

# Operating system

## Part II: Problems and Ideas in OS

By **KONG** LingBo (孔令波)

# Outline of topics covered by this course

- **INTRODUCTION** – why should we learn OS?
- **OVERVIEW** – what problems are considered by modern OS in more details?
  - And fundamental ideas to construct a modern OS?
- **EXECUTION** – CPU management
  - Process and Thread
  - CPU scheduling
- **EXECUTION** – competition (synchronization problem)
  - Synchronization
  - Deadlock

- Problems and ideas when implementing OS
  - Problems: EMM+GSD
    - Concurrency, Resource limitation, Security, GUI, ...
  - Ideas: Multiplex [复用] resource + Synchronize [同步] programs
    - **Multiplexing** resources based on interrupt mechanism
    - **Synchronize** the access of the limited resource among the related programs
- Evolution of OSs' structures
  - Construct OS with so many functions
- System Call/API
  - How to use the functions defined in OS?
- How to load and run OS?

# How to understand the problems in OS?

- Same as in other courses/theories, we could start from **SIMPLE** situation
  - 1:1:M is only for one program whose size is smaller than that of the available space in MM
    - “the available” here is because “OS also needs some MM space”
- EMM indicates the common problems: HOW
  - **EXECUTION** → CPU
    - How to configure CPU (set instruction and parameter to registers) to run your program?
      - You’ve understood this through CO ([Computer Organization: 计算机组成原理](#)) course (☺ I hope)
  - **MAPPING 2** → from File to Hard disk space
    - Your program is first represented as files in hard disk (usually). How to organize many files?
  - **MAPPING 1** → from File to Main Memory space
    - The fact from von Neumann gives that your programs should be copied into MM space

However, OS is more complicated

- **Complexity is caused by the resource limitation**, just as we meet in our real life
  - EMM is still the hint to understand the complicated situation
- Mapping 1 → “**LARGE PROGRAM V.S SMALL MM**” problem
  - 8 GB now is popular, but

[《刺客信条：起源》文件大小曝光：42.3GB\\_游民星空...](https://www.gamersky.com/news/201710/963237.shtml)

游戏发行: Ubisoft

2018-6-5 · 进入10月,《刺客信条：起源》的发售日就开始进入倒计时了。如果你正打算为它来清理下硬盘空间,那么可以参考下曝光的文见尺寸。

<https://www.gamersky.com/news/201710/963237.shtml> ▼



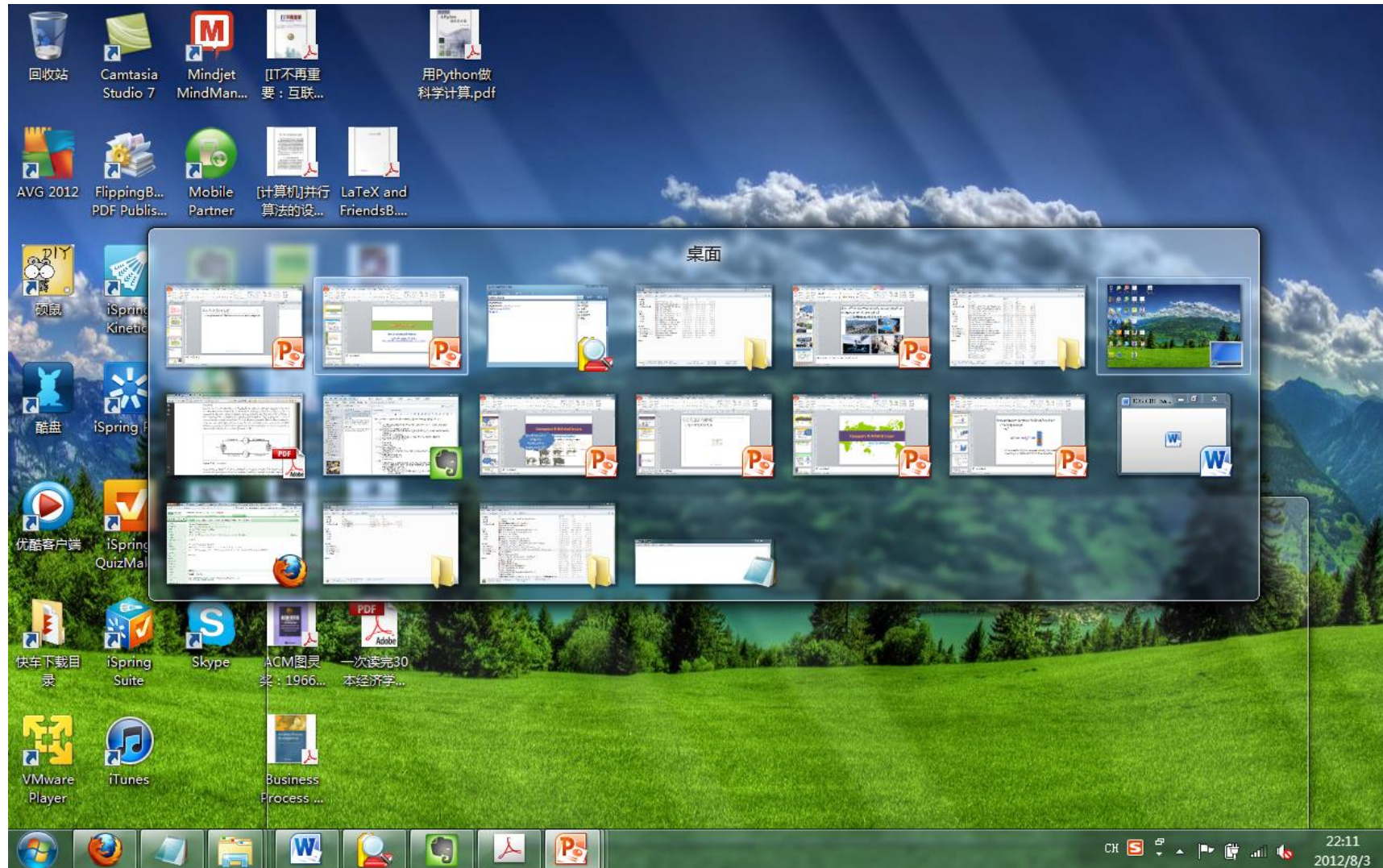
# Modern OS is more complicated



- And, the concurrency makes OS more complicated.
  - By **CONCURRENCY**, it means
    - **MANY** programs are running **IN A PERIOD**; but **ONLY ONE** is running **AT A TIME**!
    - You’ve benefited from this by using modern OSs – Windows, Ubuntu, Mac OS
      - You can listen to some music, “**at same time**”, you can edit some documents, maybe you can also read some materials for references
  - **Concurrency is now the indispensable property of modern OSs!**



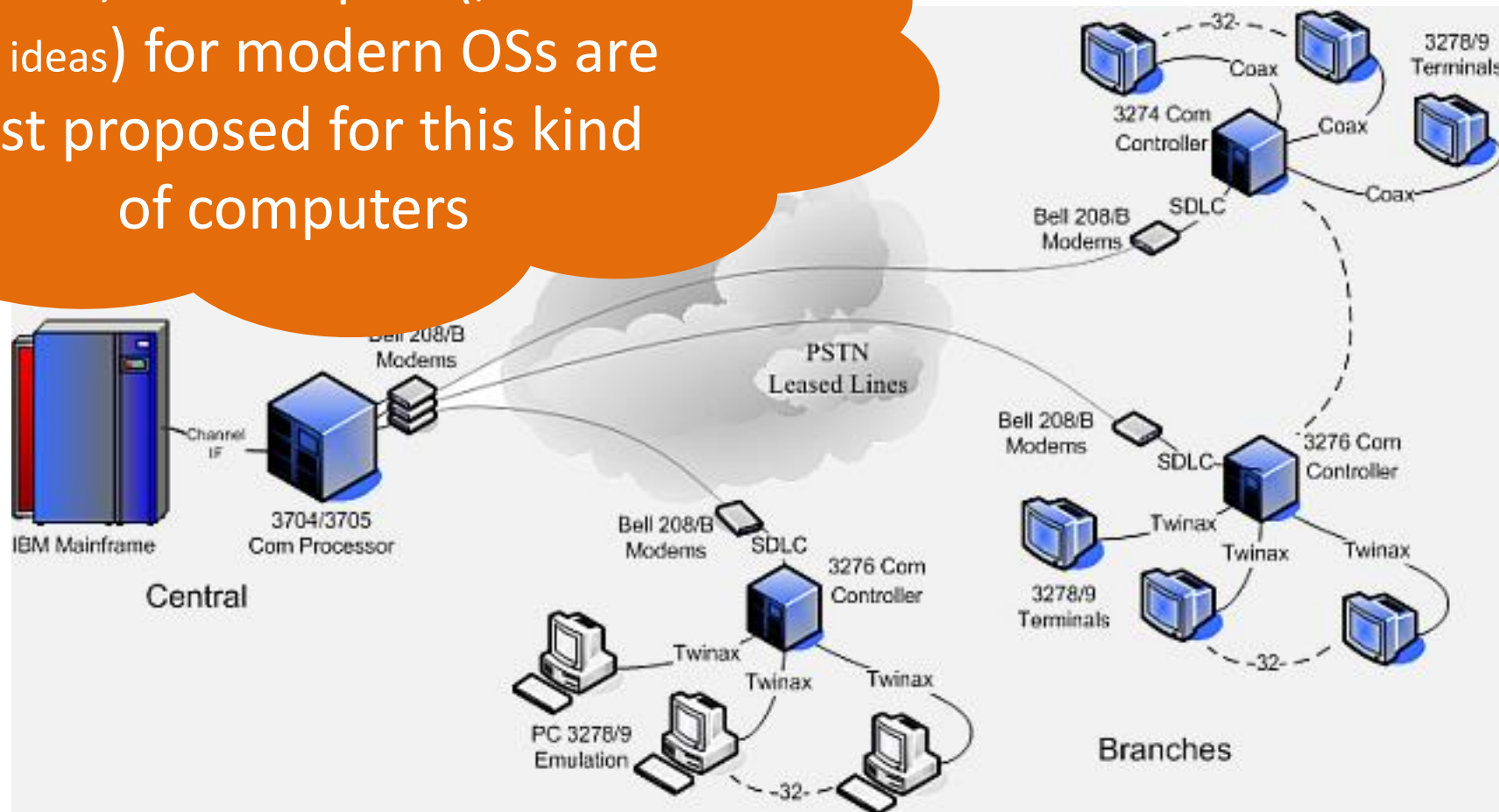
# Concurrency is now the indispensable property of modern OSs



Operating system

- Concurrency is more critical for larger computers providing service at the same time, like previous mainframe

In fact, most topics (problems and ideas) for modern OSs are first proposed for this kind of computers



同步数据链路控制 (Synchronous Data Link Control或簡稱SDLC)  
公共交换电话网 (Public Switched Telephone Network或簡稱PSTN)



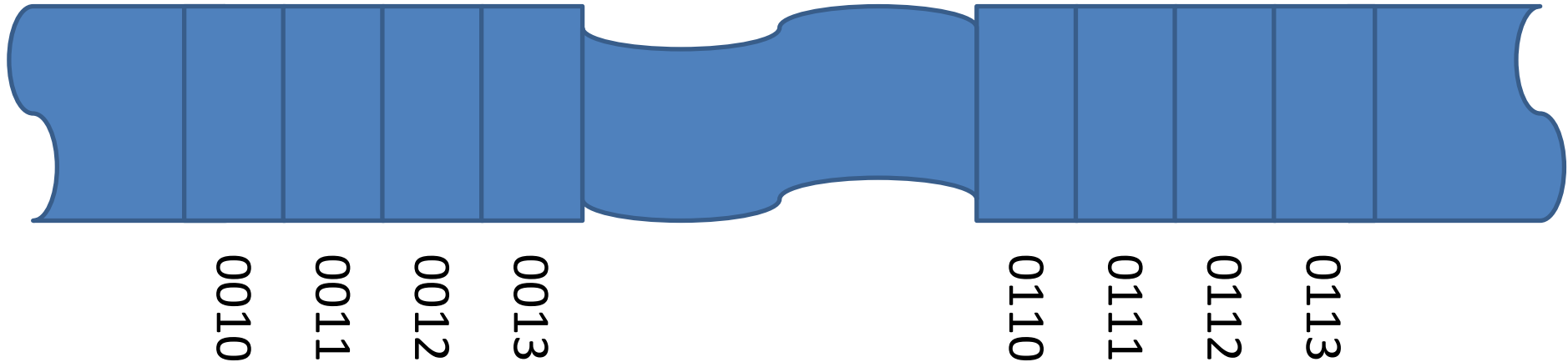
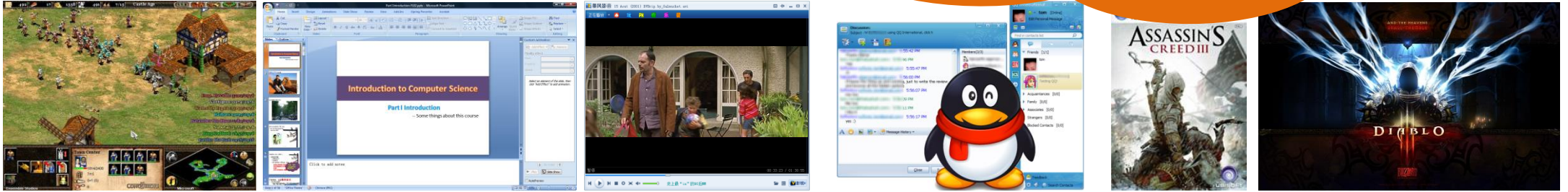
- **However, benefit always comes along with pains!**  
Concurrency leads modern OS to be more complicated
- Mapping 1 → “**SMALL MM v.s LARGE PROGRAM**” **problem** becomes more rigid even with larger and larger MM

1999	A high-end Personal Computer	128,000,000	128MB
2001	High-end Computers	Over 4,000,000,000	4 GB
2002	Large-scale Mainframe Servers	64,000,000,000	64 GB
Future	Terabyte Memory Computers	Over 1,000,000,000,000	1 TB

- Main memory [主存]
  - Linear addressed space!



User's requests could always break down the ability of the provider's ☹



Operating system

- Idea to overcome “**SMALL MM v.s LARGE PROGRAM**” **problem**?
- MULTIPLEXing MM!
  - Cut programs into smaller segments, and copy the segment (containing instructions for execution) when needed
  - This is like you paint a long picture on a small desktop
    - Of course you should find other position to keep the rest.

