Estructuras 2

Sistemas operativos y distribuidos

Gustavo Distel
gd@cs.uns.edu.ar

DCIC - UNS

Estructuras

Contenido

- Servicios del sistema operativo.
- Interfaz de usuario y sistema operativo.
- Llamadas al sistema.
- Servicios del sistema.
- Enlazadores (*Linkers*) y cargadores (*Loaders*).
- ¿Por qué las aplicaciones son específicas del sistema operativo?
- Diseño e implementación del sistema operativo.
- Estructura del sistema operativo.
- Depuración del sistema operativo.

Estructuras

Introducción

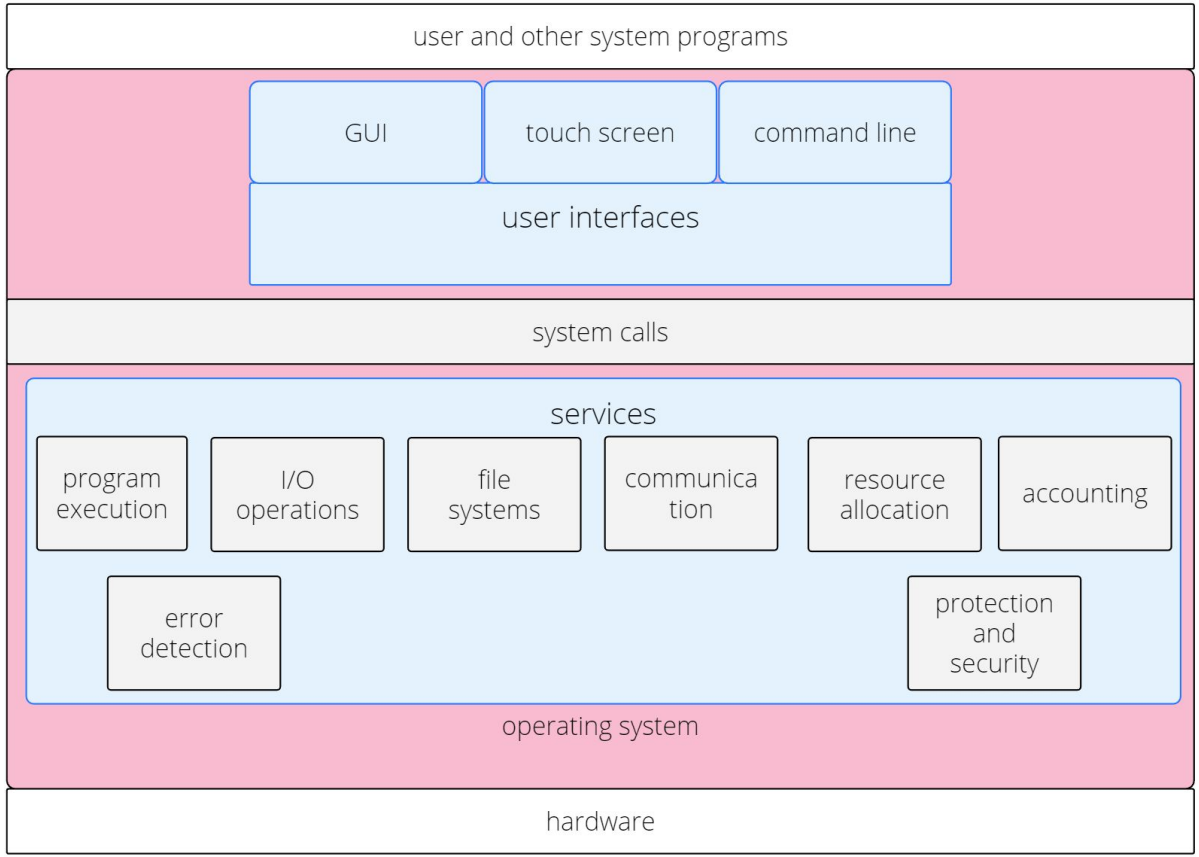
- Un SO se puede analizar desde varios puntos de vista:

Los servicios que provee.

La interfaz a los usuarios y programadores.

Sus componentes e interconexiones.

Servicios del sistema operativo



Servicios del sistema operativo

Los servicios del SO proveen funciones que son útiles para el usuario y los programadores. Se pueden identificar las siguientes clases de servicios:

- Interfaz de usuario (UI)
  - *Graphical user interface (GUI).*
  - *Touch-screen interface.*
  - *Command-line interface (CLI).*
- Ejecución de programas → Cargar un programa en memoria y ejecutarlo.
- Operaciones de E/S → Un proceso puede requerir E/S. Por razones de eficiencia y protección los usuarios generalmente no tienen acceso directo a los dispositivos de E/S.
- Manipulación del sistema de archivos → Un proceso puede requerir leer y escribir archivos y directorios. Muchos SOs proveen varios sistemas de archivos.

Servicios del sistema operativo

- Comunicaciones → Un proceso puede intercambiar información con otro proceso, ya sea en la misma computadora o por red.
    - *Shared memory.*
    - *Message passing.*
- Detección de errores → El SO debe detectar y corregir errores:
    - *CPU*.
    - *HW* de la memoria.
    - Dispositivos de E/S (disco, red o una impresora).
    - Programa del usuario (desbordamiento aritmético o un intento de acceder a una ubicación de memoria ilegal).

Servicios del sistema operativo

Existe otro conjunto de funciones del SO, cuyo propósito no es ayudar al usuario, sino garantizar un funcionamiento eficiente del propio sistema.

- Asignación de recursos:
  - Ciclos de la *CPU* → *CPU-scheduling*.
  - Memoria principal.
  - Almacenamiento de archivos.
  - Dispositivos de E/S.
- *Logging* → Registra el número y tipo de recursos que usa cada proceso. Útil para los administradores de sistemas.
- Protección y seguridad.

Interfaz de usuario

Command-Line Interface (CLI)

- Linux, UNIX y Windows → Tratan al *CLI* como un programa especial que se ejecuta cuando se inicia un proceso o cuando un usuario inicia sesión.
- En sistemas con varios *CLI* para elegir (ej. *Unix-like*) se conocen como *shells*:
  - *Bourne Shell (sh).*
  - *Bourne Again Shell (bash).*
  - *C Shell (csh).*
  - *Korn Shell (ksh).*
- Windows: *PowerShell* y *Command Prompt*.
- La función principal del *CLI* es obtener y ejecutar el siguiente comando ingresado por el usuario. Los comandos pueden ser implementados de dos maneras:
  - El *CLI* contiene el código para ejecutar el comando.
  - Como un programa de sistema.

Interfaz de usuario

Command-Line Interface (CLI)

Terminal — -zsh — 119×30

```
Documents       Library         Music           Public
hacker@hackers-Mac ~ % echo $SHELL
/bin/zsh
hacker@hackers-Mac ~ % uptime
 7:21  up 11 mins, 2 users, load averages: 0.68 1.36 0.90
hacker@hackers-Mac ~ % df -kh
Filesystem      Size   Used  Avail Capacity iused               ifree %iused  Mounted on
/dev/disk1s5   101Gi  9.3Gi   86Gi    10%  468194 9223372036854307613    0%   /
devfs          183Ki  183Ki    0Bi   100%     633                   0  100%   /dev
/dev/disk1s1   101Gi  4.4Gi   86Gi     5%   36466 9223372036854739341    0%   /System/Volumes/Data
/dev/disk1s4   101Gi  1.0Mi   86Gi     1%       1 9223372036854775806    0%   /private/var/vm
map -hosts       0Bi    0Bi    0Bi   100%       0                   0  100%   /net
map auto_home    0Bi    0Bi    0Bi   100%       0                   0  100%   /home
hacker@hackers-Mac ~ % ps
  PID TTY           TIME CMD
  836 ttys000    0:00.10 -zsh
hacker@hackers-Mac ~ %
hacker@hackers-Mac ~ % uname
Darwin
hacker@hackers-Mac ~ % uname -a
Darwin hackers-Mac.local 19.0.0 Darwin Kernel Version 19.0.0: Fri May 24 17:36:10 PDT 2019; root:xnu-6041.0.0.111.5~1/R
ELEASE_X86_64 x86_64
hacker@hackers-Mac ~ %
```

MAC OS Terminal

Interfaz de usuario

Command-Line Interface (CLI)

Windows PowerShell

```
PS C:\Users\hacker> Get-Process | select -first 5

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
    727      63   179300     195972       9,83   3960   1 AcroRd32
    526      26    11652      29184       0,50   5748   1 AcroRd32
    393      23    13548      32148       3,16   7476   1 ApplicationFrameHost
    156       9     1424       6820              3304   0 armsvc
    141       9     1700       8456       0,03   7868   1 browser_broker


PS C:\Users\hacker> ls | select -first 5


    Directory: C:\Users\hacker


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        11/8/2017     11:41                .android
d-----         1/8/2017     12:06                .AndroidStudio2.3
d-----        20/3/2018     11:34                .AndroidStudio3.0
d-----         2/5/2017     10:32                .eclipse
d-----        20/3/2018     11:42                .gradle


PS C:\Users\hacker> dir | select -first 5


    Directory: C:\Users\hacker


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        11/8/2017     11:41                .android
d-----         1/8/2017     12:06                .AndroidStudio2.3
d-----        20/3/2018     11:34                .AndroidStudio3.0
d-----         2/5/2017     10:32                .eclipse
d-----        20/3/2018     11:42                .gradle
```

Interfaz de usuario

Command-Line Interface (CLI)

