

Operating system structures

Tran, Van Hoai

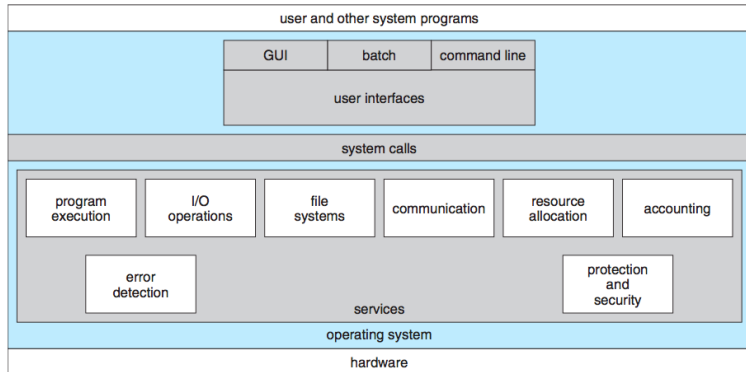
Faculty of Computer Science & Engineering
HCMC University of Technology

E-mail: hoai@hcmut.edu.vn
(partly based on slides of Le Thanh Van)

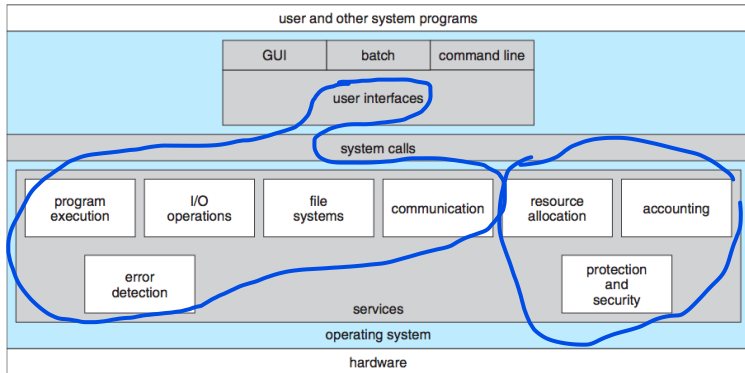
- 1 Operating system services
- 2 System calls and programs
- 3 Operating system structure
- 4 Advanced issues

- 1** Operating system services
- 2 System calls and programs
- 3 Operating system structure
- 4 Advanced issues

Operating system services



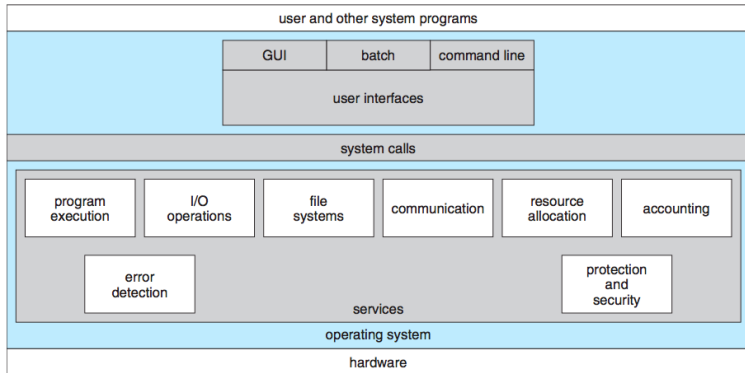
Operating system services



■ Functional services

■ Non-functional services

Operating system services



■ Functional services

- User interface (Graphical User Interface, Batch Interface, Command Line Interface),
- Program execution, I/O operations,
- File-system manipulation,
- Communications, Error detection

■ Non-functional services

- Resource allocation,
- Accounting, Protection and Security

What is a program ?

A computer program is a collection of instructions that performs a specific task when executed by a computer.

What is a program ?

A computer program is a collection of instructions that performs a specific task when executed by a computer.

