Advanced Operating Systems
2.OS Structure.8

Chun-Han Lin (林均翰)
CSIE, NTNU

Key Points 2.8

- POSIX standard

POSIX Standard (1/5)

- **Portable Operating System Interface for UNIX (POSIX)**
  - **An attempt to standardize a “UNIXy” interface**
- **Conformance: IEEE POSIX 1003.1 and ISO/IEC 9945.**
  - **Latest version from 2008**
  - **Originally one document consists of a core programming interface.**
    - **Now 19 separate docs**
  - **Many OS provide “partial conformance” (including Linux).**

POSIX Standard (2/5)

- What does POSIX define?
  - POSIX.1: core services
    - Process creation and control
    - Signals
    - Floating point exceptions, segmentation/memory violations, illegal instructions, bus errors
    - Timers
    - File and directory operations
    - Pipes
    - C library (standard C)
    - I/O port interface and control
    - Process triggers
  - POSIX.1b: real-time extensions
  - POSIX.2: shell and utilities

POSIX Standard (3/5)

- **Process primitives**
  - **Fork, execl, execlp, execv, execve, execvp, wait, waitpid**
  - **Exit, kill, sigxxx, alarm, pause, sleep**
- **File access primitives**
  - **Opendir, readdir, rewinddir, closedir, chdir, getcwd, open, creat, umask, link, mkdir, unlink, rmdir, rename, stat, fstat, access, fchmod, chown, utime, ftruncate, pathconf, fpathconf**



POSIX Standard (4/5)

- **I/O primitives**
  - **Pipe, dup, dup2, close, read, write, fcntl, lseek, fsync**
- **C-language primitives**
  - **Abort, exit, fclose, fdopen, fflush, fgetc, fgets, fileno, fopen, fprintf, fputc, fputs, fread, freopen, fscanf, fseek, ftell, fwrite, getc, getchar, gets, perror, printf, putc, putchar, puts, remove, rewind, scanf, setlocale, siglongjmp, sigsetjmp, tmpfile, tmpnam, tzset**

POSIX Standard (5/5)

- Synchronization
  - Sem_init, sem_destroy, sem_wait, sem_trywait, sem_post, pthread_mutex_init, pthread_mutex_destroy, pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock
- Memory management
  - Mmap, mprotect, msync, munmap.
- How to get information on a system call?
  - Type “man callname”, i.e. “man open”
  - System calls are in section “2” of the man pages.

Review 2.8

- **POSIX standard**

Advanced Operating Systems
2.OS Structure.9

Chun-Han Lin (林均翰)
CSIE, NTNU

Key Points 2.9

- Portability

- Examples of Windows and Unix system calls

Portability

- POSIX does provide some portability.
  - But is still pretty high level
  - Does not specify file systems, network interfaces, power management, other important things
  - Many variations in compilers, user programs, libraries, other build environment aspects
- UNIX portability
  - C-preprocessor conditional compilation
  - Conditional and multi-target makefile rules
  - GNU configure scripts to generate makefiles
  - Shell environment variables: LD_LIBRARY_PATH, LD_PRELOAD, …

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Review 2.9

- Portability

- Examples of Windows and Unix system calls

Advanced Operating Systems
2.OS Structure.10

Chun-Han Lin (林均翰)

