Advanced Operating Systems
2.OS Structure.1

Chun-Han Lin (林均翰)

CSIE, NTNU

Key Points 2.1

- Chapter goals
- OS resources
- What is a resource?

Goals

- **OS Resources**
- **API Access to Hardware Resources**
- **OS Design**
- **Modern Architectures**



Recall: UNIX System Structure

User mode

Applications (the users)

Standard libs
shells and commands
compilers and interpreters
system libraries

Kernel mode

Kernel

system-call interface to the kernel

signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory

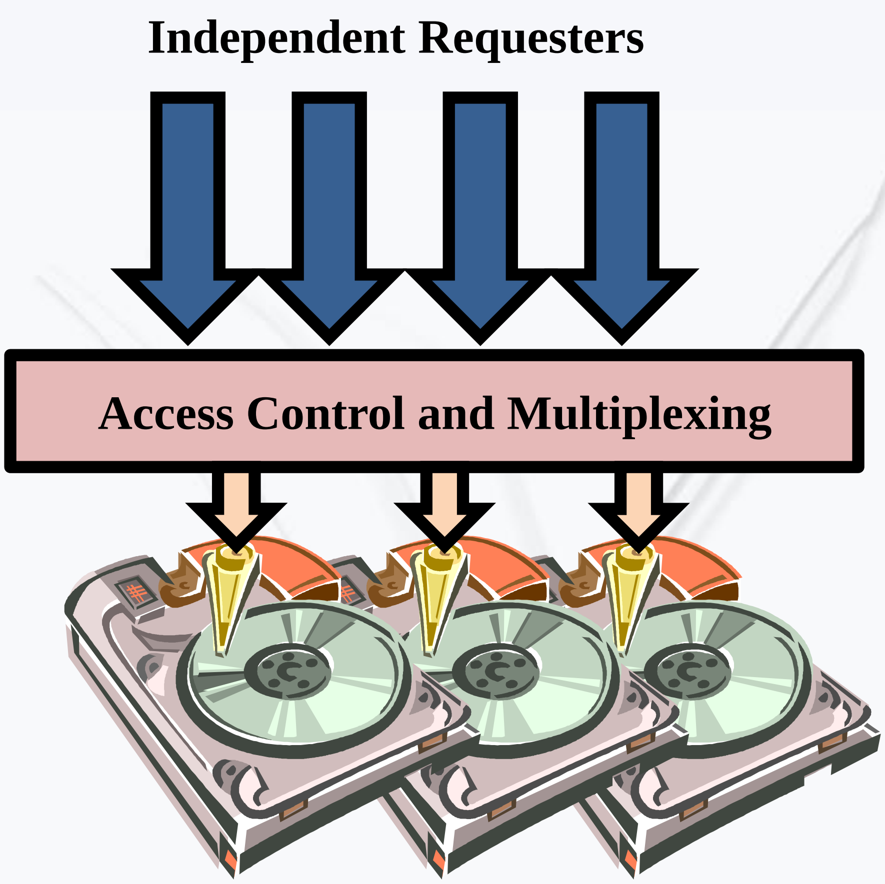
kernel interface to the hardware

Hardware

terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory

OS Resources – at the center of it all!

- What do modern OS do?
  - Control access to resources!
- Control of resources
  - Access / No access / Partial access
    - Check every access to see if it is allowed
  - Resource multiplexing
    - When multiple valid requests occur at same time – how to multiplex access?
    - What fraction of resource can requester get?
  - Performance isolation
    - Can requests from one entity prevent requests from another?
- What or who is a requester?
  - Process? user? public key?
  - Think of this as a “principle”.



