

操作系统

第1章 操作系统引论

朱小军

南京航空航天大学计算机科学与技术学院

2025年春

操作系统的源起与发展

(方便与效率)

没有操作系统的时代



ENIAC, 1946

冯诺依曼计算机体系结构

- 存储程序的思想



EDVAC

此时的计算机类似于组原课设

- 你制造出了什么？
 - 支持若干条指令的CPU
 - 外加总线、内存、输入设备、输出设备构成一台计算机
- 但，程序员写程序很不方便
 - 不同设备的操作方式不一样
 - 同样是输出设备，磁带、软盘、硬盘、显示器的机器指令不一；处理所有情况会导致程序庞大
- 为了方便程序员编程，提出了操作系统概念
 - 将最底层操作硬件设备的代码封装，提供统一的接口；即提供一个虚拟机

《CODE》一书观点

追求效率

(计算机昂贵、人工廉价的年代)

如果多人需要使用计算机？

- 人工操作阶段：排队



Name	Tape no.	Resources
Alice	000123	...
Bob	020132	...

- 缺点
 - 手动更换任务，耗时易出错
 - 计算机被闲置！

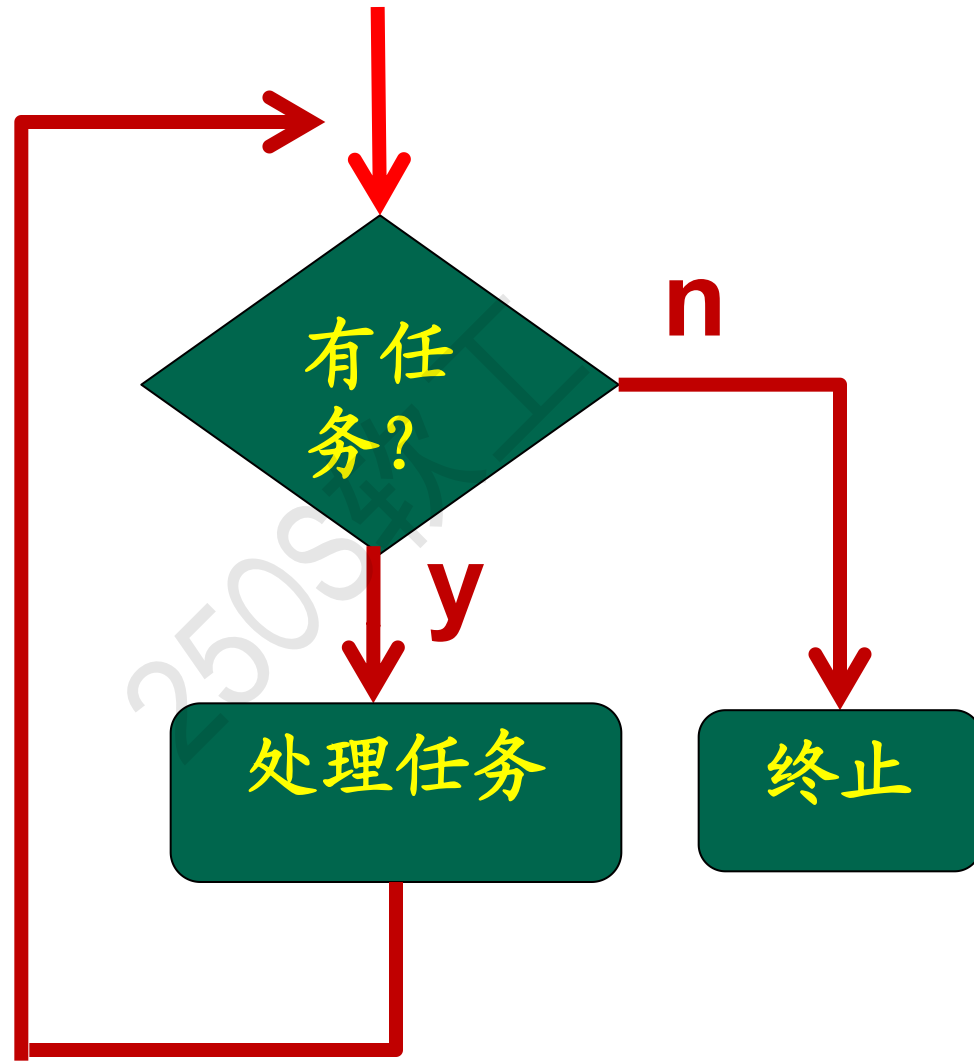
解决策略？

单道批处理系统



Name	Tape no.	Resources
Alice	000123	...
Bob	020132	...