Terminal - gdistel@battlecruiser:~

File Edit View Terminal Tabs Help

Linux XFCE Terminal

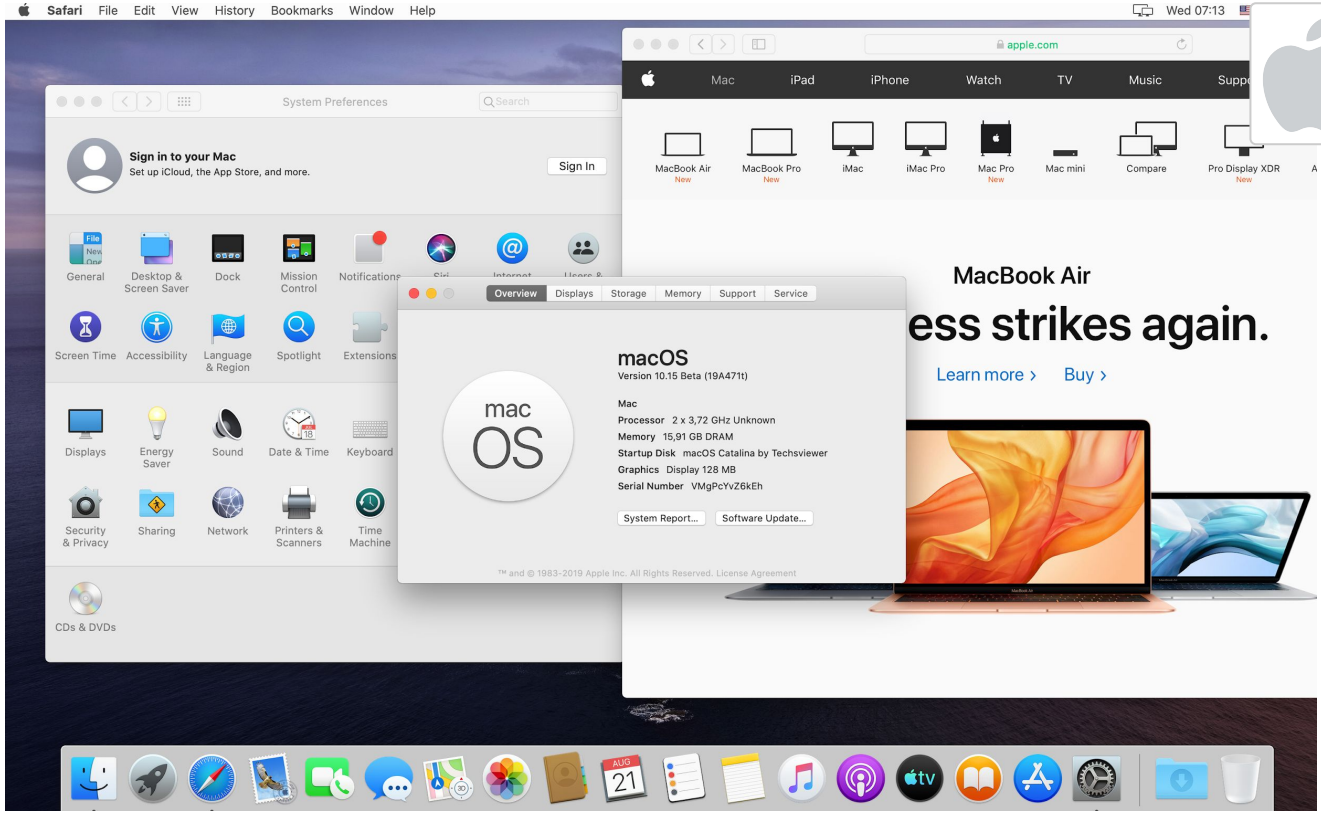
```
  1  [||                                        1.3%]   Tasks: 141, 681 thr; 1
  2  [||                                        1.3%]   Load average: 0.13 0.39
  3  [||                                        2.0%]   Uptime: 03:33:54
  4  [||                                        2.6%]
  Mem[||||||||||||||||||||||||||||||||||||  4.76G/14.6G]
  Swp[                                        0K/7.57G]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
22948 gdistel    20   0  220M  4776  3692 R  2.0  0.0  0:00.80 htop
 5182 gdistel    20   0 1261M  329M  140M S  1.3  2.2  5:40.65 /opt/google/chrome-unstable/chrome
19843 gdistel    20   0 1828M  134M  104M S  0.7  0.9  0:49.99 doublecmd
22855 gdistel    20   0  635M 51904 36912 S  0.7  0.3  0:01.74 xfce4-terminal
 5253 gdistel    20   0 1087M  215M 87216 S  0.7  1.4  0:27.28 /opt/google/chrome-unstable/chrome --type=rendere
19783 gdistel    20   0 1397M  500M  159M S  0.0  3.3  4:58.15 /opt/google/chrome-unstable/chrome --type=rendere
 1520 gdistel    20   0 1011M  162M  122M S  0.0  1.1  5:19.72 /usr/libexec/Xorg vt2 -displayfd 3 -auth /run/use
 1191 root       20   0 15220  9280  7372 S  0.0  0.1  0:38.80 /sbin/dhclient -d -q -sf /usr/libexec/nm-dhcp-hel
21036 gdistel    20   0 1131M  276M 87672 S  0.0  1.8  2:07.44 /opt/google/chrome-unstable/chrome --type=rendere
 5255 gdistel    20   0 1360M  317M 87032 S  0.0  2.1  1:21.65 /opt/google/chrome-unstable/chrome --type=rendere
19925 gdistel    20   0 1889M 1002M  131M S  0.0  6.7  0:46.95 /usr/lib64/libreoffice/program/soffice.bin --impr
 5265 gdistel    20   0  998M  316M 98720 S  0.0  2.1  2:35.63 /opt/google/chrome-unstable/chrome --type=rendere
 1589 gdistel    20   0 2231M 25792 19212 S  0.0  0.2  0:16.12 /usr/bin/startdde
 5220 gdistel    20   0  560M   99M 67284 S  0.0  0.7  0:59.51 /opt/google/chrome-unstable/chrome --type=utility
 1792 root       20   0 1208M 20468 15156 S  0.0  0.1  0:08.91 /usr/libexec/deepin-daemon/dde-system-daemon
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Interfaz de usuario

Graphical User Interface (GUI)

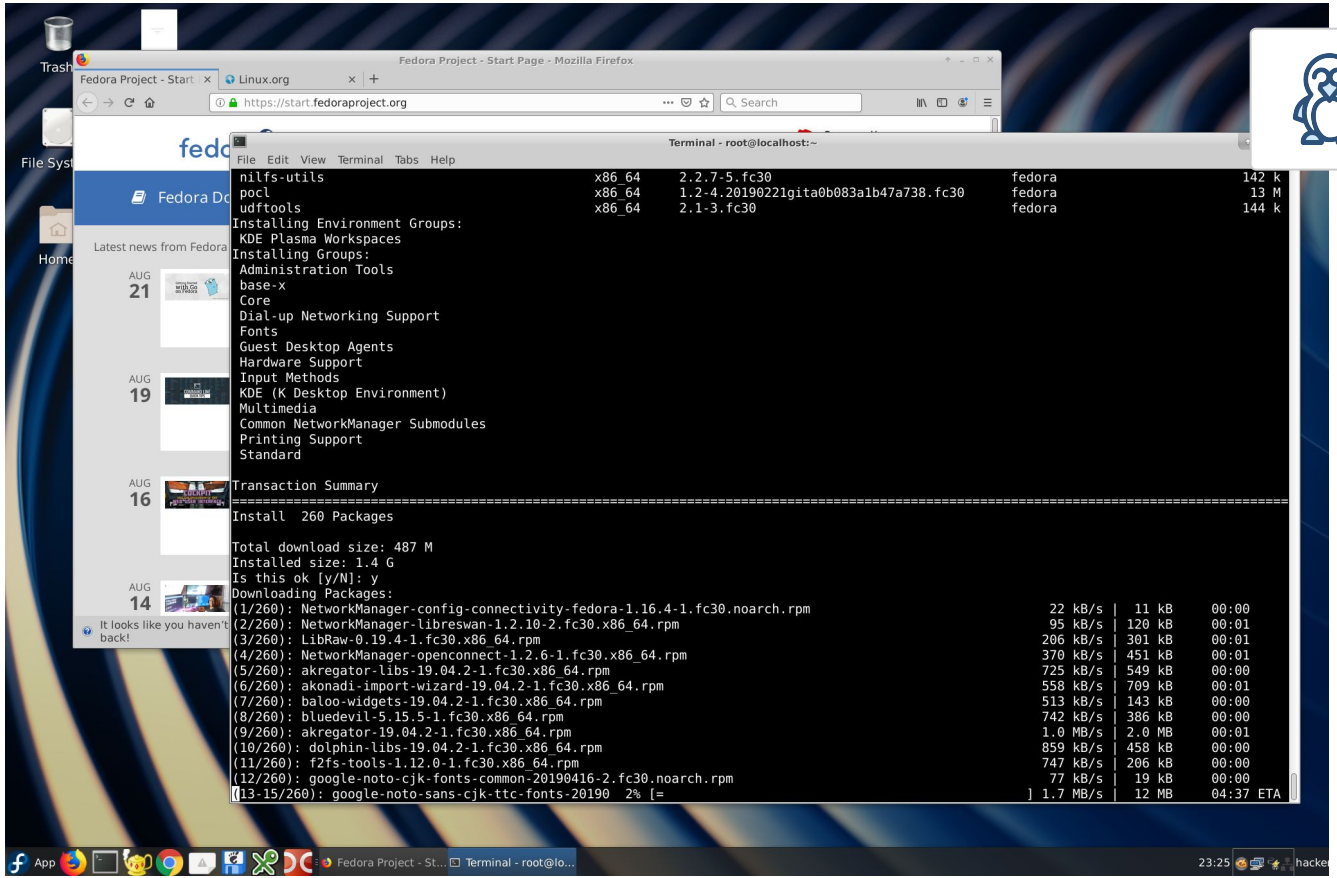
Metáfora de escritorio con ventanas, menús, íconos y uso de mouse además de teclado.



MAC OS Catalina

Interfaz de usuario

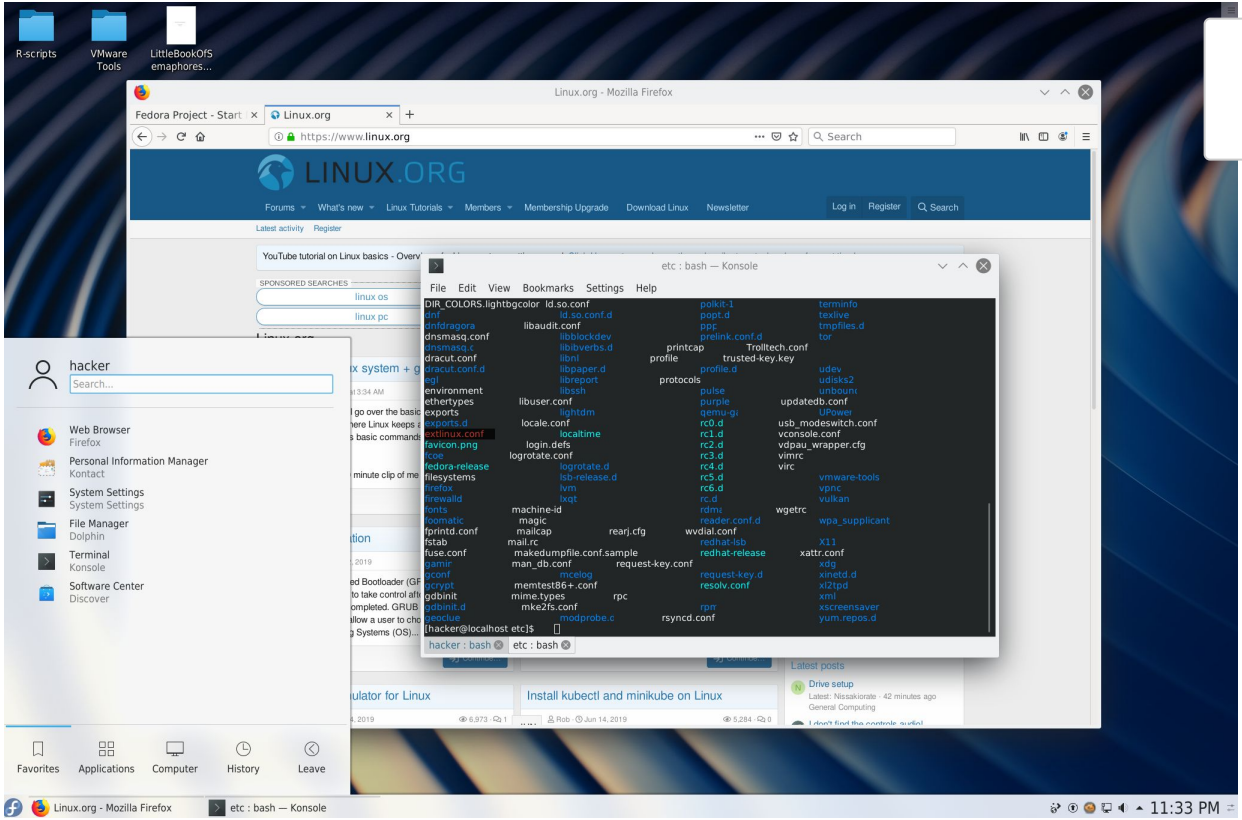
Graphical User Interface (GUI)



XFCE en Fedora

Interfaz de usuario

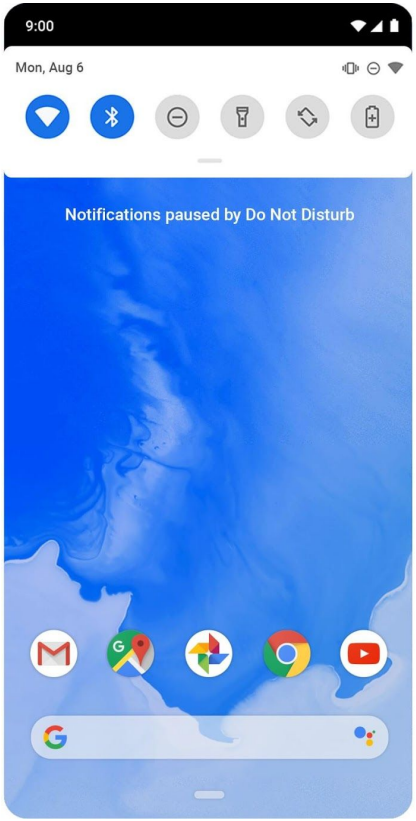
Graphical User Interface (GUI)

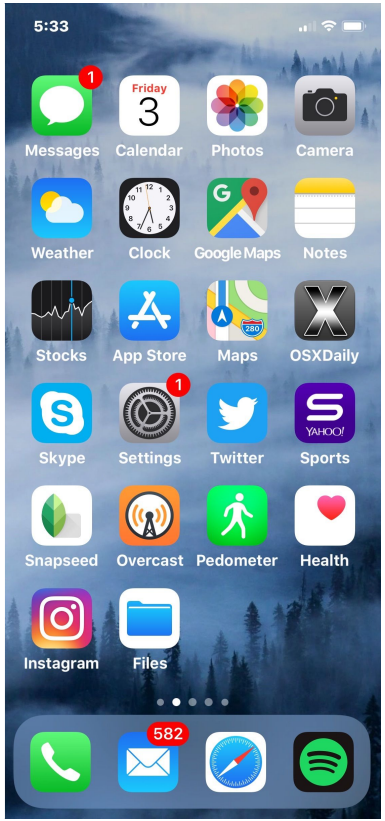


KDE en Fedora

Interfaz de usuario

Touch-Screen Interface

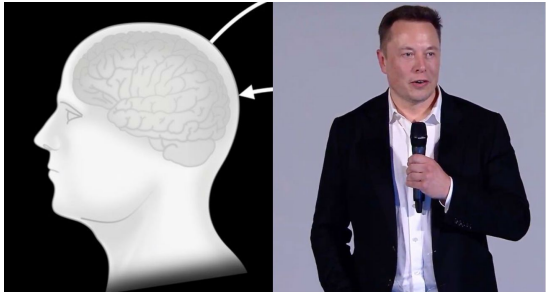


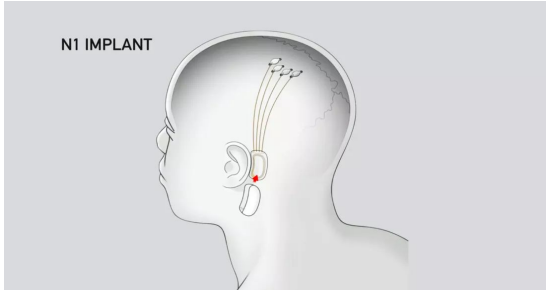


Interfaz de usuario

Adicional: *Neuralink*

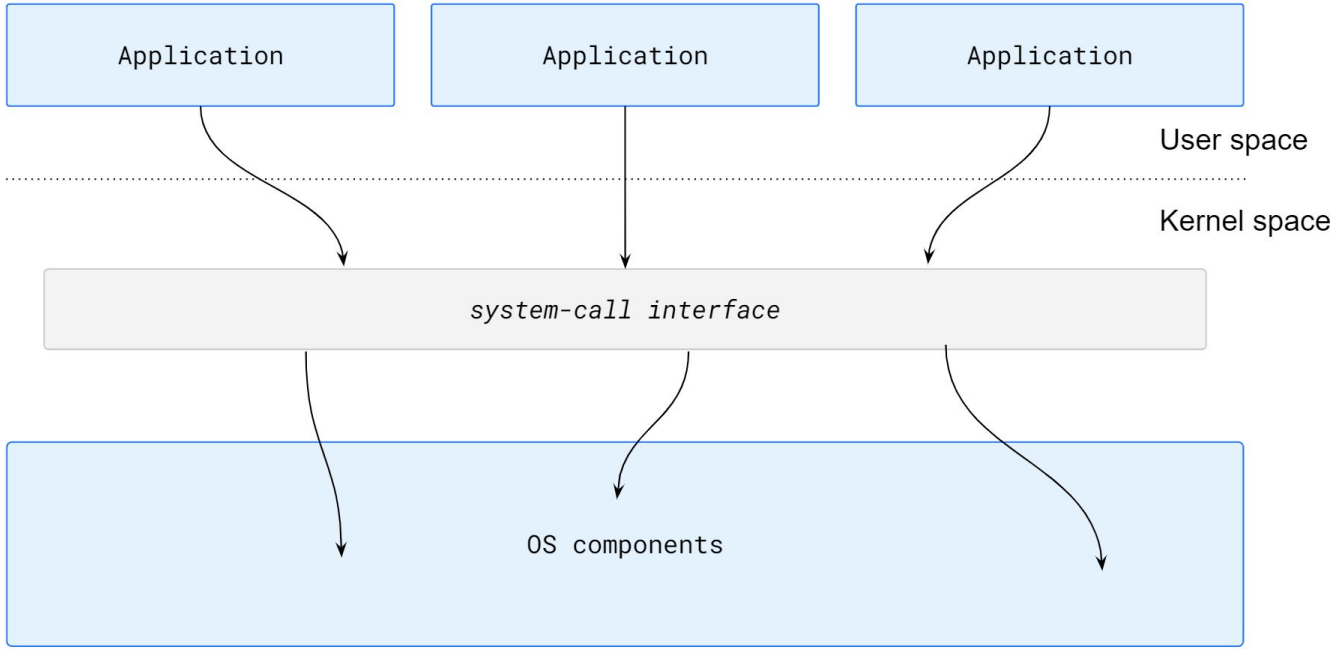
- *Neuralink* es una empresa norteamericana de neurotecnología fundada por Elon Musk y ocho socios, para el desarrollo de **interfaces cerebro-computadora implantables**.
- Musk expresó que el objetivo de la empresa a corto-plazo es hacer dispositivos para tratar enfermedades serias del cerebro, con el objetivo eventual del **perfeccionamiento humano**.
- En julio de 2019, *Neuralink* realizó una presentación en vivo en la Academia de Ciencias de California.
    - La tecnología futura propuesta implica un módulo colocado fuera de la cabeza que recibe de forma inalámbrica información de hilos de electrodos delgados y flexibles **incrustados en el cerebro**.
- Fuente: https://en.wikipedia.org/wiki/Neuralink





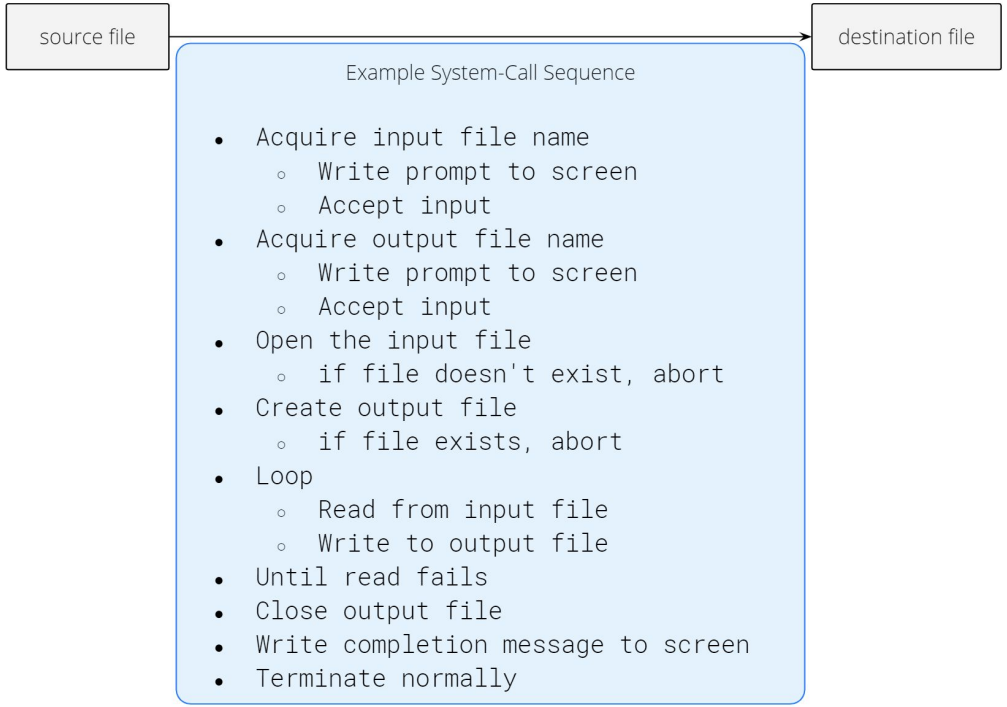
System Calls