Prof. Chun-Han Lin, CSIE, NTNU

What is a resource? (1/3)

- Processor, memory, and cache
  - Multiplex: scheduling and virtual memory
  - Abstraction: process, thread
  - Need kernel level to multiplex?
    - Need to sandbox somehow
    - Kernel control of memory, prevent certain instructions



What is a resource? (2/3)

- Network
  - Multiplex: queues and input filters
  - Abstraction: socket API
  - Need kernel level to multiplex?
    - Not necessarily: new hardware has on-chip filters.
    - Setup cost, but not necessarily a per-packet cost
    - Is network really secure anyway? Need crypto



What is a resource? (3/3)

- **Disk**
  - **Multiplex: buffer cache**
  - **Abstraction: file system API**
  - **Need kernel level to multiplex?**
    - **Traditionally all access control through kernel.**
    - **What about assigning unlimited access to partitions?**

Review 2.1

- Chapter goals
- OS resources
- What is a resource?

Advanced Operating Systems
2.OS Structure.2

Chun-Han Lin (林均翰)

CSIE, NTNU



Key Points 2.2

- **More complex resources: OS services**

More Complex Resources: OS Services (1/3)

- System services are really complex resources.
- File system (uses disk drive)
  - File API: create, read, write, delete
  - Access control: user, group, world, read/write/execute
- Windows system (uses graphics card)
  - Windowing API: write text, draw/fill in figures
  - Access control: per window (user created)

More Complex Resources: OS Services (2/3)

- Database (uses disk drive/memory/network)
  - DB API: SQL queries and transactions
  - Access control: per user, group, others

- Lock service (memory)
  - Lock API: acquire (read, write), release
  - Access control: by group/per user

More Complex Resources: OS Services (3/3)

- **Access is controlled through syscall interfaces (in kernel level).**
  - **Funnel all access through trusted and verified API**
    - **Kernel controls access to API, verifies identity.**
    - **Service controls access to resources using identity.**
  - **Service decides multiplexing/isolation policies.**
    - **Often based on first-come-first-serve policy!**

Review 2.2

- More complex resources: OS services

Advanced Operating Systems
2.OS Structure.3

Chun-Han Lin (林均翰)

CSIE, NTNU

Key Points 2.3

- **Protection vs Management**

- **Traditional approach to handling resource**

Protection vs Management (1/2)

- Kernels mix protection, performance isolation, and management.
  - Protection: Should a principle have access to a given resource?
    - Yes or no?
    - Based on local password file? thumbprint? cryptographic key?
  - Performance isolation: How much of bandwidth-limited resource should a principle have access to?
    - 50% CPU
    - As much network as desired
    - Fraction of paging disk for virtual memory?

Protection vs Management (2/2)

- **Management: How should the principle use this resource?**
  - **Scheduling, policies**
  - **Use my CPU resources to meet real-time deadlines vs highest throughput scheduling**
  - **Keep certain pages in memory**

- **Problem with putting all three of these together is that API are limited, complex, insufficient, …**