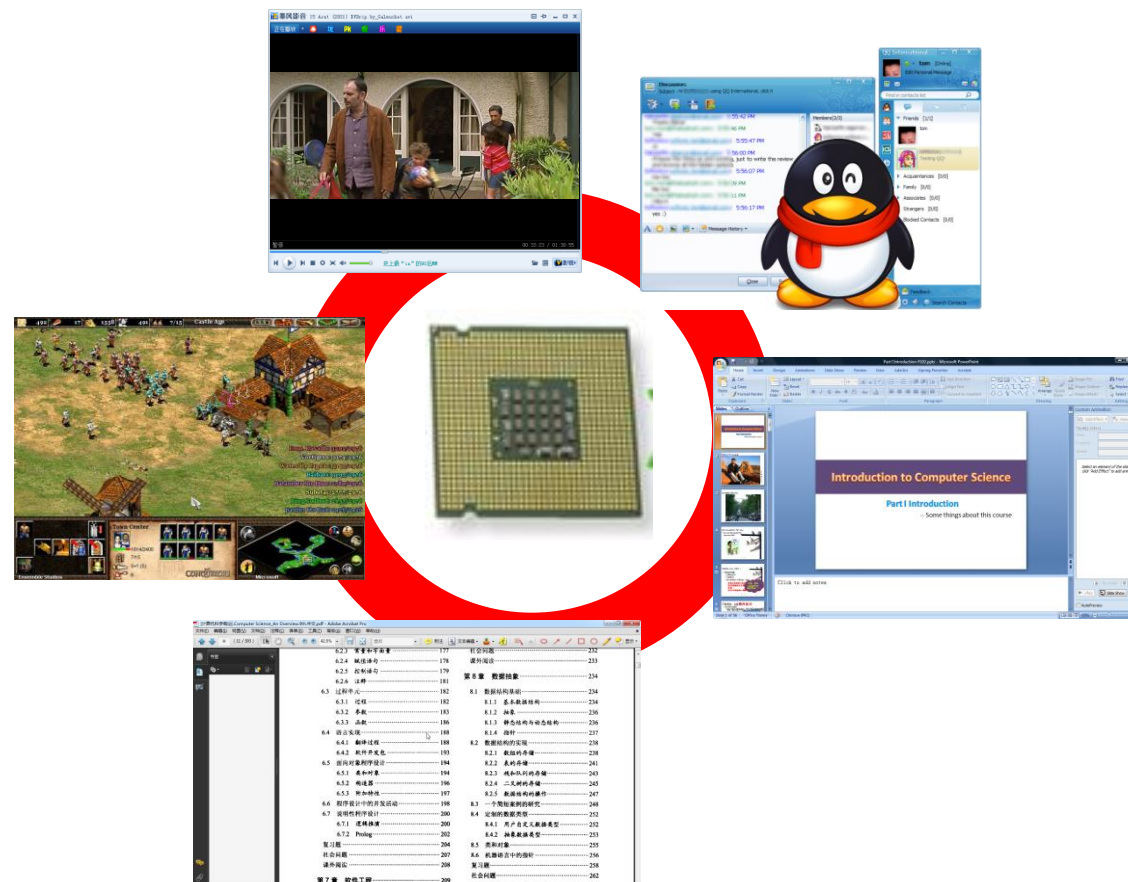


- Pay attention to another keyword – **PARALLEL**
  - Don't be confused with **CONCURRENCY**.
- Parallel means to do many things at same time!
  - In computers, this always means a computer has many CPUs. This is the property of modern server computers
  - This also means many children draw the long picture 😊



Operating system

- Execution → “1 CPU vs MANY PROGRAMS” problem



Operating system



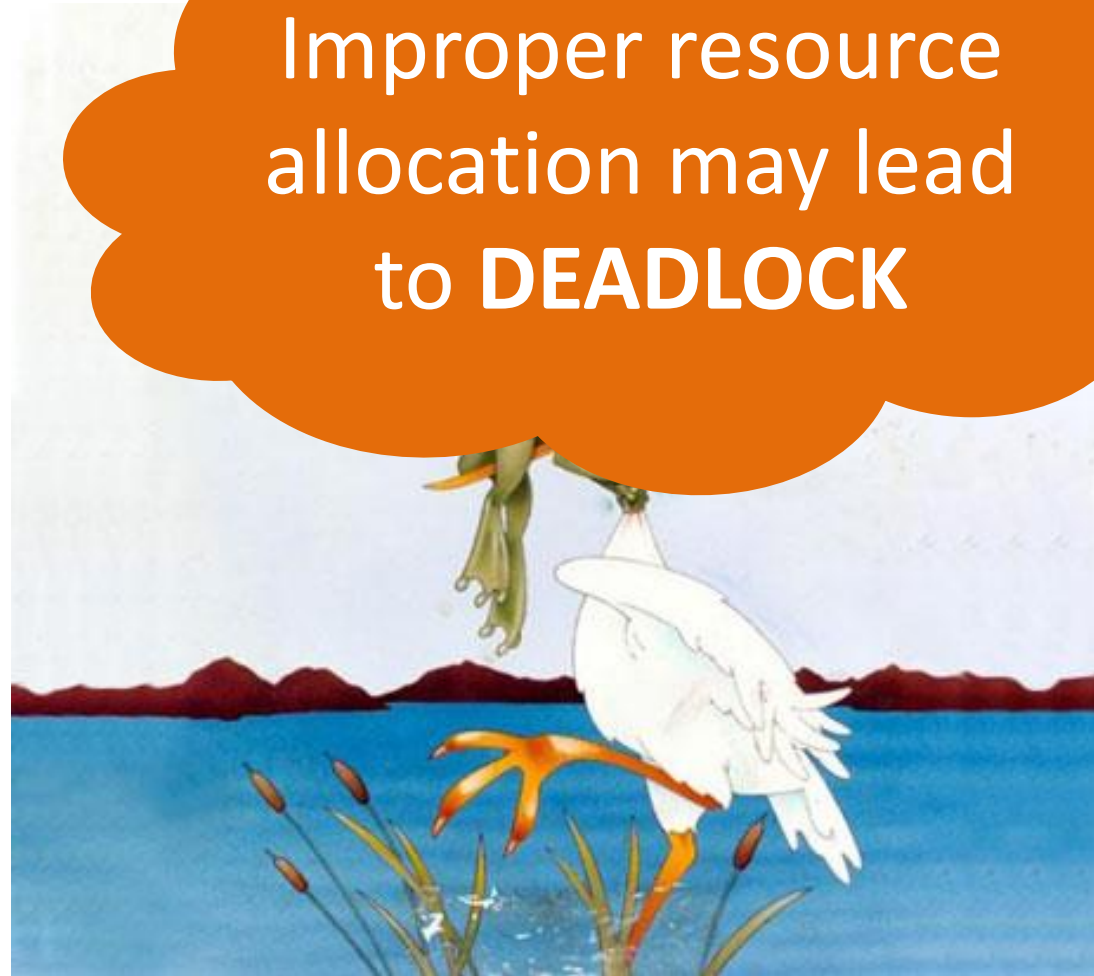
- Idea to overcome “**1 CPU v.s MANY PROGRAMS**” **problem?**
- MULTIPLEXing CPU!
  - This means to switch CPU among those concurrent programs quickly
  - This is like you paint several pictures on a small desktop, not one picture after another, but “one part of one picture after another part of other pictures”
    - Of course you should find other position to keep those pictures.



# Concurrency also leads to other challenges

- **PROBLEM** of **RESOURCE COMPETITION**
  - Each program needs some resources – memory, HDD, CPU, etc.
  - There must be competition among programs when running them concurrently

Improper resource allocation may lead to **DEADLOCK**



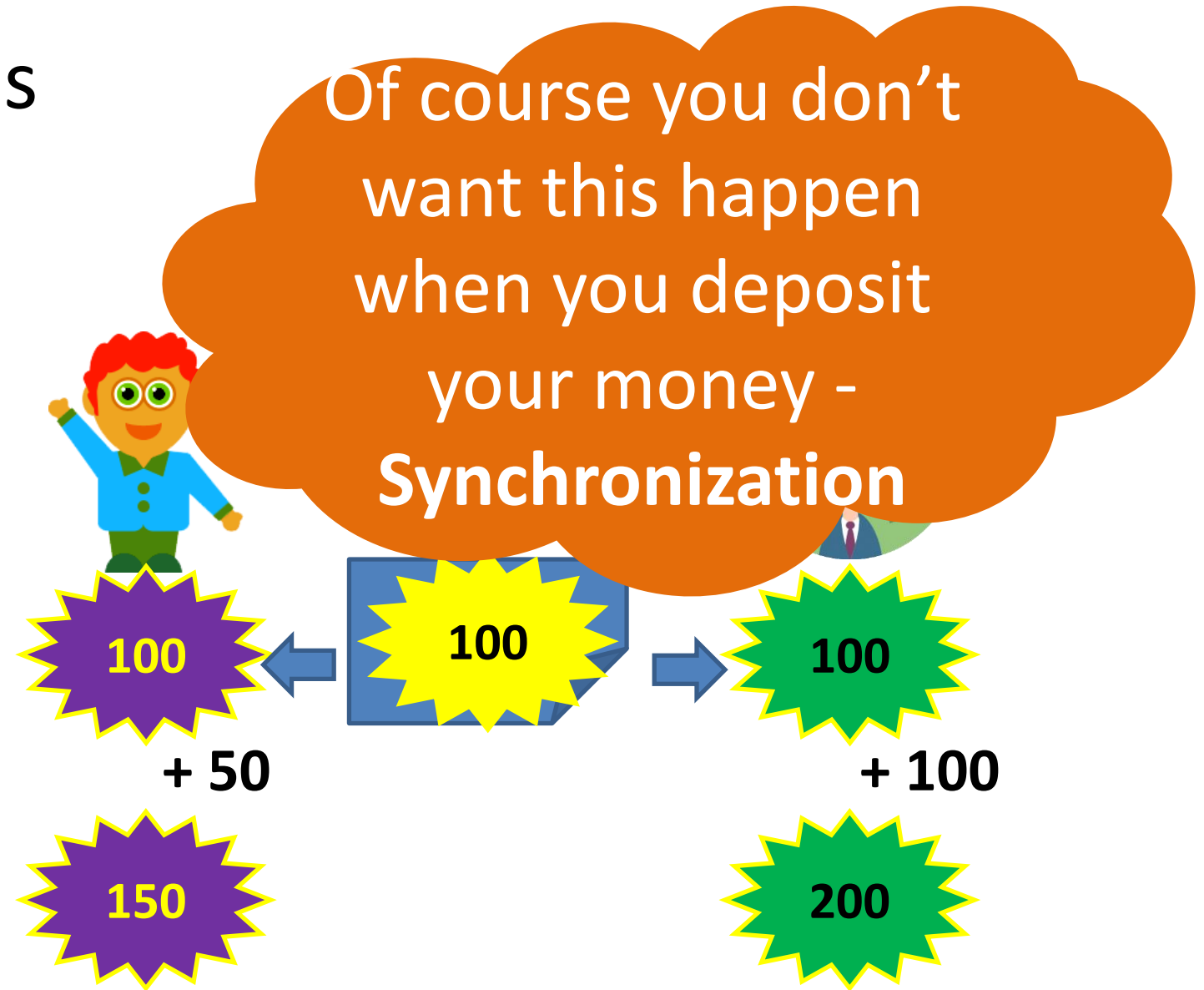
Operating system

R3



# Other challenges

- **PROBLEM** of Resource competition
  - **Data inconsistency** is another issue we should consider carefully when competing.



- Idea to overcome “**RESOURCE COMPETITION (DATA INCONSISTENCY AND**

- **SYNCH**

Hint is that you should conclude the conditions causing this kind of situation, as you cope with the real life problems , such as the light could not be turned on, etc.

- - The ... be mutually exclusively.
- More than one resources:
  - ?? Complicated, and touch later ☺
  - Could you infer some methods from your own traffic experience?





- By “**mutually exclusively**”, it’s like you and other persons share one washing machine, WC, coffer, etc.

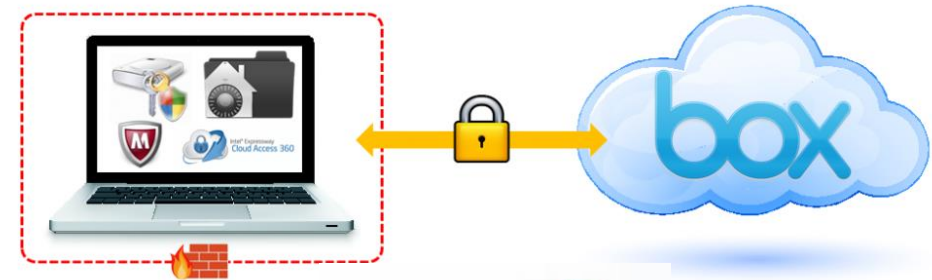
coffer ['kɒfə] n. 保险柜, 保险箱

- No others could use this resource when you are using it.
- Could you infer the key for this?
  - The **lock** (password)!
  - and the rules to use it
    - **Apply for** the lock **first**. You can use the resource only if you succeed.
    - You should **release** the lock **after** you finish to use the resource.



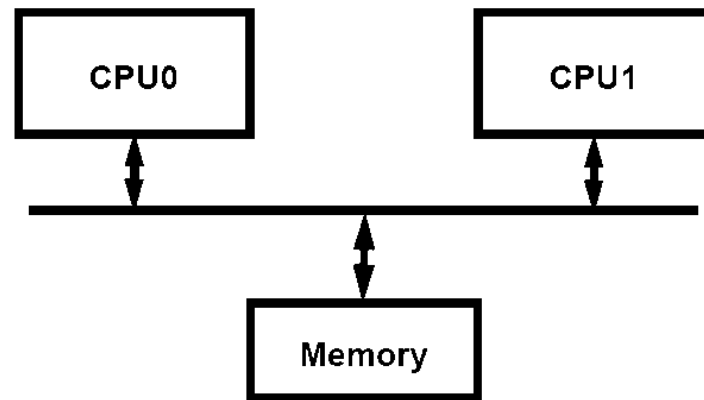
# SECURITY?

- **PROBLEM:** The essence is the data security!
  - **INSIDE** – in a computer system
    - Everyone can only access his/her own data! – **AUTHORIZATION.**
  - **OUTSIDE** – during the data transmission
    - No one could hack the data flowing in the network – **ENCRYPTION.**



# Other considerations

- Advanced topic – multi-processor system
  - All those issues listed before (multiplexing, Synchronization, deadlock, protection etc. ) should be reconsidered



← this part will be covered in this course, which depends ....

