Ch. 02 Operating-System Structures

Chanho Lee
Soongsil University

1

Objectives

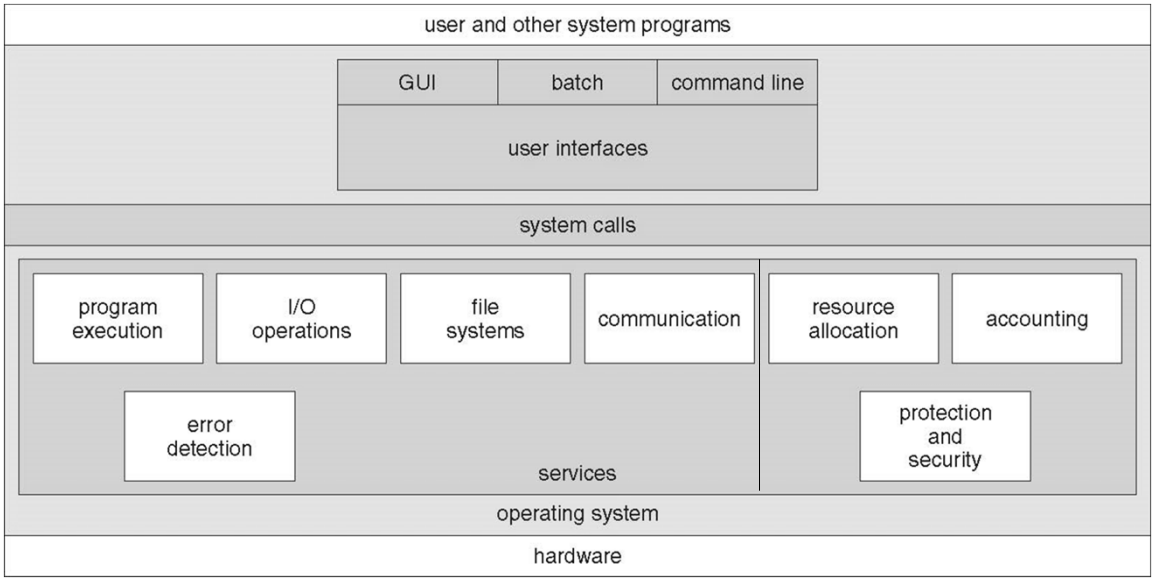
- Services of an OS
- System calls
- Various ways of structuring an OS
- OS installation, customization and booting
- Monitoring OS performance

2

3주차 Services, interface and system calls

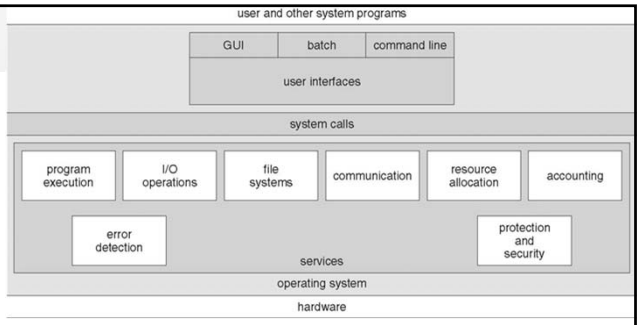
3

2.1 Operating-System Services



4

2.1 Operating-System Services

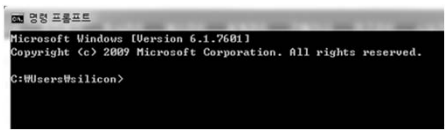


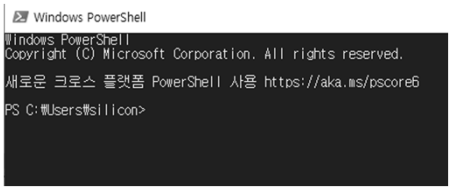
- Services
  - ▶Program execution
  - ▶I/O operations
    - KBD/mouse/touch, network, display, printer
  - ▶File-system manipulation
    - read and write, create and delete, and so on
  - ▶Communications
    - IPC(Inter-Process Communication): shared memory, message passing
  - ▶Error detection
    - HW failure, I/O error, user program error → Error handling
  - ▶Resource allocation
  - ▶Logging: tracking records for accounting or statistics
  - ▶Protection and security: authorization and authentication