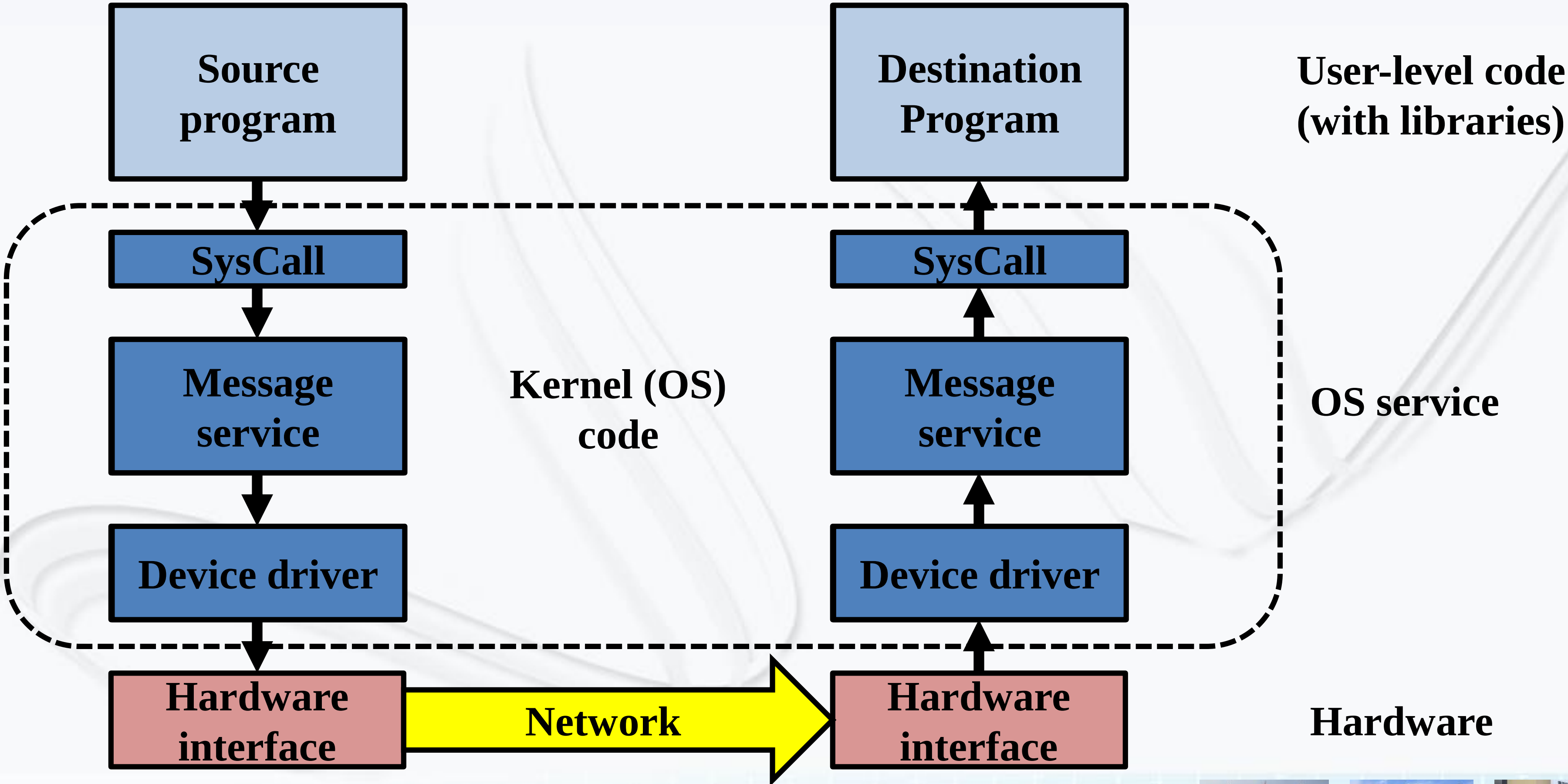


Traditional Approach to Handling Resource (1/2)

- **Example: send a message from one processor to another**
  - **Check permission, format the message**
  - **Enforce forward progress, handle interrupt**
  - **Prevent Denial Of Service (DOS) and/or deadlock**

Traditional Approach to Handling Resource (2/2)