单道批处理系统



单道批处理系统如何实现？

- 批处理和用户任务共享内存
- 如何避免用户任务修改操作系统？



效率还有提升空间吗？

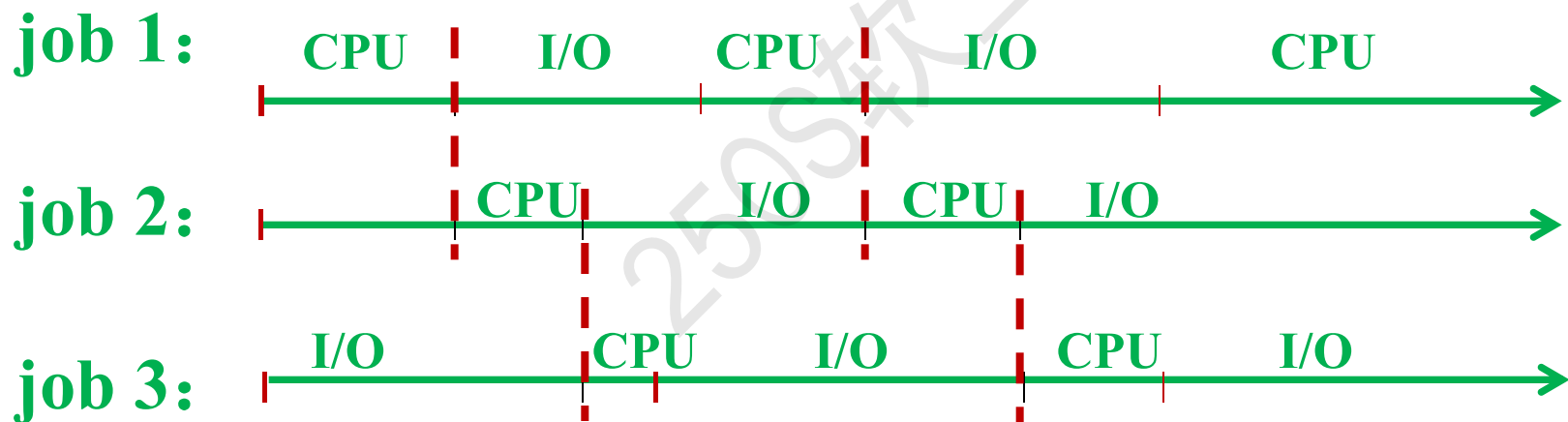
- 当一个任务在执行IO操作时，CPU是空闲的
- 怎么可以让CPU闲置！

多道批处理系统

内存



执行时序



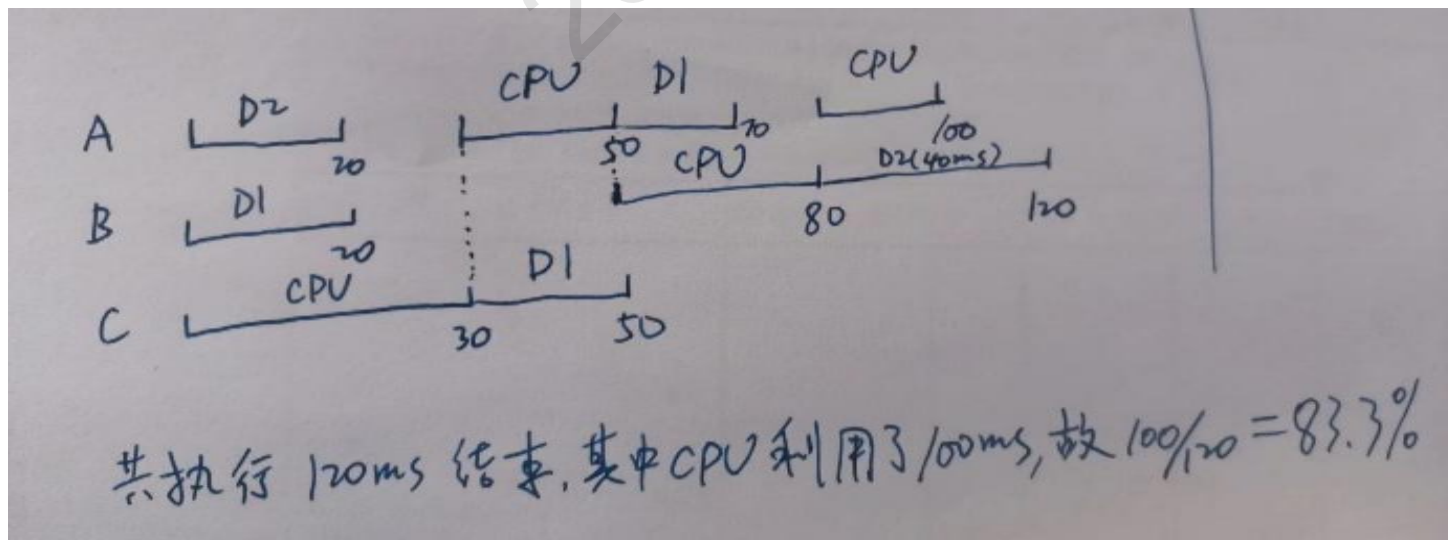
- CPU利用率高: CPU的使用时长/总时长 (注意, 本课程只考虑单CPU)
- 给定任务, 计算CPU利用率, 是常考的计算题

例题（某年期末考试题）

某计算机系统中有一个CPU，一台输入设备D1和一台输出设备D2。现有三个任务（优先级A大于B，B大于C）：

- 进程A的运行轨迹为：D2（20ms）、CPU（20ms）、D1（20ms）、CPU（20ms）；