- OS is able to **load** a **program** into memory and to **run** that program
- The program is able to **end** its execution (either normally or abnormally)

I/O operations

- A running program may access I/O devices, e.g., recording DVD)
- For efficiency and protection, OS must provide a means (for programs) to do I/O

Functional services

I/O operations & File-system manipulation

I/O operations

- A running program may access I/O devices, e.g., recording DVD)
- For **efficiency and protection**, OS must provide a means (for programs) to do I/O

File-system manipulation

- Programs need operations on files/directories: list, create, delete, read, write, permission management

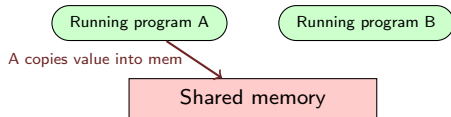
- There are many communications
 - between running programs on a machine
 - between running program on different machines
- There are two main types of communications
 - **shared memory**: read/write on a shared of memory
 - **message passing**: packets with **predefined formats** are exchanged between running programs

Running program A

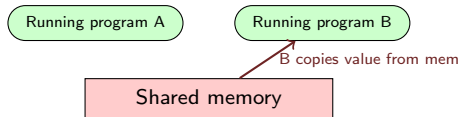
Running program B

Shared memory

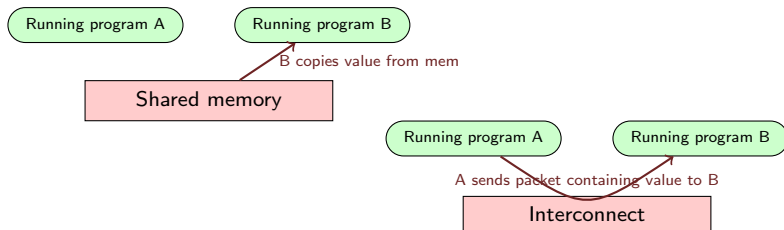
- There are many communications
 - between running programs on a machine
 - between running program on different machines
- There are two main types of communications
 - **shared memory**: read/write on a shared of memory
 - **message passing**: packets with **predefined formats** are exchanged between running programs



- There are many communications
 - between running programs on a machine
 - between running program on different machines
- There are two main types of communications
 - **shared memory**: read/write on a shared of memory
 - **message passing**: packets with **predefined formats** are exchanged between running programs



- There are many communications
 - between running programs on a machine
 - between running program on different machines
- There are two main types of communications
 - **shared memory**: read/write on a shared of memory
 - **message passing**: packets with **predefined formats** are exchanged between running programs



- Errors can occur everywhere
 - Hardware: CPU, memory, I/O devices,...
 - User program: arithmetic overflow, illegal access of memory, division by zero,...
- OS should take appropriate actions to detect and correct errors constantly.

Non-functional services

Resource allocation

- OS manages multiple resources (hardware, software) and allocates them to **multiple users** and **multiple running programs**
- Special codes are needed to make allocation efficiently, e.g., CPU scheduling, printers allocation

Accounting

- Recording which users to use how much and what kind of resources
- Usage statistics is useful to **reconfigure** for improvement of computing services

Protection and security

- Information of multiple users on a networked computer should be controlled by its owner.

User interface

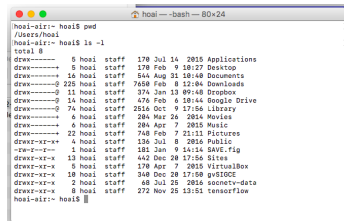
- **Command interpreter** - a special program to allows users to directly enter commands (or programs) (by text) to be performed by the operating system

- Multiple command interpreters (also **shell**) in a modern operating system

Example: Bourne shell,

Bash shell, C shell,

Bourne-Again shell, Korn shell



```
hoai~ -bash -- 80x24
hoai-sir:- hoai$ pwd
/Users/hoai
hoai-sir:- hoai$ ls -l
total 8
drwx-----  5 hoai  staff   178 Jul 14  2015 Applications
drwx-----  5 hoai  staff   178 Feb  9 18:27 Desktop
drwx----- 16 hoai  staff   544 Aug 31 18:48 Documents
drwx----- 225 hoai  staff  7658 Feb  8 12:04 Downloads
drwx-----  2 hoai  staff   374 Jan 13 09:48 Dropbox
drwx-----  2 hoai  staff   476 Feb  6 19:44 Google Drive
drwx-----  74 hoai  staff  2514 Oct  9 17:56 Library
drwx-----  6 hoai  staff   284 Mar 26  2014 Movies
drwx-----  6 hoai  staff   284 Apr  7  2015 Music
drwx----- 22 hoai  staff   748 Feb  7 21:11 Pictures
drwxr-xr-x  4 hoai  staff   136 Jul  8  2016 Public
-rw-r--r--  1 hoai  staff   181 Jan  9 14:14 SAVE.fig
drwxr-xr-x 13 hoai  staff   442 Dec 20 17:56 Sites
drwxr-xr-x  5 hoai  staff   128 Apr  7  2015 VirtualBox
drwxr-xr-x 18 hoai  staff   348 Dec 28 17:58 gvSIGICE
drwxr-xr-x  2 hoai  staff    68 Jul 25  2016 secretv-data
drwxr-xr-x  8 hoai  staff   272 Nov 25 13:51 tensorflow
hoai-sir:- hoai$
```

