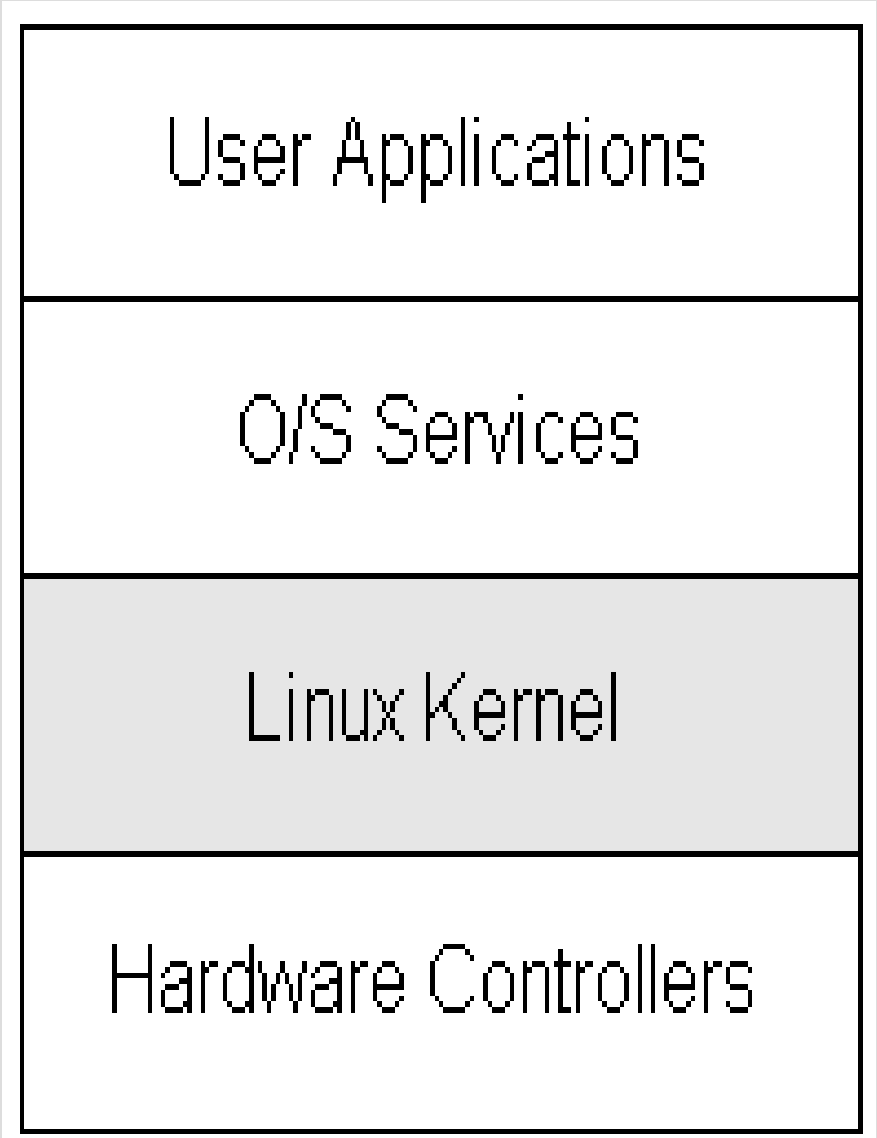
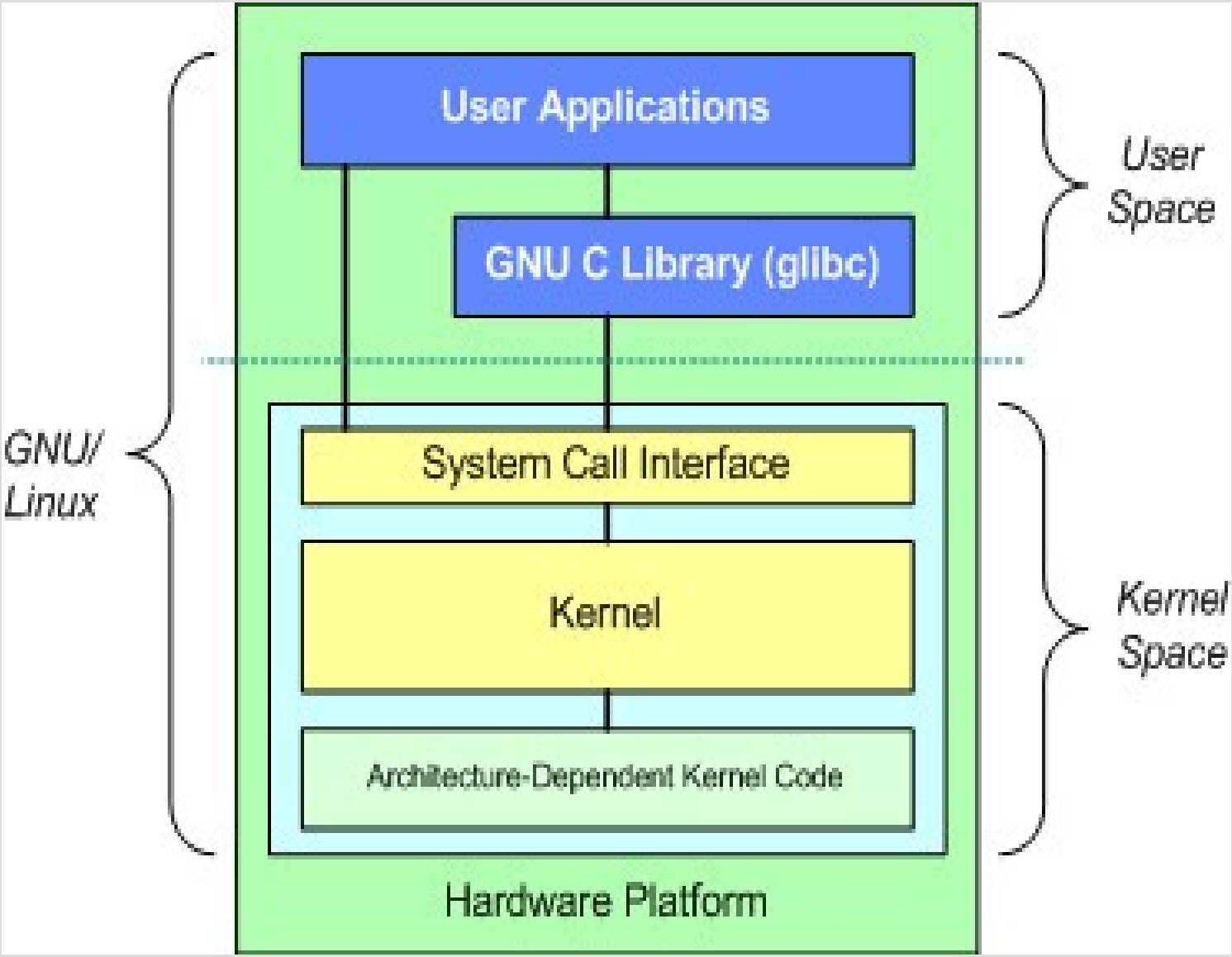
Un sistema GNU/Linux