# 3 hints to follow!

- How to manage resources for

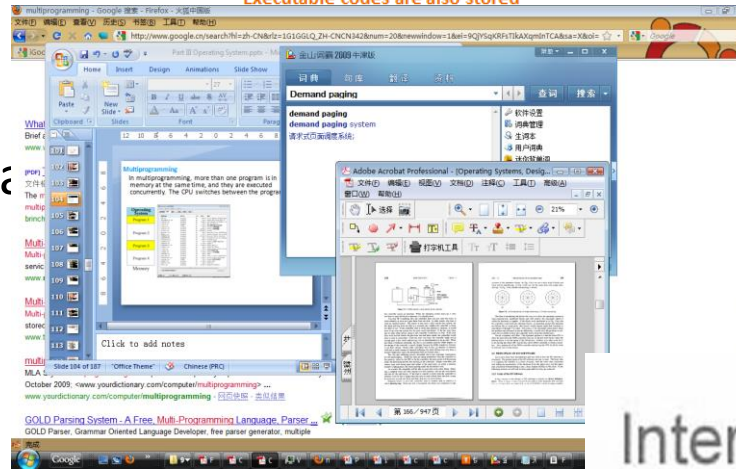
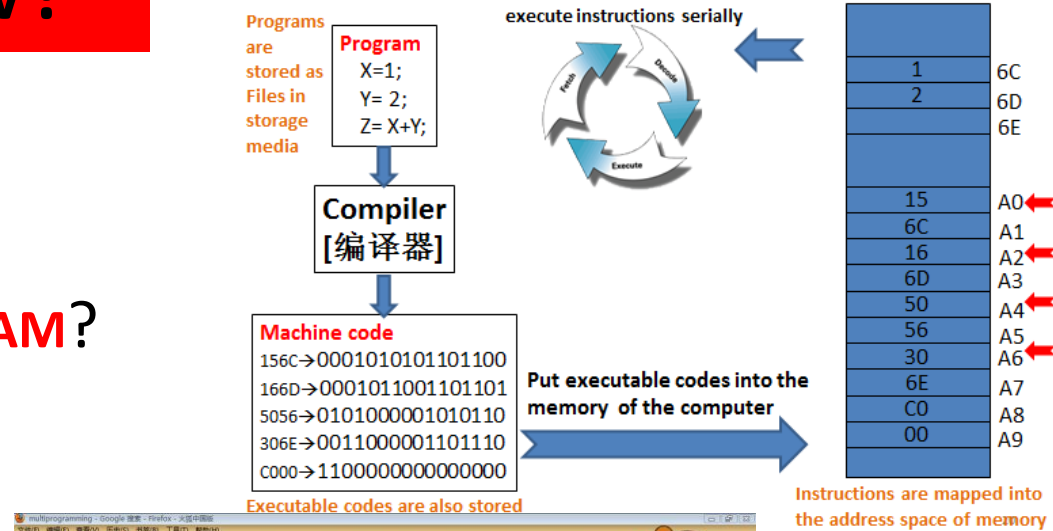
1. Running **A PROGRAM**?

2. Supporting **CONCURRENCY**?

- 1 CPU, many programs
- Small MM, large program
- Synchronization problem

3. Keeping **SECURITY**?

- Control illegal access



and,

Not “Electro-Magnetic Measurement 电磁测量”

- 3 core tasks – **EMM**
  - **EXECUTION** → CPU
    - Configurations for executing one program
    - Switching execution of concurrent programs
  - **MAPPING 2** → from File to Hard disk space
    - How to represent files in hard disks?
      - This is the fundamentals to understand storing data in other media
  - **MAPPING 1** → from File to Main Memory space
    - Problem of Small MM v.s Large size of programs
      - How to allocate space
        - » for a program?
        - » for many programs?



and,

Not “Greenwich Sidereal Date: 格林尼治恒星日期”

- 3 value-added tasks – **GSD**

- **GRAPHIC USER INTERFACE**

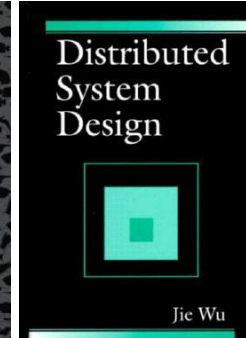
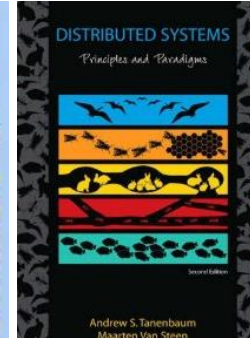
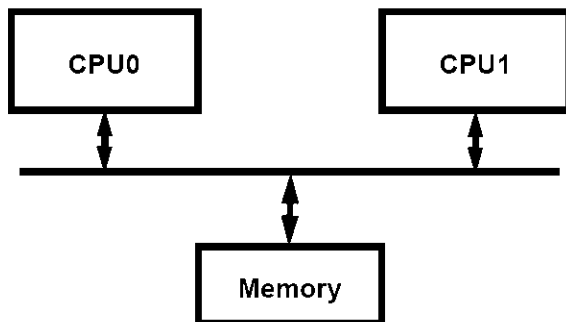
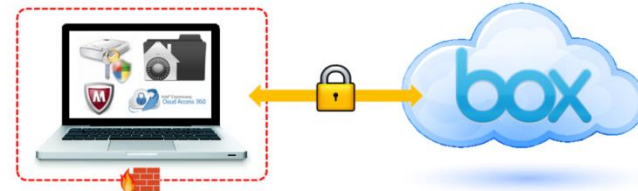
- How to provide friendly interface for users?

- **SECURITY**

- Inside and Outside?

- Extension of EMM to **DISTRIBUTED** systems

- Multiprocessor system → Parallel
    - Many programs could run at the same time now!
  - Data consistency problem and Deadlock are more urgent!



Support concurrent execution of many programs  
[with limited resources] [Local or Remote]

**Friendly Interface:**

GUI system (based on  
OO)

**Distributed Computing:**

Providing services for  
remote computers

**Carry out the security strategy:**

Control the resource access safely

**Carefully sharing resources among programs:**

Multiplexing + Synchronization

**Manage the resources:**

Define the necessary information (Data  
Structure) for those resources

OS's responsibility  
[programming]

Limited resources: 1:1:M – von Neumann

- Other fundamental problems
  - How to identify the program you want to run?
  - How to represent and organize files in HDD?
    - With the help of CO to understand how those bits are recorded
  - How to allocate limited MM to programs?
  - How to execute the ready program?
    - You have learned in Computer Organiza...
  - How to support GUI?
  - How to carry out inner and o...
  - How to carry out CPU switching, ...
- All we need is to design **data structure and related operations**



Yes, DSA is important!

# From 0 to 1 – “Write your own Operating System”



1.mp4



1-B -  
Addendum.mp4



2 - Install your  
OS in a Virtual  
Machine.mp4



3 - Memory  
Segments, Global  
Descriptor  
Table.mp4



4 -  
Hardware-Commu-  
nication  
Ports.mp4



5 -  
Interrupts.mp4



6- Keyboard.mp4



7 - Mouse.mp4



8 - Abstractions  
for Drivers.mp4



9 - Tidying  
up.mp4



10 - Peripheral  
Component  
Interconnect  
(PCI).mp4



11 - Base  
Address  
Registers.mp4



12 - Graphics  
Mode (VGA).mp4



13 - GUI  
Framework  
Basics.mp4



14 - Desktop and  
Windows.mp4



15 -  
Multitasking.mp4



16 - Dynamic  
Memory  
Management  
Heap.mp4



17 -  
Networking.mp4



18 - Network  
continued.mp4



19 - Hard  
Drives.mp4



20 - System calls,  
POSIX  
compliance.mp4



A Website for the  
operating system  
programming  
tutorial.mp4



A01 - Ethernet  
Frames.mp4



A02 - Address  
Resolution  
Protocol  
(ARP).mp4



A03 - Internet  
Protocol  
(IPv4).mp4



A04 - Internet  
Control Message  
Protocol  
(ICMP).mp4



A05 - User  
Datagram  
Protocol  
(UDP).mp4



A06 -  
Transmission  
Control Protocol  
(TCP).mp4



A07 -  
Transmission  
Control Protocol  
(TCP).mp4



A08 -  
Transmission  
Control Protocol  
(TCP).mp4



A09 - TCP + a  
little HTTP.mp4



B01 - Partition  
Table.mp4



B02 - File  
Allocation Table  
(FAT32).mp4



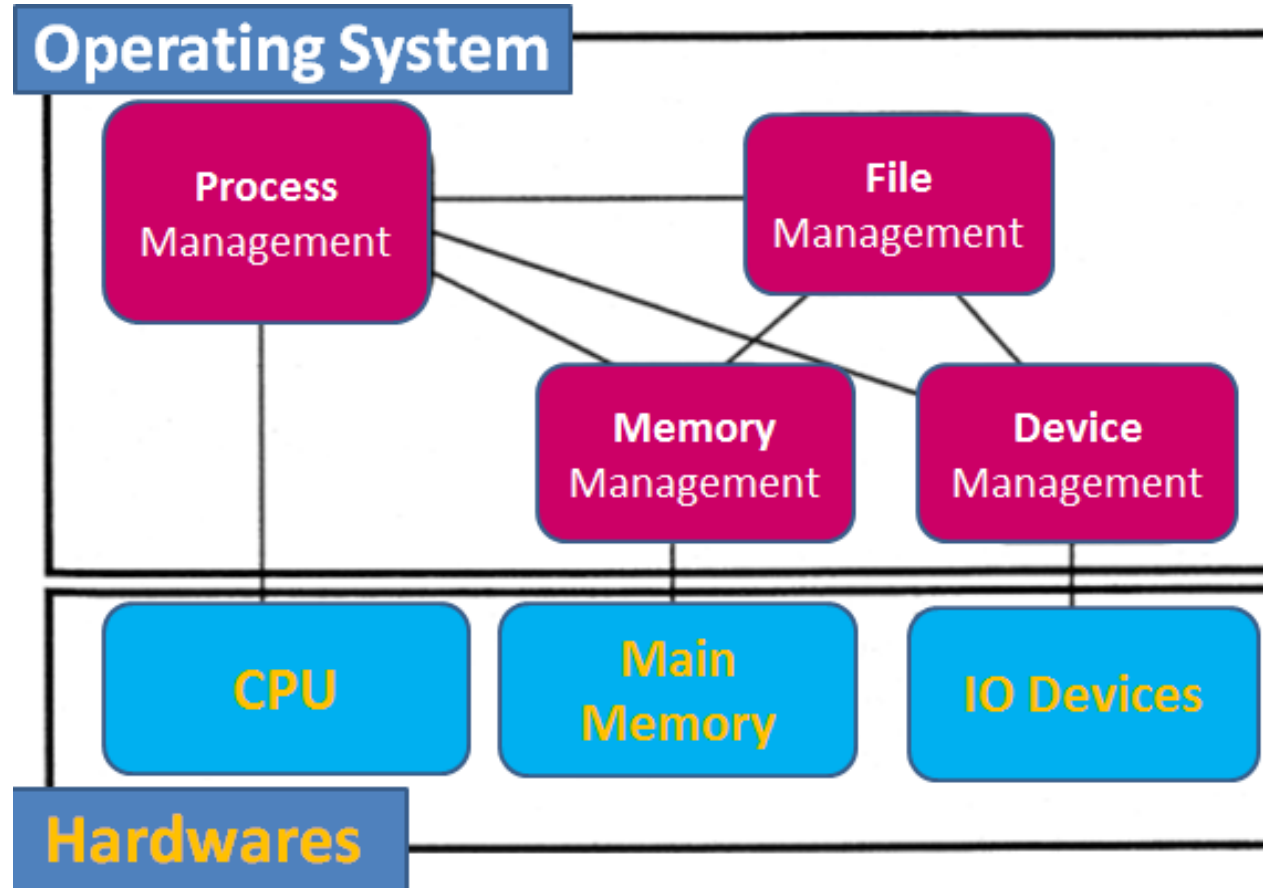
B03 - File  
Allocation Table  
(FAT32).mp4

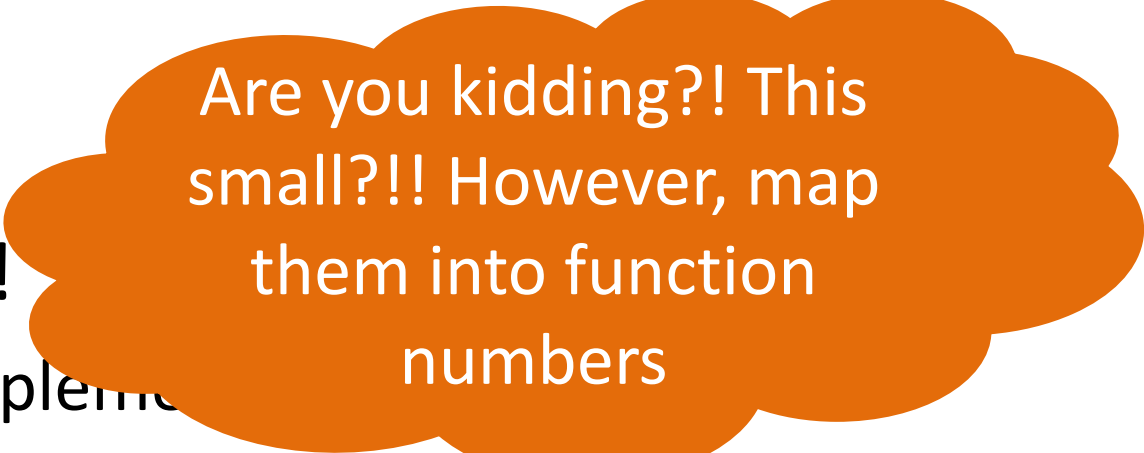
- Problems and ideas when implementing OS
  - Problems: EMM+GSD
  - Ideas: Multiplex resource + Synchronize programs
- Construct OS
  - Evolution of OSs's structures
    - Simple structure/monolithic structure
    - Modular, Layered, Microkernel
    - Trend – Virtual machine
- How to use the functions defined in OS?
  - System Call/API
- How to load and run OS?