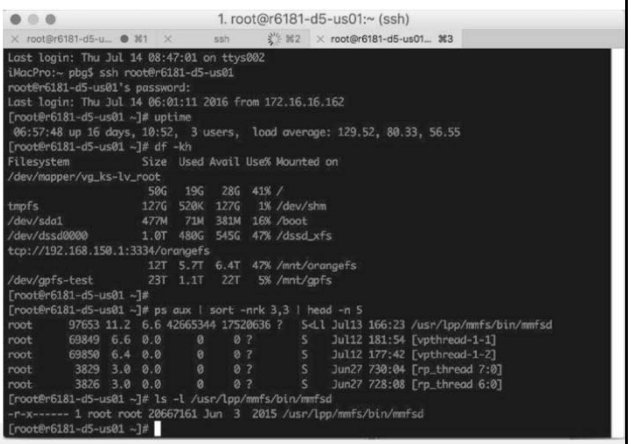
5

2.2 User and Operating-System Interface

- Command Interpreters
  - ▶get and execute the user-specified command: usually file manipulation including execution
  - ▶commands are self-containing (Windows) or running system programs (UNIX/Linux)
  - ▶Command Prompt/PowerShell(Windows),
  - ▶C shell(csh), Bourne-Again shell(bash), Korn shell(Ksh), and others (UNIX/Linux)







6

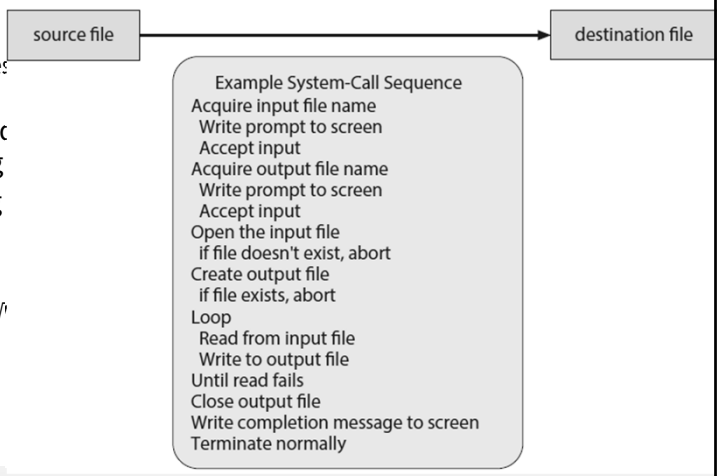
2.2 User and Operating-System Interface

- Graphical User Interface
  - ►Xerox Alto computer in 1973
  - ►Apple Macintosh computers in the 1980s
  - ►K Desktop Environment (or KDE) and the GNOME desktop by the GNU project (UNIX/Linux)
- Touch-Screen Interface
  - ►touch and gestures
- Choice of Interface
  - ►personal preference
  - ►GUI and touch-screen interface
  - ►Power users: CLI using shell scripts

7

2.3 System Calls

- Interface to the services
  - ►functions written in C and C++ or assembly language for certain tasks.
- Example: file copy
  - ►Obtaining file names
    - cp in.txt out.txt : part of the command
    - Program asks: typed by the user: requires characters, or windows pop-up)
  - ►Opening and reading the input file, and another system call, and error handling
  - ►Loop for reading and writing, checking
  - ►Closing files
  - ►Display messages
  - ►All of the above requires system calls √



8

2.3 System Calls

- Application Programming Interface (API)
  - ▶specifies a set of available functions including the parameters and the return values
  - ▶Windows API for Windows, the POSIX API for POSIX-based systems (UNIX, Linux, and macOS), and the Java API for the Java virtual machine
  - ▶example of a standard API, `read()` function

```
#include <unistd.h>
```

number of bytes read,0,-1 file descriptor Data buffer max number of bytes (buffer)

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

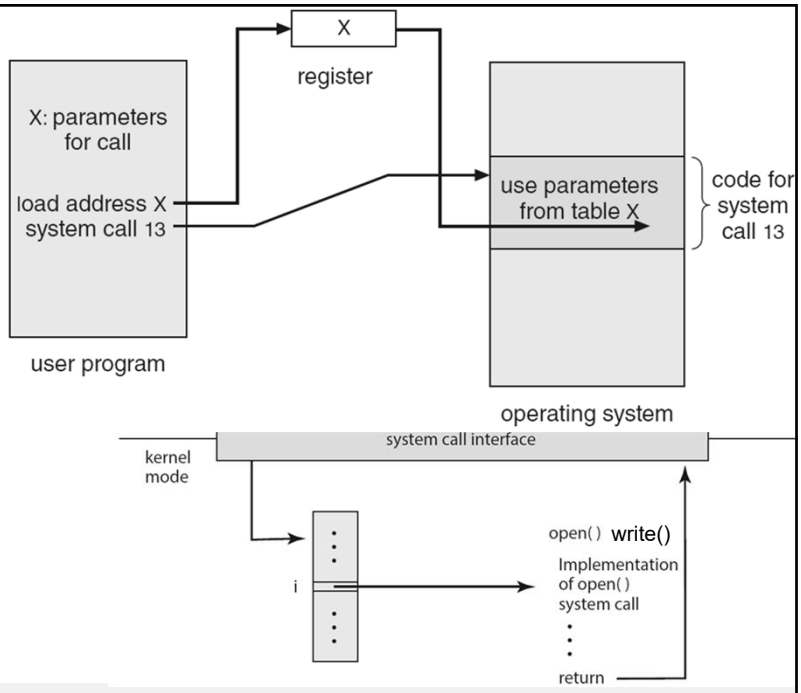
return value | function name | parameters