- 进程B的运行轨迹为：D1（20ms）、CPU（30ms）、D2（40ms）；
- 进程C的运行轨迹为：CPU（30ms）、D1（20ms）。

CPU的利用率是83.3%。（保留1位小数）



如何实现多道批处理系统？

- 内存同时容纳多个任务
 - 写程序的时候可能不知道会分配到内存哪个区域，怎么处理？
 - 一个程序出错会否影响到另一个程序？
- 选择哪个任务运行？
 - 万一有一个任务耗时很长，后续的短任务怎么办？

多道批处理系统的缺点：

- 如果多人共享一台计算机，则用户感觉到的周转时间较长
- 如果一个程序员在调程序，同时有物理、天文等工作人员提交了计算任务。。。

分时操作系统

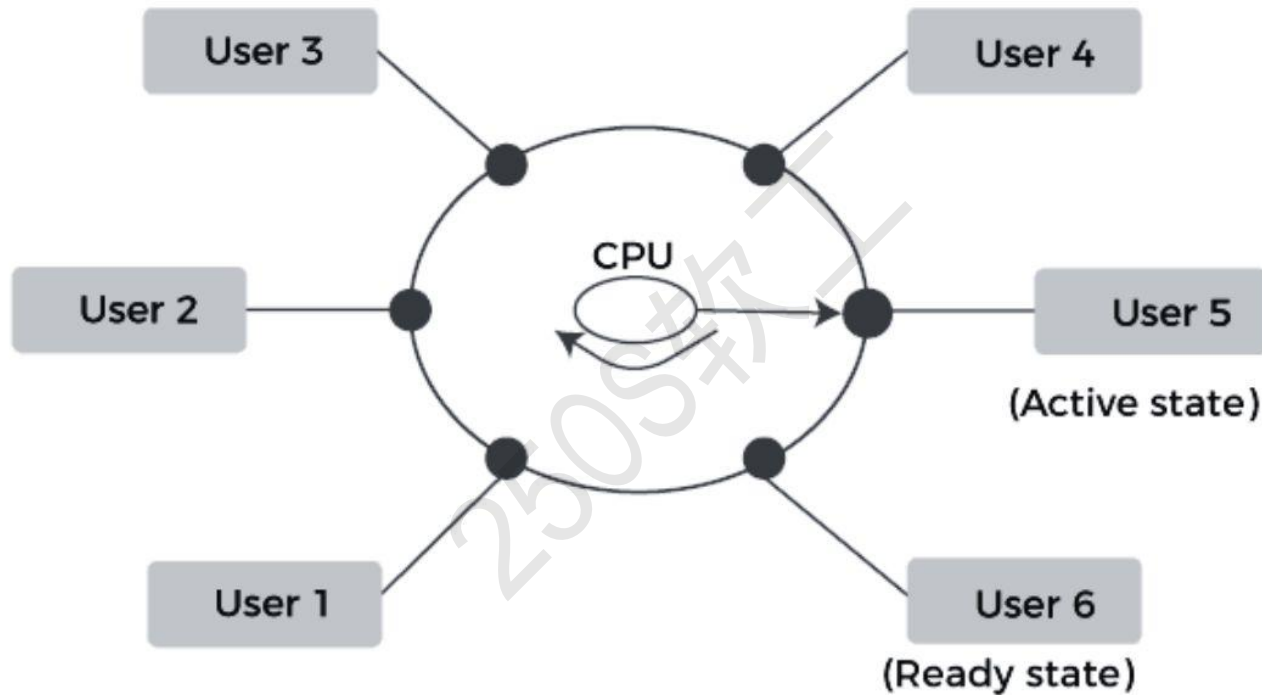
(当时称为交互式系统)