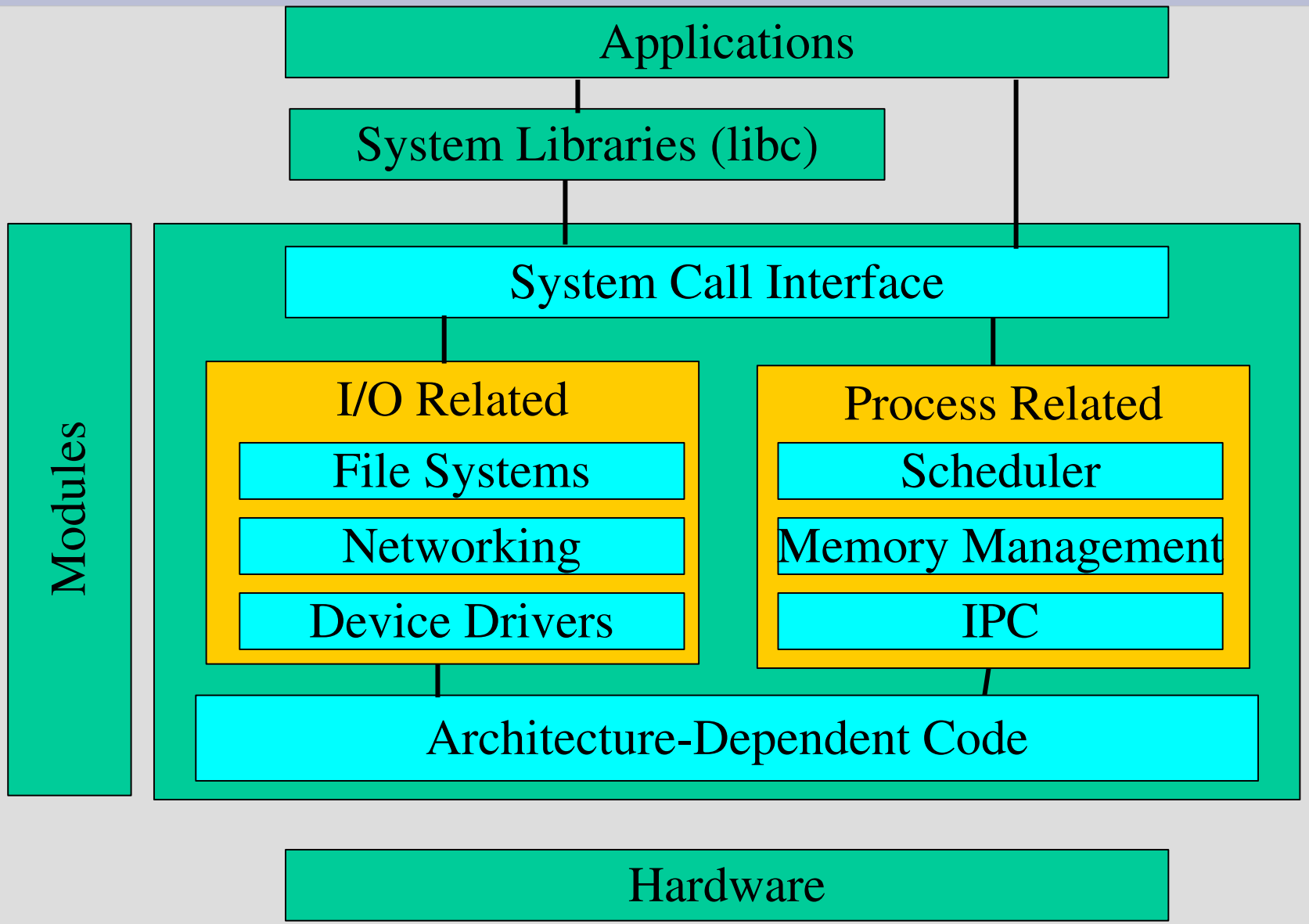


- procesador de texto, web browser
- shell, sistema de ventanas, glibc, etc
- driver de teclado, scheduler, file systems
- controlador de teclado, timer, APIC

nos acercamos un poco...



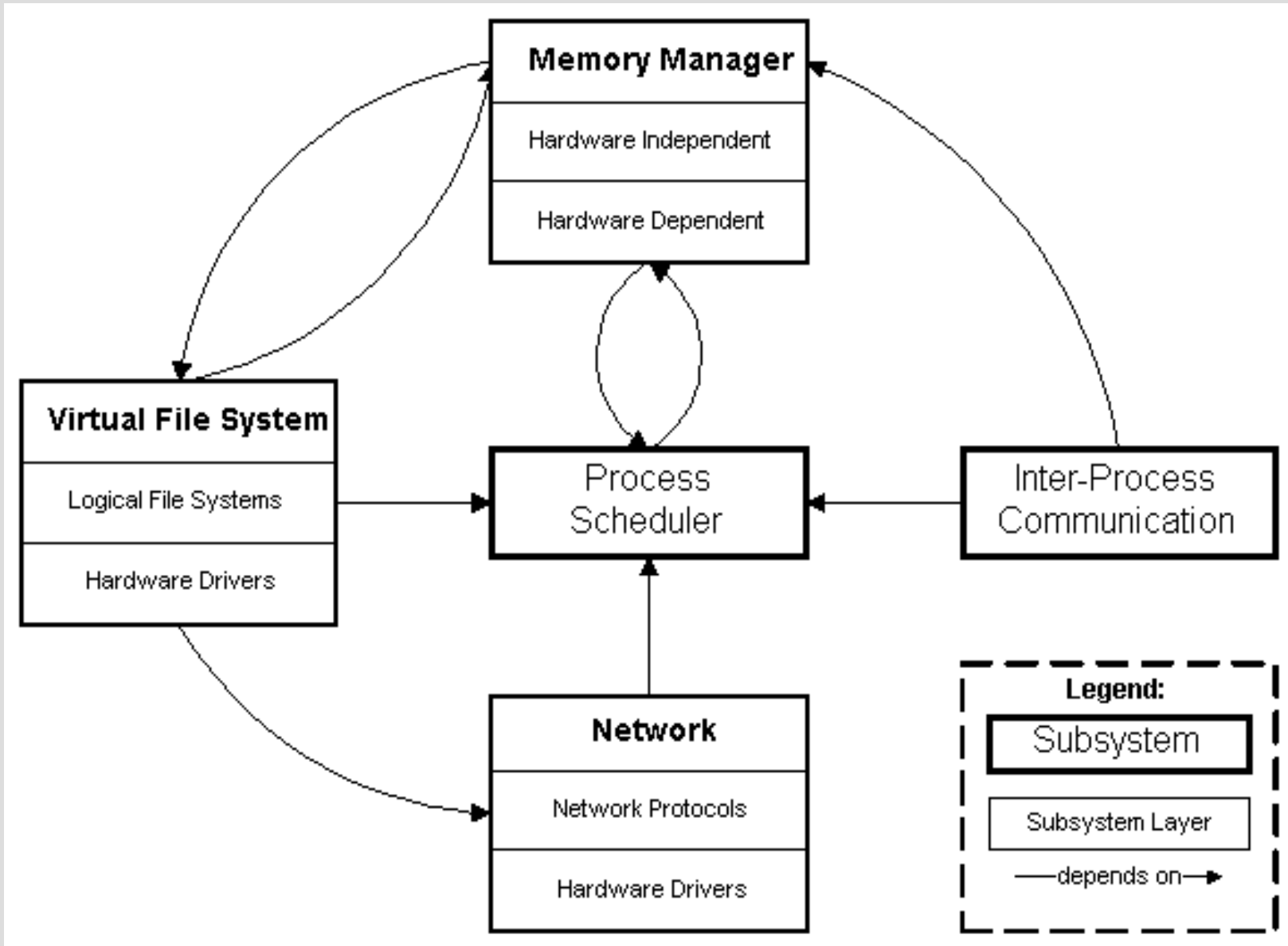
...hasta ver los subsistemas.



