wheel   512 10 nov  18:00 CVS
-rw-r--r--  1 fpscha  wheel    62  7 nov  09:53 GNUmakefile
-rw-r--r--  1 fpscha  wheel  1192 30 ene  2008 Makefile
-rw-r--r--  1 fpscha  wheel  9063 10 nov  18:00 alcanzarar.c
-rw-r--r--  1 fpscha  wheel  1404 20 may  2007 alcanzarar.h
-rw-r--r--  1 fpscha  wheel  3663  3 dic  18:23 cmdline.h
-rw-r--r--  1 fpscha  wheel  1902 20 may  2007 estadisticas.c
-rw-r--r--  1 fpscha  wheel   752 20 may  2007 estadisticas.h
-rw-r--r--  1 fpscha  wheel  7770 21 ene  2008 estado.c
-rw-r--r--  1 fpscha  wheel  3199 21 ene  2008 estado.h
-rw-r--r--  1 fpscha  wheel 13612 30 sep  18:18 estado_visitado.c
-rw-r--r--  1 fpscha  wheel  3782 30 sep  18:18 estado_visitado.h
-rw-r--r--  1 fpscha  wheel  6228 30 ene  2008 estados_visitados.c
-rw-r--r--  1 fpscha  wheel   946 16 sep  19:01 estados_visitados.h
-rw-r--r--  1 fpscha  wheel  5071 25 ene  2008 estados_visitados_inclusion
.c
-rw-r--r--  1 fpscha  wheel  1241 20 may  2007 estados_visitados_medir_com
paraciones.c
-rw-r--r--  1 fpscha  wheel  3496  9 sep  15:57 main.c
drwxr-xr-x  3 fpscha  wheel   512  3 dic  18:25 obj
-rw-r--r--  1 fpscha  wheel  8780  6 oct  18:44 tags
-rw-r--r--  1 fpscha  wheel  1073  9 sep  15:42 zeus2.ggo
[1012]funes:/home/fpscha/facultad/Zeus/zeus2/código/mc>vi zeus2.ggo
```

(18) Qué es y qué no es un SO (cont.)



(19) Elementos básicos de un SO

- Drivers: programas que son parte del sistema operativo y manejan los detalles de bajo nivel relacionados con la operación de los distintos dispositivos.
- Núcleo o Kernel: es el SO propiamente dicho, su parte central. Se encarga de las tareas fundamentales y contiene los diversos subsistemas que iremos viendo en la materia.
- Intérprete de comandos o Shell: un programa más, que muchas veces es ejecutado automáticamente cuando comienza el SO, que le permite al usuario interactuar con SO. Puede ser gráfico o de línea de comandos. Ejemplos en Unix: `sh`, `ssh`, `ksh`, `bash`.
- Proceso: un programa en ejecución más su espacio de memoria asociado y otros atributos.

(20) Elementos básicos de un SO (cont.)

- Archivo: secuencia de bits con un nombre y una serie de atributos que indican permisos, etc.
- Directorio: colección de archivos y directorios que contiene un nombre y se organiza jerárquicamente.
- Dispositivo virtual: una abstracción de un dispositivo físico bajo la forma, en general, de un archivo, de manera tal que se pueda abrir, leer, escribir, etc.
- Sistema de archivos: es la forma de organizar los datos en el disco para gestionar su acceso, permisos, etc.

(21) Elementos básicos de un SO (cont.)

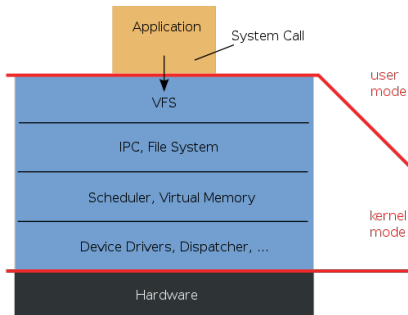
- Directorios del sistema: son directorios donde el propio SO guarda archivos que necesita para su funcionamiento, como por ejemplo, `/boot`, `/devices` o `C:\Windows\system32`.
- Binario del sistema: son archivos, que viven en los directorios del sistema. Si bien no forman parte del kernel, suelen llevar a cabo tareas muy importantes o proveer las utilidades básicas del sistema. Ejemplo:
 - `/usr/sbin/syslogd`: es el encargado de guardar los eventos del sistema en un archivo.
 - `/bin/sh`: el Bourne Shell.
 - `/usr/bin/who`: indica qué usuarios están sesionados en el sistema.
- Archivo de configuración: es un archivo más, excepto porque el sistema operativo saca de allí información que necesita para funcionar. Por ejemplo, `/etc/passwd` o `C:\Windows\system32\user.dat`.

(22) Elementos básicos de un SO (cont.)

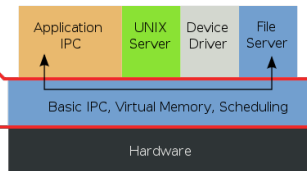
- Usuario: la representación, dentro del propio SO, de las personas o entidades que pueden usarlo. Sirve principalmente como una forma de aislar información entre sí y de establecer limitaciones.
- Grupo: una colección de usuarios.

(23) Kernel Monolítico vs. μ kernels

**Monolithic Kernel
based Operating System**



**Microkernel
based Operating System**



- La idea de microkernel está basada en contrarrestar algunas de las desventajas de un *kernel monolítico* buscando lograr:
 - Mucho menos código privilegiado.
 - Facilidad de actualizaciones.
 - Mayor flexibilidad y extensibilidad.
 - Crash de servicios no tira abajo todo el sistema.
 - Diferentes “sabores” de los servicios.

- Para lograr esto la idea era tener un kernel que hiciera:
 - Manejo básico de memoria.
 - IPC liviano.
 - Manejo básico de E/S.
- Todo lo demás sería provisto por servicios.

(26) μ kernels en la práctica

- En la práctica esto resultó mucho más lento que en los kernels monolíticos.
- Si bien hubo una segunda generación de μ kernels que trató de solucionar estos problemas (y en algún punto lo logró), la idea nunca terminó de ser exitosa del todo desde un punto de vista práctico.
- Con algunas excepciones notables:
 - QNX, un Unix RT con arquitectura μ kernel, diseñado especialmente para sistemas embebidos.
 - MacOS tiene algo de Mach, un microkernel.

(27) El legado de los μ kernels

- De todas maneras algunas ideas sí se tomaron:
 - IPC más rápido.
 - Módulos de kernel.
 - Tratar de sacar *algunos* servicios del kernel (por ejemplo, *portmapper* de RPC).

(28) Haciendo un poco de limpieza



(29) Paseando por Marte



(30) Dónde estamos

- Vimos
 - Qué es un SO.
 - Un administrador de recursos.
 - Una interfaz de programación.
 - Un poco de su evolución histórica.
 - Su misión fundamental.
 - SO batch e interactivos.
 - Hablamos de multiprogramación.
 - Qué cosas son parte del SO y cuáles no.
 - Kernel monolítico y microkernel
- En el taller:
 - Vamos a ver las herramientas básicas que nos brinda un SO.
 - Eso sirve *per se*, pero también porque son necesarias para explorar al propio SO.
- La próxima clase:
 - Vamos a empezar a analizar al SO en tanto interfaz de programación y ver qué funcionalidades nos brinda.
 - Vamos a analizar el concepto de proceso en detalle.