(计算机逐渐便宜、人工逐渐昂贵)



John McCarthy (1927-2011)
Turing Award (1971)

分时操作系统的基本思想



提问：和多道批处理有何不同？

分时操作系统引入的新问题？

- 如何打断任务的运行？
- 因为随时可能被打断，所以会出现一系列并发错误，如死锁

小结

- 裸机→(库函数?)操作系统
 - 单道批处理操作系统
 - 多道批处理操作系统
 - 分时操作系统
- 编译输出的是能在操作系统上运行的二进制代码
- 操作系统就是当年换磁带的那个人
- 每次操作系统升级都会带来新的问题
- 本课程处理的问题到分时操作系统为止

不要反复造轮子

不能闲置计算机!

不能闲置CPU!

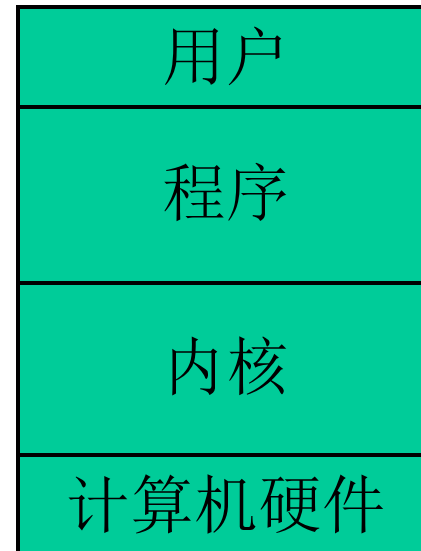
不能闲置程序员!

所以，什么是操作系统？！！！！

什么是操作系统？

- 操作系统（OS）的例子？
- 没有一个广泛接受的定义
- **“Acts as an intermediary between a user of a computer and the computer hardware.”** ——
《Operating Systems Concept》

OS处于用户与
计算机硬件之间





You

what is operating system



ChatGPT

An operating system (OS) is a software program that acts as an intermediary between computer hardware and software applications. It manages the hardware resources of a computer system, provides a platform for software applications to run, and facilitates user interaction with the computer.

Here are some key functions and components of an operating system:

1. **Process Management:** The OS manages processes, which are instances of executing programs. It allocates CPU time to processes, schedules them for execution, and provides mechanisms for inter-process communication and synchronization.
2. **Memory Management:** The OS manages system memory, allocating memory to processes as needed and ensuring efficient utilization of available memory resources. This includes virtual memory management, which allows the OS to use disk space as an extension of physical memory.
3. **File System Management:** The OS provides a file system that organizes and stores data on storage devices such as hard drives, SSDs, and flash drives. It provides mechanisms for creating, reading, writing, and deleting files, as well as managing directory structures and access permissions.
4. **Device Management:** The OS manages input and output (I/O) devices such as keyboards, mice, displays, printers, and network interfaces. It provides device drivers to facilitate communication between hardware devices and software applications.
5. **User Interface:** The OS provides a user interface that allows users to interact with the computer system. This can range from a command-line interface (CLI) where users enter text commands to a graphical user interface (GUI) with windows, icons, menus, and pointing devices.
6. **Security:** The OS enforces security policies to protect the system and its data from unauthorized access, malware, and other threats. This includes user authentication, access control mechanisms, encryption, and security updates.
7. **Networking:** The OS provides networking capabilities, allowing the computer to connect to other computers and devices over local area networks (LANs) or the internet. This includes protocols for communication, network configuration, and network services.