- Las llamadas al sistema proporcionan una interfaz a los servicios de un SO.



System Calls

- ¿Cómo se utilizan las llamadas al sistema?: Ej.: Un programa que lee datos de un archivo y los copia en otro: `cp in.txt out.txt`



System Calls

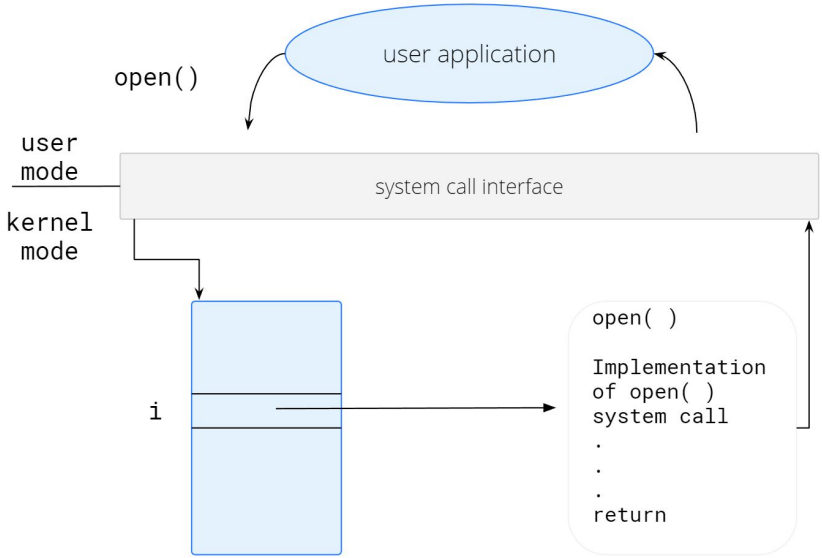
Application Programming Interface (API)

- El desarrollo de aplicaciones es a través de una *API*.
- La *API* especifica un conjunto de funciones que están disponibles para un programador, incluídos los parámetros que se pasan a cada función y los valores de retorno esperados.
- *APIs* más comunes:
  - *API* Windows para sistemas Windows.
  - *API* POSIX para sistemas basados en POSIX (UNIX, Linux y macOS).
  - *API* Java para programas que se ejecutan en la máquina virtual de Java.
- El acceso a una *API* es a través de una librería provista por el SO. Ej: `libc` (o `glibc`) para C en *Unix-like*.
- Las funciones de una *API* normalmente invocan las *system calls* en nombre del programador.
  - Ejemplo: `API Windows → CreateProcess() → NTCreateProcess()`
- Ventajas:
  - Portabilidad.
  - Las *system calls* suelen ser más detalladas y difíciles de manejar que las que provee una *API*.

System Calls

run-time environment (RTE)

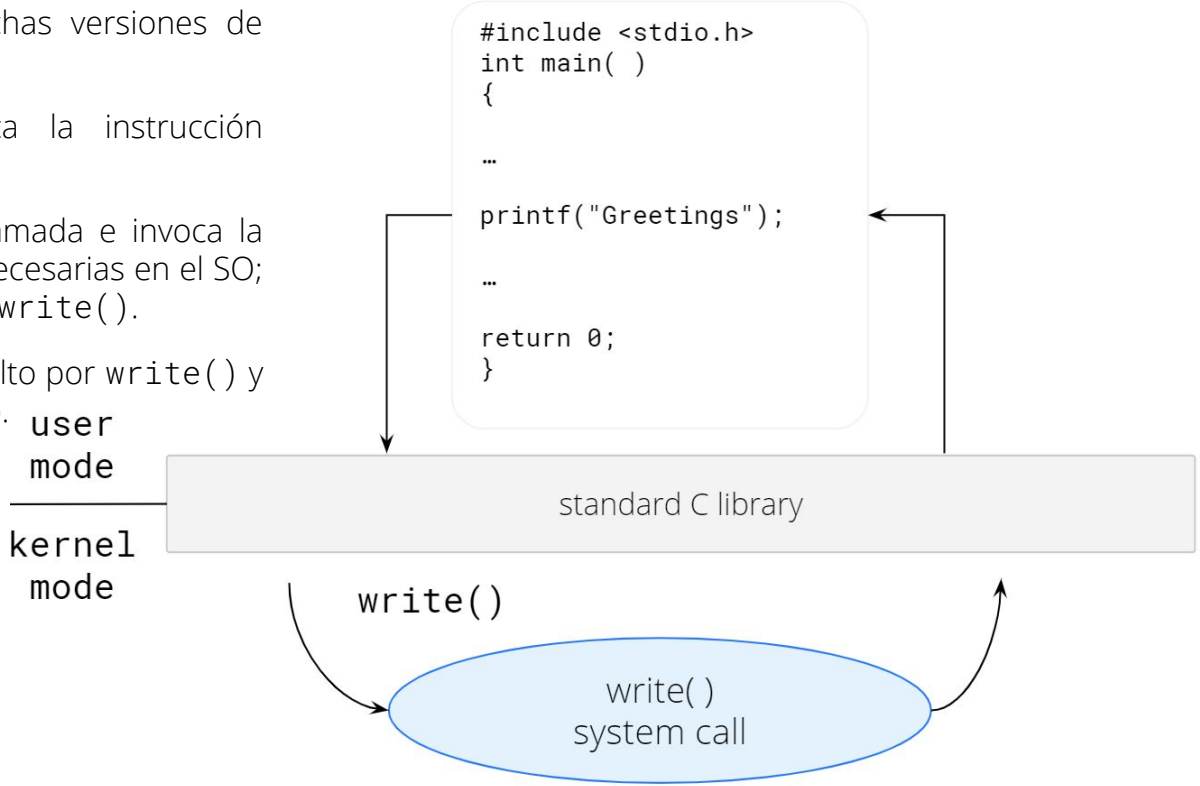
- Conjunto de *software* necesario para ejecutar aplicaciones escritas en un lenguaje de programación determinado, incluídos sus compiladores o intérpretes, así como otro *software*, como bibliotecas (*libraries*) y cargadores (*loaders*).
- Proporciona una *system-call interface* que sirve como enlace a las *system-call* del SO.
- Intercepta las llamadas de función en la *API* e invoca a las *system-calls* necesarias dentro del SO.
- Por lo tanto, la mayoría de los detalles de la interfaz del SO están ocultos al programador por la *API* y son gestionados por el *RTE*.
- Métodos usados para pasar parámetros al SO:
  - Registros.
  - Memoria.
  - Pila.



System Calls

La librería estándar C

- La librería estándar C provee una parte de la interfaz de llamada al sistema para muchas versiones de UNIX y Linux.
- Ej.: un programa en C invoca la instrucción `printf()`.
- La biblioteca C intercepta esta llamada e invoca la llamada (o llamadas) del sistema necesarias en el SO; en este caso, la llamada al sistema `write()`.
- La biblioteca C toma el valor devuelto por `write()` y lo devuelve al programa de usuario.



System Calls

Tipos

- *Process control*:
  - `create process, terminate process.`
  - `load, execute.`
  - `get process attributes, set process attributes.`
  - `wait event, signal event.`
  - `allocate and free memory.`
- *File management*:
  - `create file, delete file.`
  - `open, close.`
  - `read, write, reposition.`
  - `get file attributes, set file attributes.`
- *Device management*:
  - `request device, release device.`
  - `read, write, reposition.`
  - `get device attributes, set device attributes.`
  - `logically attach or detach devices.`

System Calls

Tipos

- *Information maintenance*:
  - `get time or date, set time or date.`
  - `get system data, set system data.`
  - `get process, file, or device attributes.`
  - `set process, file, or device attributes.`
- *Communications*:
  - `create, delete communication connection.`
  - `send, receive messages.`
  - `transfer status information.`
  - `attach or detach remote devices.`
- *Protection*:
  - `get file permissions.`
  - `set file permissions.`

System Calls

Ejemplos de llamadas al sistema de Windows y linux

	Windows	Unix-like

Process control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File management	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Device management	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