CSIE, NTNU

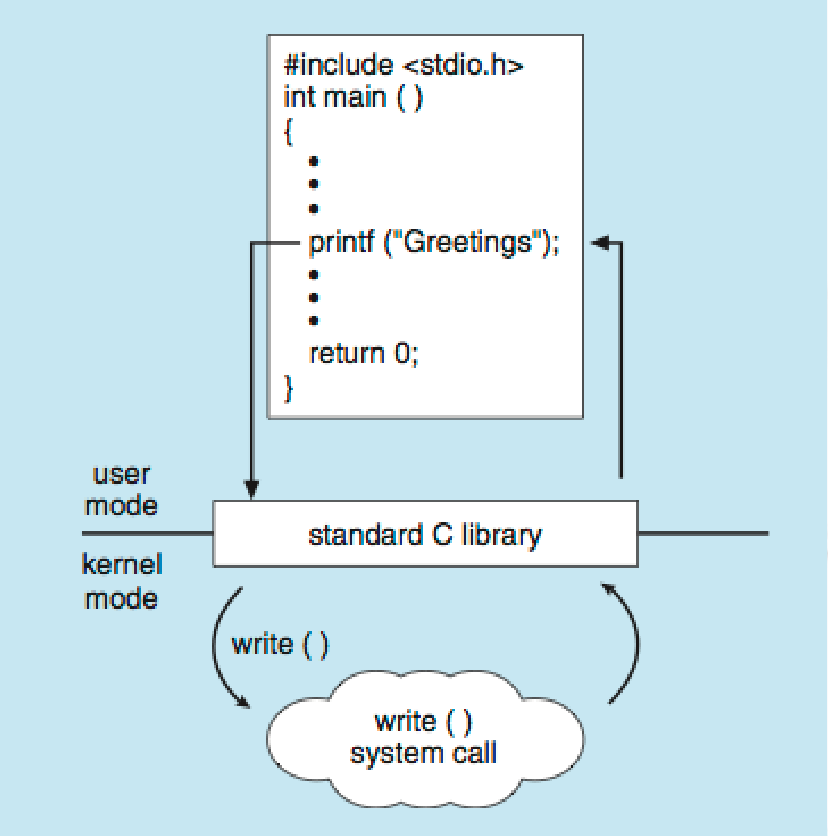


Key Points 2.10

- Standard C library example
- Implementation issues
- OS structure

Standard C Library Example

- C programs invoke printf() library call, which calls write() system call.



Implementation Issues (How is OS implemented?) (1/2)

- **Policy vs mechanism**
  - **Policy: what do you want to do?**
  - **Mechanism: how are you going to do it?**
  - **Should be separated, since both change.**

- **Algorithm used**
  - **Linear, tree-based, log structured, …**

Implementation Issues (How is OS implemented?) (2/2)

- Event model used
  - Threads vs event loops

- Backward compatibility issues
  - Very important for Windows 2000/XP

- System generation/configuration
  - How to make generic OS fit on specific hardware?

OS Structure (What is the organizational principle?)

- **Simple**
  - Only one or two levels of code
- **Layered**
  - Lower levels independent of upper levels
- **Microkernel**
  - OS built from many user-level processes
- **Modular**
  - Core kernel with dynamically loadable modules
- **Exokernel (research paper)**



Review 2.10

- **Standard C library example**
- **Implementation issues**
- **OS structure**



Advanced Operating Systems
2.OS Structure.11

Chun-Han Lin (林均翰)