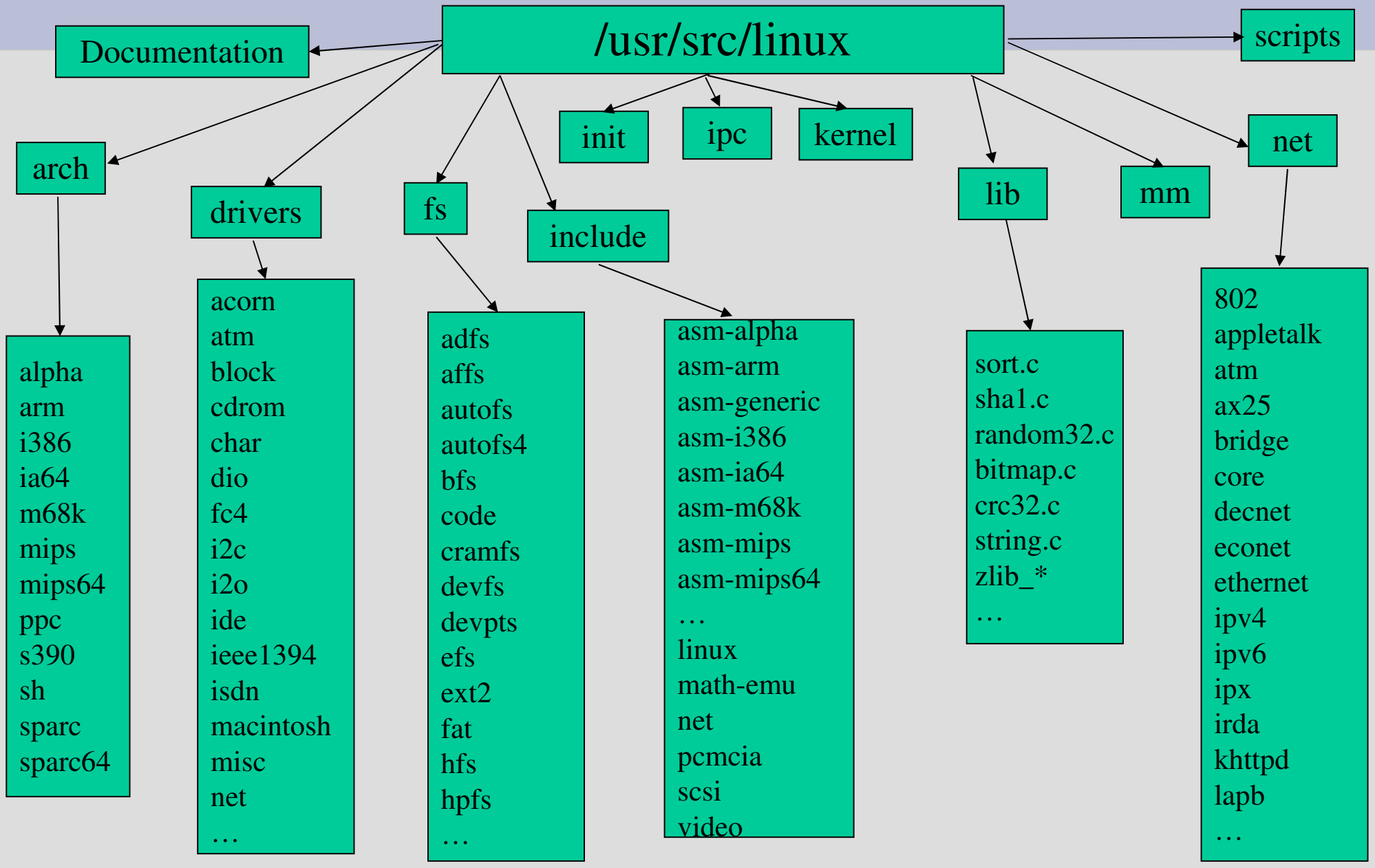
Subsistemas del kernel

- Core
- Memory Manager (MM)
- Virtual File System (VFS)
- Network Interface (NET)
- Inter-Process Communication (IPC)

Arquitectura del Kernel



Esquema del código fuente



Descripción de Subsistemas

- Funcionalidad / Responsabilidad
- Interface externa
- Descripción
- Arquitectura
- Código Fuente

Core / Scheduler Funcionalidad

- Permite crear nuevos threads y procesos
- Planifica la CPU y realiza el switch
- Rutea las señales a los procesos de usuario
- Administra el hardware timer
- Libera los recursos de un proceso cuando éste termina su ejecución
- Proveé soporte para módulos del kernel

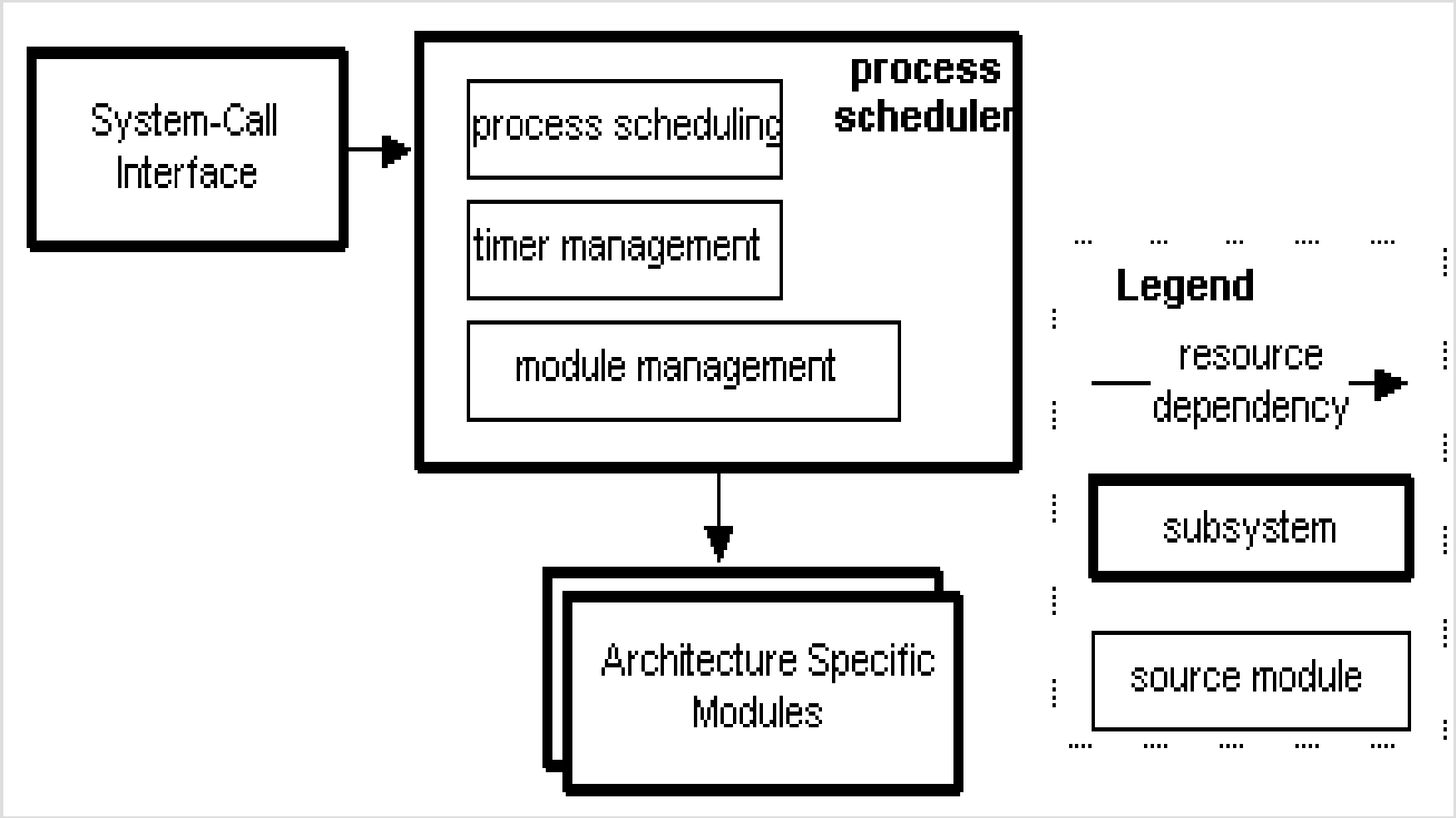
Core / Scheduler
Interface Externa

- **fork(), wait(), exit()**: crear y terminar procesos.
- **setitimer() and getitimer():** manejo de timers.
- **signal():** asocia un handler a una señal.
- **sleep():** pone a dormir al proceso invocante.
- **create_modules, init_modules, delete_module():** crea, inicializa y destruye módulos dinámicos del kernel.