User interface

- **Command interpreter** - a special program to allows users to directly enter commands (or programs) (**by text**) to be performed by the operating system

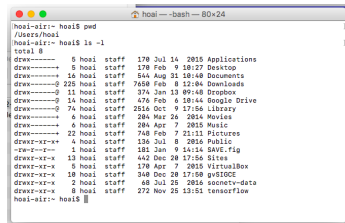
- Multiple command interpreters (also **shell**) in a modern operating system

Example: Bourne shell,

Bash shell, C shell,

Bourne-Again shell, Korn shell

- **GUI** - a user friendly graphical interface, input/output is performed in a **more interactive** way



```
hoai-sir:- hoai$ pwd
/Users/hoai
hoai-sir:- hoai$ ls -l
total 8
drwx----- 5 hoai staff 178 Jul 14 2015 Applications
drwx----- 5 hoai staff 178 Feb  9 18:27 Desktop
drwx----- 16 hoai staff 544 Aug 31 18:48 Documents
drwx----- 225 hoai staff 7658 Feb  8 12:04 Downloads
drwx----- 11 hoai staff 374 Jan 13 09:48 Dropbox
drwx----- 14 hoai staff 476 Feb  6 19:44 Google Drive
drwx----- 74 hoai staff 2514 Oct  9 17:56 Library
drwx----- 6 hoai staff 284 Mar 26 2014 Movies
drwx----- 6 hoai staff 284 Apr  7 2015 Music
drwx----- 22 hoai staff 748 Feb  7 21:11 Pictures
drwxr-xr-x  4 hoai staff 136 Jul  8 2016 Public
-rw-r--r--  1 hoai staff 181 Jan  9 14:14 SAVE.fig
drwxr-xr-x 13 hoai staff 442 Dec 20 17:56 Sites
drwxr-xr-x  5 hoai staff 128 Apr  7 2015 VirtualBox
drwxr-xr-x 18 hoai staff 348 Dec 28 17:58 gvSIGCE
drwxr-xr-x  2 hoai staff 68 Jul 25 2016 secretv-data
drwxr-xr-x  8 hoai staff 272 Nov 25 13:51 tensorflow
hoai-sir:- hoai$
```

Interface choice

Kind of interface (CLI or GUI) is mostly one of **personal preference**

- System administrators or power users often prefer CLI
 - With deep system knowledge, using CLI is **more efficient and secure**
 - **Multiple** shell commands can be combined into a program, called **shell script** to perform more complex tasks.
- Normal users often choose GUI

Interface choice

Kind of interface (CLI or GUI) is mostly one of **personal preference**

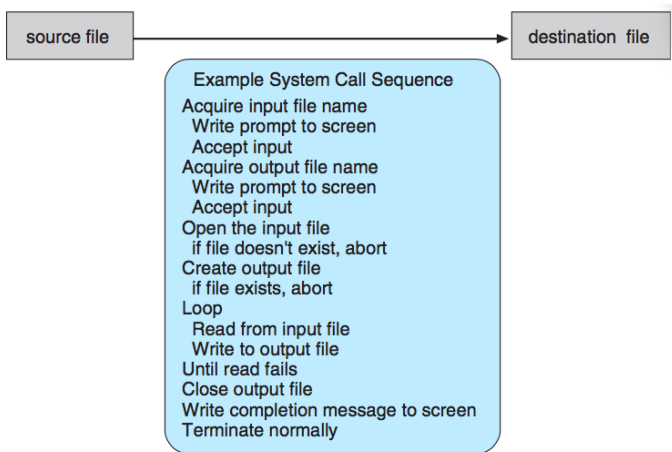
- System administrators or power users often prefer CLI
 - With deep system knowledge, using CLI is **more efficient and secure**
 - **Multiple** shell commands can be combined into a program, called **shell script** to perform more complex tasks.
- Normal users often choose GUI

UI can vary from system to system. Therefore, it is **not a direct function** of operating system.