- **Traditional approach: use a system call + OS services**



Review 2.3

- Protection vs Management

- Traditional approach to handling resource

Advanced Operating Systems
2.OS Structure.4

Chun-Han Lin (林均翰)
CSIE, NTNU

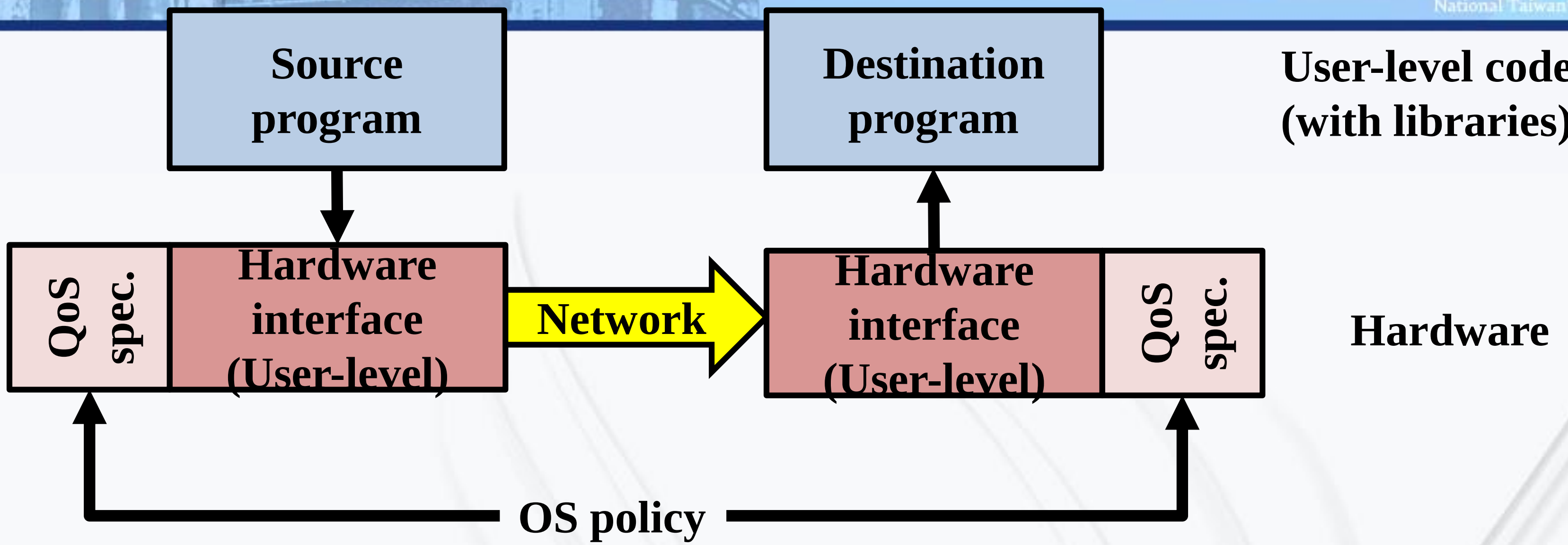


Key Points 2.4

- **Alternative: Mechanisms in hardware**

- **System calls: Details**

Alternative: Mechanisms (or even Policy!?) in Hardware



- **Permit user-level code to send messages**
  - Have hardware check permission and/or rate
  - Have hardware enforce format/consistency
  - Have hardware guarantee forward progress
  - Have hardware deliver messages/interrupts to user code
- OS sets registers to control behavior based on policy.

System Calls: Details (1/2)

- Challenge: interaction despite isolation
  - How to isolate processes and their resources?
    - While still permitting them to request help from the kernel
    - Let them interact with resources while maintain usage policies such as security, QoS, …
  - Let processes interact with one another in a controlled way
    - Through messages, shared memory, …
- Enter the system call interface
  - Layer between the hardware and user-space processes
  - Programming interface to the services provided by OS

System Calls: Details (2/2)

- System calls are mostly accessed by programs via a high-level application program interface (API) rather than directly.
  - Get at a system call by linking with a library in glibc

Call to printf() → printf() in the C library → write() system call

- Three most common APIs are …
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  - Java API for Java virtual machine (JVM)

Review 2.4

- **Alternative: Mechanisms in hardware**

- **System calls: Details**

Advanced Operating Systems
2.OS Structure.5

Chun-Han Lin (林均翰)