Information maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

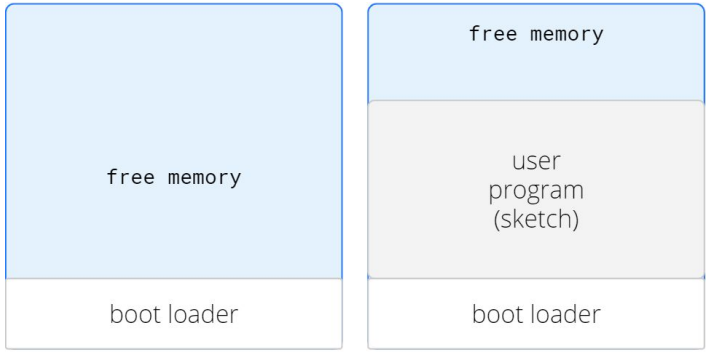
Communications	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shm_open()`
		`mmap()`

Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`

System Calls

Control de procesos - Ejecución Arduino

- El Arduino es una plataforma de *HW* simple, con un microcontrolador y sensores de entrada que responden a una variedad de eventos, Ej.: cambios en la luz, temperatura y presión barométrica.
- Para escribir un programa para Arduino, primero se escribe en una PC y luego se carga el programa compilado (*sketch*) desde la PC a la memoria *flash* del Arduino a través del *USB*.

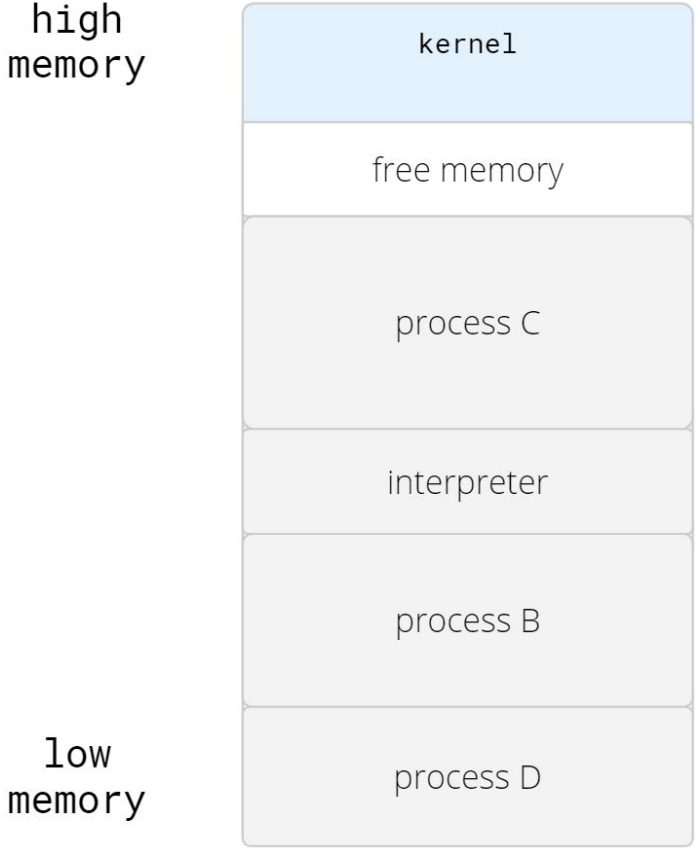


- La plataforma estándar Arduino no proporciona un SO, en cambio, un pequeño *software* conocido como *boot loader* carga el *sketch* en una región de memoria del Arduino.
- Una vez ejecutado el *sketch*, queda a la espera de eventos a los que está programado para responder. Ej.: si el sensor de temperatura detecta que la temperatura ha excedido un cierto umbral, el *sketch* puede hacer que el Arduino arranque el motor para un ventilador.
- Un Arduino se considera un sistema *single-tasking*, ya que solo un *sketch* puede estar presente en la memoria a la vez; si se carga otro *sketch*, reemplaza el *sketch* existente.
- El Arduino no proporciona una interfaz de usuario más allá de los sensores de entrada de *HW*.

System Calls

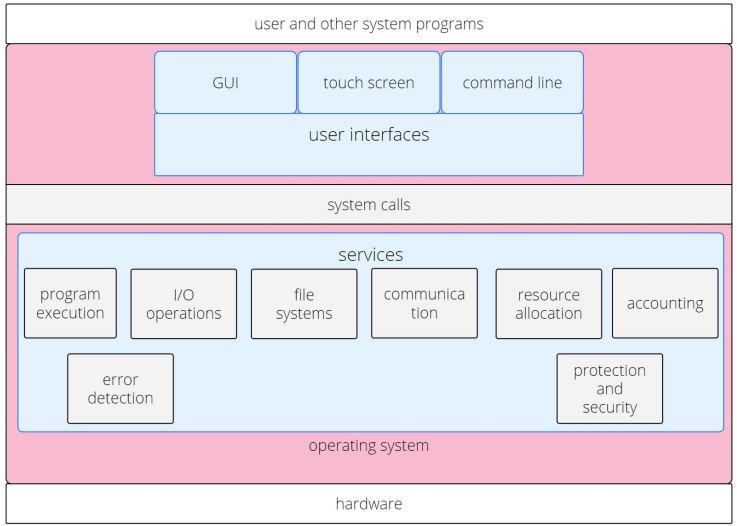
Control de procesos - *FreeBSD*

- FreeBSD es un derivado de BSD, la versión de UNIX® desarrollada en la Universidad de California, Berkeley.
- Multitarea (*multitasking*).
- Inicio de sesión del usuario → invoca la *shell* de elección del usuario.
- La *Shell* ejecuta la llamada al sistema `fork()` para crear el proceso:
  - Ejecuta `exec()` para cargar el programa en el proceso.
  - *Shell* espera que el proceso finalice o continúa con los comandos del usuario.
- El proceso sale con:
  - `código = 0, sin error`
  - `código > 0, código de error`



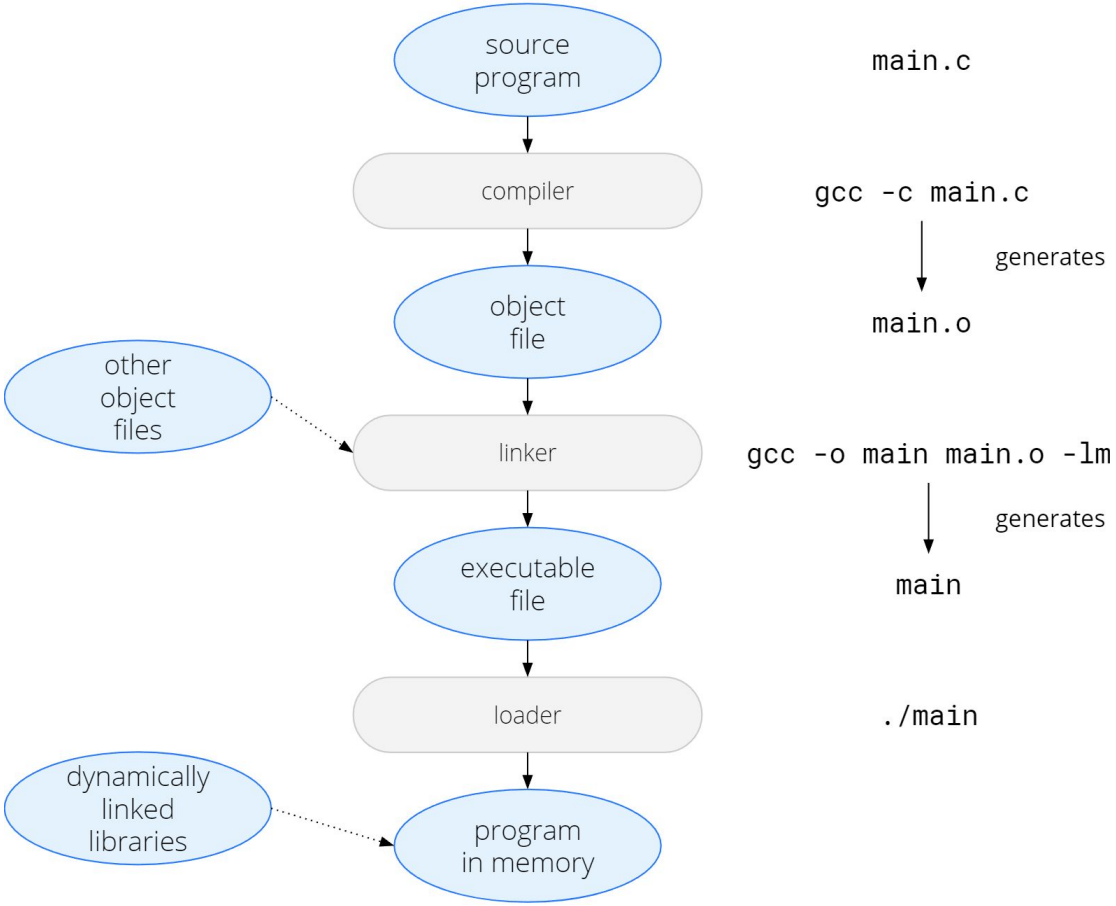
Servicios del sistema

- También conocidos como *system utilities*, proporcionan un entorno conveniente para el desarrollo y ejecución del programa.
- Algunos son simplemente interfaces de *system calls* y otros son más complejos.
- Categorías:
  - Gestión de archivos.
  - Información de estado.
  - Modificación de archivos.
  - Soporte de lenguajes de programación.
  - Carga y ejecución de programas.
  - Comunicaciones.
  - Servicios de segundo plano (*daemos*).



- Programas de aplicación: navegadores *web*, procesadores de texto, hojas de cálculo, sistemas de bases de datos, compiladores, paquetes de análisis estadístico y juegos.
- La vista del SO para la mayoría de los usuarios está definida por las aplicaciones y los programas del sistema, y no por las *system calls*.

Linkers and Loaders



¿Aplicaciones específicas para un determinado SO?

- Por lo general, las aplicaciones compiladas en un SO no son ejecutables en otros SOs.
- Un problema (entre muchos) es que cada SO proporciona un conjunto único de llamadas al sistema.
- Sin embargo, una aplicación puede ejecutarse en múltiples SO de alguna de las siguientes maneras:
  - Aplicación escrita en un lenguaje interpretado (como *Python* o *Ruby*), con intérprete disponible para varios SO. Desventajas: *performance*, intérprete limitado.
  - Aplicación escrita en un lenguaje que incluya una máquina virtual que contenga la aplicación en ejecución. La máquina virtual es parte del *RTE* completo del lenguaje (Ej.: *JAVA*). *RTE* disponible para varios SO. Desventajas: idem lenguaje interpretado.