两种流行的观点

- 自顶向下的观点：OS实现了对计算机资源的抽象，用户使用的不是作为硬件设备的机器，而是OS 所提供的接口；
- 自下往上的观点：OS 是资源的管理者，让计算机中的各类资源，如CPU、存储器、I/O设备以及文件等得到充分的利用

操作系统的特权

需求分析

- 操作系统需要常驻内存，并且要保护硬件资源
- 到分时操作系统时，操作系统还需要有打断其他正在运行程序的能力
 - 如何打断？
 - 硬件需要提供特殊能力

特权1：可以执行特权指令

- 指令集中，部分指令很危险，划分为特权指令，比如读写设备的指令in、out
- 只有操作系统允许执行特权指令
- 类比：
 - 在我国，持枪是一种危险行为（特权指令），只有警察（操作系统）允许执行
 - 警察持枪和幼儿园小朋友持枪，哪个让人害怕？

如何实现特权指令？

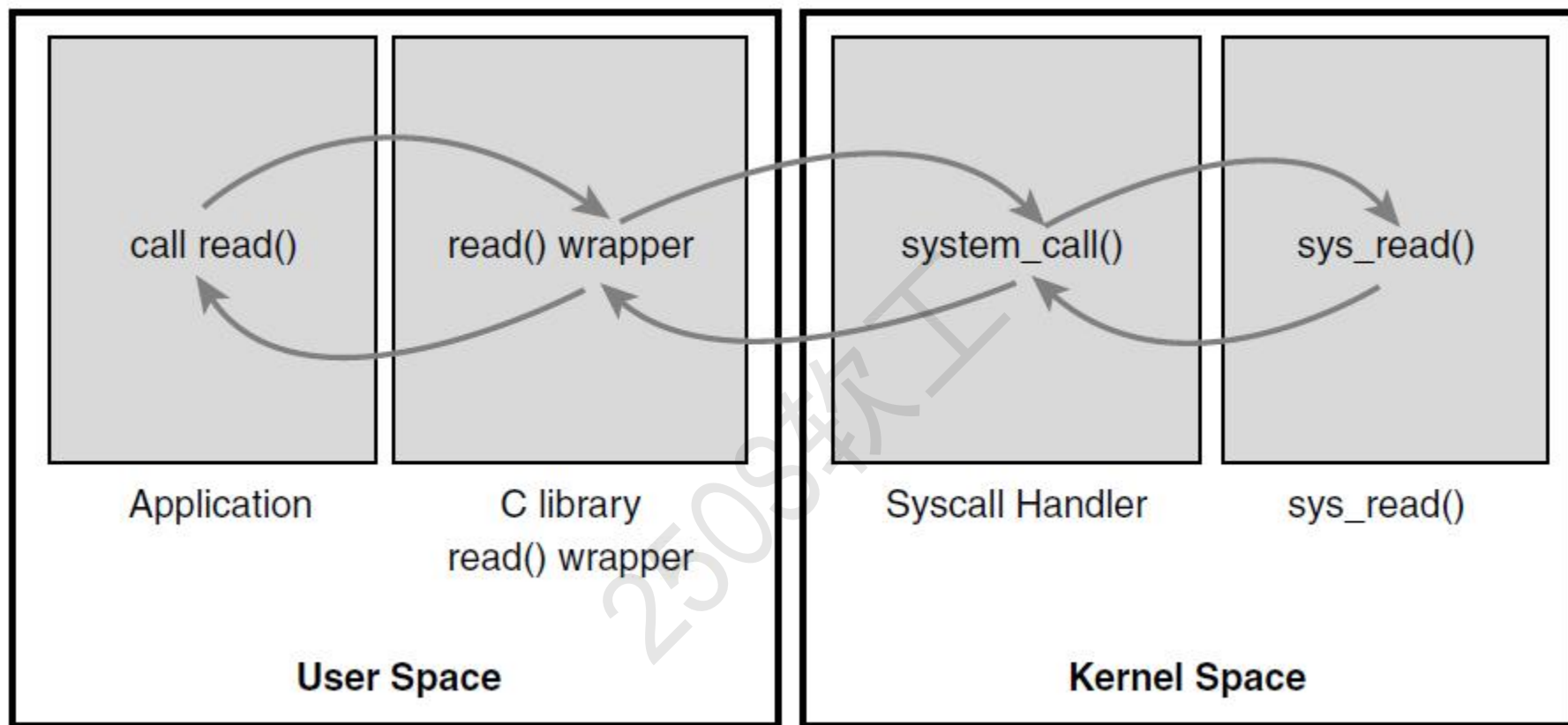
- CPU的运行模式：内核态与用户态
- CPU在内核态时，允许执行所有指令
- CPU在用户态时，只能执行非特权指令
 - 应用程序所用到的指令均为非特权指令
 - 如果应用程序直接使用特权指令，报错

```
1 #include "stdio.h"
2
3 void main(){
4     printf("hello\n");
5     asm("cli");
6     printf("world\n");
7 }
```