CSIE, NTNU



Key Points 2.5

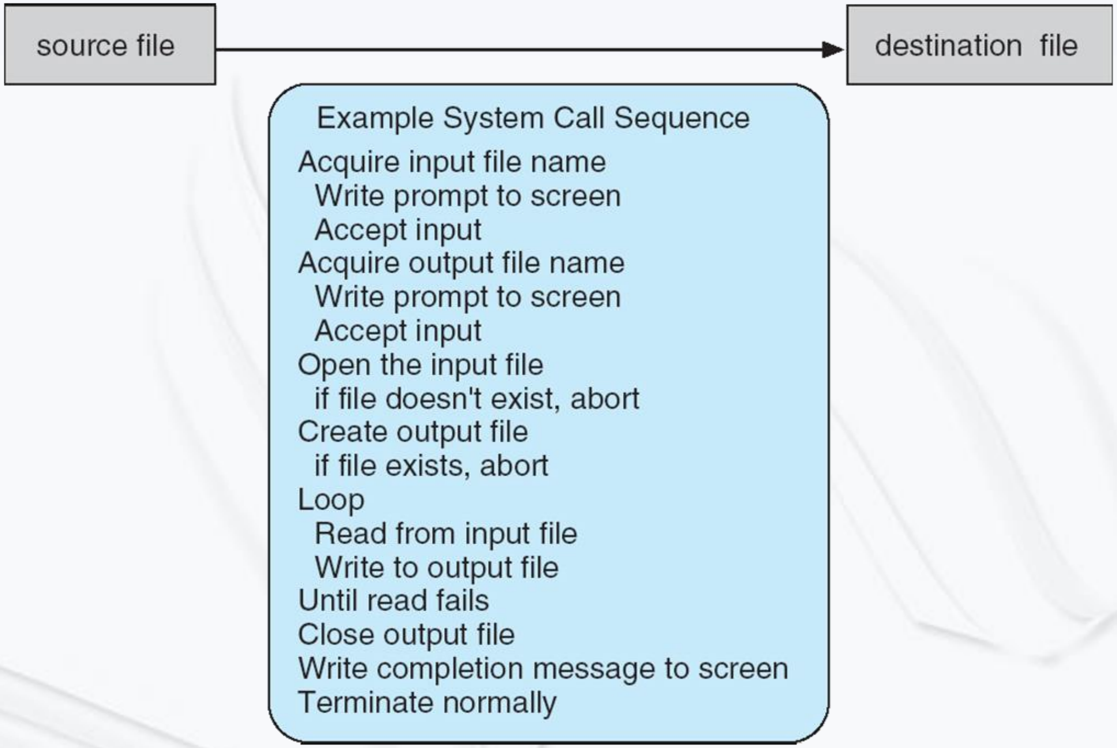
- Example of system call usage

- Example: Use strace to trace Syscalls

- Example of standard API

Example of System Call Usage

- System call sequence to copy the contents of one file to another file



- Many crossings of the user/kernel boundary!
  - The cost of traversing this boundary can be high.

Example: Use strace to Trace Syscalls (1/2)

- prompt% strace wc production.log
  - execve("/usr/bin/wc", ["wc", "production.log"], [/* 52 vars */]) = 0
  - brk(0) = 0x1987000
  - mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8f7000
  - access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
  - open("/etc/ld.so.cache", O_RDONLY) = 3
  - fstat(3, {st_mode=S_IFREG|0644, st_size=137151, ...}) = 0
  - mmap(NULL, 137151, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff24b8d5000
  - close(3) = 0
  - open("/lib64/libc.so.6", O_RDONLY) = 3
  - read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360\355\241,0\0\0\0"..., 832) = 832
  - fstat(3, {st_mode=S_IFREG|0755, st_size=1922112, ...}) = 0
  - mmap(0x302ca00000, 3745960, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x302ca00000
  - mprotect(0x302cb89000, 2097152, PROT_NONE) = 0
  - mmap(0x302cd89000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x189000) = 0x302cd89000
  - mmap(0x302cd8e000, 18600, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x302cd8e000
  - close(3) = 0
  - mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8d4000
  - mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8d3000
  - mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8d2000
  - arch_prctl(ARCH_SET_FS, 0x7ff24b8d3700) = 0
  - mprotect(0x302cd89000, 16384, PROT_READ) = 0
  - mprotect(0x302c81f000, 4096, PROT_READ) = 0
  - munmap(0x7ff24b8d5000, 137151) = 0
  - brk(0) = 0x1987000
  - brk(0x19a8000) = 0x19a8000
  - …

Example: Use strace to Trace Syscalls (2/2)