- 1 Operating system services
- 2 System calls and programs**
- 3 Operating system structure
- 4 Advanced issues

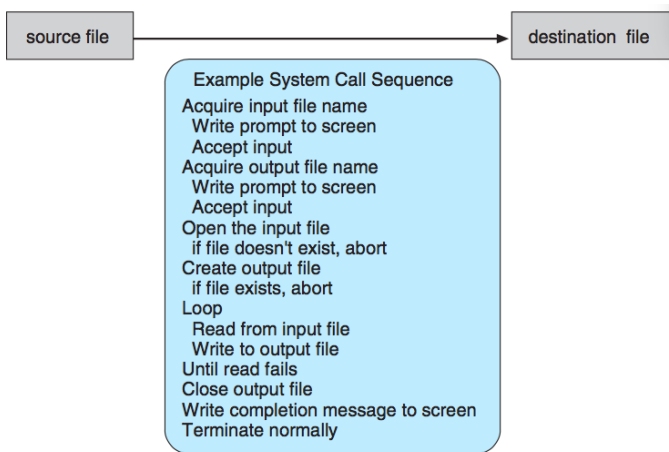
Example on system call demand

File-to-file copy



Example on system call demand

File-to-file copy



Even a simple program executes **many** system calls

API vs. system call interface

Application Programming Interface (API)

Set of functions **available to application programmers**

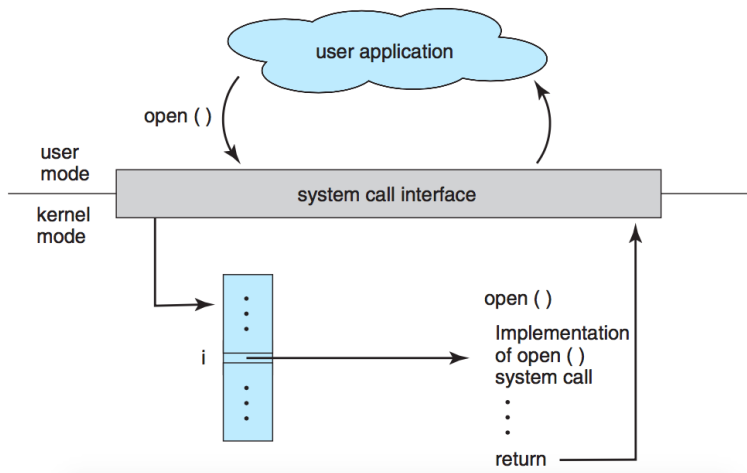
- Using API to increase **program portability**: ability to compile and run on systems with the same API **without code modification**
- Working with API is easier than with actual system calls
- 3 most common APIs: Windows API, POSIX API, Java API

System-call interface

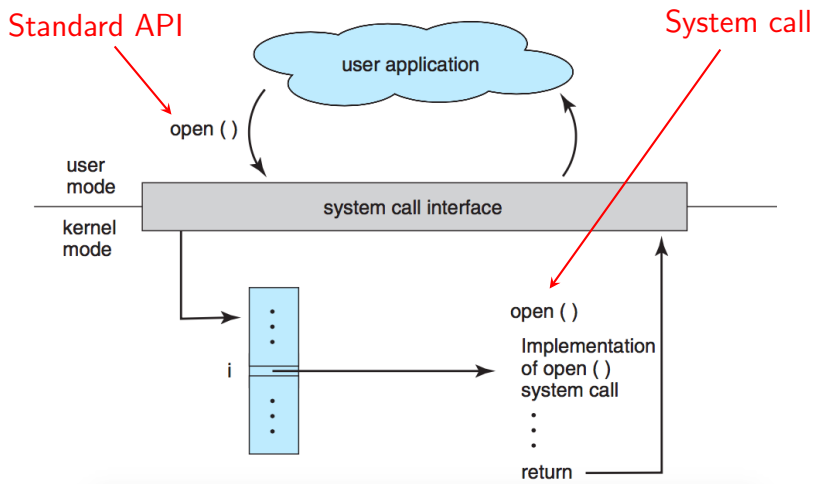
A run-time support system (e.g., `libc`) serves as a link between API and system calls in operating system