# We've learned – 4 components

- We can distill 4 kinds of resources / concepts
  - **CPU, Main Memory, Hard Disk, File**
- Modern OS has 4 fundamental components





Are you kidding?! This small?!! However, map them into function numbers

- However, OS is more complex!
  - There are so many functions implemented
- Kernel size (**EMM+S**, without those drivers, etc.) of many OSs
  - Unix 32/V (VAX) 180k
  - BSD 4.3 (VAX) 640k
  - Mach 2.0 (VAX) 1000k
  - SunOS 4.03 (Sun 3, Sun 4) 2400k
  - Windows NT (x86) 4000k

<http://www.gordoni.com/os-sizes.html>

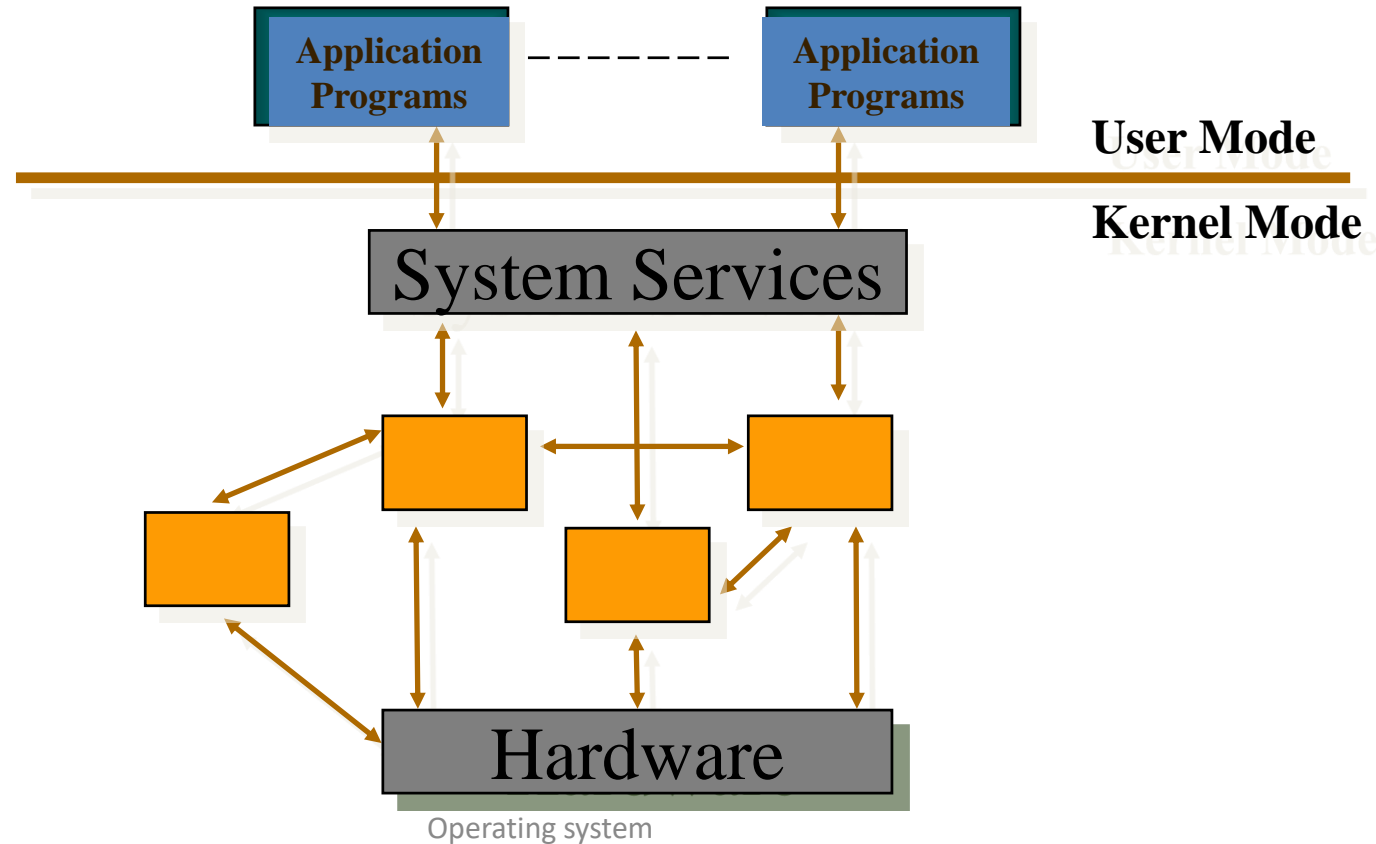
- Based on my experience in Code::Blocks, the C code of whole classic linked list is 3KB. Therefore,
  - Unix 32/V (VAX) ~60
  - BSD 4.3 (VAX) ~214
  - Mach 2.0 (VAX)
  - SunOS 4.03 (Sun 3, Sun 4)
  - Windows NT (x86)
- Have you ever experienced to manage 200 entities?
  - Like 200 Books? Cloths? persons? Especially they have some kind of relationships?



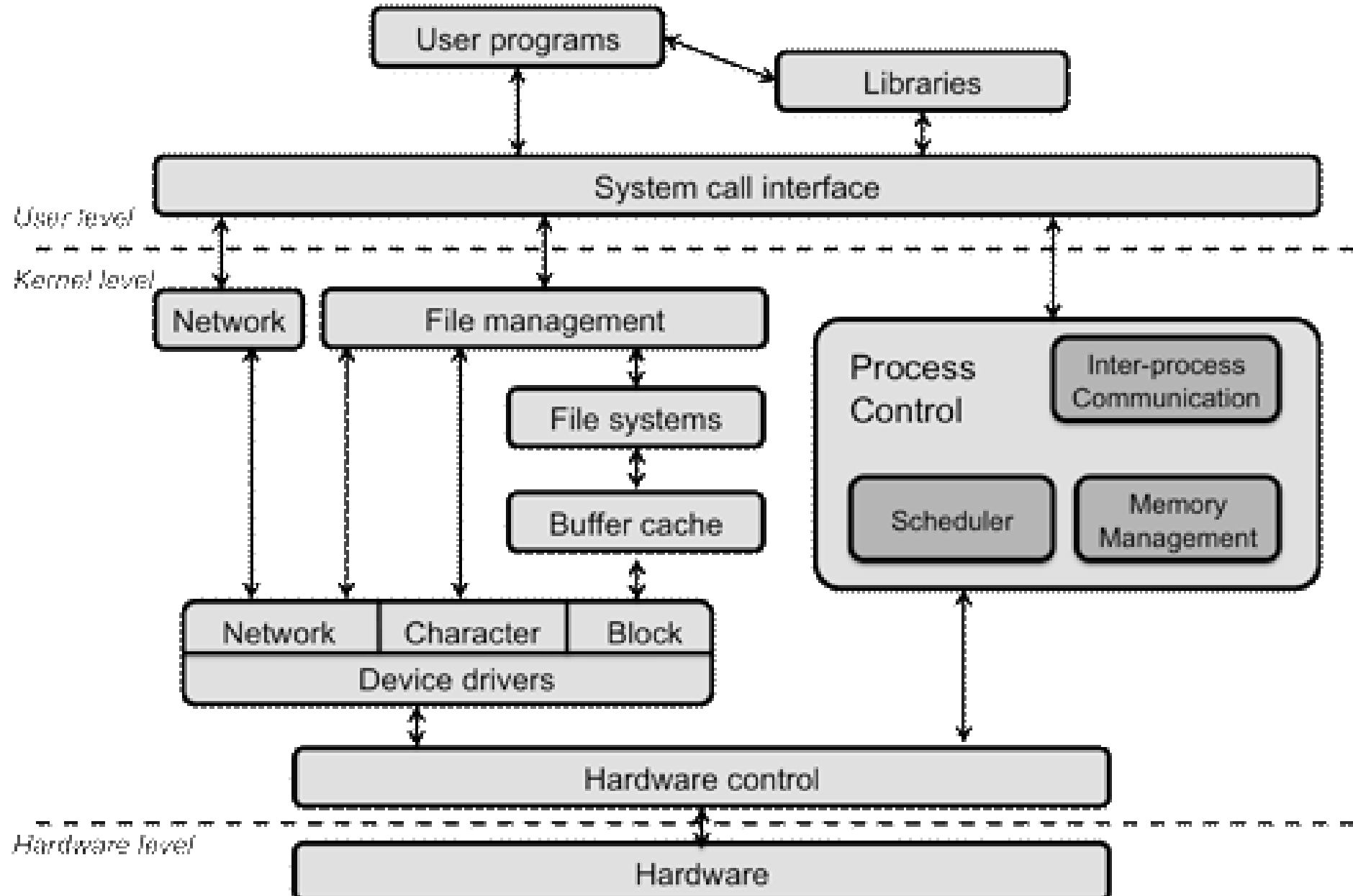
Organized  
structure! OS uses  
this too!

- Modular structure (Monolithic)

- Better application Performance
- Difficult to extend
- Ex: **MS-DOS**

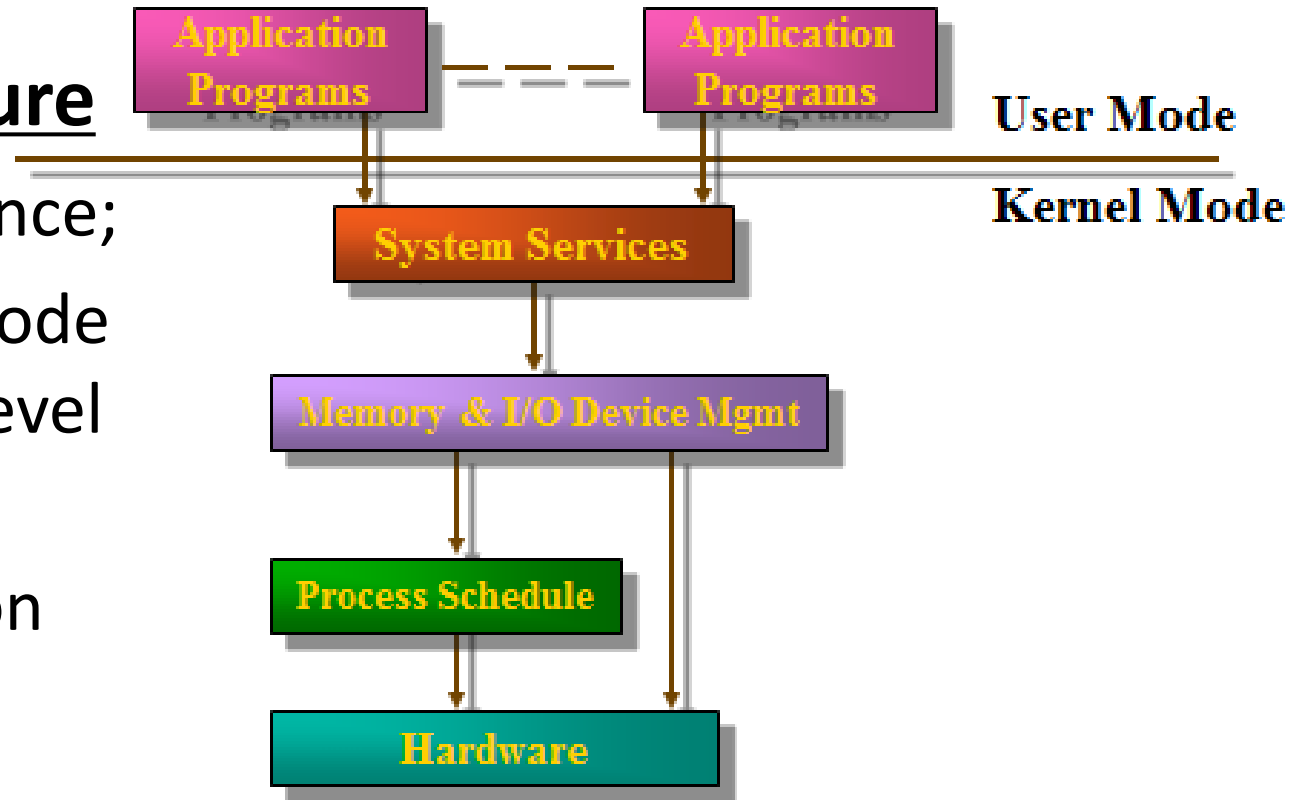


# Structure of a typical operating system



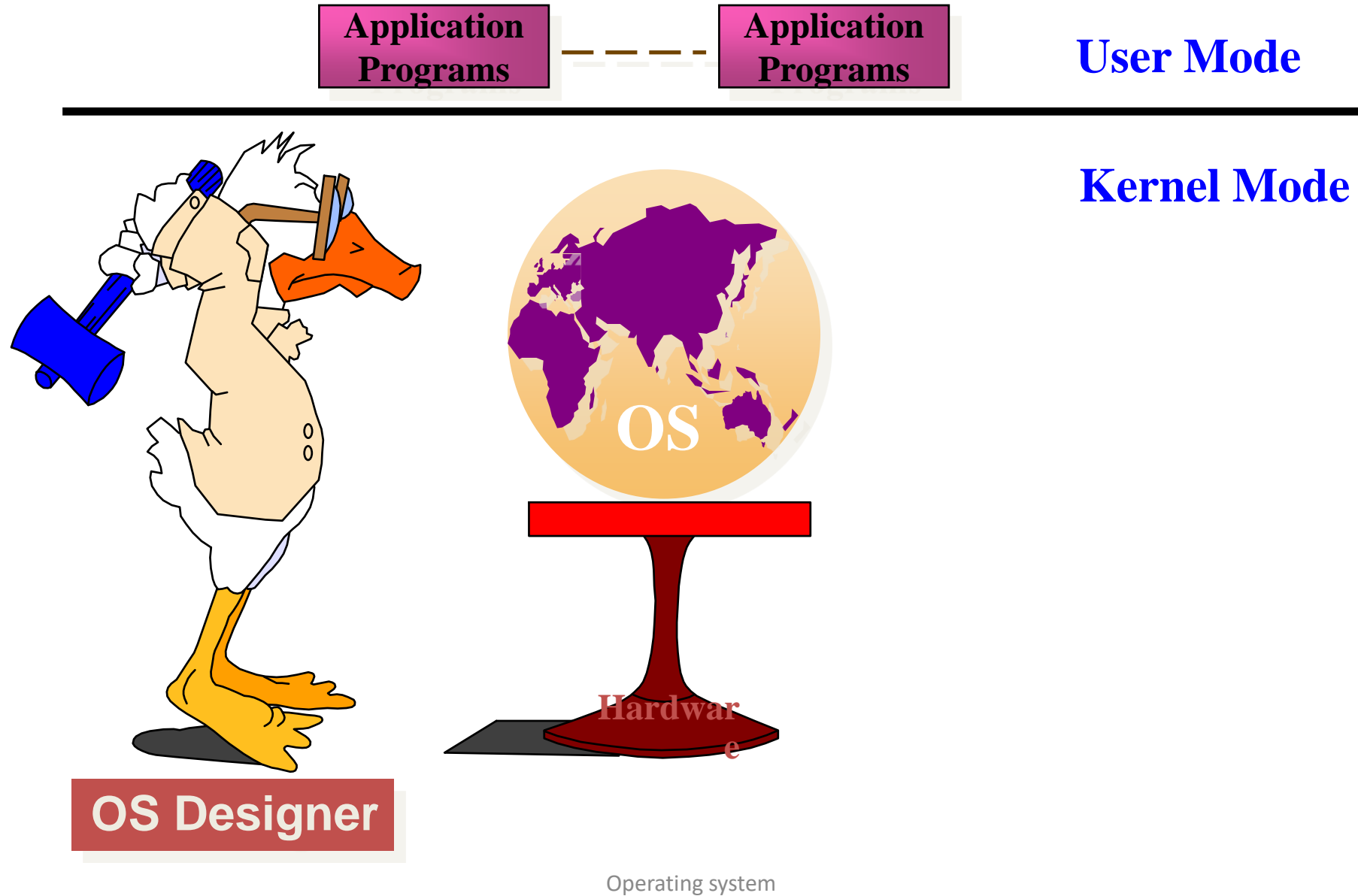
- **Layered structure**

- Easier to enhance;
- Each layer of code access lower level interface
- Low-application performance
- Ex: Old Unix