- …
- open("/usr/lib/locale/locale-archive", O_RDONLY) = 3
- fstat(3, {st_mode=S_IFREG|0644, st_size=99158576, ...}) = 0
- mmap(NULL, 99158576, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff245a41000
- close(3)   = 0
- stat("production.log", {st_mode=S_IFREG|0644, st_size=526550, ...}) = 0
- open("production.log", O_RDONLY)   = 3
- read(3, "# Logfile created on Fri Dec 28 "..., 16384) = 16384
- open("/usr/lib64/gconv/gconv-modules.cache", O_RDONLY) = 4
- fstat(4, {st_mode=S_IFREG|0644, st_size=26060, ...}) = 0
- mmap(NULL, 26060, PROT_READ, MAP_SHARED, 4, 0) = 0x7ff24b8f0000
- close(4)   = 0
- read(3, "m: cannot remove `/tmp/fixrepo/g"..., 16384) = 16384
- read(3, "a36de93203e0b4972c1a3c81904e': P"..., 16384) = 16384
- read(3, "xrepo/git-tess/gitolite-admin/.g"..., 16384) = 16384
- Many repetitions of these reads
- read(3, "ixrepo/git-tess/gitolite-admin\n "..., 16384) = 16384
- read(3, "ite/redmine/vendor/plugins/redmi"..., 16384) = 16384
- read(3, "ited with positive recursionChec"..., 16384) = 16384
- read(3, "ting changes to gitolite-admin r"..., 16384) = 2262
- read(3, "", 16384)   = 0
- fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
- mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8ef000
- write(1, " 4704 28993 526550 production."..., 36 4704 28993 526550 production.log) = 36
- close(3)   = 0
  close(1)   = 0
- munmap(0x7ff24b8ef000, 4096)   = 0
- close(2)   = 0
- exit_group(0)



Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

return value	function name	parameters
`ssize_t`	`read`	`(int fd, void *buf, size_t count)`

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

Review 2.5

- Example of system call usage
- Example: Use strace to trace Syscalls
- Example of standard API

Advanced Operating Systems
2.OS Structure.6

Chun-Han Lin (林均翰)

CSIE, NTNU



Key Points 2.6

- **System call implementation**
- **API – system call – OS relationship**
- **System call parameter passing**

System Call Implementation (1/2)