  - ▶API function invokes system calls
    - Ex. Windows function CreateProcess() → NTCreateProcess() system call
  - ▶Increases portability

9

2.3 System Calls

- Run-time environment (RTE)
  - ▶full suite of software needed to ex
  - ▶provides a system-call interface:
  - ▶the caller need only obey the API details are hidden and managed b
  - ▶Passing of parameters √
    - Through registers or
    - Address of a block in memory
      - – Linux: <=5: registers
        >5: block method
    - Push and pop from a stack



10

2.3 System Calls: Types of System Calls

- 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

11

2.3 System Calls: Types of System Calls

- 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

12

2.3 System Calls: Types of System Calls

- EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix
Process control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File management	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device management	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communications	CreatePipe()	pipe()
	CreateFileMapping()	shm_open()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

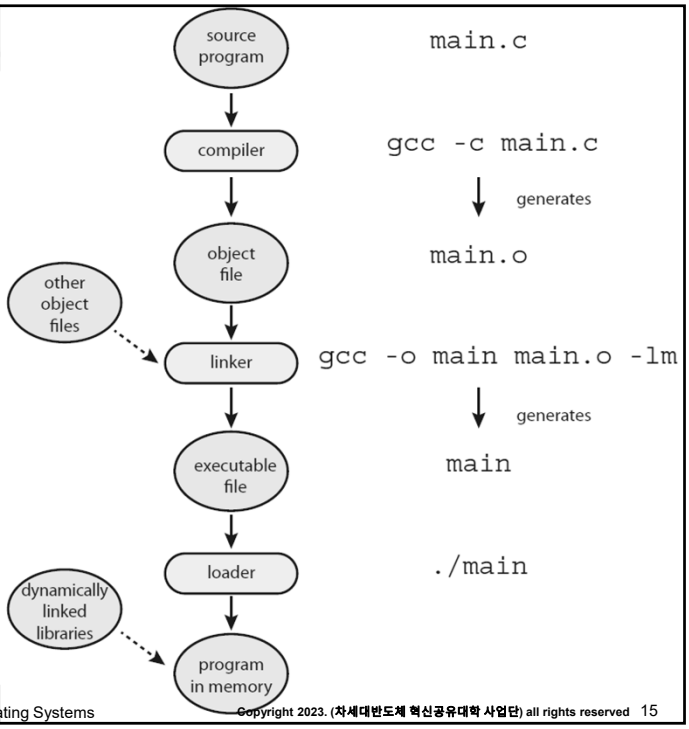
13

3주차 Linker/Loader, OS structure

14

2.5 Linkers and Loaders

- Compiler: Relocatable object file
  - ▶Relative addresses and relocatable
- Linker: single binary executable file
- Loader: load the binary executable file into memory
  1. creates a new process: fork()
  2. invokes the loader: exec()
- Relocation: linking and loading
  - ▶assigns final addresses to the program parts
  - ▶adjusts code and data in the program to match those addresses
- dynamically linked libraries (DLLs)
  - ▶Windows
  - ▶linker inserts relocation information instead of libraries



15

2.5 Linkers and Loaders

- Object files and executable files
  - ▶Typically standard formats : ELF (Executable and Linkable Format) for UNIX and Linux
  - ▶include the compiled machine code and a symbol table containing metadata about functions and variables
  - ▶ELF relocatable and executable file
  - ▶ELF executable file: entry point
    - address of the first instruction to be executed
  - ▶Portable Executable (PE) format: Windows
  - ▶Mach-O format: macOS