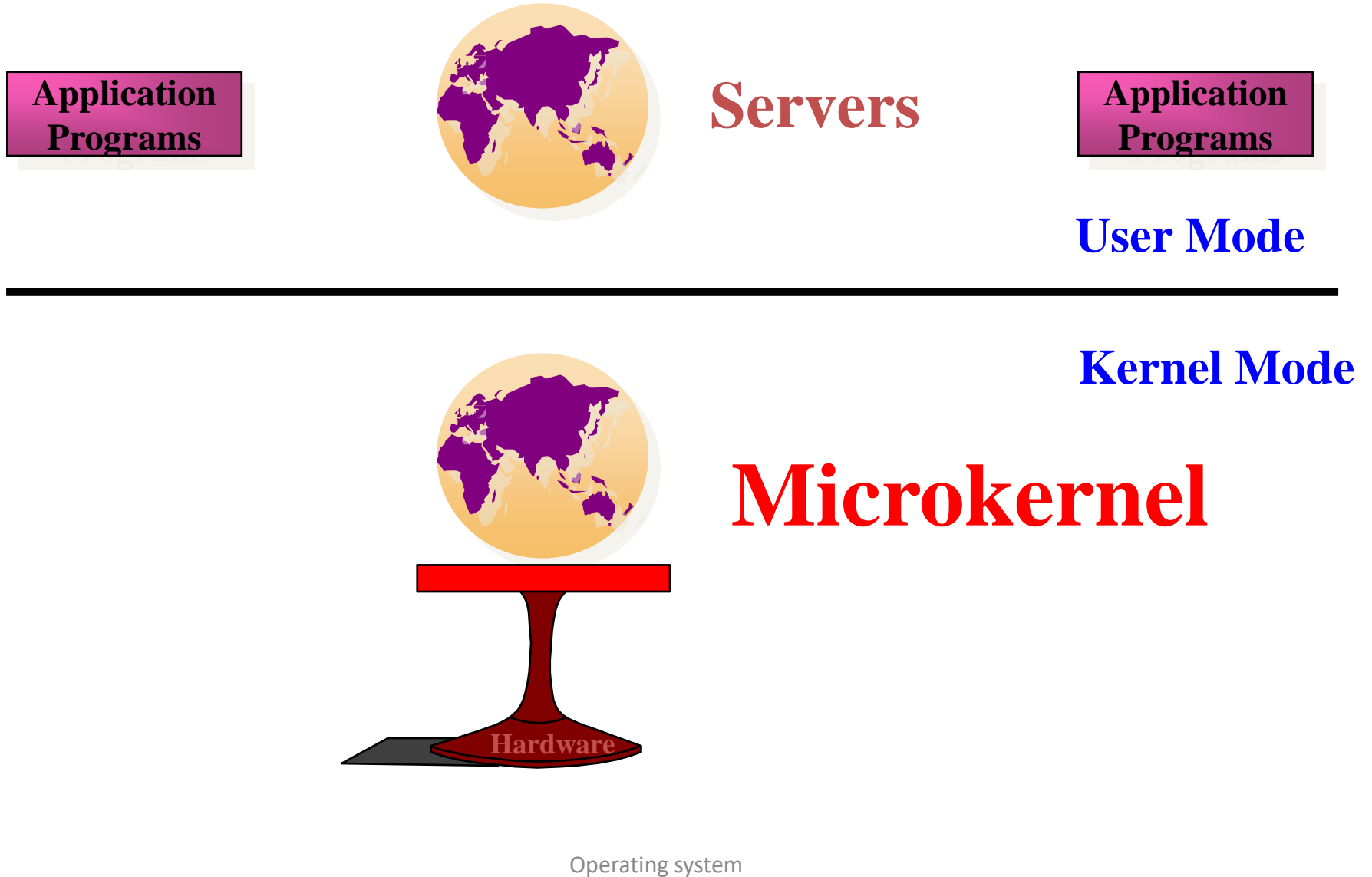




# Traditional OS

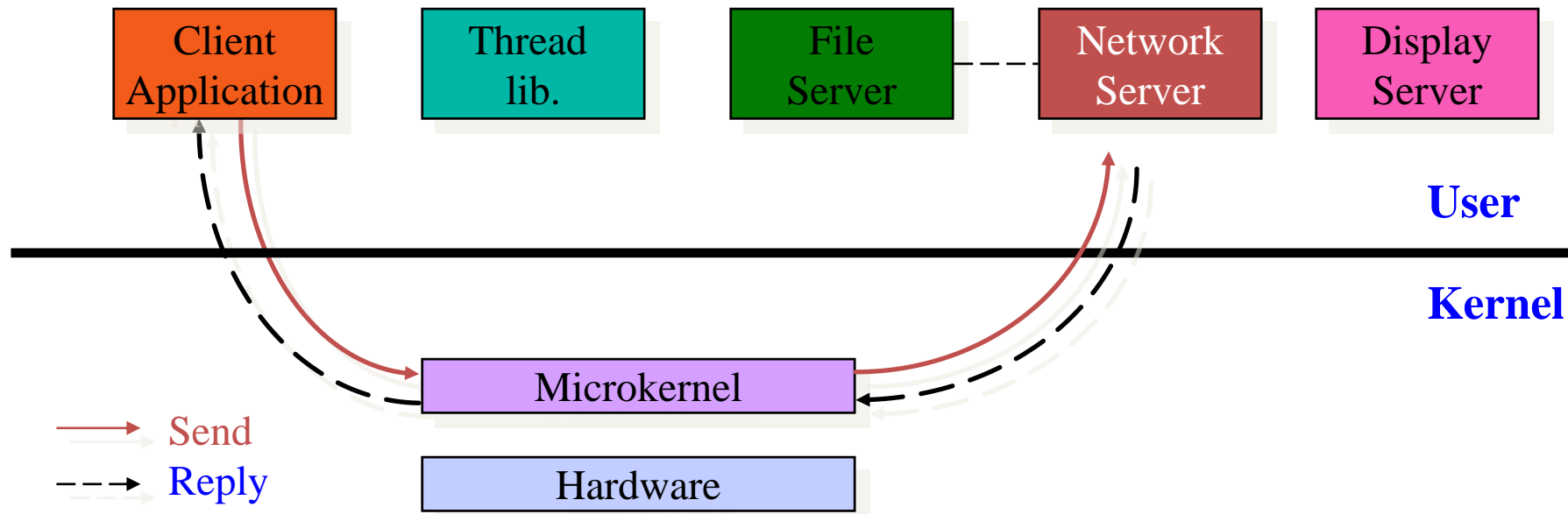


# New trend in OS design



- Microkernel/Client Server OS 1990s

- Tiny OS kernel providing basic primitive (**process**, **memory**, **IPC – Inter-process communication**)
- Traditional services becomes subsystems
- OS = Microkernel + User Subsystems
- Ex: **Mach**, **PARAS**, **Chorus**, etc.



---

# Microkernels

# 22

## CONTENTS

22.1	The evolution from RIG through Accent to Mach	506
22.2	Mach	511
22.3	CHORUS	516
22.4	Distributed systems at Cambridge	520
22.5	Summary	527

---

Conventional operating systems have become large and unwieldy. They are difficult to comprehend, develop and maintain. The addition of new facilities often means that a given task can be done in a number of different ways. Section 2.4 introduced the microkernel approach: it is argued that the kernel should have the minimum functionality necessary to support:

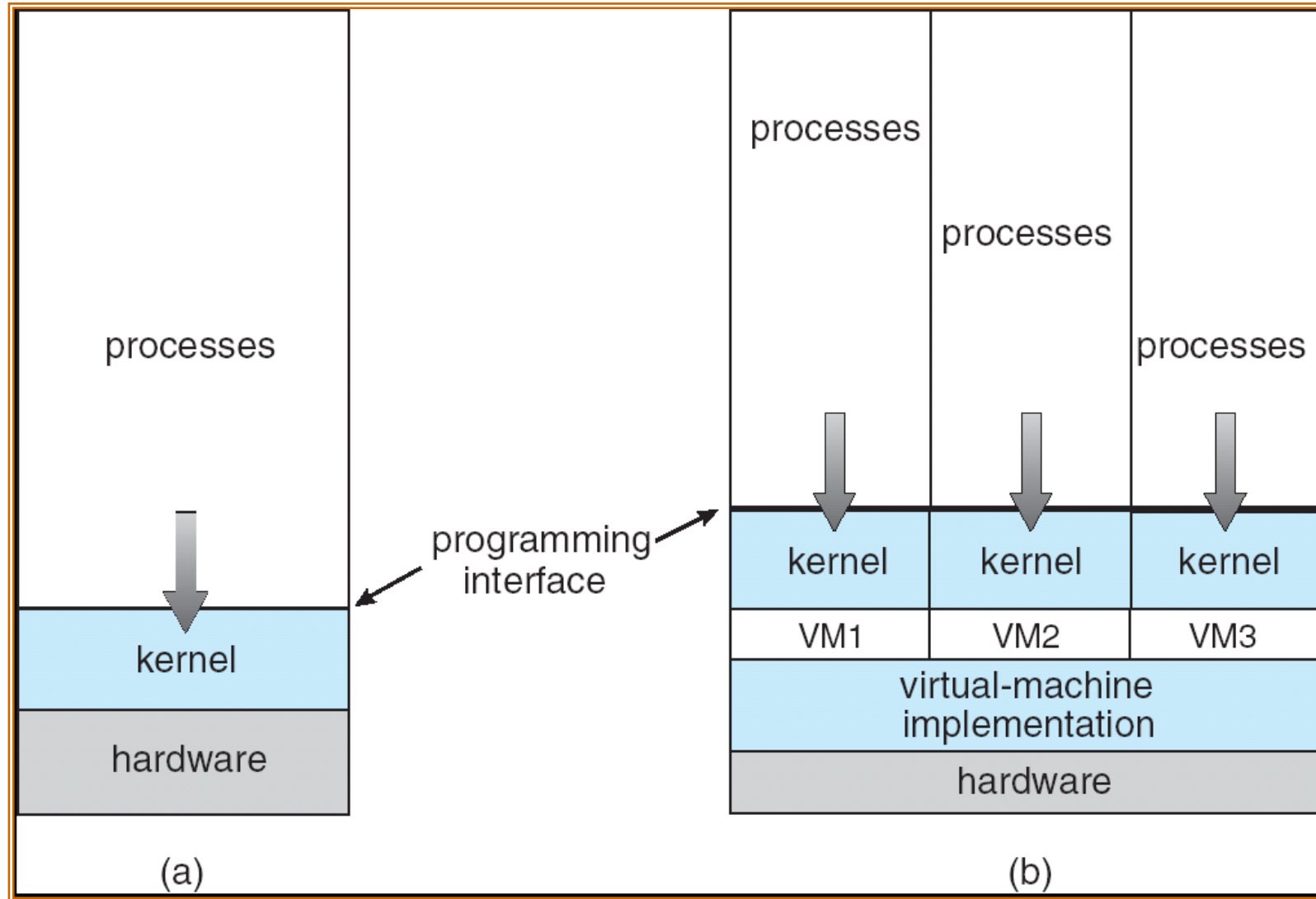
- the management of the hardware interface;
- processes (or whatever term is used to indicate the unit of computation and scheduling on a processor);
- inter-process communication;
- minimal memory management: handling MMU exceptions for example.

Other functions such as the file service and much of memory management and communications handling can be provided above the kernel and run as user-level processes. The efficiency penalty of offering services at user level is offset by the greater efficiency of a streamlined kernel.

# Virtual Machines

- **Virtual Machines**: simulate OSs on one computer hardware
  - Hardware is abstracted into several different execution environments
    - Virtual machines
  - Each virtual machine provides an interface that is identical to the bare hardware
  - A *guest* process/kernel can run on top of a virtual machine.
    - We can run several operating systems on the same *host*.
    - Each virtual machine will run another operating system





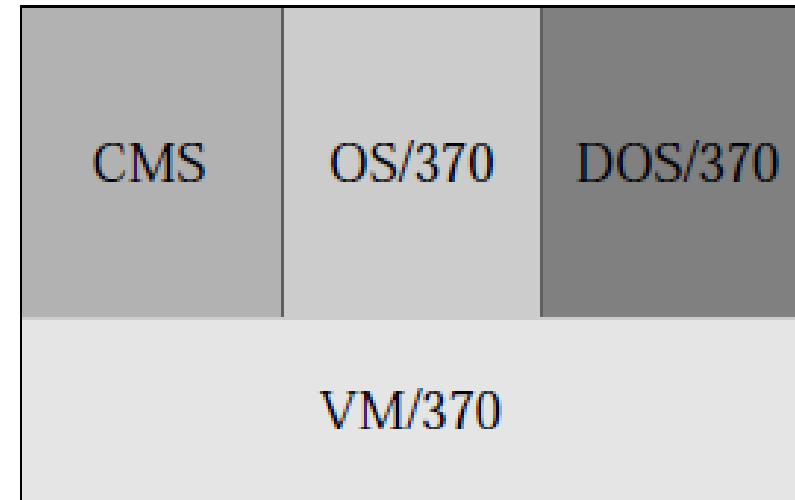
Non-virtual Machine

Operating system

Virtual Machine

# However, VM idea is not new at all

- It was first proposed in **1970s**
  - The Conversational Monitor System(CMS) is a single-user operating system, while the OS/370 and DOS/370 are multiprogramming operating systems.
  - A user process is unaware of the presence of the VM/370—it sees only the guest OS that it uses.