Core / Scheduler
Descripción del Subsistema

- Carga, ejecución y terminación correcta de procesos
- El algoritmo de scheduling es invocado explícita e implícitamente en la ejecución de un proceso
    - Interrupcion del timer, despues de cada system call, etc
    - System calls que invocan directamente al scheduler (sleep())
- Carga y descarga de módulos dinámicos.
- Vemos las señales en la parte de IPC

Core / Scheduler Arquitectura



Core / Scheduler linux/kernel

- El código central del kernel
- sched.c – “el archivo principal del kernel”
  - scheduler, wait queues, timers, alarmas, task queues
- control de procesos
  - fork.c, exec.c, signal.c, exit.c
  - acct.c, capability.c, exec_domain.c
- soporte de módulos del kernel
  - kmod.c, ksyms.c, module.c
- timers, E/S, softirqs, imprimir en pantalla, otras operaciones...
  - time.c, resource.c, dma.c, softirq.c, itimer.c
  - printk.c, info.c, panic.c, sysctl.c, sys.c

Administrador de Memoria Funcionalidad

- Espacio de direccionamiento virtual. Los procesos pueden referenciar más memoria de la que existe físicamente.
- Protección. La memoria de un proceso no debe ser modificada arbitrariamente por otros procesos. Tambien se protegen los espacios de código y sólo lectura.
- Mapeo de Memoria. Los Procesos pueden mapear un archivo en un área de memoria virtual y acceder al archivo como si fuera RAM.
- Memoria compartida. Se permite a los procesos compartir alguna porción de su memoria.

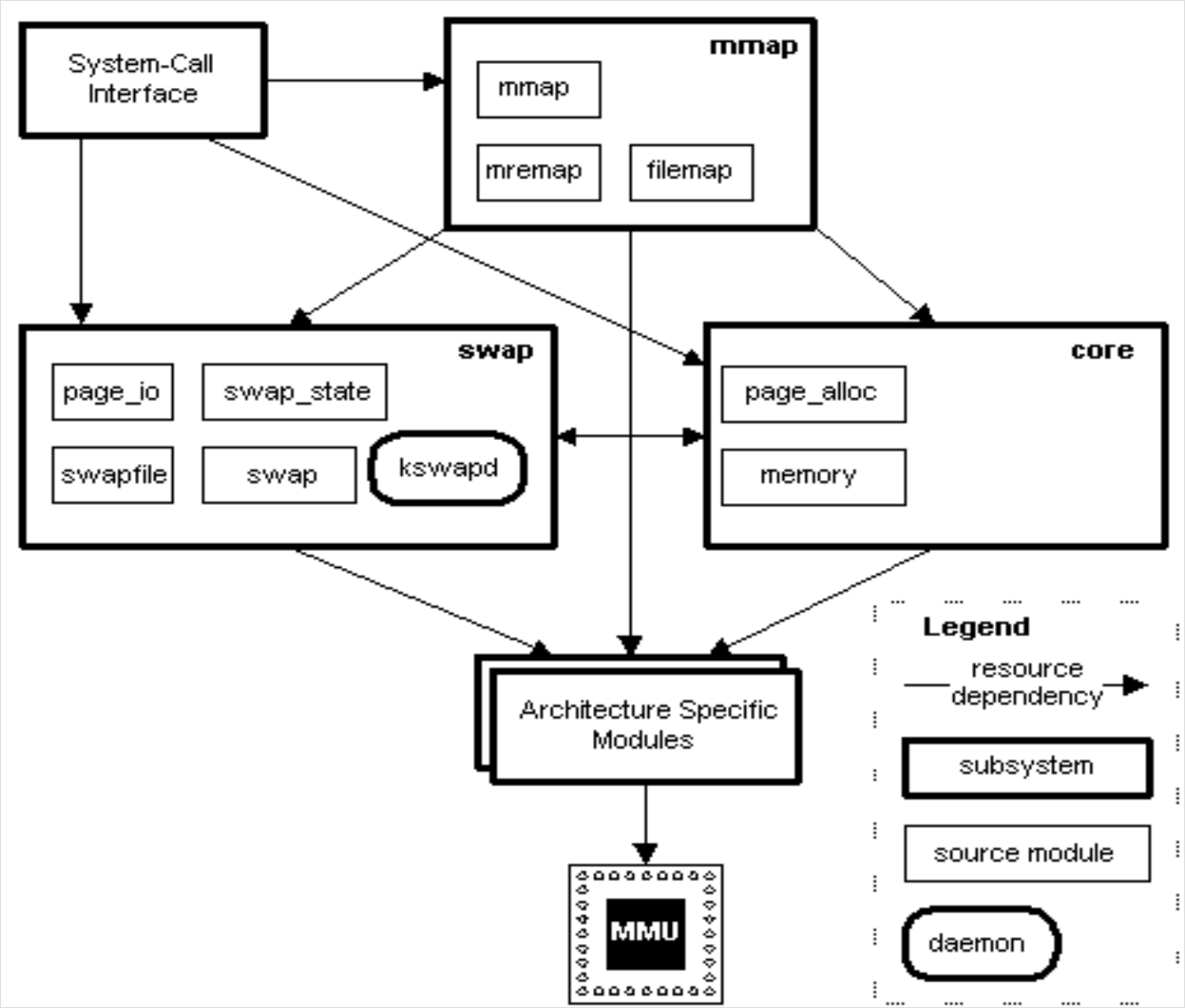
Administrador de Memoria Interface Externa

- **brk()** (malloc() / free())**:** alocar o liberar memoria para un proceso.
- **mmap()** / **munmap()** / **msync()** / **mremap():** mapear archivos en regiones de memoria virtual.
- **mprotect():** modifica la protección de una región de memoria virtual
- **mlock()** / **mlockall()** / **munlock()** / **munlockall()**: Permiten bloquear una pagina en memoria para que no sea swapeada a disco.
- **swapon()** / **swapoff():** agregar o remover espacios de swap.

Administrador de Memoria
Descripción del Subsistema

- Se implementa una capa dependiente de la plataforma que ofrece una interface común.
- El MMU traduce direcciones virtuales a direcciones físicas. Los procesos no conocen direcciones físicas, entonces se pueden mover.
- page faults: por paginas swapeadas, por referencias inválidas o por protección.
- kswapd es un thread del kernel que swapea páginas poco referenciadas.

Administrador de Memoria
Arquitectura



Administrador de Memoria linux/mm

- paging and swapping
  - swap.c, swapfile.c (paging devices), swap_state.c (cache)
  - vmscan.c – paging policies, kwapd
  - page_io.c – low-level page transfer
- allocation and deallocation
  - slab.c – slab allocator
  - page_alloc.c – page-based allocator zone allocator
  - vmalloc.c – kernel virtual-memory allocator
- memory mapping
  - memory.c – paging, fault-handling, page table code
  - filemap.c – file mapping
  - mmap.c, mremap.c, mlock.c, mprotect.c

Sistema de Archivos Virtual Funcionalidad

- Multiples dispositivos de hardware
- Multiples sistemas de archivos logicos
- Multiple executable formats (a.out, ELF, java)
- Homogeneidad: presenta una interface común a todos los archivos y dispositivos de hardware
- Performance, integridad y seguridad

Sistema de Archivos Virtual Interface Externa

Proveé operaciones para el uso de archivos y directorios. Se incluyen las normalmente provistas por sistemas compatibles con POSIX.