CSIE, NTNU

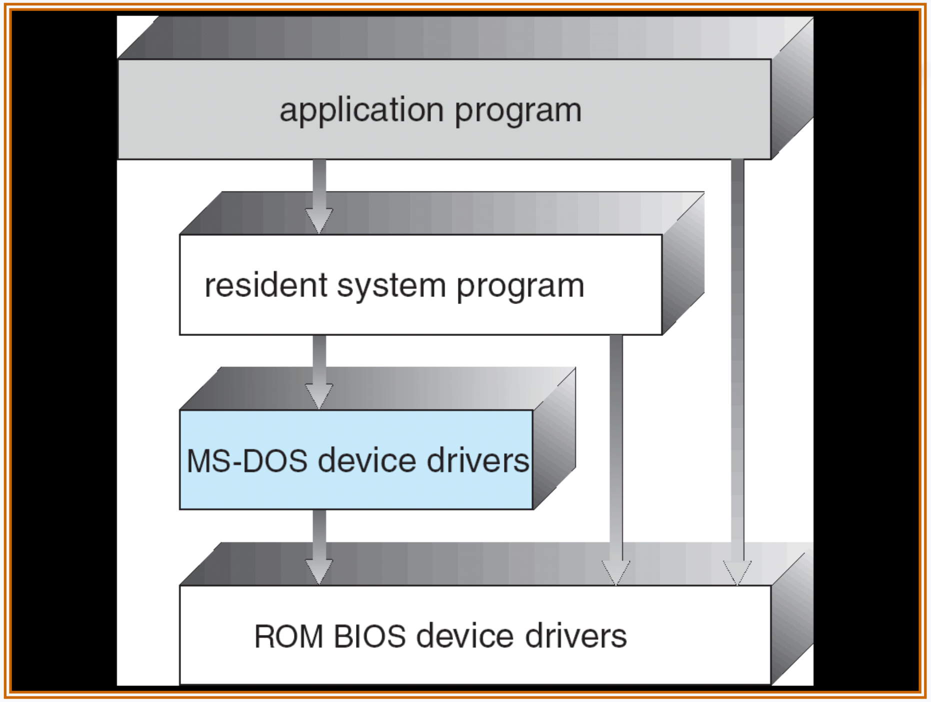


Key Points 2.11

- **Simple structure**

Simple Structure (1/2)

- **All aspects of OS are linked together in one binary.**
  - API are not carefully designed (and/or lots of global variables).
  - Interfaces and levels of functionality are not well separated.
  - No address protection



Simple Structure (2/2)

- Example: MS-DOS
  - Provide the most functionality in the least space
  - Made sense in early days of personal computers with limited processors, e.g., 6502
- Advantages
  - Low memory footprint
- Disadvantages
  - Very fragile, no enforcement of structure/boundaries
- What about language enforcement? Microsoft singularity?

Review 2.11

- **Simple structure**