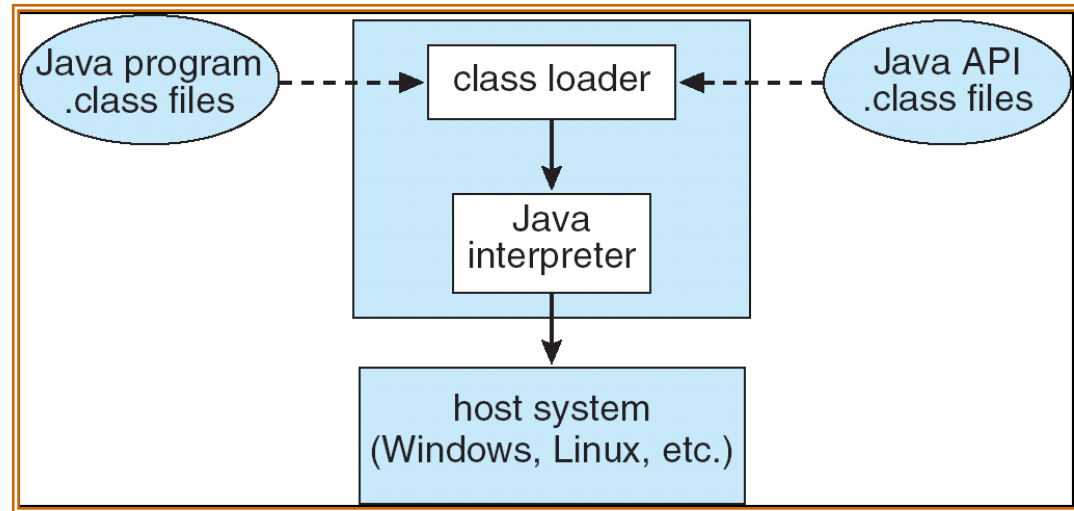
**Figure 4.5** Virtual machine operating system VM/370.

Users can choose  
what OS they want to  
use when logging in

# VM: Examples

- VMware
  - Abstracts Intel X86 hardware
- Java virtual machine
  - Specification of an abstract computer
- .NET Framework

## The Java Virtual Machine

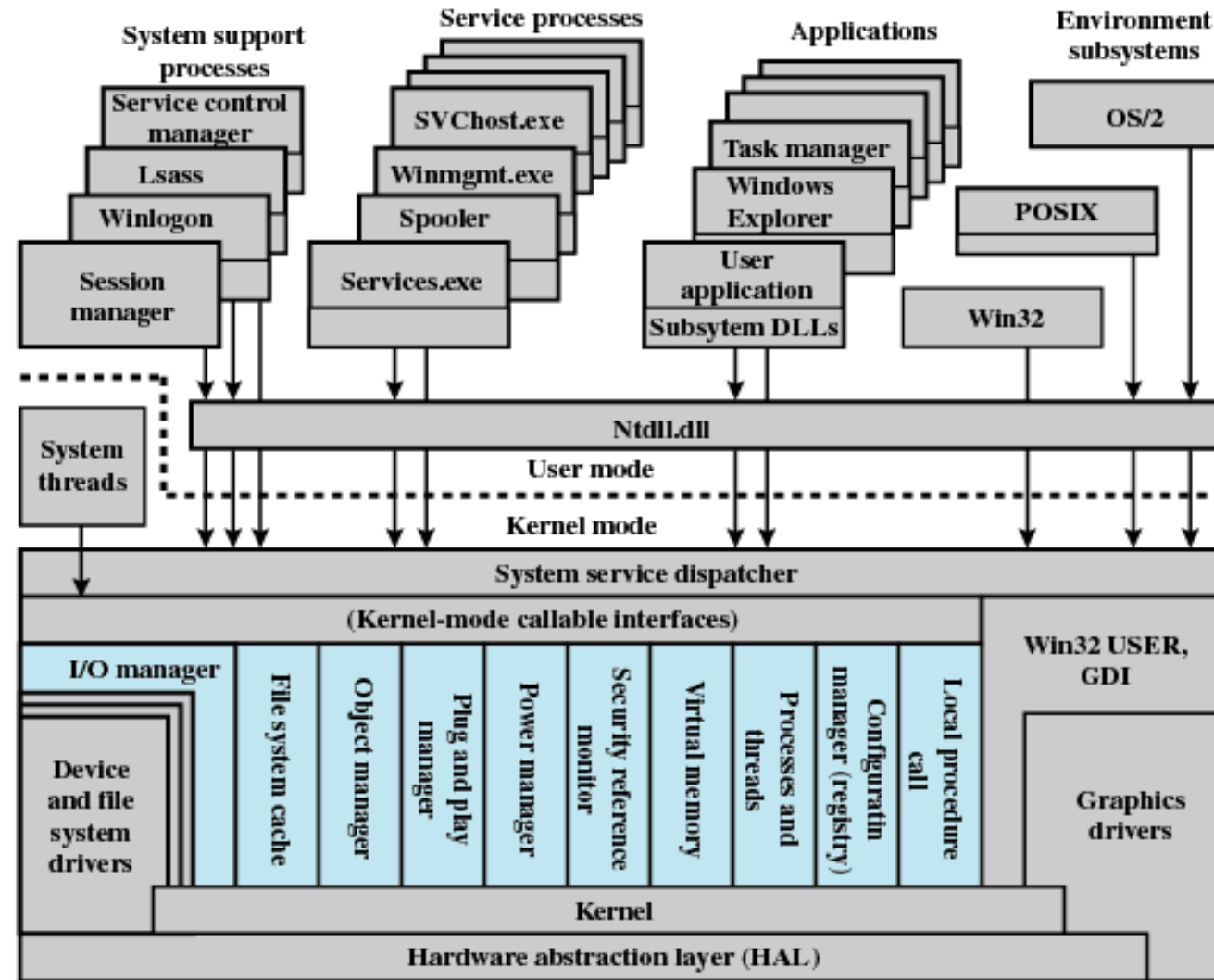


Java consists of

1. Programming language specification
2. Application programming interface (API)
3. Virtual machine specification

**The Java Virtual Machine allows Java code to be portable between various hardware and OS platforms.**

# Hybrid



Lsass = local security authentication server  
 POSIX = portable operating system interface  
 GDI = graphics device interface  
 DLL = dynamic link libraries

Colored area indicates Executive

**Figure 2.13 Windows 2000 Architecture [SOLO00]**

- **内核(kernel)**
  - 操作系统中实现基本功能的代码，它常驻内存并运行于特权方式。
- **单体内核(monolithic kernel)/宏内核(macro-kernel)/分层内核(Layered kernel)**
  - 内核实现操作系统的所有基本功能（包括调度、文件系统、连网、设备驱动程序、内存管理等等）
  - 一般用一个进程实现，内核代码共享同一个地址空间，每一模块可以调用任意其它模块和使用内核所有核心数据
  - 效率高，但难于修改和扩充
  - 例子：Unix/BSD、Linux/Android、Mac OS 1.0~8.5.1、DOS、Windows 1~3.1/95/98/ME
- **微内核（microkernel）**
  - 内核只实现最基本功能（包括地址空间、IPC[InterProcess Communication，进程间通信]和基本调度）
  - 更多的功能代码组织为多个进程，各自独立使用自己的地址空间，运行于非特权方式
  - 客户/服务器模式，设计简单灵活，适用于嵌入式与分布式环境，但效率稍低
  - 例子：Windows NT 3.1/3.5/3.51、Mac OS 8.6~9.2.3（nanokernel 纳米/超微内核）、Minix、Mach（马赫[奥地利物理家/1倍音速]，卡内基·梅隆大学1985年开发的一种分布式与并行操作系统的研究内核，最后的版本为1994年推出的3.0）、QNX（加拿大量子/QNX软件系统公司1982年开发的一种嵌入式商用类Unix操作系统，2010年被黑莓公司获得，当前最新版本是2014年3月推出的6.6）
- **混合内核（hybrid kernel）**
  - 微内核与宏内核的结合（也可算作单体内核中的一类）
  - 具有微内核结构，按宏内核实现
  - 例子：Windows NT(4.0/2000/XP/Vista/7/8/10)、XNU（Darwin 的核心，源自Mach和BSD，用于苹果公司的Mac OS X、iOS、watchOS和tvOS）

2019.8.9

<https://github.com/huawei-iot/HarmonyOS>

<https://www.bilibili.com/video/av63030192/>

# 华为 鸿蒙 HarmonyOS

## HarmonyOS 鸿蒙

HarmonyOS 鸿蒙

基于微内核的全场景分布式OS

分布架构

天生流畅

内核安全

生态共享

鸿蒙OS实现模块化解耦 对应不同设备可弹性部署



智慧屏专有服务



穿戴设备专有服务



车机专有服务



音箱专有服务



手机专有服务

程序框架

基础服务

内核



# 2021年6月2日

<https://news.mydrivers.com/1/760/760939.htm>

## • 华为正式发布鸿蒙手机操作系统：告别安卓 打响国产之战

- 2021年6月2日，在备受瞩目的 HarmonyOS 2 及华为全场景新品发布会上，华为正式发布了数款出厂搭载鸿蒙 OS（HarmonyOS）的智能手机新品：Mate 40 系列、Mate X2 和 Nova 8 Pro 的新版本——同时，华为 P50 系列手机新品也在发布会上正式亮相



<https://repo.openeuler.org/>

- EulerOS – Linux CentOS based OS by HuaWei



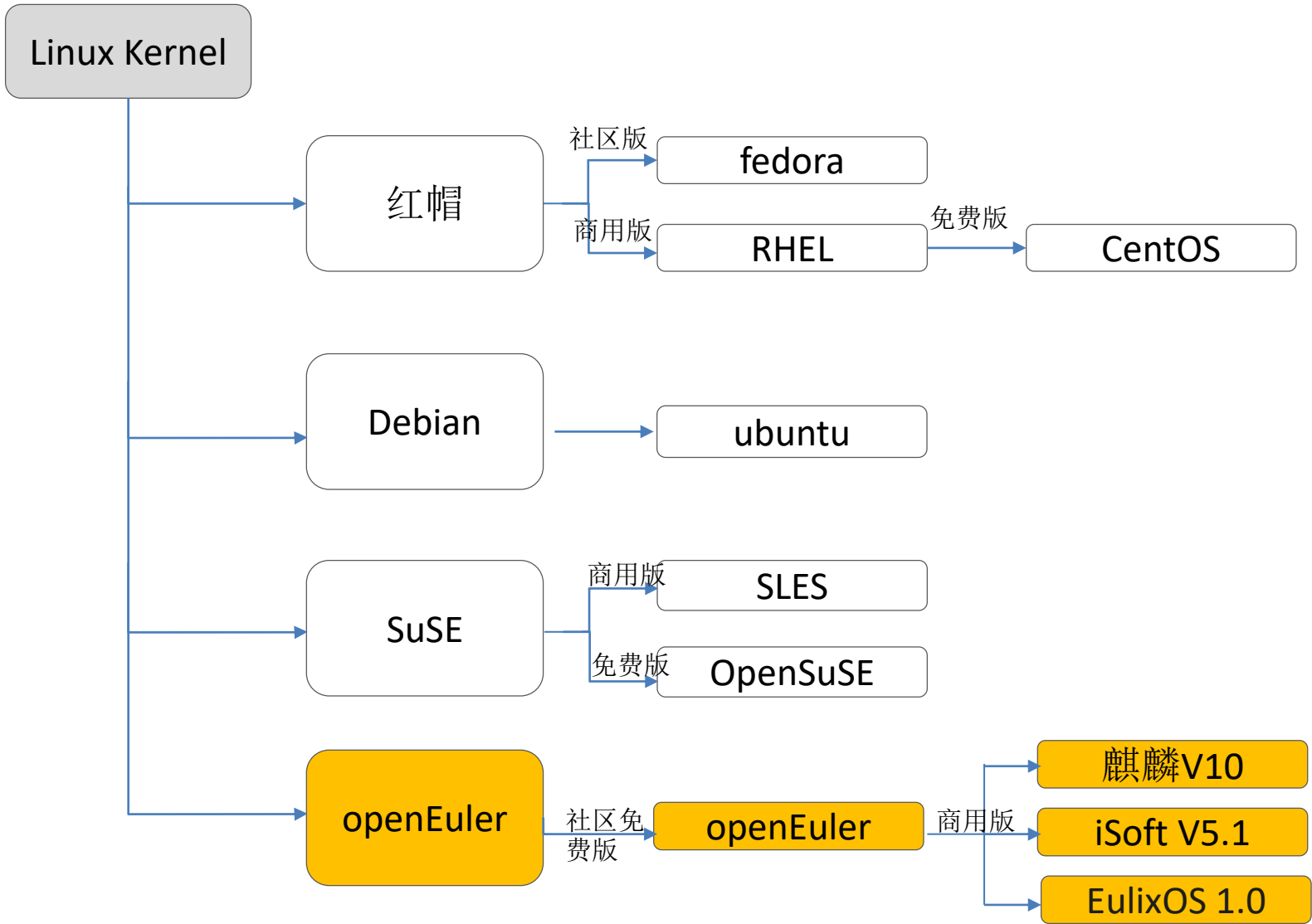
<https://developer.huaweicloud.com/ict/cn/site-euleros/euleros>



- OpenEuler – the EulerOS community version
  - The openEuler 20.03 LTS version is a standard release version that meets open scenario requirements, which has a lifecycle of four years.