- Each system call is identified by **an index number**

A call to system call `open()`



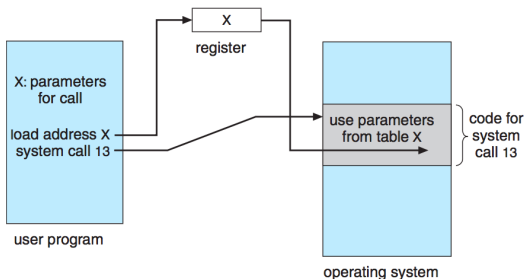
A call to system call `open()`



Parameters passing to system call

There are 3 ways.

- By registers: **not enough** for large parameters
- Stored in block (or table), passing address of the block in parameter



- Pushed onto program's **stack**, and popped off by OS

Types of system calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

What is a program ?

A computer program is a collection of instructions that performs a specific task when executed by a computer.

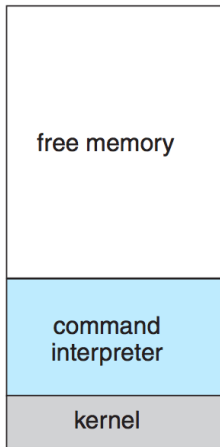
What is a process ?

A program **loaded into memory** and **executing** is called a **process**

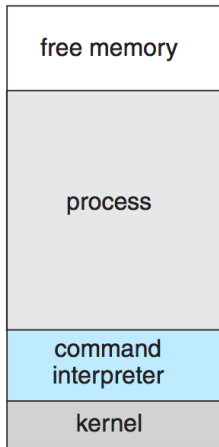
- `end()`, `abort()`
- `load()`, `execute()`
- `create_process()`,
`terminate_process()`
- `get_process_attributes()`,
`set_process_attributes()`
- `wait_for_time()`
- `wait_event()`, `signal_event()`
- allocate and free memory