16

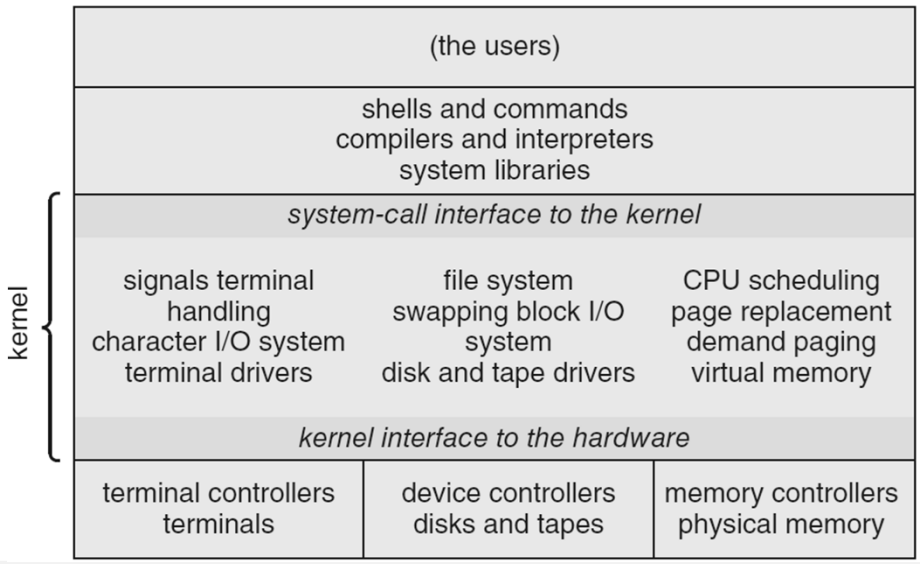
2.6 Why Applications Are Operating-System Specific

- unique set of system calls and so on
- Applications available to run on multiple operating systems
  - ►interpreted language (such as Python):
    - reads each line,
    - executes equivalent instructions on the native instruction set, and
    - calls native operating system calls
    - Performance problem
  - ►language that includes a virtual machine containing the running application
    - virtual machine is part of the language's full RTE: Ex. Java
    - RTE porting to many OSes
    - Performance problem
  - ►use a standard language or API
    - the compiler generates binaries in a machine- and operating-system specific language.
    - ported to each operating system
    - Time-consuming

17

2.8 Operating-System Structure

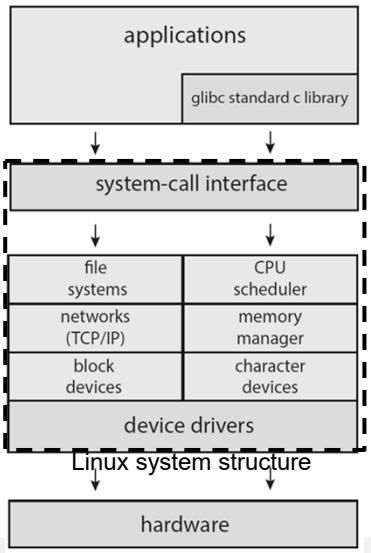
- Traditional UNIX system structure.



18

2.8 Operating-System Structure

- Monolithic Structure
  - ►no structure: place all of the functionality of the kernel into a single, static binary file
  - ►Ex. original UNIX
    - the kernel and the system programs
    - Layered kernel: modular design (modified during run time); monolithic (in a single address space)
    - Difficult to implement and extend
    - Distinct performance advantage: very little overhead in the system-call interface, and communication within the kernel is fast
  - ►tightly coupled system : changes to one part of the system can have wide-ranging effects on other parts