Advanced Operating Systems
2.OS Structure.12

Chun-Han Lin (林均翰)
CSIE, NTNU

Key Points 2.12

- **Layered structure**
- **Monolithic structure: UNIX system structure**

Layered Structure (1/2)

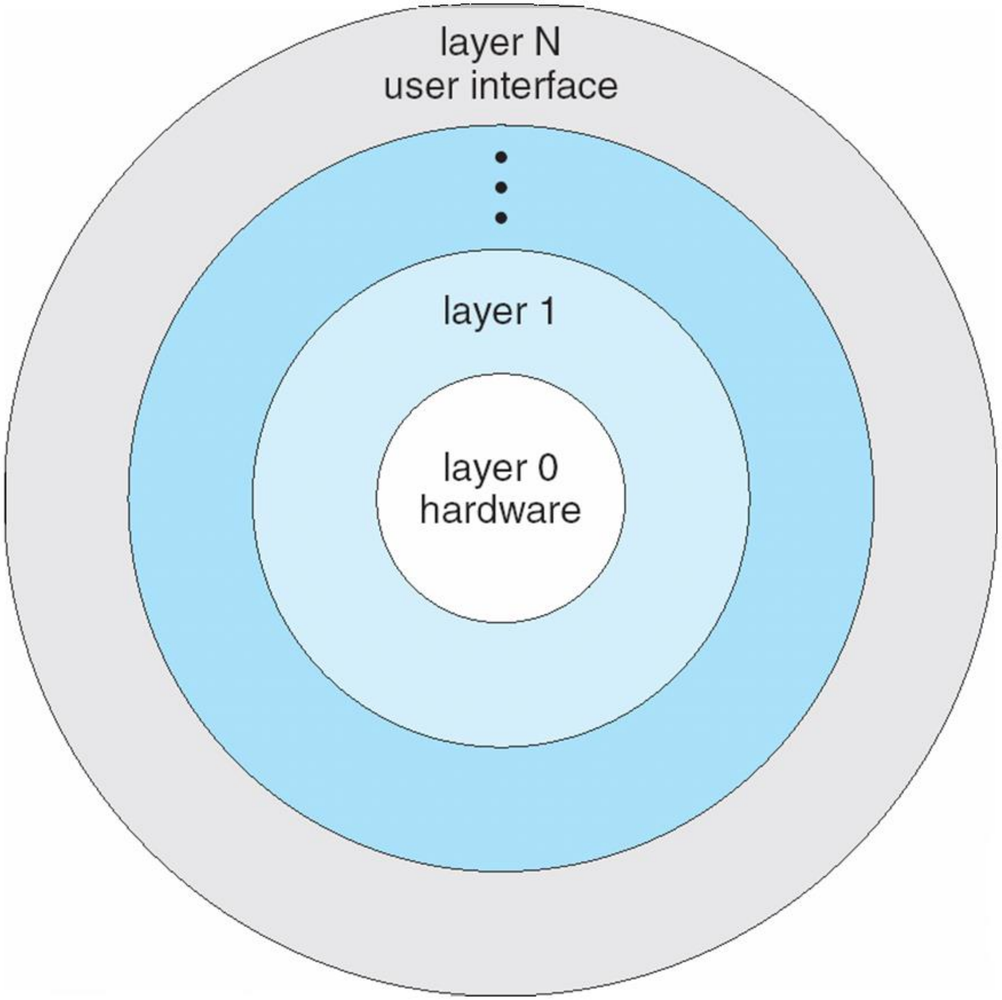
- OS is divided into many layers or levels.
  - Each is built on top of lower layers.
  - Bottom layer (layer 0) is hardware.
  - Highest layer (layer N) is the user interface.
- Each layer uses functions, operations, and services of only lower-level layers.
  - Advantage: Modularity ⇒ Easier debugging/maintenance
  - Not always possible: Does process scheduler lie above or below virtual memory layer?
    - Need to reschedule processor while waiting for paging
    - May need to page in information about tasks