Execution on single-task OS

MS-DOS



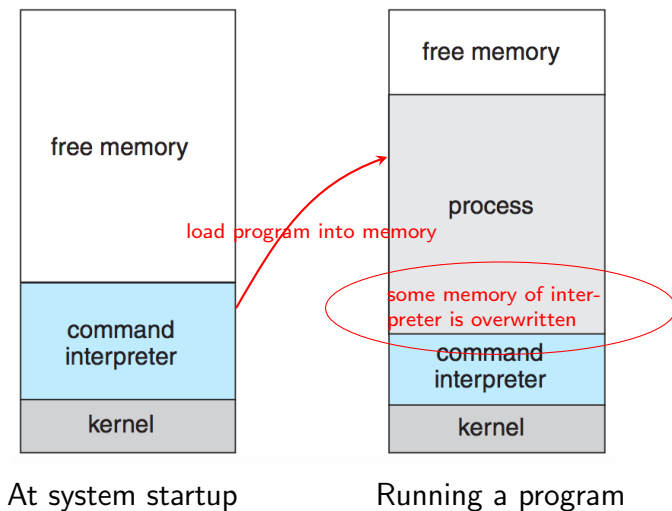
At system startup



Running a program

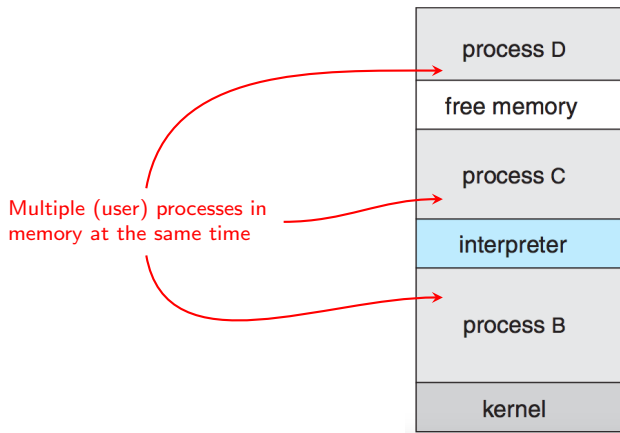
Execution on single-task OS

MS-DOS



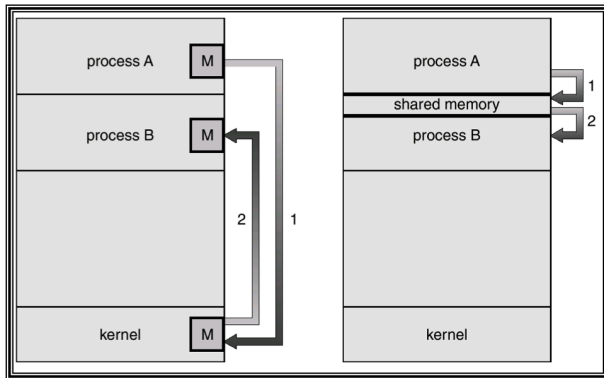
Execution on multiple-task OS

FreeBSD



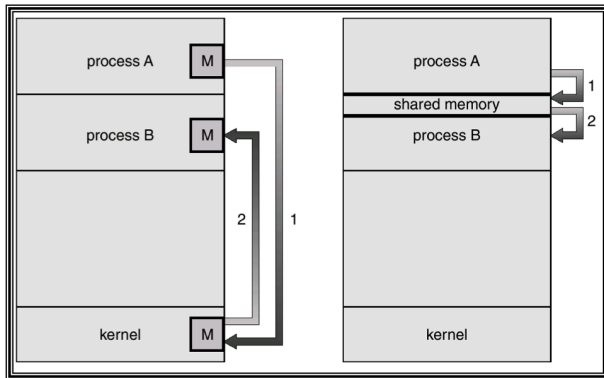
Interprocess communication

- **Interprocess** communication may take place using either message passing or shared memory



Interprocess communication

- **Interprocess** communication may take place using either message passing or shared memory



Message passing

Shared memory

System programs = system utilities

Providing a **convenient environment** for program development and execution

There are categories as follows.

- File manipulation
- Status information
- File modification
- Programming-language support
- Program loading and execution
- Communications
- Background services

System programs

System programs = system utilities

Providing a **convenient environment** for program development and execution

There are categories as follows.

- File manipulation
- Status information
- File modification
- Programming-language support
- Program loading and execution
- Communications
- Background services

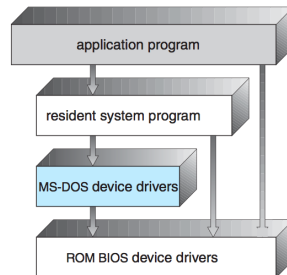
The view of most users on the operating system is defined by system programs, not actual system calls.

- 1 Operating system services
- 2 System calls and programs
- 3 Operating system structure**
- 4 Advanced issues

Simple structure

MS-DOS

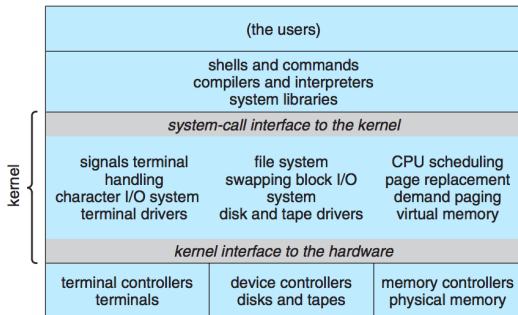
- MS-DOS: written to provide the most functionality in the **least** space
 - **not divided** into modules
 - interfaces and levels of functionality are **not well separated**



Simple structure

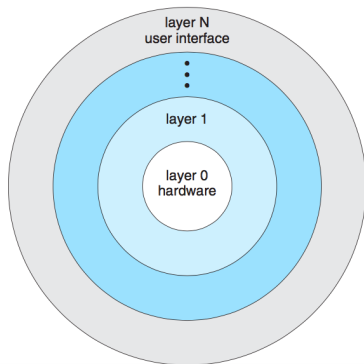
Traditional UNIX

- Limited structure with limited functionality
- Consists of 2 parts: kernel and system programs