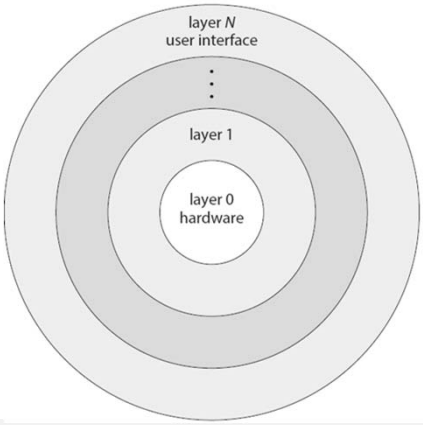


Linux system structure

19

2.8 Operating-System Structure

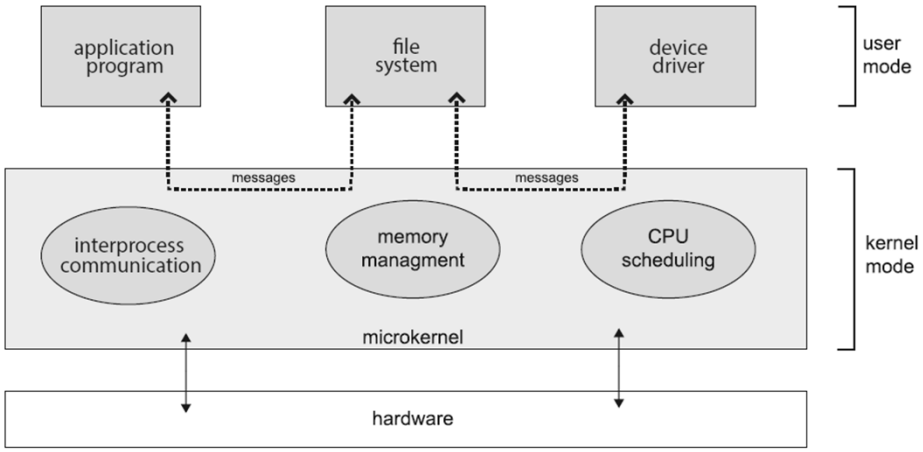
- Layered Approach
  - ►loosely coupled
  - ►separate, smaller components that have specific and limited functionality
  - ►consists of data structures and a set of functions that can be invoked by higher-level layers
  - ►simplicity of construction and debugging: layer by layer
  - ►EX. computer networks (such as TCP/IP) and web applications
  - ►challenges of appropriately defining the functionality of each layer
  - ►poor performance due to the overhead to traverse through multiple layers → small number of layers



20

2.8 Operating-System Structure

- Microkernels
  - ▶Removing all nonessential components
    → small kernel + user level programs in separate address spaces
  - ▶Easy to extending the operating system: new services in user space
  - ▶Not likely, but easy to modify kernels
  - ▶Secure and reliable
  - ▶Ex. Mach: NextStep



21

2.8 Operating-System Structure

- Modules
  - ▶loadable kernel modules (LKMs): the best current methodology
    - UNIX, such as Linux, macOS, and Solaris, as well as Windows
  - ▶kernel has a set of core components and
  - ▶can link in additional services via modules, either at boot time or during run time
  - ▶Idea: the kernel provides core services, while other services are implemented dynamically
  - ▶resembles a layered system: each kernel section has defined, protected interfaces; but it is more flexible: any module can call any other module
  - ▶similar to the microkernel approach: the primary module has only core functions and knowledge of how to load and communicate with other modules; but more efficient: modules do not need to invoke message passing
  - ▶Ex. Linux
    - LKMs can be loaded and removed at anytime
    - allow a dynamic and modular kernel, while maintaining the performance of a monolithic system

22

2.8 Operating-System Structure

- Hybrid Systems
  - ►combine different structures: performance, security, and usability issues
  - ►Ex. Linux: monolithic + modular
  - ►Ex. Windows: largely monolithic + some typical behavior of microkernel + dynamically loadable kernel modules

23

2.9 Building and Booting an Operating System

- System Boot
  1. BIOS/UEFI
  2. Bootstrap program or boot loader locates the kernel: boot manager, GRUB
  3. The kernel is loaded into memory and started.
  4. The kernel initializes hardware.
  5. The root file system is mounted.

24

Exercises, problems and projects

- Exercises
  - ►2.1, 2.3, 2.7, 2.8
  - ►2.10, 2.12

2.10 Describe three general methods for passing parameters to the operating system.
2.12 What are the advantages and disadvantages of using the same system call interface for manipulating both files and devices?

- Problems
  - ►2.24
  - ►FileCopy.c: copy 10 integers per loop from in.txt to out.txt
  - ►in.txt: 1~100, 10 integers per line, 10 lines

2.24 In Section 2.3, we described a program that copies the contents of one file to a destination file. This program works by first prompting the user for the name of the source and destination files. Write this program using the POSIX. Be sure to include all necessary error checking, including ensuring that the source file exists. Once you have correctly designed and tested the program, if you used a system that supports it, run the program using a utility that traces system calls. Linux systems provide the `strace` utility. These tools can be used as follows (assume that the name of the executable file is `FileCopy`):

Linux:

```
strace ./FileCopy
```


25

Summary

- OS: provides services for the execution of programs
- Interacting with an OS: CLI, GUI, and touchscreen
- System calls: an interface to the services by an OS. Use API.
- System calls: (1) process control, (2) file management, (3) device management, (4) information maintenance, (5) communications, and (6) protection.
- The standard C library provides the system-call interface for UNIX and Linux
- System programs in OS provide utilities to users.
- A linker: relocatable object modules → a single binary executable file.
- A loader loads the executable file into memory: eligible to run
- Applications are OS specific: binary formats, instruction sets, system calls
- OS structure: monolithic, layered, microkernel, modular

26