# openEuler和主流OS系的关系

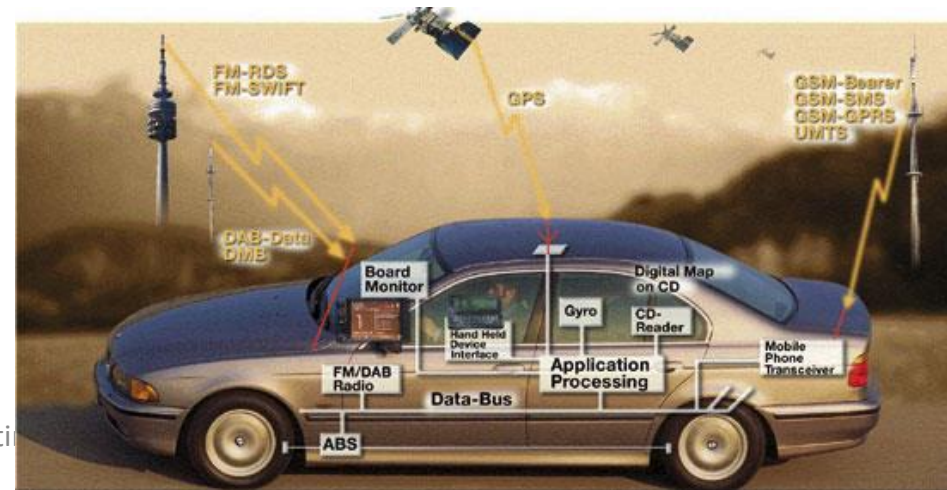


- openEuler与SuSE、Debian、RedHat的上游社区都是kernel社区  
[www.kernel.org](http://www.kernel.org)
- openEuler社区发行LTS免费版本，使能OSV发展商业发行版，如麒麟软件、普华、中科软、万里开源等

# Implementing OS



- Traditionally, operating systems have been written in assembly language.
- Now, however, they are most commonly written in higher-level languages such as **C** or **C++**
  - This is why C is so important, and now is popular again because of the boom of embedding system – system programming always needs C!



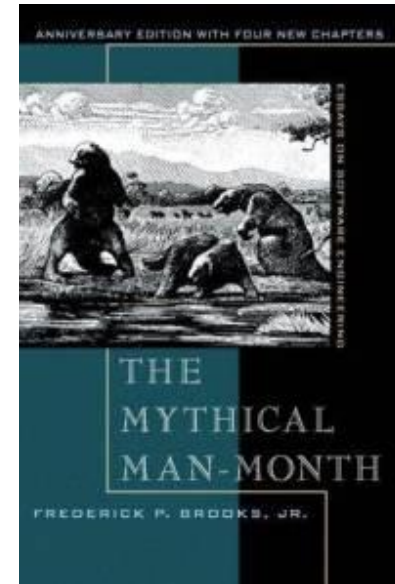
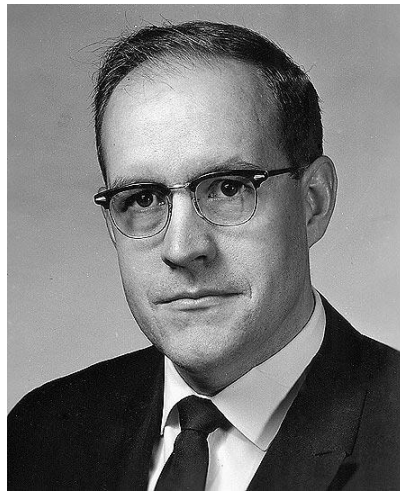


# Anecdote: The complexity of implementing OS

- Even triggers the consideration of **Software Engineering!**
  - IBM 360

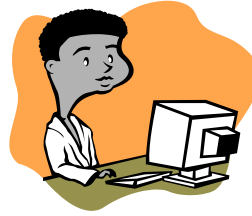


- Frederick Phillips Brooks, “**The Mythical Man-Month**” 《人月神话》





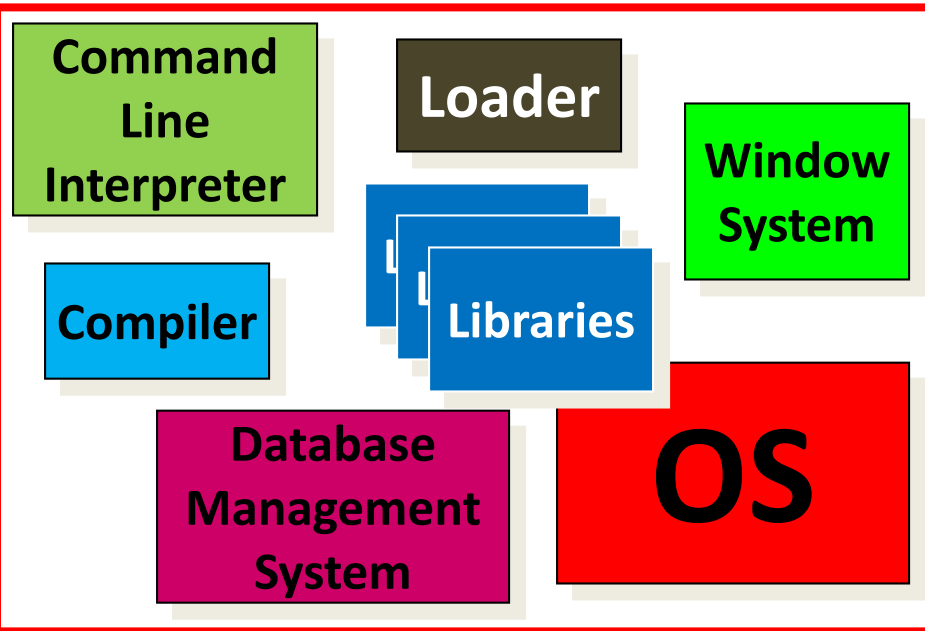
- Problems and ideas when implementing OS
  - Problems: EMM+GSD
  - Ideas: Multiplex resource + Synchronize programs
- Construct OS
  - Evolution of OSs's structures
- How do users use the functions defined in OS?
  - System Call/API ← Just as Function call you've learned
- How to load and run OS?
  - POST → BIOS/CMOS → (Boot) loader
  - Firmware (ROM)



Application  
Programmer

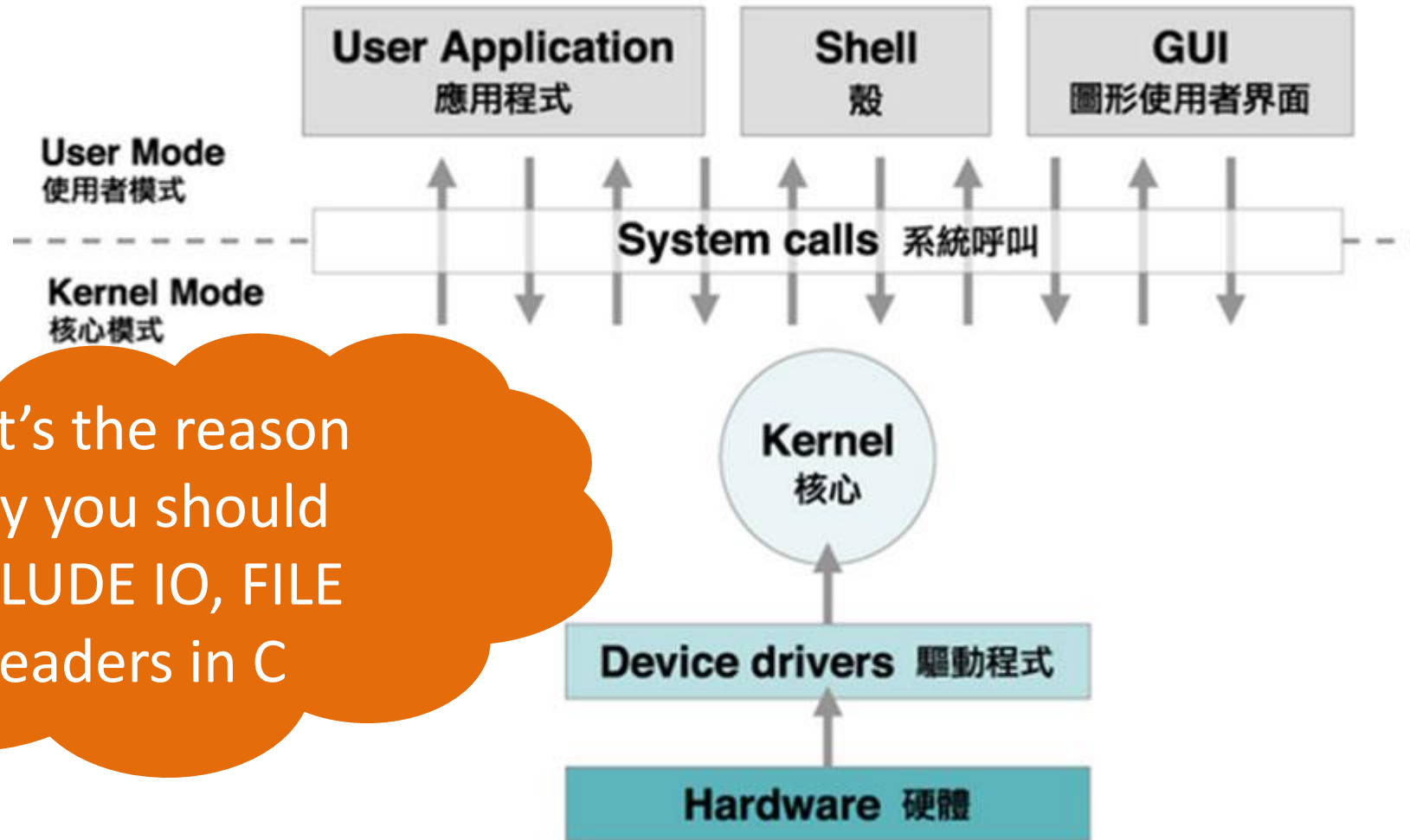
## Software API

System Software



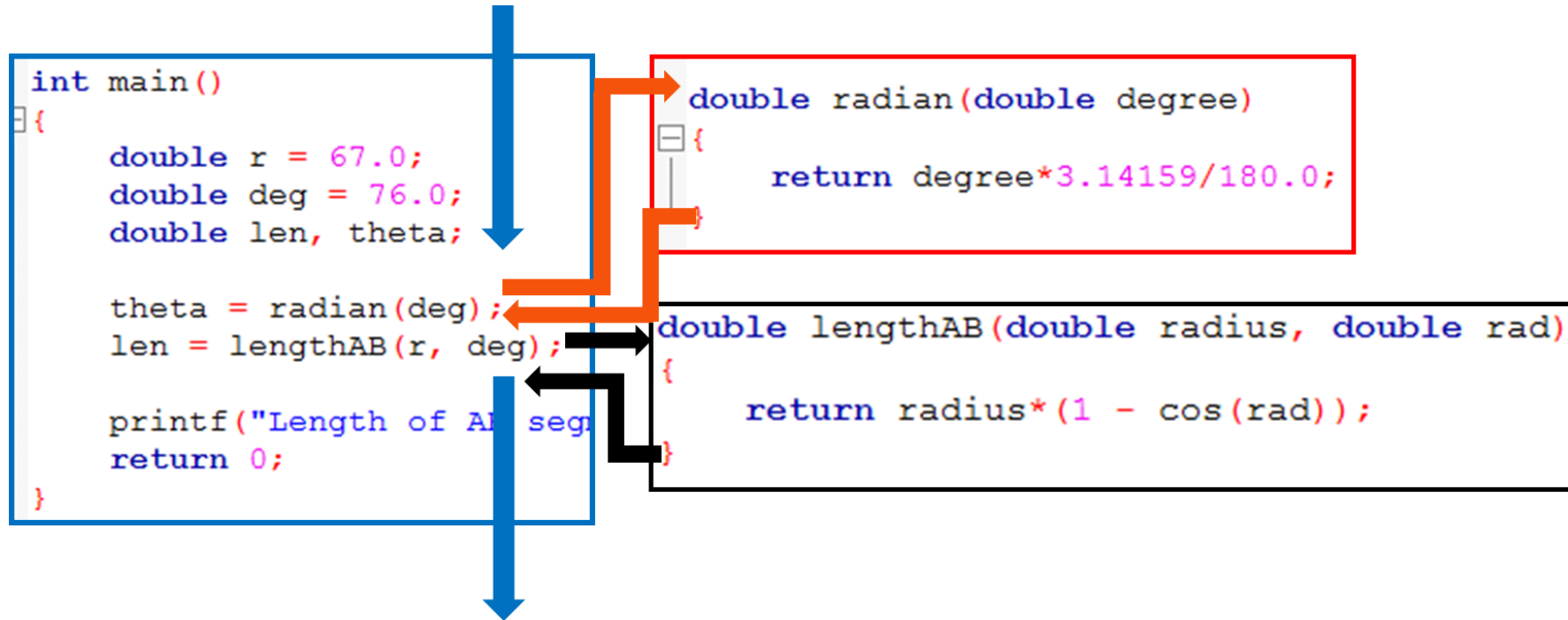
Hardware

That's the reason  
why you should  
INCLUDE IO, FILE  
headers in C



<http://mlinking.blog.163.com/blog/static/18580192220131101314948/>

# You have known something about program

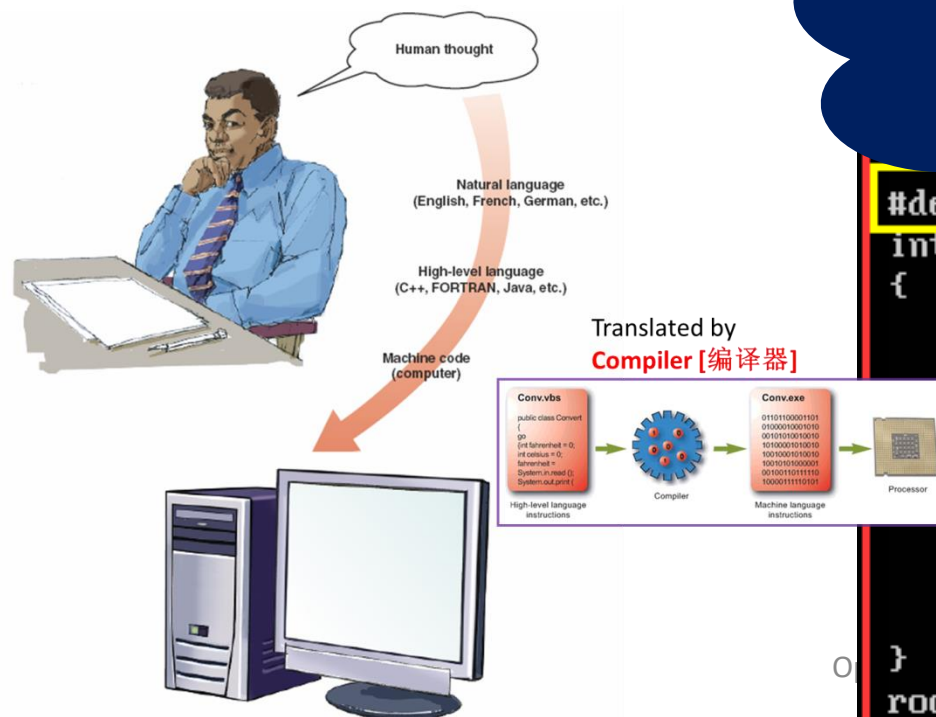


- On the other side, computer can execute the predefined codes/instructions
  - Which could control the circuits and drive the devices to store, print, transfer (network)...