Layered Structure (2/2)

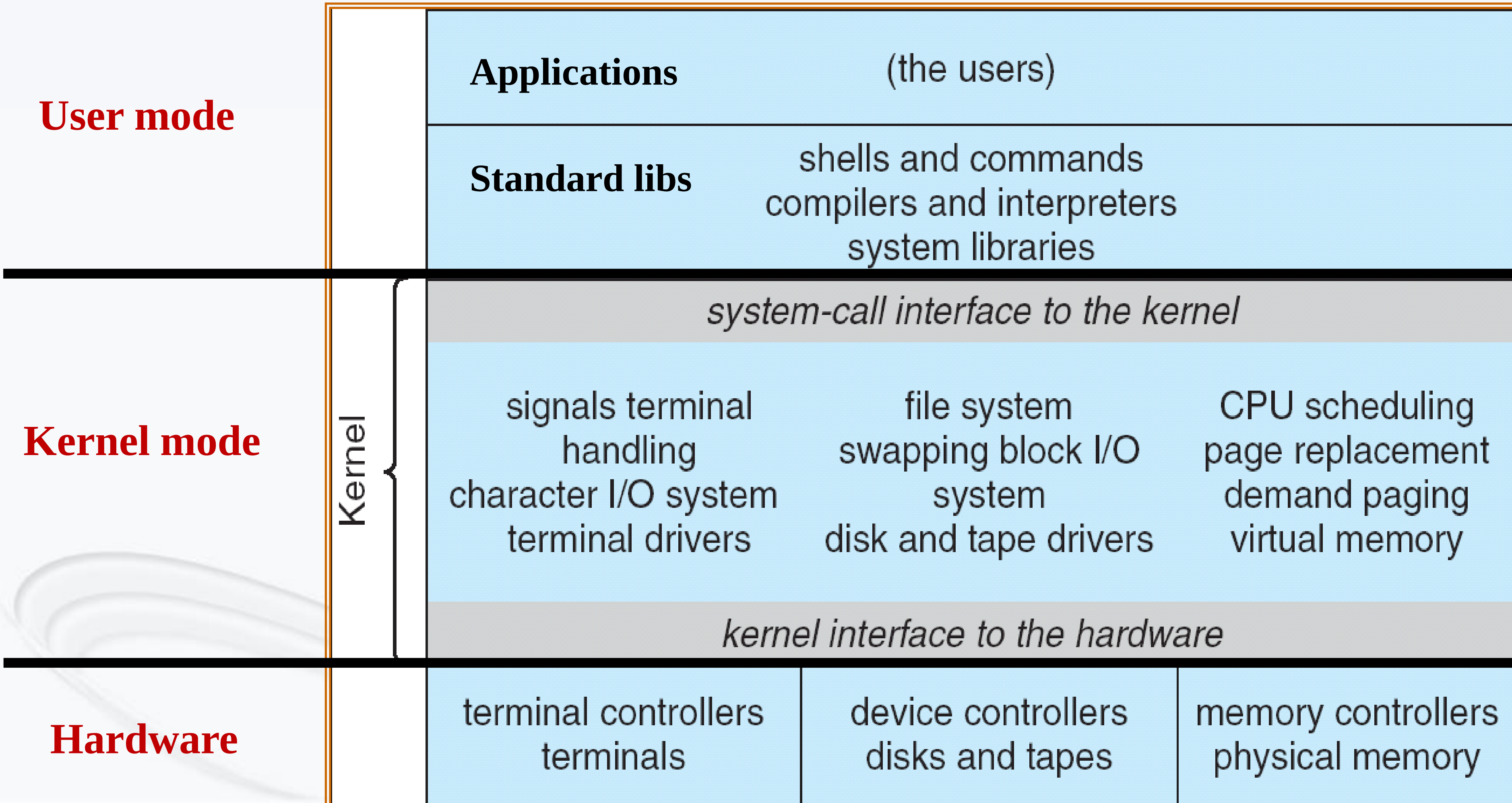
- **Important: Machine-dependent vs independent layers**
  - Easier migration between platforms
  - Easier evolution of hardware platform
  - Good idea for you as well!