  - Desarrollado con lenguaje estándar o *API* en el que el compilador genera archivos binarios en un lenguaje específico de la máquina y el SO. Desventajas: portación a cada SO (compilar, testear).
- Otros desafíos en aplicaciones multiplataforma: *APIs*, formatos binarios, instrucciones *CPUs*.

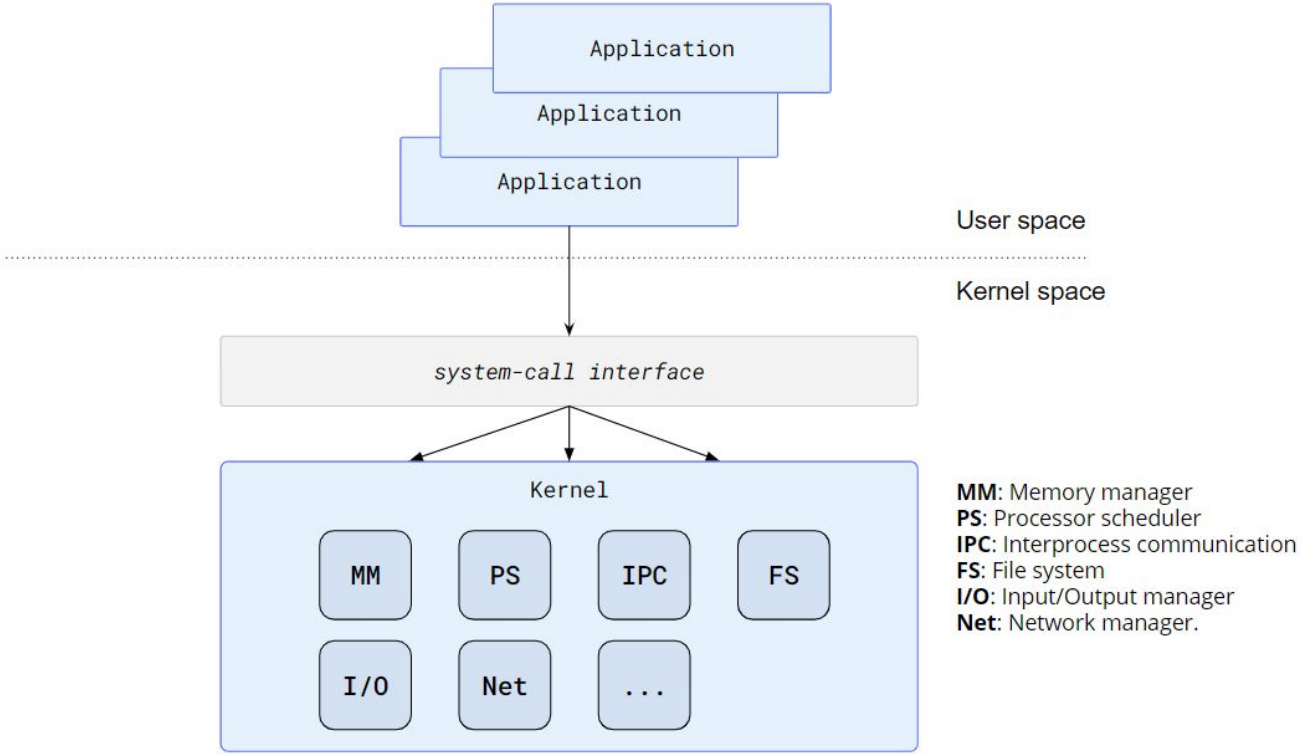
Diseño e implementación de un SO

- Metas de diseño
- Mecanismos y Políticas
- Implementación

Estructura del sistema operativo

Monolítica

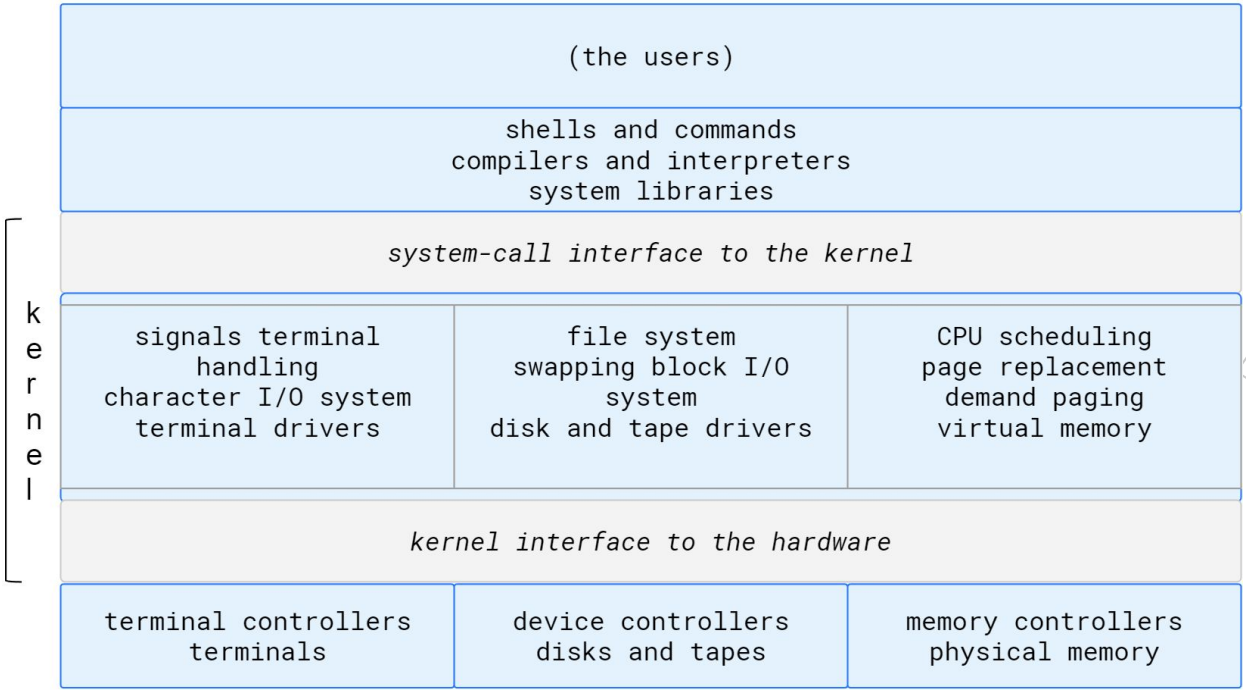
- Coloca toda la funcionalidad del *kernel* en un único archivo binario estático que se ejecuta en un solo espacio de direcciones.



Estructura del sistema operativo

Monolítica

- Unix

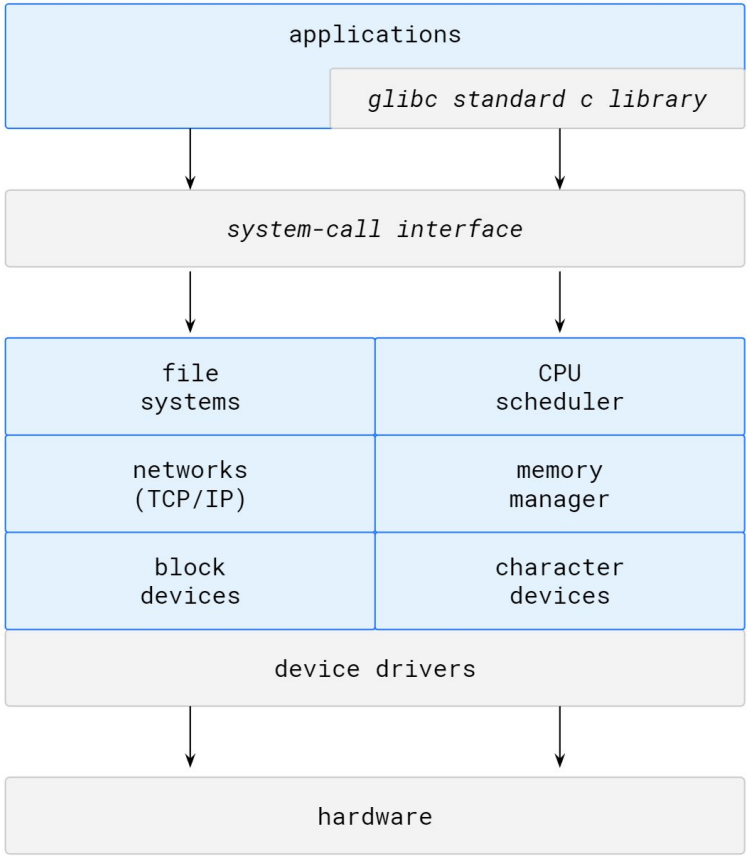


Tradicional UNIX OS

- Se puede ver como un sistema tipo en capas.
- *Kernel*: todo lo que está por debajo de la *system-call interface* y encima del HW.

Estructura del sistema operativo

Monolítica - Linux



- Linux está basado en Unix.
- *Kernel* monolítico: se ejecuta completamente en modo *kernel* en un solo espacio de direcciones.
- Tiene un diseño modular que permite modificar el *kernel* durante la ejecución.

Estructura del sistema operativo

Monolítica

Desventajas:

- Difícil de implementar.
- Difícil de extender.

Ventajas:

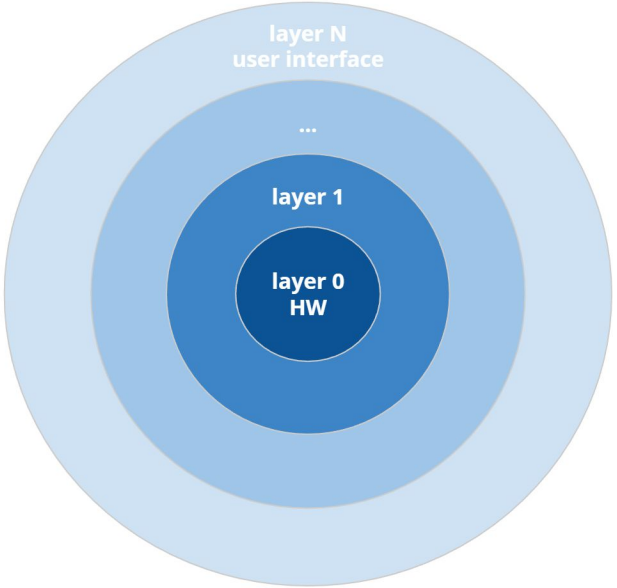
- Sencillo.
- Buen rendimiento por la baja sobrecarga en la interfaz del *system call.*
- La comunicación dentro del *kernel* es rápida.

- El enfoque monolítico se conoce a menudo como un sistema fuertemente acoplado (*tightly coupled*) porque los cambios en una parte del sistema pueden afectar a otras partes.
- Alternativamente, se puede diseñar un sistema debilmente acoplado (*loosely coupled*), que se divide en componentes separados más pequeños que tienen una funcionalidad específica y limitada. Todos los componentes juntos forman el *kernel*.
- La ventaja del enfoque modular es que los cambios en un componente afectan sólo a ese componente y no a otros, permitiendo crear y cambiar el funcionamiento interno del sistema.
- Un sistema puede ser modular de varias maneras; un método es el enfoque en capas.

Estructura del sistema operativo

Enfoque en capas

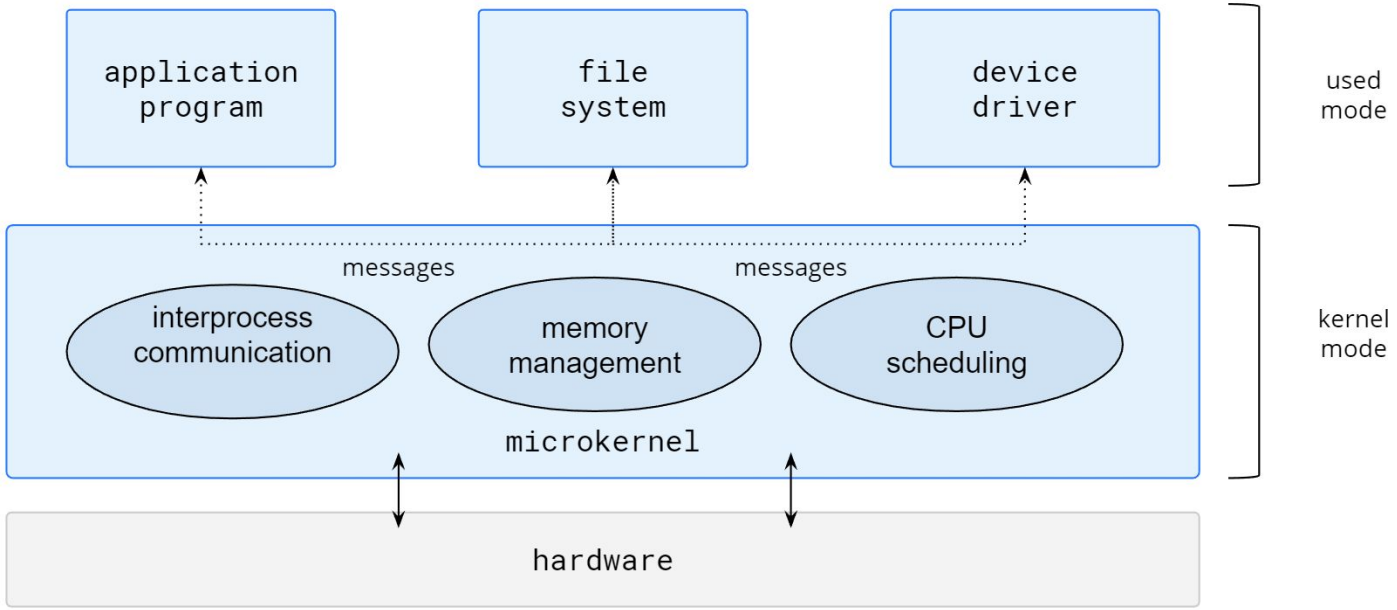
- El SO se divide en varias capas (niveles). La capa inferior (capa 0) es el HW; la más alta (capa N) es la interfaz de usuario.
- La capa M consiste en estructuras de datos y un conjunto de funciones que pueden ser invocadas por capas de nivel superior. La capa M, a su vez, puede invocar operaciones en capas de nivel inferior.
- La principal ventaja del enfoque en capas es la simplicidad de construcción y depuración.
- Pocos SOs utilizan un enfoque de capas puras.
  - Es difícil definir adecuadamente la funcionalidad de cada capa.
  - El rendimiento general es deficiente debido a la sobrecarga al requerir que un programa de usuario atraviese varias capas para obtener un servicio del SO.
- En los SO actuales solo se usan pocas capas con mucha funcionalidad, obteniendo ventajas de modularización y evitando problemas de definición e integración.



Estructura del sistema operativo

Microkernels

- Elimina todos los componentes no esenciales del *kernel*, implementándolos como programas a nivel de usuario que residen en espacios de direcciones separadas, resultando es un *kernel* más pequeño.
- No hay consenso sobre qué servicios deben permanecer en el *kernel* y cuáles deben implementarse en el espacio del usuario.



Estructura del sistema operativo