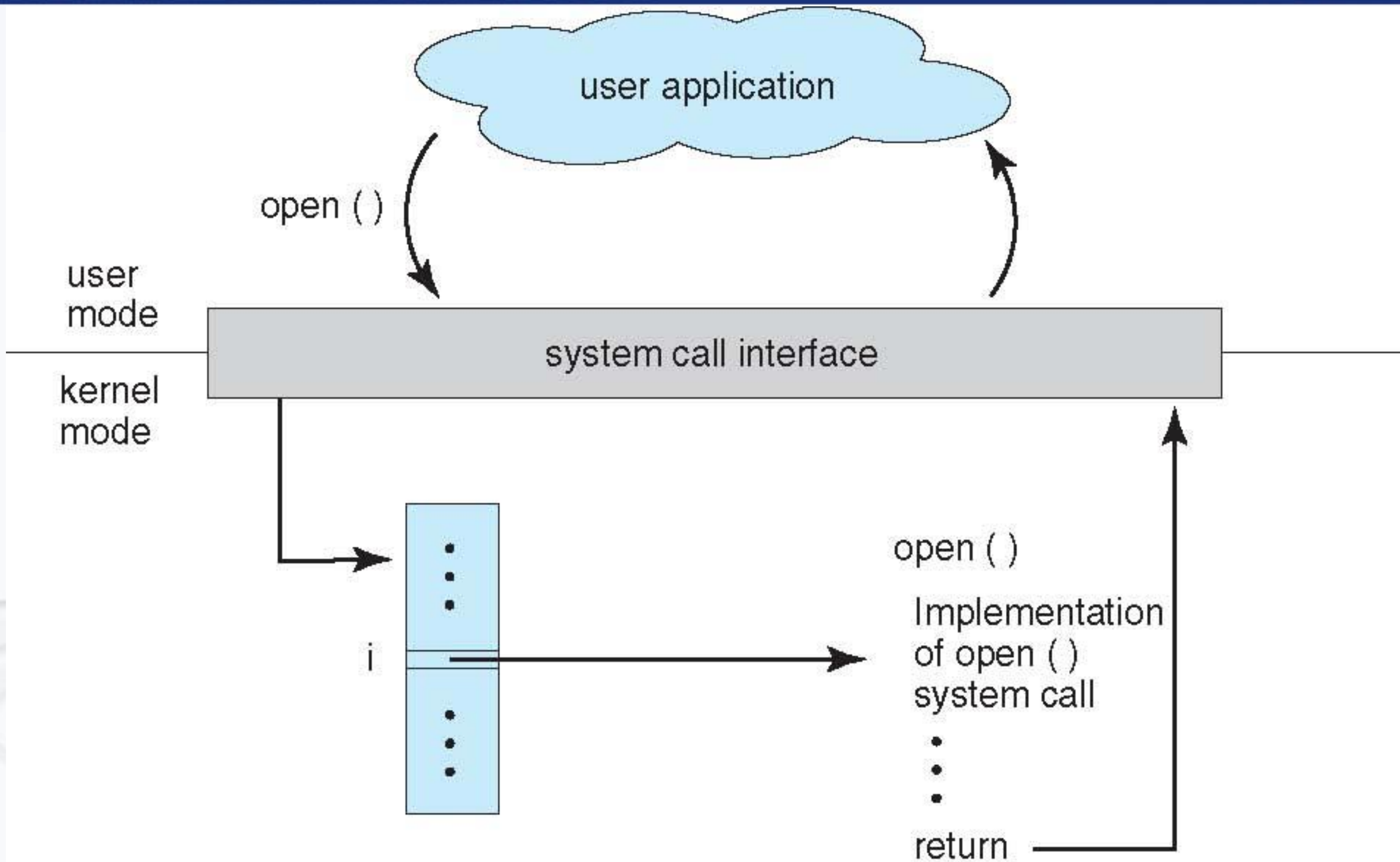
- Typically, a number is associated with each system call.
  - System-call interface maintains a table indexed according to these numbers.
  - The fact that the call is by “number”, is essential for security reasons!

- The system call interface invokes an intended system call in OS kernel and returns status of the system call and any return values.
  - Return value: often a long (integer)
    - Return of zero is usually a sign of success, but not always
    - Return of -1 is almost always reflects an error
  - On error – return code placed into global “errno” variable
    - Can translate into human-readable errors with the “perror()” call

System Call Implementation (2/2)

- The caller needs to know nothing about how the system call is implemented.
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface are hidden from programmers by API.
    - Managed by a run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship



System Call Parameter Passing (1/2)

- **Often, more information is required than a simple identity of the desired system call.**
  - **Exact type and amount of information vary according to OS and call**