- Las operaciones con archivos: open, close, read, write, seek, tell.

- Las operaciones con directorios: readdir, creat, unlink, chmod, stat.

Sistema de Archivos Virtual
Descripción del Subsistema

Linux necesita soportar muchos sistemas de archivos lógicos y muchos dispositivos de hardware. Para esto utiliza dos capas de software que pueden ser extendidas muy fácilmente.

- **La capa de drivers de dispositivos** presenta a todos los dispositivos fisicos con una interface comun.

    - Tres tipos de drivers: character, block y network

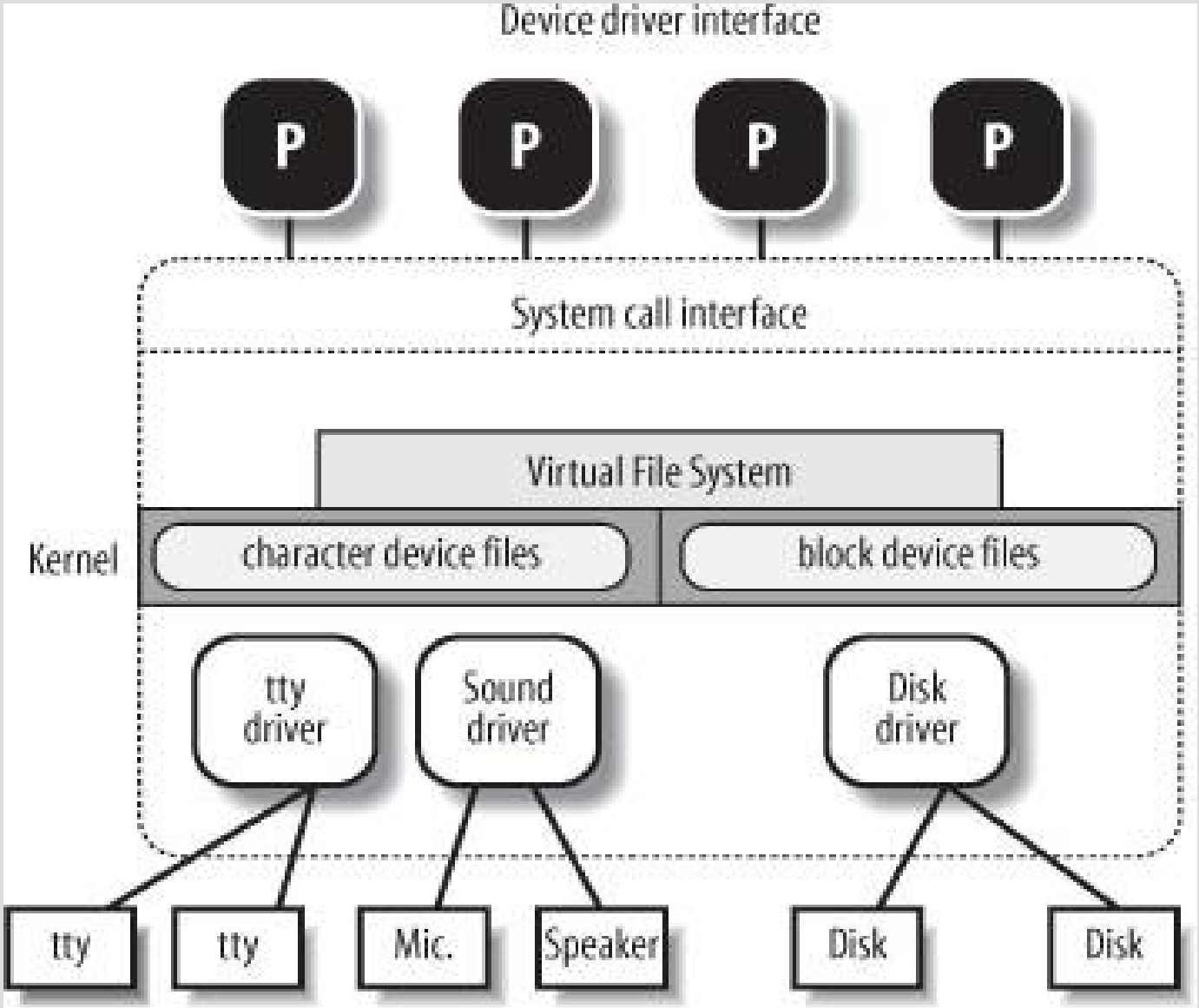
- **La capa de sistema de archivos virtual (VFS)** representa a todos los sistemas de archivos logicos con una interface unica.

    - Page cache (buffer cache) y el daemon pdflush.

Drivers de dispositivos

- El kernel interactúa con los dispositivios de E/S a través de los drivers de dispositivos.
- Se incluyen en el kernel y consisten en estructuras de datos y funciones que controlan uno o más dispositivios, como discos, teclados, mouses, interfaces de red, etc
- Cada driver además se comunica con el resto del kernel (incluso con otros drivers) usando una interfac específica.