Translation/Mapping  
from source code to  
machine codes/driver  
functions

You need to learn

- Computer organization
- Compiler



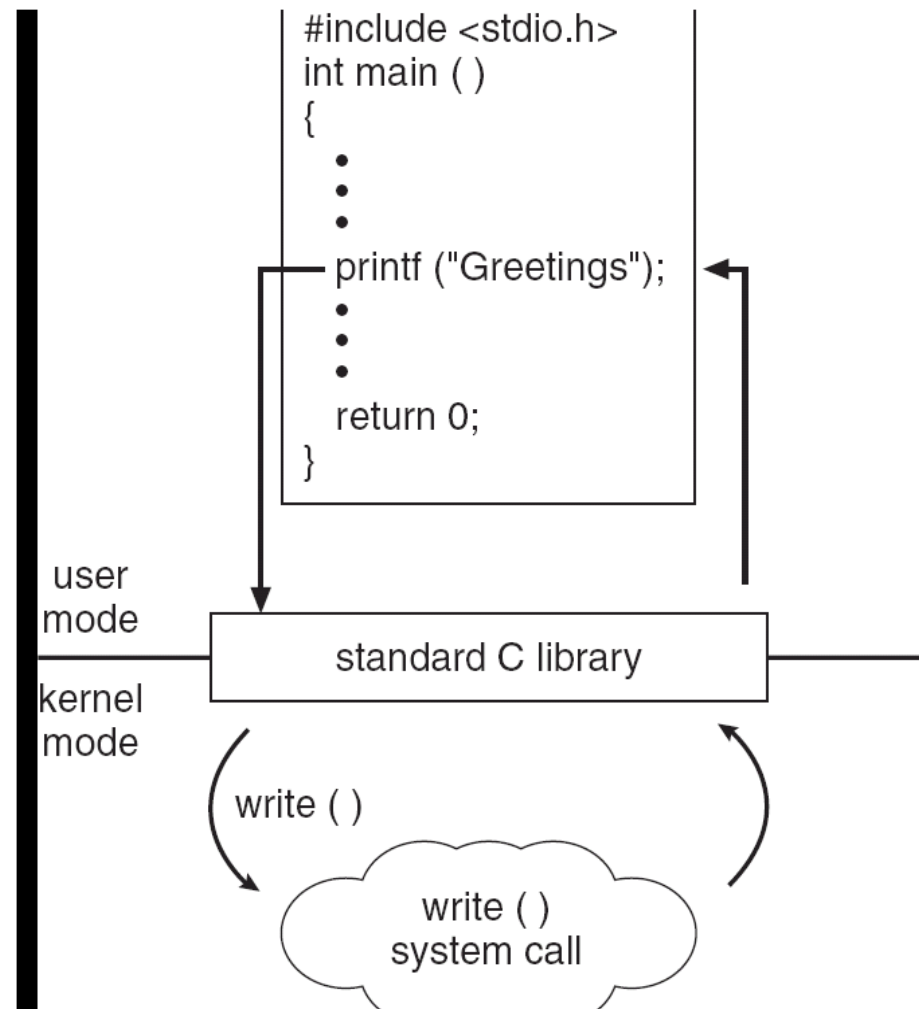
```
#define DEVICE_PATH "/dev/lcd"

int main(int argc, char *argv[])
{
    int fd;
    printf("open function\n");
    fd = open(DEVICE_PATH, O_RDONLY);
    if (fd == -1) {
        perror("open");
    }
    printf("open function finish\n");
    return 0;
}

root@ubuntu:/home/test/drv1#
```

interface between user programs and OS services

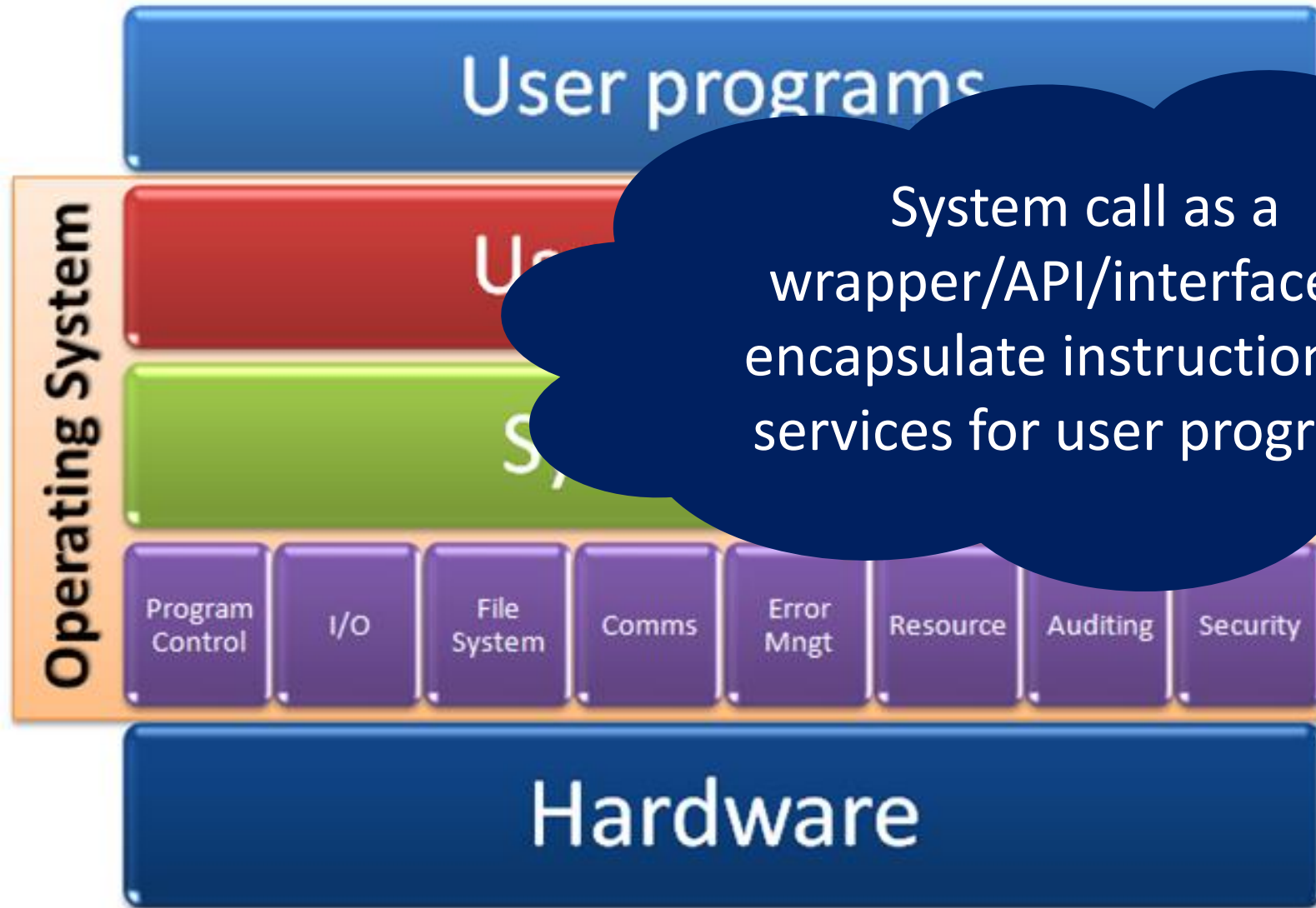
- C program invoking **printf()** library call, which calls **write()** system call



Advice to learn a HPL:

- Besides the learning of the basic programming skills (syntax, function declaration, supported data types & operations, etc.), you should learn how to use the ready-to-use functions provided in predefined libraries.
- Learn the methods used for the problems in ACM, mathematics





System call as a wrapper/API/interface to encapsulate instructions as services for user programs

- Problems and ideas when implementing OS
  - Problems: EMM+GSD
  - Ideas: Multiplex resource + Synchronize programs
- Construct OS
  - Evolution of OSs's structures
- How to use the functions defined in OS?
  - System Call/API
- How to load and run OS?
  - POST → BIOS/CMOS → (Boot) loader
  - Firmware (ROM)

# Before you use your OS

- When you turn on the power of your computer, the ***first program*** that runs is usually a set of instructions kept in the computer's **read-only memory (ROM)**.
  - This code examines the system hardware to make sure everything is functioning properly.
    - 3 subprograms: **POST, BIOS, Boot loader**
      - Sometimes, loader is believed as part of BIOS in some textbooks
    - This **power-on self test (POST)** checks the CPU, memory, and **basic input-output systems (BIOS)** for errors and stores the result in a special memory location (**CMOS**).
    - Once the **POST** has successfully completed, the software loaded in **ROM** (sometimes called the **BIOS** or **firmware** [固件]) will begin to activate the computer's disk drives.

ROM PCI/ISA BIOS (2A69KG0D)  
CMOS SETUP UTILITY  
AWARD SOFTWARE, INC.

**STANDARD CMOS SETUP**

BIOS FEATURES SETUP

CHIPSET FEATURES SETUP

POWER MANAGEMENT SETUP

PNP/PCI CONFIGURATION

LOAD BIOS DEFAULTS

LOAD PERFORMANCE DEFAULTS

INTEGRATED PERIPHERALS

SUPERVISOR PASSWORD

USER PASSWORD

IDE HDD AUTO DETECTION

SAVE & EXIT SETUP

EXIT WITHOUT SAVING

Esc : Quit

↑ ↓ → ← : Select Item

F10 : Save & Exit Setup

(Shift) F2 : Change Color

Time, Date, Hard Disk Type...



CMOS Setup Utility - Copyright (C) 1984-2003 Award Software  
Advanced BIOS Features

First Boot Device	[Floppy]
Second Boot Device	[CDROM]
Third Boot Device	[HDD-0]
Password Check	[Setup]
Full Screen LOGO Show	[Enabled]

Item Help

Mem Level >

Select Boot Device  
Priority

[Floppy]  
Boot from floppy

[LS120]  
Boot from LS120

[HDD-0]  
Boot from First HDD

[HDD-1]  
Boot from second HDD

↑↓→←: Move   Enter: Select   +/-/PU/PD: Value   F10: Save   ESC: Exit   F1: General Help  
F5: Previous Values   F6: Fall-Safe Defaults   F7: Optimized Defaults

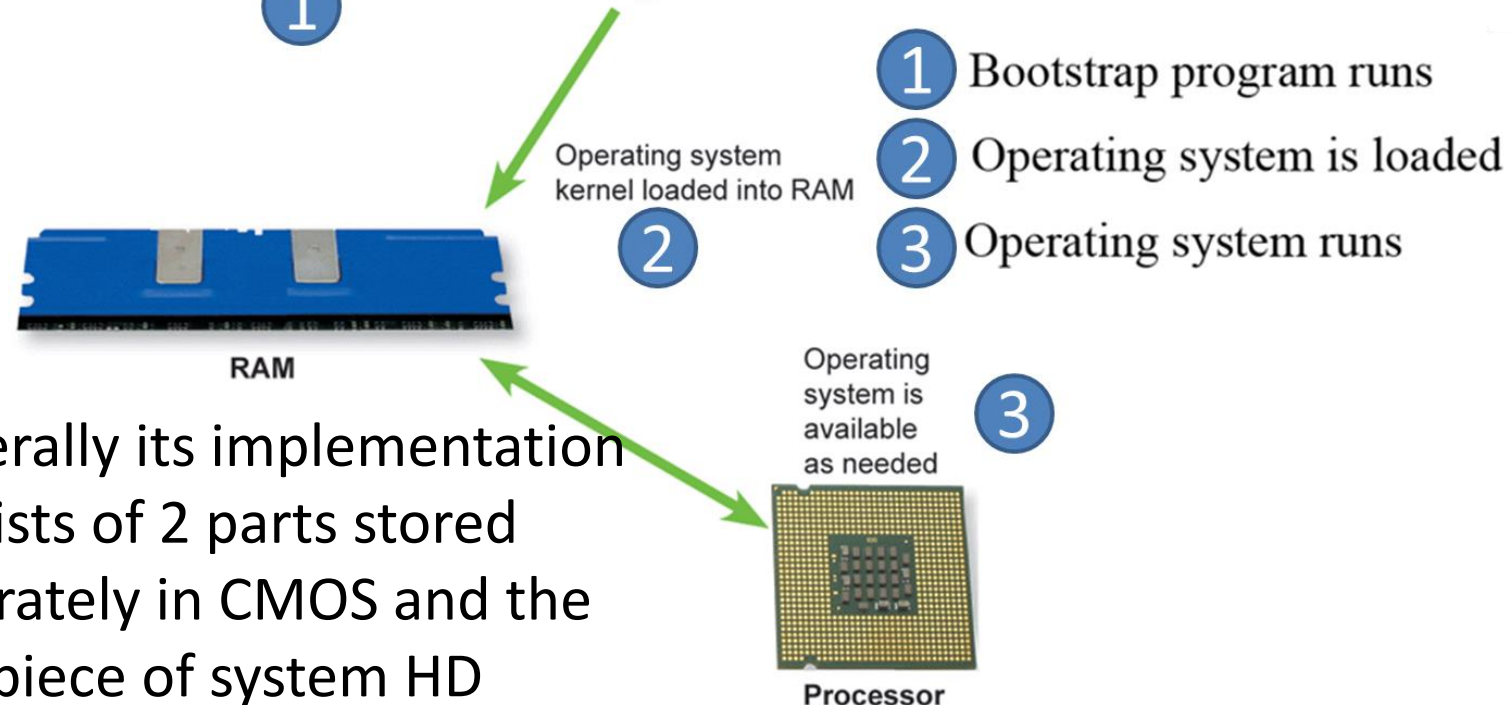
# Before you use your OS

- The **bootstrap loader** is a small program that has a single function: **It loads the operating system into memory and allows it to begin operation.**



**FIGURE 4-13**

The bootstrap program copies the operating system into RAM, where it can be directly accessed by the processor to carry out input, output, or storage operations.



Generally its implementation consists of 2 parts stored separately in CMOS and the first piece of system HD

- Then it's OS!