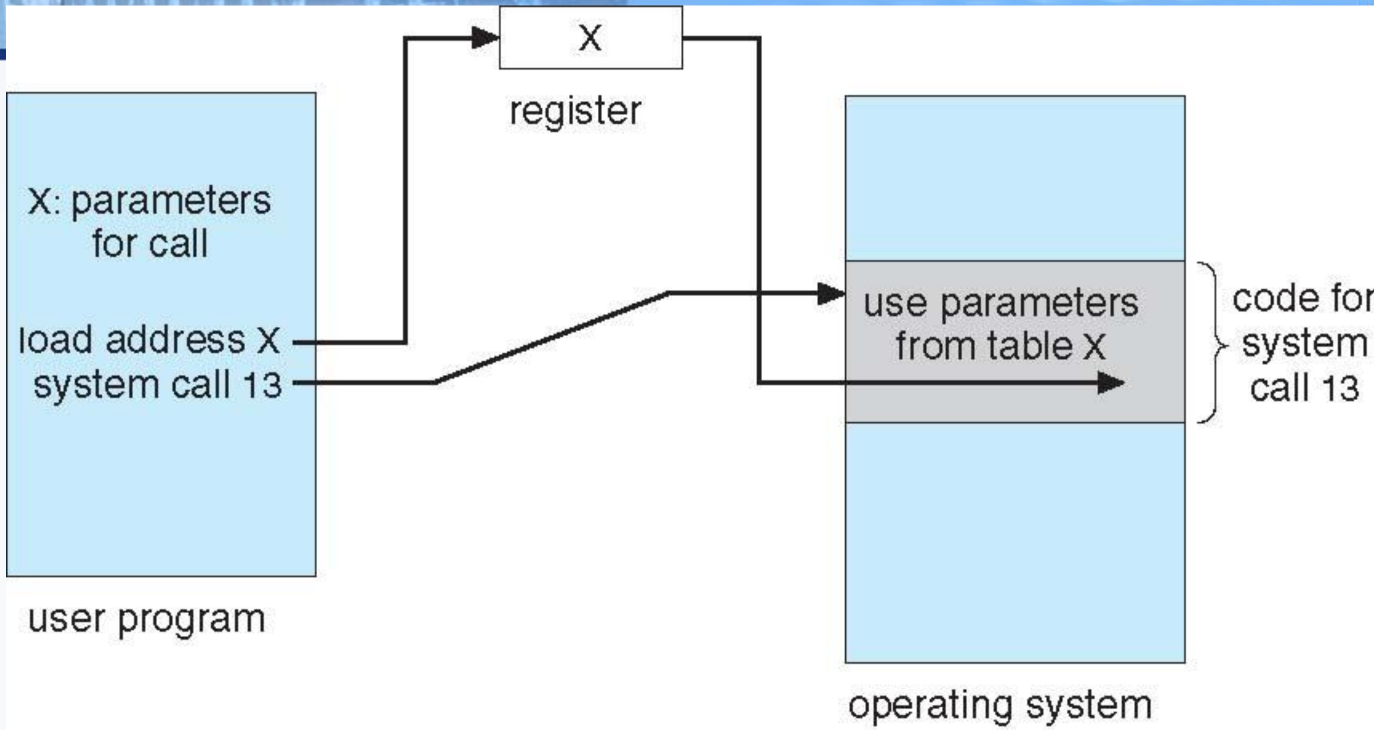
- **Three general methods are used to pass parameters to OS.**
  - **Simplest: pass the parameters in registers**
    - **In some cases, may be more parameters than registers**

System Call Parameter Passing (2/2)

- Parameters are stored in a block or table in memory, and the address of block is passed as a parameter in a register.
  - This approach is taken by Linux and Solaris.
- Parameters are placed, or pushed, onto the stack by the program and popped off the stack by an OS.
- Block and stack methods do not limit the number or length of parameters being passed.



Parameter Passing via Table



- **Kernel must always verify parameters passed to it by the user.**
  - **Are parameters in a reasonable range?**
  - **Are memory addresses actually owned by the calling user? (rather than bogus addresses)**

Review 2.6

- **System call implementation**
- **API – system call – OS relationship**
- **System call parameter passing**

Advanced Operating Systems
2.OS Structure.7

Chun-Han Lin (林均翰)

CSIE, NTNU

Key Points 2.7

- **Types of system calls**

Types of System Calls (1/3)

- **Process control**
  - End, abort
  - Load, execute
  - Create process, terminate process
  - Get process attributes, set process attributes
  - Wait for time
  - Wait event, signal event
  - Allocate and free memory

  - Dump memory if error
  - Debugger for determining bugs, single step execution
  - Locks for managing access to shared data between processes

Types of System Calls (2/3)

- **File management**
  - Create file, delete file
  - Open, close file
  - Read, write, reposition
  - Get and set file attributes
- **Device management**
  - Request device, release device
  - Read, write, reposition
  - Get and set device attributes
  - Logically attach or detach devices
- **Information maintenance**
  - Get and set time or date
  - Get and set system data
  - Get and set process, file, or device attributes

Types of System Calls (3/3)

- Communications
  - Create, delete communication connection
  - Send and receive messages if message passing model to host name or process name
    - From client to server
  - Shared-memory model create and gain access to memory regions
  - Transfer status information
  - Attach and detach remote devices
- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Review 2.7

- **Types of system calls**