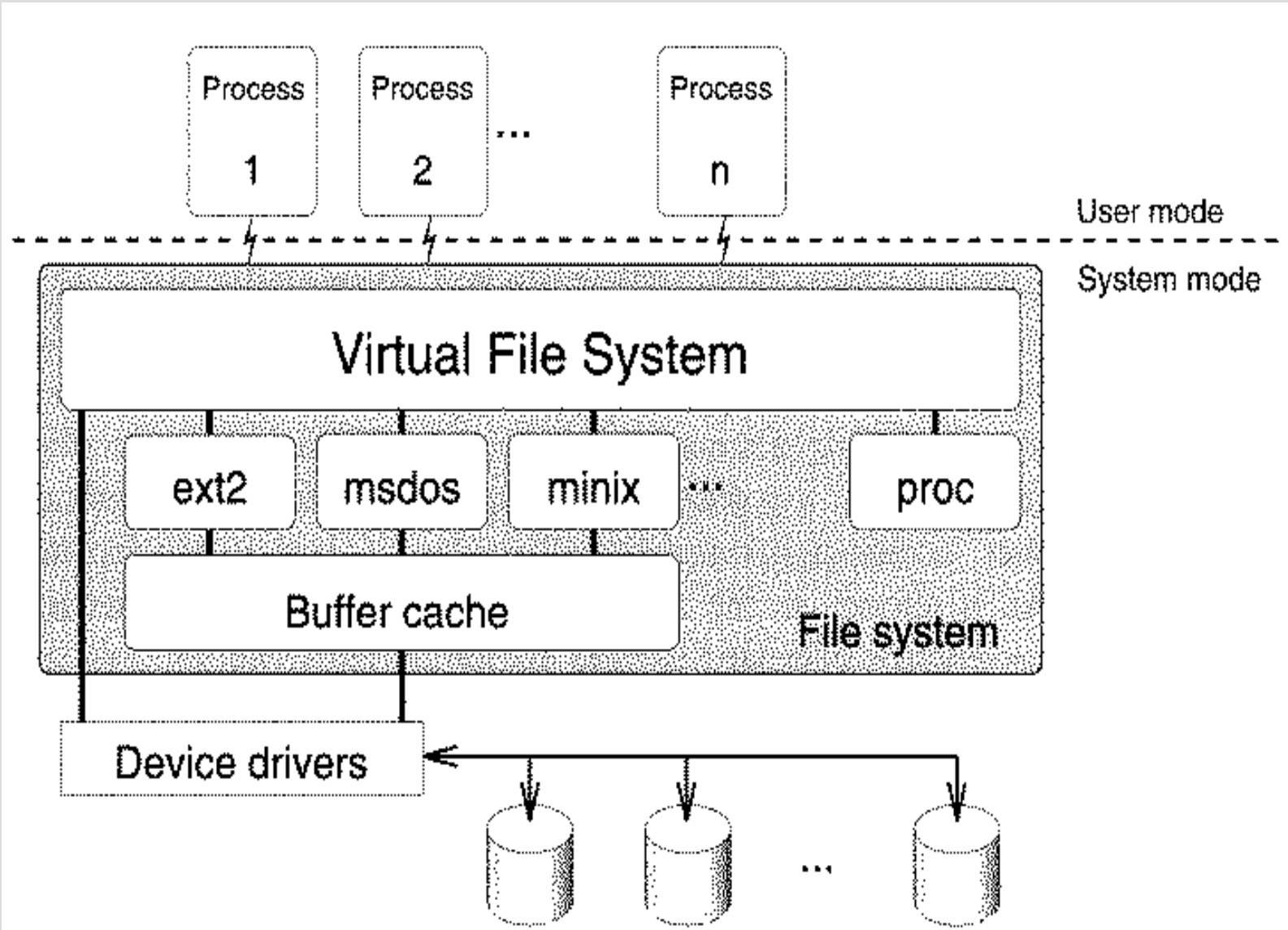
Drivers de dispositivos



Drivers de dispositivos linux/drivers

- largest amount of code in the kernel tree (~1.5M)
- device, bus, platform and general directories
- drivers/char – n_tty.c is the default line discipline
- drivers/block – elevator.c, genhd.c, linear.c, ll_rw_blk.c, raidN.c
- drivers/net –specific drivers and general routines Space.c and net_init.c
- general:
  - cdrom, ide, isdn, parport, pcmcia,
  - pnp, sound, telephony, video
- buses – fc4, i2c, nubus, pci, sbus, tc, usb
- platforms – acorn, macintosh, s390, sgi

Esquema de VFS



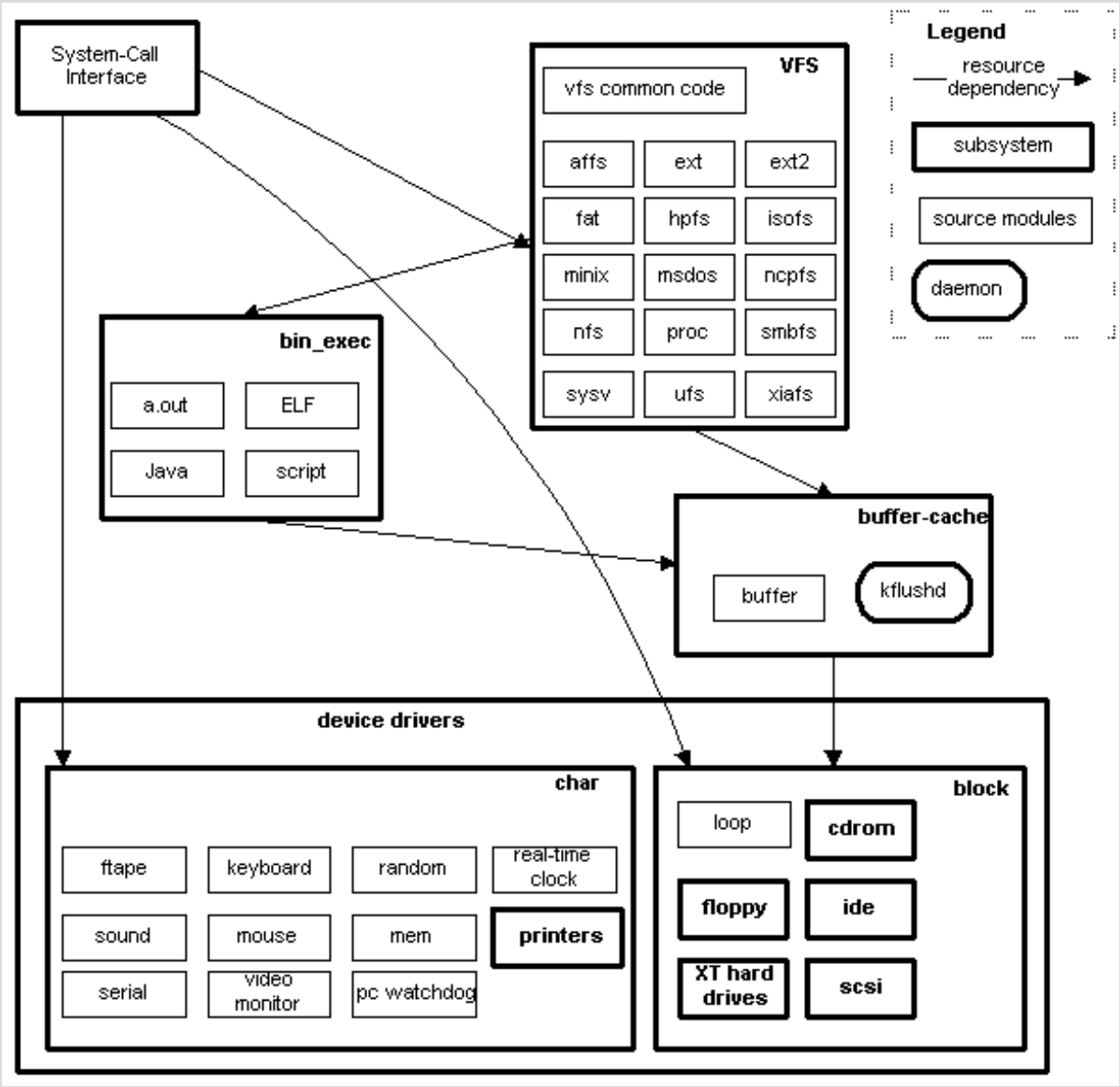
read()

sys_read()

vfs_read()

disk_dev_read()

Sistema de Archivos Virtual Arquitectura



Sistema de Archivos Virtual linux/fs

- contains:
  - virtual filesystem (VFS) framework
  - subdirectories for actual filesystems
- vfs-related files:
  - exec.c, binfmt_*.c - files for mapping new process images
  - devices.c, blk_dev.c – device registration, block device support
  - super.c, filesystems.c
  - inode.c, dcache.c, namei.c, buffer.c, file_table.c
  - open.c, read_write.c, select.c, pipe.c, fifo.c
  - fcntl.c, ioctl.c, locks.c, dquot.c, stat.c

Comunicación Entre Procesos Funcionalidad

- Señales: mensajes (muy cortos) asincrónicos enviados a procesos.
- Locks sobre regiones de archivos o archivos enteros.
- Pipes y Pipes nombradas (fifo): permiten transferencia unidireccionales de bytes entre dos procesos.
- System V IPC: Semáforos, Colas de Mensajes y Memoria Compartida.
- Sockets Unix

Comunicación Entre Procesos Interface Externa

- send_sig(), signal(): envio de señales y registro de handlers.
- open(), sys_fcntl(): Lock archivos enteros o regiones.
- pipe(), mknod(), read(), write(): crear pipes y pipes nombradas, y leer o escribir en ellas.
- ipc(): interface común para los mecanismos IPC System V.
- socketcall(): llamada para usar Unix Domain Sockets.