- Can utilize hardware enforcement
  - x86 processor: 4 “rings”
  - Call gates

Layered OS



Monolithic Structure: UNIX System Structure (1/2)



Monolithic Structure: UNIX System Structure (2/2)

- Two-layered structure: user vs kernel
  - All codes representing protection and management of resources are placed in same address space.
  - Compromise of one component can compromise whole OS
- Clear division of labor?
  - The producer of OS and the user of OS

Review 2.12

- **Layered structure**

- **Monolithic structure: UNIX system structure**

Advanced Operating Systems
2.OS Structure.13

Chun-Han Lin (林均翰)

CSIE, NTNU

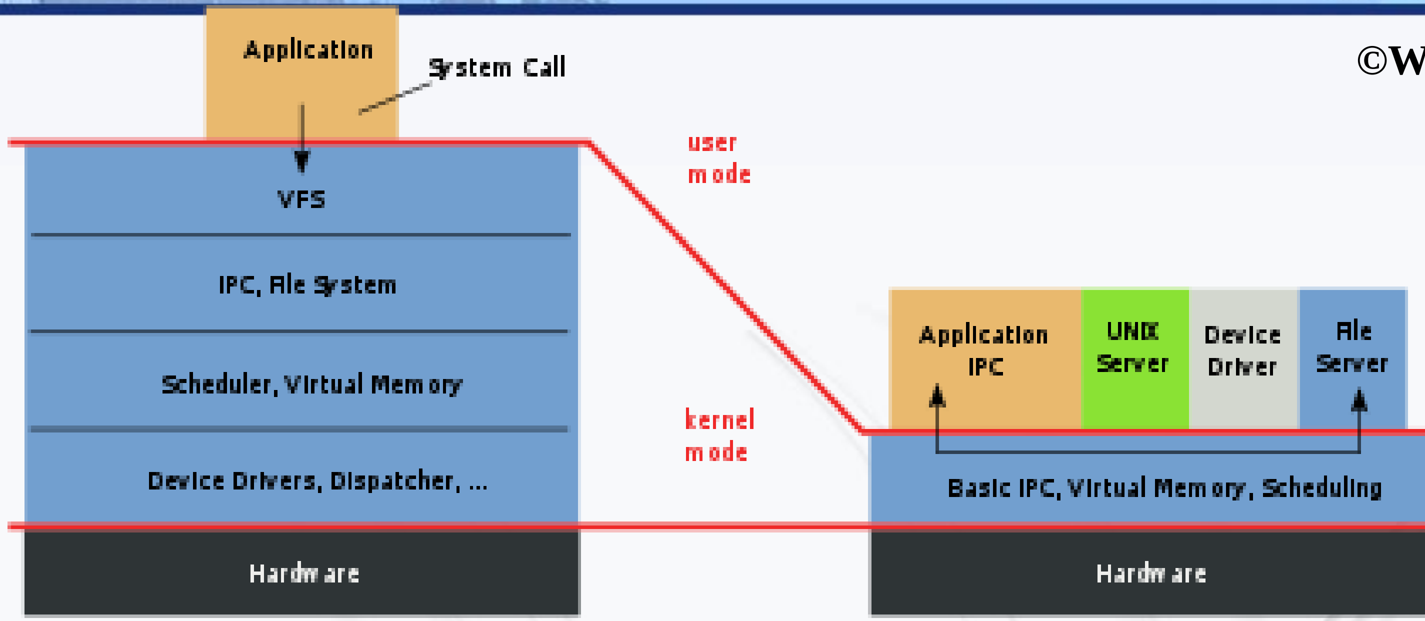


Key Points 2.13

- Microkernel structure

Microkernel Structure (1/2)

Monolithic kernel



Microkernel

- **Move functionality from the kernel into user space**
  - **Small core OS running at kernel level**
  - **OS services are built from many independent user-level processes.**
  - **Communication between modules with message passing**

Microkernel Structure (2/2)

- **Advantages**
  - **Easier to extend a microkernel**
  - **Easier to port OS to new architectures**
  - **More reliable (less code is running in kernel mode)**
  - **Fault isolation (parts of kernel protected from other parts)**
  - **More secure**
- **Disadvantages**
  - **Performance overhead can be severe for naïve implementation.**