Microkernels

- La función principal del *microkernel* es proporcionar comunicación entre el programa cliente y los diversos servicios que también se ejecutan en el espacio del usuario.
- Por ejemplo, si un programa cliente quiere acceder a un archivo, debe interactuar con el servidor de archivos. El programa cliente y el servicio nunca interactúan directamente, sino que se comunican indirectamente intercambiando mensajes con el *microkernel*.
- Ventajas:
  - Más fácil de extender → nuevos servicios agregados en espacio de usuario.
  - Más fácil de portar el SO a nuevas arquitecturas.
  - Más confiable (se ejecuta menos código en modo *kernel*).
  - Más seguro.
- Desventajas:
  - Sobrecarga de comunicación entre *user mode* y *kernel mode*.

Estructura del sistema operativo

Módulos

- La mejor metodología actual para el diseño de SOs involucra el uso de *loadable kernel modules* (*LKM* - módulos de *kernel* cargables).
- El *kernel* tiene un conjunto de componentes centrales y puede enlazar servicios adicionales a través de módulos, ya sea en el arranque o durante la ejecución (dinámicamente, sin necesidad de recompilar el *kernel*).
- Este diseño es común en implementaciones modernas de UNIX, como Linux, macOS y Solaris e incluso en Windows.
- Se asemeja a un sistema en capas en que cada sección del *kernel* tiene interfaces definidas y protegidas, pero es más flexible que un sistema en capas ya que cualquier módulo puede llamar a cualquier otro módulo.
- Se asemeja al enfoque de *microkernel* en el sentido de que el módulo primario solo tiene funciones básicas y sabe cómo cargar y comunicarse con otros módulos, pero es más eficiente porque los módulos no necesitan invocar el paso de mensajes para comunicarse.
- Linux utiliza *loadable kernel modules* principalmente para soportar controladores de dispositivos y sistemas de archivos.

Estructura del sistema operativo

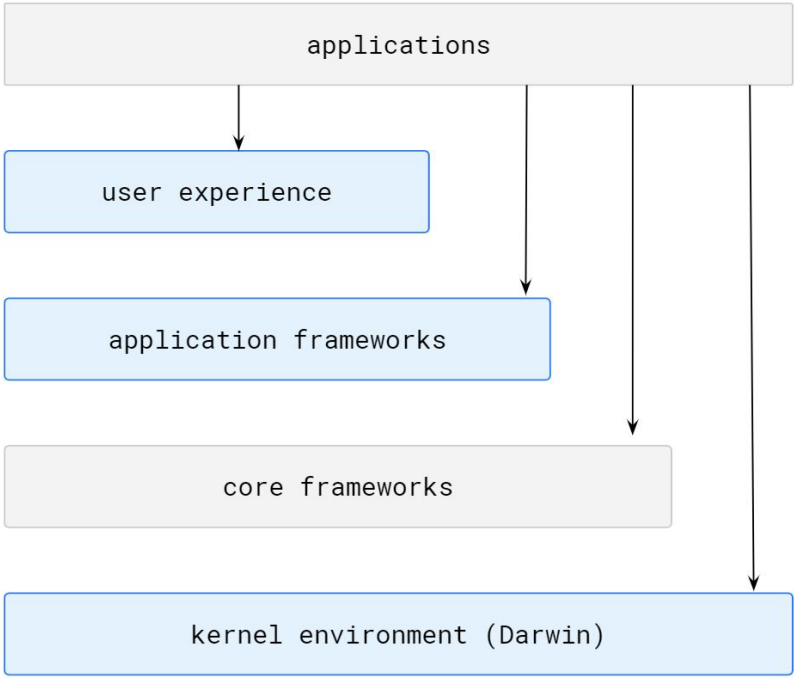
Sistemas híbridos

- En la práctica, muy pocos SOs adoptan una estructura única y estrictamente definida. En su lugar, combinan diferentes estructuras, lo que da como resultado sistemas híbridos que abordan los problemas de rendimiento, seguridad y uso.
  - Linux:
    - Es monolítico, porque al tener el SO en un solo espacio de direcciones proporciona un rendimiento muy eficiente.
    - También es modular, ya que una nueva funcionalidad se puede agregar dinámicamente al *kernel*.
  - Windows:
    - En gran parte monolítico, también por motivos de rendimiento.
    - Tiene aspectos de sistemas *microkernel*, ya que se encuentra incluído el soporte para subsistemas separados (conocidos como personalidades del SO) que se ejecutan como procesos en modo de usuario.
    - Windows también proporciona soporte *LKM*.

Estructura del sistema operativo

Sistemas híbridos - macOS e iOS

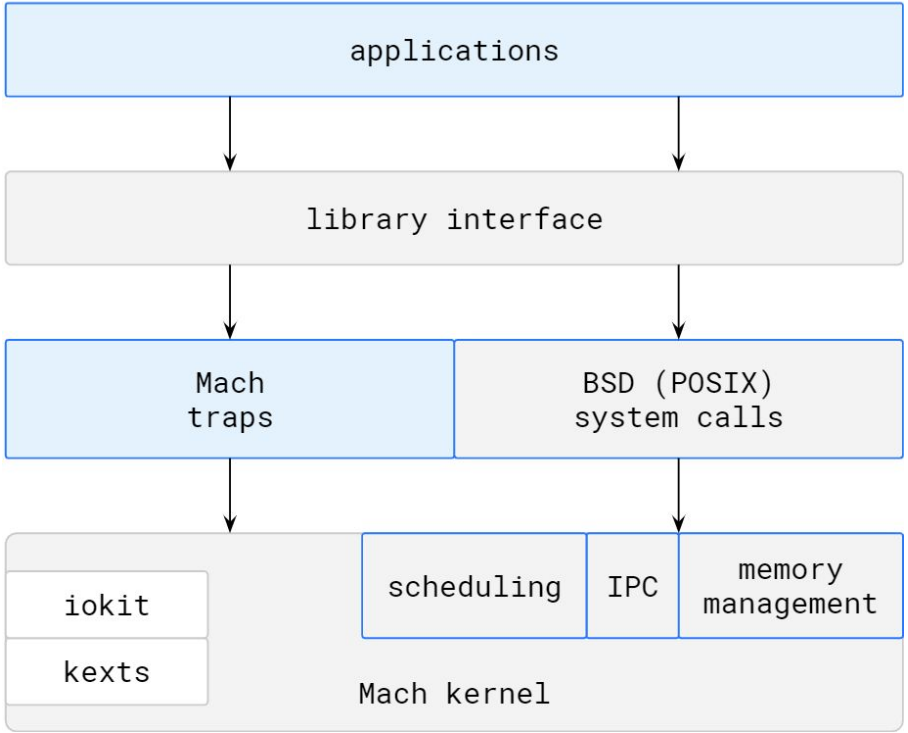
- macOS e iOS tienen un desarrollo común, con aspectos diferentes para los distintos dispositivos que los soportan.
- ***User experience***: Interfaz de SW, Interacción usuarios dispositivos. *Aqua* e *Springboard*.
- ***Application frameworks***: *Cocoa* y *Cocoa Touch*; *Api* para *Objective-C* y *Swift*.
- ***Core frameworks***: Soporte gráfico (*OpenGL*) y multimedia.
- ***Kernel* environment**: Darwin; incluye *Mach microkernel* y el *kernel BSD* UNIX.
- Una aplicación puede comunicarse directamente con cualquier capa.
- maOS → Arquitectura Intel.
- iOS → Arquitectura ARM.



Estructura del sistema operativo

Sistemas híbridos - Darwin

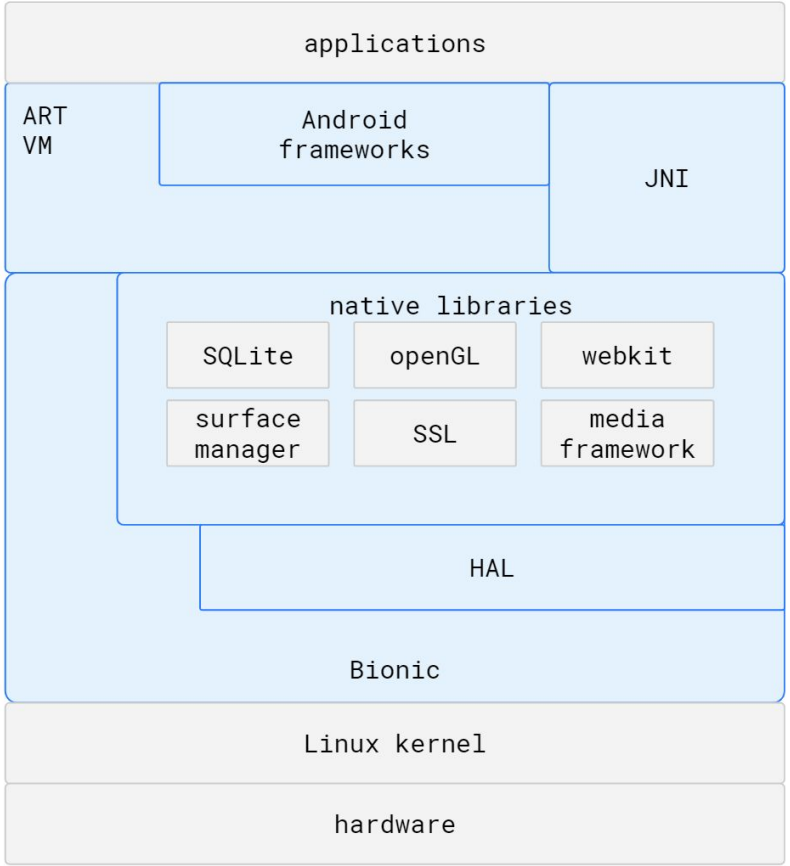
- Estructura híbrida y en capas, con el *microkernel Mach* y el *kernel BDS UNIX.*