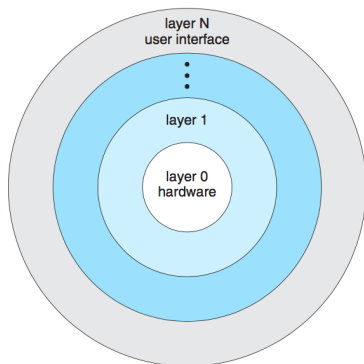
- Considered to be layered to some extent: system-call interface, kernel and hardware interface
- **Enormous amount** of functionality combined in one level → **monolithic** structure

Layered approach



- OS is broken into several layers
- Layer M provides data structures & routines for upper layer, and can invoke operations of lower layer

Layered approach



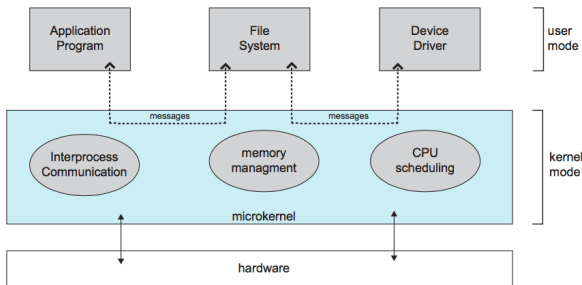
- OS is broken into several layers
- Layer M provides data structures & routines for upper layer, and can invoke operations of lower layer

Advantages (due to **modularity**): **easy to debug**; **simple in design and implementation**; **secure**

Disadvantages: **Not easy to define layers** (which layer is above/below which layer); **inefficiency**

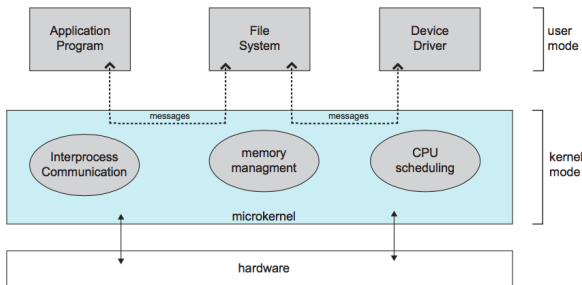
Microkernel structure

- Moves as much from the kernel into “user” space
- Communication takes place between user modules using **message passing**



Microkernel structure

- Moves as much from the kernel into “user” space
- Communication takes place between user modules using **message passing**

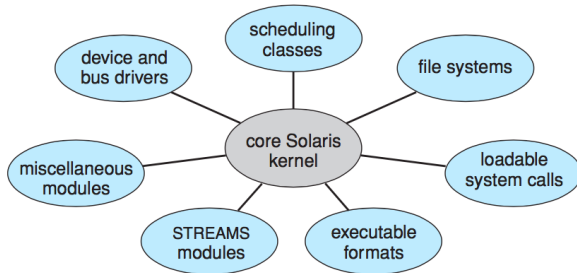


Advantages (due to **modularity**): **easy to extend kernel**; **easier to port OS to new architectures**; **more reliable** (less code running in kernel); **more secure**

Disadvantages: **system-function overhead**

Modules-based structure

- Best current methodology is **loadable kernel modules**
 - Only need **core services**
 - Additional services can be loaded as modules in boot time and run time
 - Additional services have to be **recompiled** to add new features
- Idea of modules-based kernel more flexible than layered approach and also similar to microkernel structure



Hybrid structure

- In practice, very few OS adopt a single, strict defined structure

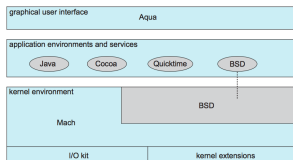


Figure 2.16 The Mac OS X structure.



Figure 2.17 Architecture of Apple's iOS.

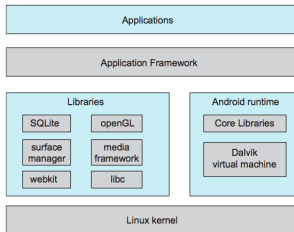


Figure 2.18 Architecture of Google's Android.

Outline

- 1 Operating system services
- 2 System calls and programs
- 3 Operating system structure
- 4 Advanced issues**