Review 2.13

- Microkernel structure

Advanced Operating Systems
2.OS Structure.14

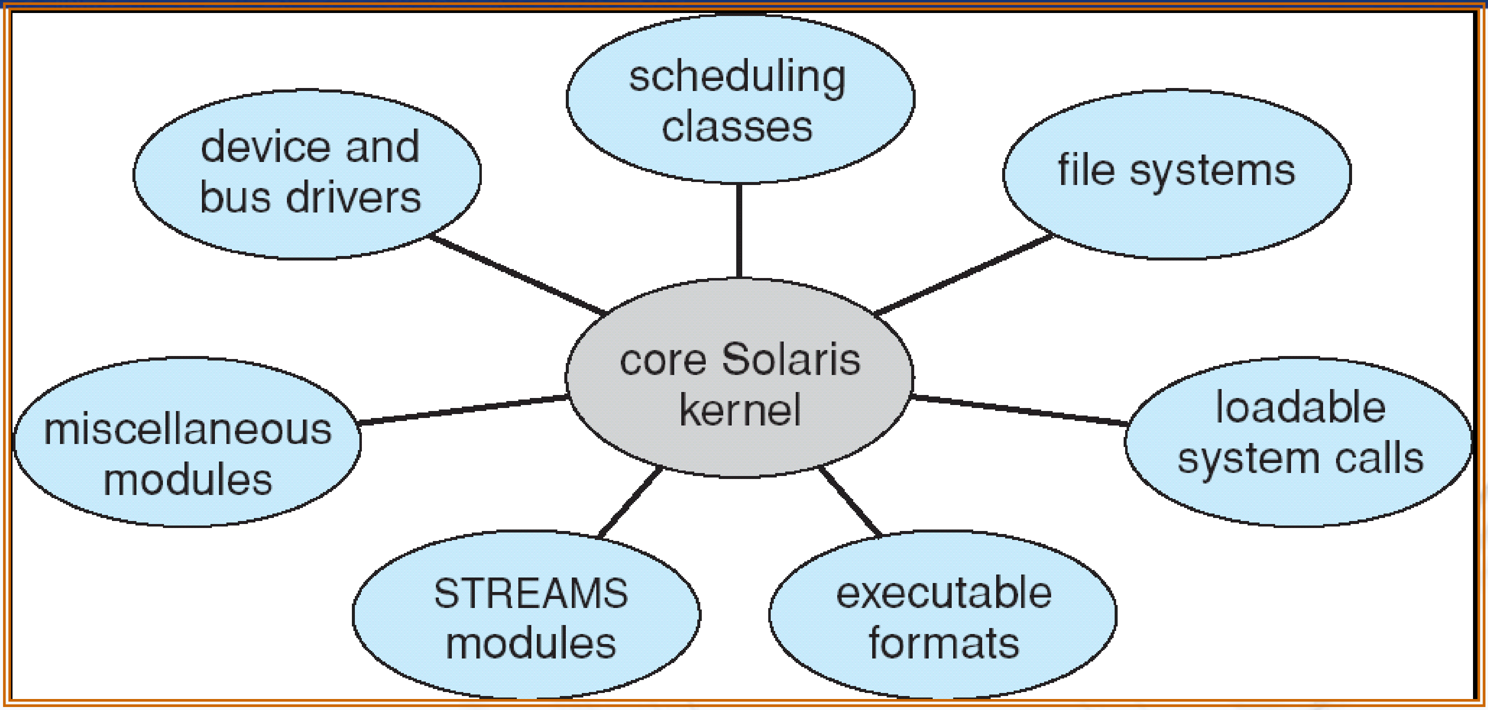
Chun-Han Lin (林均翰)
CSIE, NTNU

Key Points 2.14

- Module-based Structure
- ExoKernel
- Conclusion



Module-based Structure (1/2)



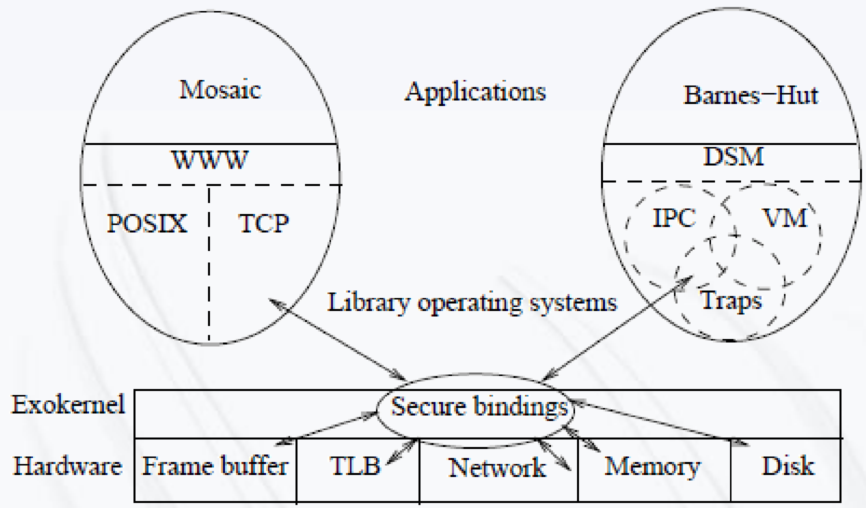
- Most modern OS implement modules.
  - Use object-oriented approach
    - Careful API design/few if any global variables

Module-based Structure (2/2)

- Each core component is separate.
  - Each talks to the others over known interfaces.
  - Each is loadable as needed within the kernel.

- Overall, similar to layers but with more flexible
  - May or may not utilize hardware enforcement

ExoKernel: Separate Protection from Management (1/2)



- **Thin layer exports hardware resources directly to users.**
  - **As little abstraction as possible**
  - **Secure protection and multiplexing of resources**

ExoKernel: Separate Protection from Management (2/2)

- LibraryOS: traditional OS functionality at user-level.
  - Customize resource management for every application
  - Is this a practical approach?

- Very low-level abstraction layer
  - Need extremely specialized skills to develop LibraryOS

Conclusion (1/2)

- Resource control: In HW or SW!
  - Access/No access/Partial access
  - Resource multiplexing
  - Performance isolation
- System-call interface
  - This is the I/O for the process “virtual machine”.
  - Accomplished with special trap instructions which vector off a table of system calls
  - Usually programmers use the standard API provided by the C library rather than direct system-call interface.



Conclusion (2/2)

- **POSIX interface**
  - **An attempt to standardize “UNIXy” OS**

- **Many organizations for OS**
  - **All oriented around resources**
  - **Common organizations: monolithic, microkernel, exokernel**

Review 2.14

- **Module-based Structure**
- **ExoKernel**
- **Conclusion**