- Provee dos interfaces de llamadas al sistema:
  - *Mach system calls: traps.*
  - *BSD system calls: POSIX.*
- *Mach* provee los principales servicios del SO: memoria/*CPU*/*IPC* y *RPCs*.
- *I/O kit*: desarrollo de *device drivers.*
- *kexts*: *kernel extensions*, módulos cargables dinámicamente.
- Combina a *Mach, BSD, I/O kit* y *kext* en un solo espacio de direcciones; no es un *microkernel* puro, ya que estos subsistemas no corren en espacio de usuario.



Estructura del sistema operativo

Sistemas híbridos - Android

- Android se ejecuta en una variedad de plataformas móviles y es de código abierto, lo que explica en parte su rápido incremento en popularidad.
- Es una pila de *SW* en capas que proporciona un conjunto rico de *frameworks* para gráficos, audio y funciones de *HW*.
- El desarrollo de *app* es en Java, sin usarse la *API* Java estándar sino una provista por **Google**.
- Las *app* Java se compilan para ejecutar en *Android RunTime ART*, una máquina virtual diseñada para Android y optimizada para dispositivos móviles.
- Se compila a Java *bytecode .class* → *ejecutable .dex*.



Estructura del sistema operativo

Sistemas híbridos - Android

- *ahead-of-time (AOT)*: los archivos .dex se compilan en código máquina nativo cuando se instalan en un dispositivo. → Ejecución más eficiente y menor consumo de energía, características que son cruciales para los sistemas móviles.
- Android incluye *frameworks* de desarrollo de navegadores *web* (*webkit*), soporte de bases de datos (*SQLite*) y soporte de red, como *sockets seguros* (*SSL*).
- Google abstrae el *hardware* físico a través de la capa de abstracción de *HW*, o *HAL*.
    - Al abstraer todo el *HW*, como la cámara, el chip GPS y otros sensores, el *HAL* proporciona a las aplicaciones una interfaz coherente, independientemente del *HW* específico.
- *Bionic*: biblioteca estándar C para Android (en Linux es *glibc*). No solo está optimizada para dispositivos móviles si no que *Bionic* permite a Google omitir la licencia GPL de *glibc*.
- En la parte inferior de la pila de *SW* de Android se encuentra el *kernel* de Linux modificado para las necesidades de los dispositivos móviles.

Debugging del sistema operativo

- El *debugging* es la actividad de encontrar y corregir errores en un sistema, tanto en *HW* como en *SW*.
- Los problemas de rendimiento (*performance*) se consideran *bugs*, por lo que el *debugging* también puede incluir ajustes que buscan mejorar el rendimiento eliminando cuellos de botella (*bottlenecks*) de procesamiento.

Análisis de fallas

- Si un proceso falla, la mayoría de los SO escriben la información del error en un archivo de registro (*log file*) para informar a los administradores o usuarios que ocurrió un problema.
- El SO también puede realizar un *core dump* (una captura de memoria del proceso) y almacenarlo en un archivo para su posterior análisis con un *debugger*.
- Una falla en el *kernel* se denomina *crash* y cuando ocurre, la información de error se guarda en un archivo y el estado de la memoria se guarda en un *crash dump*.
- El *debugging* del SO y el *debugging* de procesos utilizan diferentes herramientas y técnicas debido a la diferente naturaleza de estas dos tareas.

Debugging del sistema operativo

Monitoreo y ajuste de rendimiento

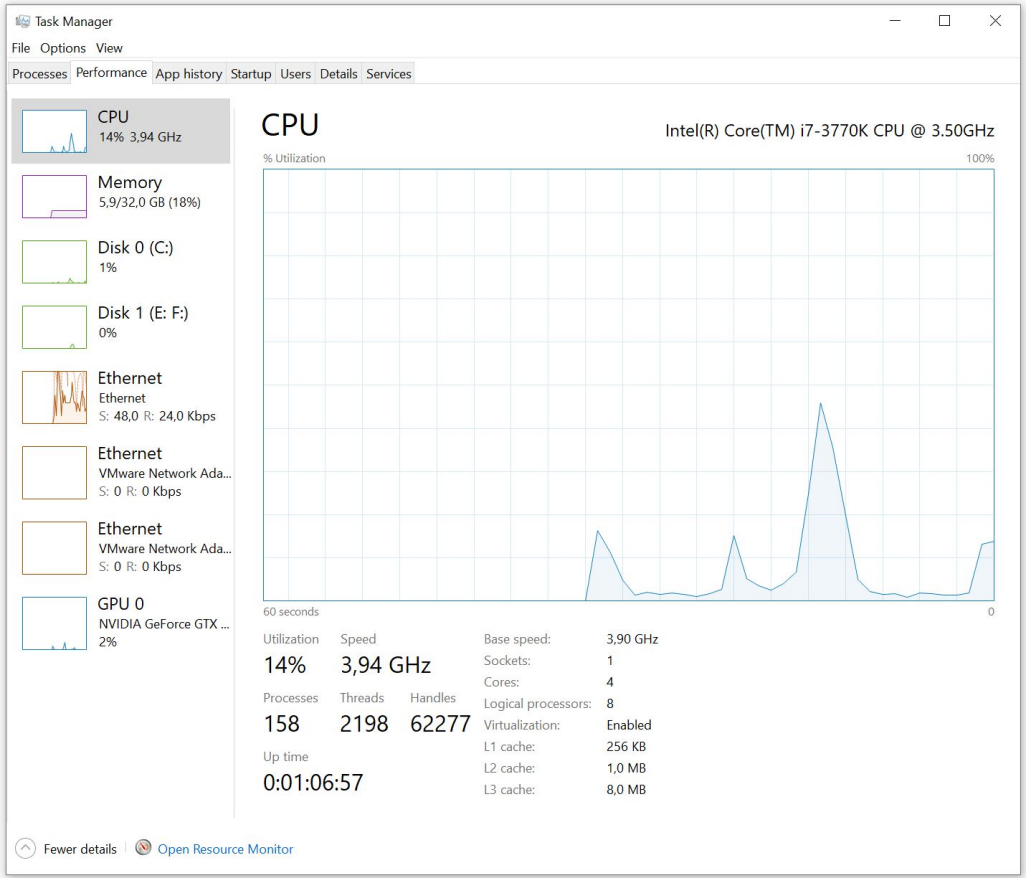
- El SO debe tener alguna forma para calcular y mostrar medidas de comportamiento del sistema.

Counters

- Los SO realizan un seguimiento de la actividad a través de una serie de contadores, Ej.: Linux:
  - *Per-Process*
    - `ps`: información de un proceso o conjunto de procesos.