Comunicación Entre Procesos Descripción del Subsistema (I)

- Las señales son usadas para notificar a los procesos de ciertos eventos y tienen el efecto de alterar el estado del proceso que la recibe según la semántica de cada señal en particular. El kernel puede enviar señales a cualquier proceso en ejecución.

- Linux permite que los procesos de usuario marquen como exclusivo el acceso a un archivo. Esto puede hacerse sobre el archivo completo o sobre regiones de archivo. Para esto se usan los locks de archivos.

Comunicación Entre Procesos Descripción del Subsistema (II)

- Los pipes y las pipes nombrados son similares en cuanto a funcionalidad e implementación, pero se diferencian por el proceso de creación. Un pipe abierto se asocia con una página de memoria que es tratada como un buffer circular sobre el cual las operaciones de write son atómicas. Cuando el buffer está lleno, el proceso que escribe se bloquea. Si se intenta leer más datos de los que están disponibles, el proceso que lee se bloquea.

- Los semáforos se implementan usando wait queues, y se basan en el modelo clásico de semáforos.

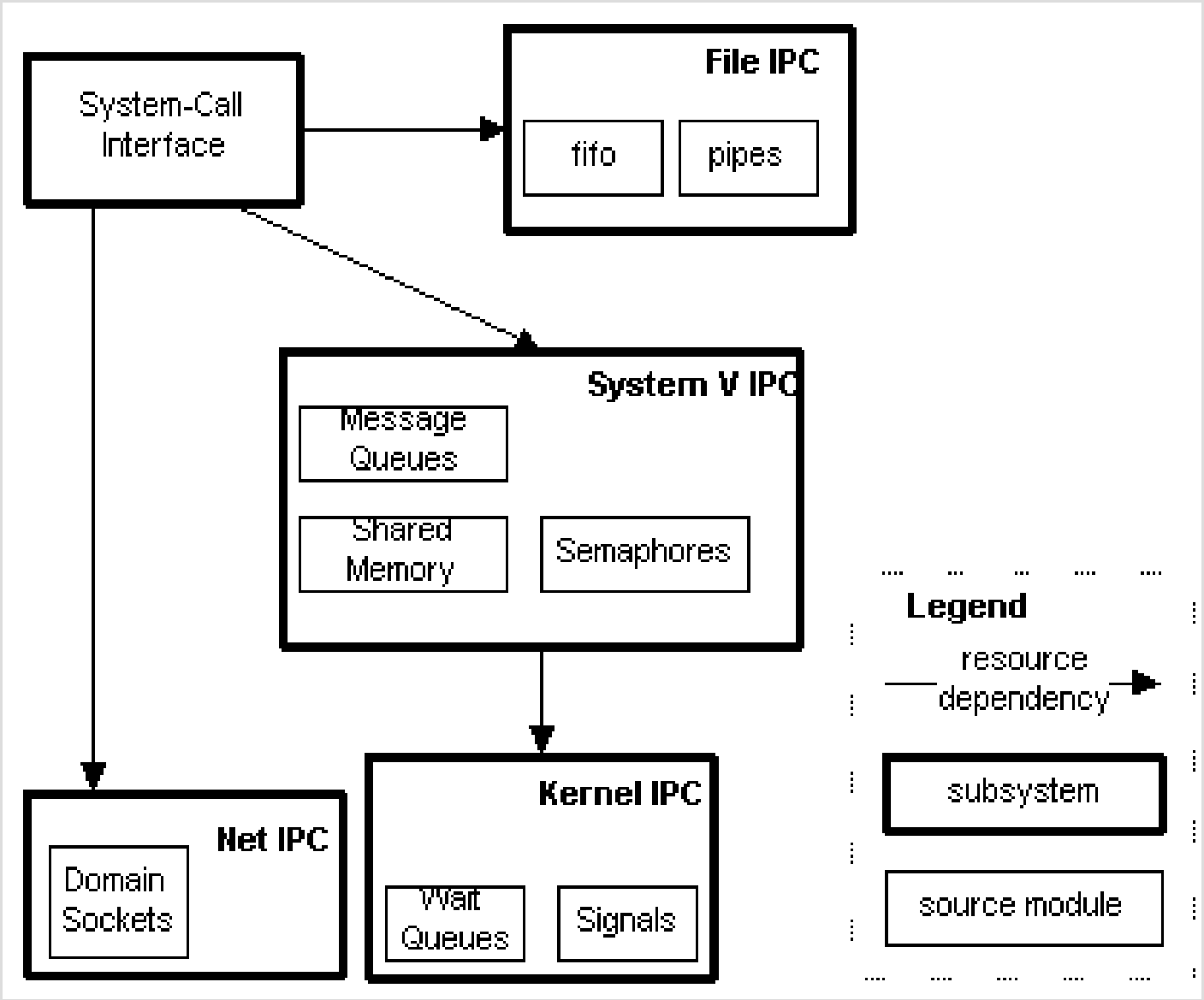
- Las colas de mensajes son listas enlazadas, en las cuales los procesos escriben o leen una secuencia de bytes. Los mensajes son recibidos en el mismo orden en el que se envían.

Comunicación Entre Procesos Descripción del Subsistema (III)

- La memoria compartida es la forma más rápida de IPC. Este mecanismo es manejado por el subsistema de administración de memoria.

- Los Unix domain sockets se implementan de manera similar a los pipes: ambos se basan en buffers circulares en una página de memoria. Sin embargo, los sockets proveén un buffer separado para cada dirección en la comunicación.

Comunicación Entre Procesos
Arquitectura



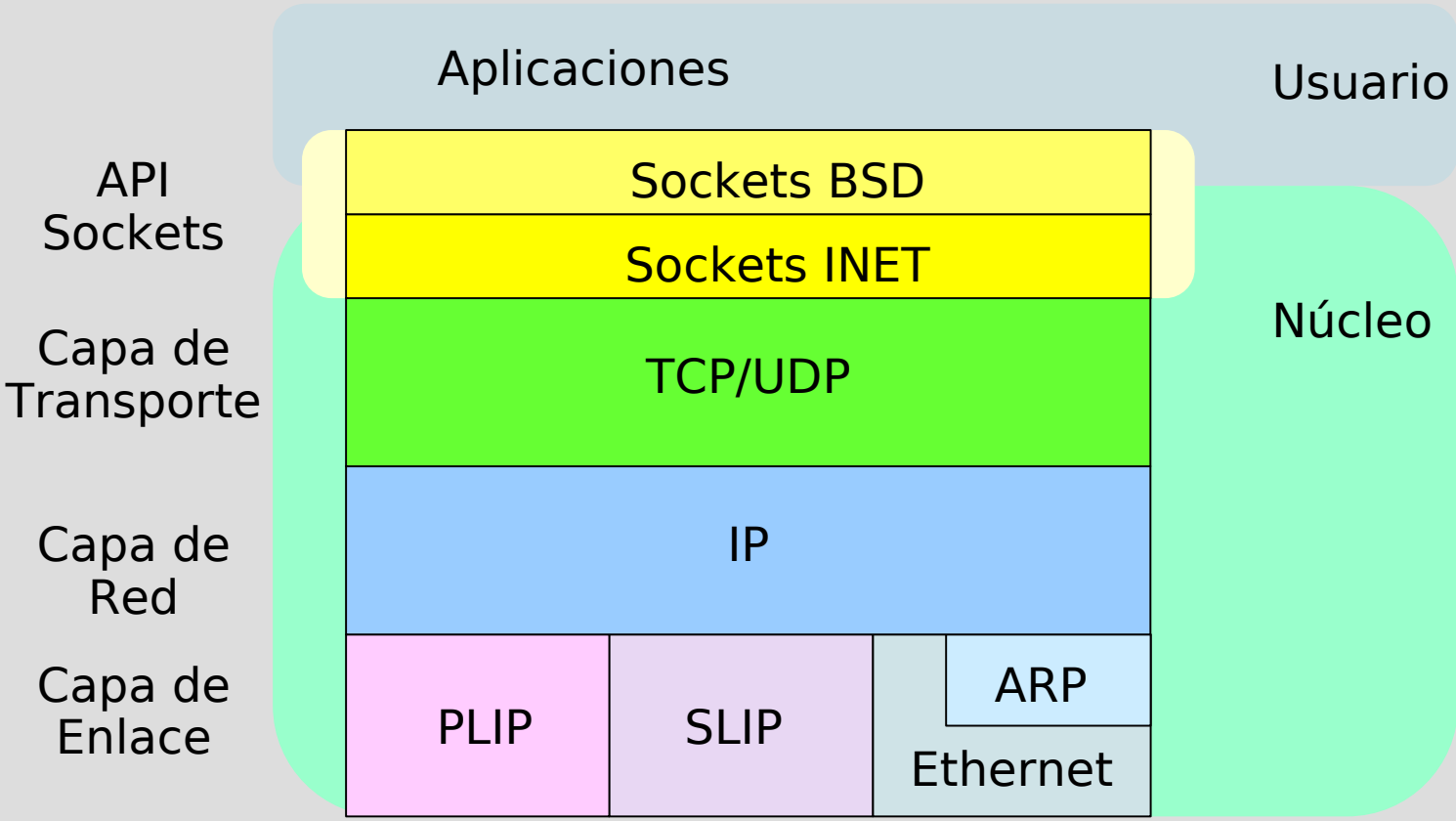
linux/ipc

- System V IPC facilities
- if disabled at compile-time, util.c exports stubs that simply return –ENOSYS
- one file for each facility:
  - sem.c – semaphores
  - shm.c – shared memory
  - msg.c – message queues

Interface a la Red
Objetivos

- El subsistema de red de Linux proveé conectividad de red y un modelo de comunicación basado en sockets BSD.
- Protocolos de red.
- Drivers de dispositivos de red

Pila TCP/IP


Aplicaciones
Usuario
API Sockets
Sockets BSD
Sockets INET
Capa de Transporte
TCP/UDP
Núcleo
Capa de Red
IP
Capa de Enlace
PLIP
SLIP
ARP
Ethernet

Interface a la Red
Interface Externa

- Los servicios de red son utilizados por el usuario a través de la interface que brindan los sockets.

- Los sockets son creados y manipulados a través de la **system call** socketcall(), aunque se suelen usar **funciones de librería** como socket(), bind(), listen(), accept(), connect().

- Los datos son enviados y recibidos usando read() y write() sobre el file descriptor del socket.

Interface a la Red
Descripción del Subsistema

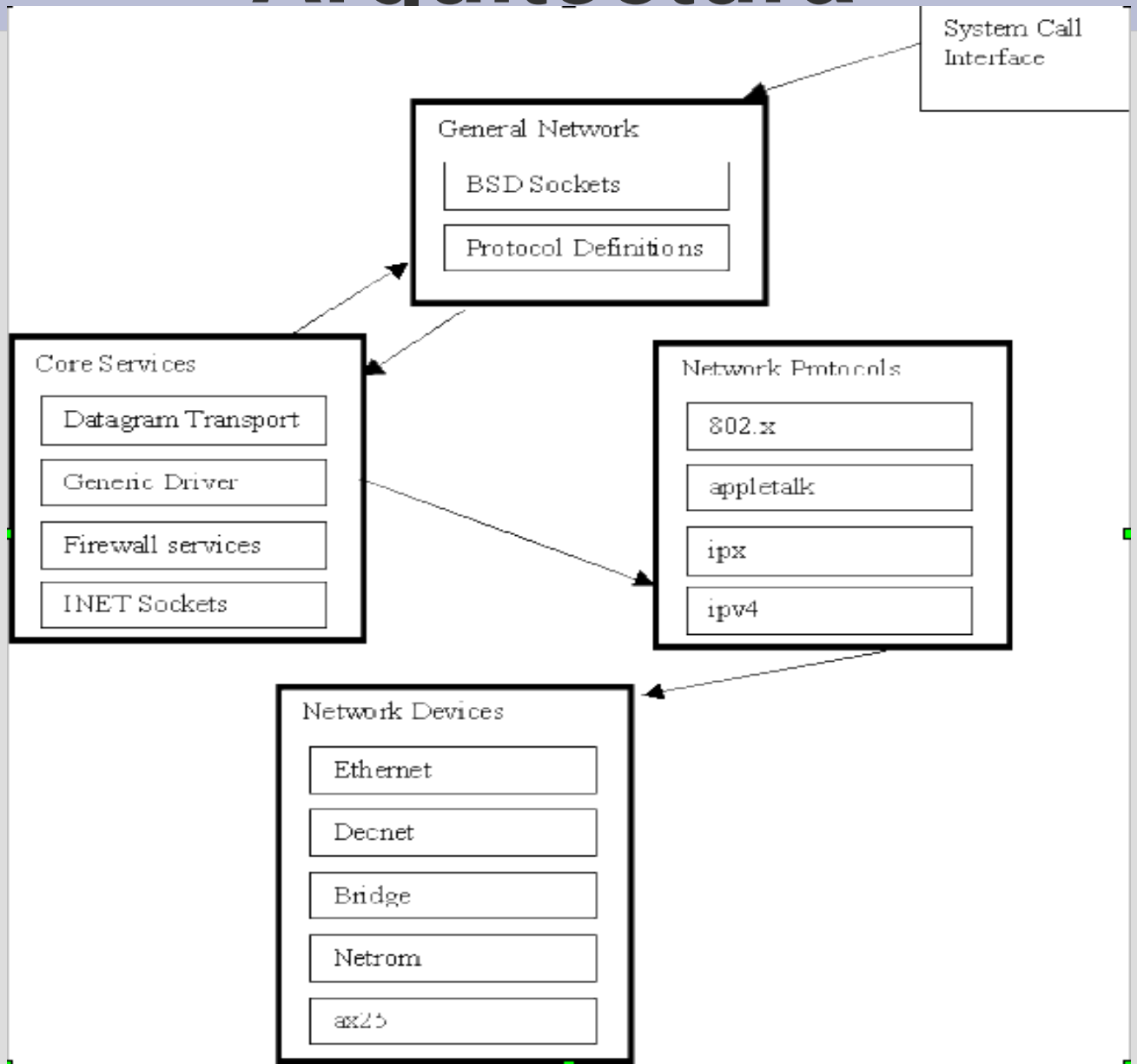
- La **interface al usuario es el modelo de socket BSD**, el cual tiene como propósito proveer una mayor portabilidad abstrayendo los detalles de las comunicaciones a través de una interface común.

- Los **sockets INET** son los que realmente manejan la comunicación entre los puntos para los **protocolos TCP y UDP.**

- Una E/S de red comienza con un **read() o un write() sobre un socket** que es manejado por **VFS**. Desde ese punto, siguen una serie de llamadas. Por ejemplo: sock_write() (capa BSD), inet_write(), tcp_write(), etc.

- Se proveén **drivers de dispositivos para todo tipo de hardware**. A las capas superiores se les presenta una **interface abstracta** para esconder las diferencias de los dispositivos.

Interface a la Red
Arquitectura



Referencias

- Conceptual Architecture of the Linux Kernel (paper)
- Concrete Architecture of the Linux Kernel (paper)
- Understanding the Linux kernel. 3era edición.
- Linux Kernel Development. 2da edición.
- Understanding Linux Network Internals.
- The Linux TCP/IP Stack: Networking for Embedded Systems.
- Internet