    - `top`: estadísticas en tiempo real de los procesos actuales.
  - System-Wide
    - `vmstat`: uso de memoria.
    - `netstat`: interfaces de red.
    - `iostat`: uso de E/S en discos.
- La mayoría de las herramientas en Linux leen info del sistema de archivos /proc. Es un "pseudo" *file system* que existe solo en la memoria del *kernel* y se utiliza principalmente para consultar diversas estadísticas del *kernel* y procesos.

Debugging del sistema operativo

Monitoreo y ajuste de rendimiento



Windows Task Manager

Debugging del sistema operativo

Monitoreo y ajuste de rendimiento

Resource Monitor

File Monitor Help

Overview CPU Memory Disk Network

CPU — 2% CPU Usage — 51% Maximum Frequency

Image	PID	Description	Status	Threads	CPU	Average CPU
SearchUI.exe	6988	Search and Cortana application	Suspended	35	0	0.00
MicrosoftEdge.exe	7388	Microsoft Edge	Suspended	36	0	0.00
MicrosoftEdgeCP.exe	8072	Microsoft Edge Content Process	Suspended	15	0	0.00
MicrosoftEdgeSH.exe	7236	Microsoft Edge Web Platform	Suspended	9	0	0.00
YourPhone.exe	3512	YourPhone.exe	Suspended	13	0	0.00
ShellExperienceHost.exe	9292	Windows Shell Experience Host	Suspended	24	0	0.00
perfmon.exe	10816	Resource and Performance Monitor	Running	20	1	2.95
explorer.exe	4928	Windows Explorer	Running	65	0	1.14
dwm.exe	1048	Desktop Window Manager	Running	22	1	0.63

Disk — 192 KB/sec Disk I/O — 2% Highest Active Time

Image	PID	File	Read (B/sec)	Write (B/sec)	Total (B/sec)	I/O Priority	Response Time (...
explorer.exe	4928	C:\Users\hacker\Downloads\wallpapers\4k\64.j...	780.434	0	780.434	Normal	2
explorer.exe	4928	C:\Users\hacker\Downloads\wallpapers\4k\64.j...	840.468	0	840.468	Normal	2
System	4	C:\Users\hacker\Downloads\Los Vengadores 3 ...	18.725	0	18.725	Normal	1
chrome.exe	11196	C:\Users\hacker\AppData\Local\Google\Chrom...	0	290.816	290.816	Normal	1
chrome.exe	11196	C:\Users\hacker\AppData\Local\Google\Chrom...	0	26.438	26.438	Normal	1
Taskmgr.exe	10728	C:\Windows\System32\dui70.dll	1.820	0	1.820	Normal	1
System	4	C:\Users\hacker\Downloads\The Ultimate Deat...	18.793	0	18.793	Normal	1
Taskmgr.exe	10728	C:\Windows\System32\d3d9.dll	1.820	0	1.820	Normal	0
System	4	C:\Users\hacker\Downloads\Jurassic.world.falle...	65.536	0	65.536	Normal	0

Network — 96 Kbps Network I/O — 0% Network Utilization

Memory — 0 Hard Faults/sec — 16% Used Physical Memory

Image	PID	Hard Faults/sec	Commit (KB)	Working Set (KB)	Shareable (KB)	Private (KB)
chrome.exe	12116	0	421.944	445.312	61.424	383.888
chrome.exe	12080	0	291.060	316.208	50.804	265.404
chrome.exe	10988	0	511.848	305.076	55.016	250.060
AcroRd32.exe	10588	0	235.400	262.012	38.648	223.364
chrome.exe	11316	0	239.128	274.288	57.012	217.276
chrome.exe	11196	0	227.260	288.372	76.068	212.304
chrome.exe	12096	0	193.944	219.728	48.124	171.604
chrome.exe	12128	0	196.536	221.968	55.276	166.692
chrome.exe	1648	0	132.744	145.908	30.956	114.952

Views

CPU 100% — 60 Seconds 0%

Disk 1 MB/sec — 0

Network 1 Mbps — 0

Memory 100 Hard Faults/sec — 0

Windows Resource Monitor

Debugging del sistema operativo

Monitoreo y ajuste de rendimiento

Tracing

Las herramientas de *tracing* recopilan datos de un evento específico, como los pasos involucrados en una *system call*.

- *Per-Process*
  - `strace`: rastrea las llamadas al sistema invocadas por un proceso.
  - `gdb`: un depurador de nivel de origen.
- *System-Wide*
  - `perf`: una colección de herramientas de rendimiento de Linux.
  - `tcpdump`: recoge los paquetes de red.

Sistema Distribuidos

Consultar transparencias del Profesor Prof. Dr. Javier Echaiz

https://cs.uns.edu.ar/~gd/soyd/clasesjavi/je.tar.gz