用户程序需要服务如何实现？

- 若用户程序的功能需要特权指令，如IO？
 - 向操作系统发出请求，称为系统调用
 - 发请求的指令是特权指令还是非特权指令？
 - 不同操作系统提供的系统调用不一样（这是应用程序不能跨平台运行的重要原因）
- Windows程序能在Linux下直接运行吗？报什么错？
 - WSL是一个特殊的Linux系统，可以运行windows程序

系统调用-Linux中的典型实现



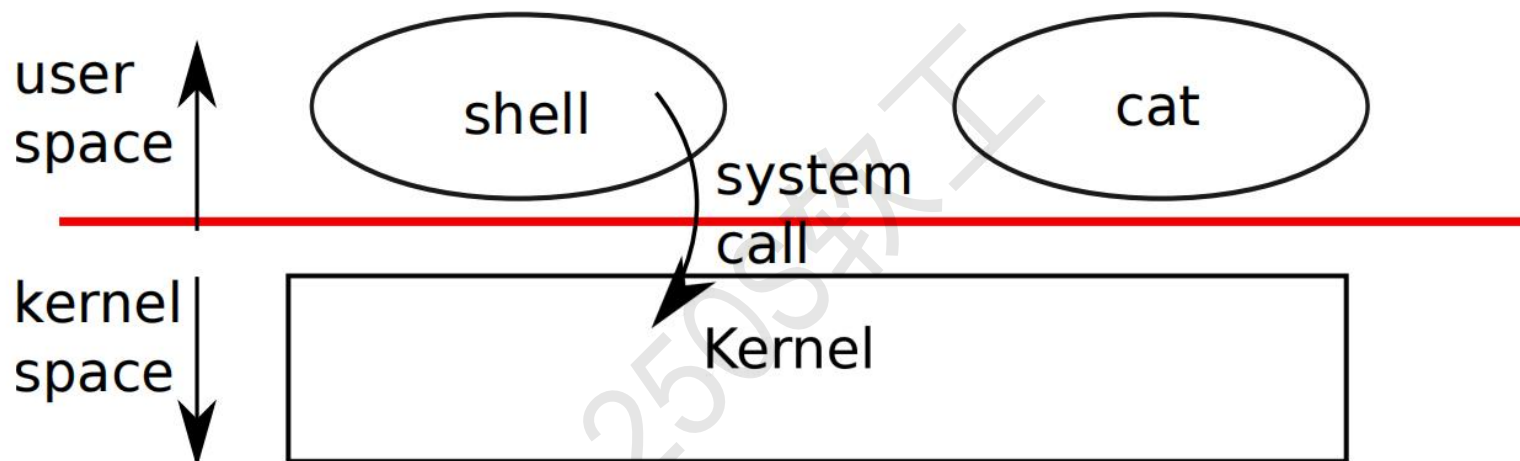
CPU运行在用户态

CPU运行在内核态

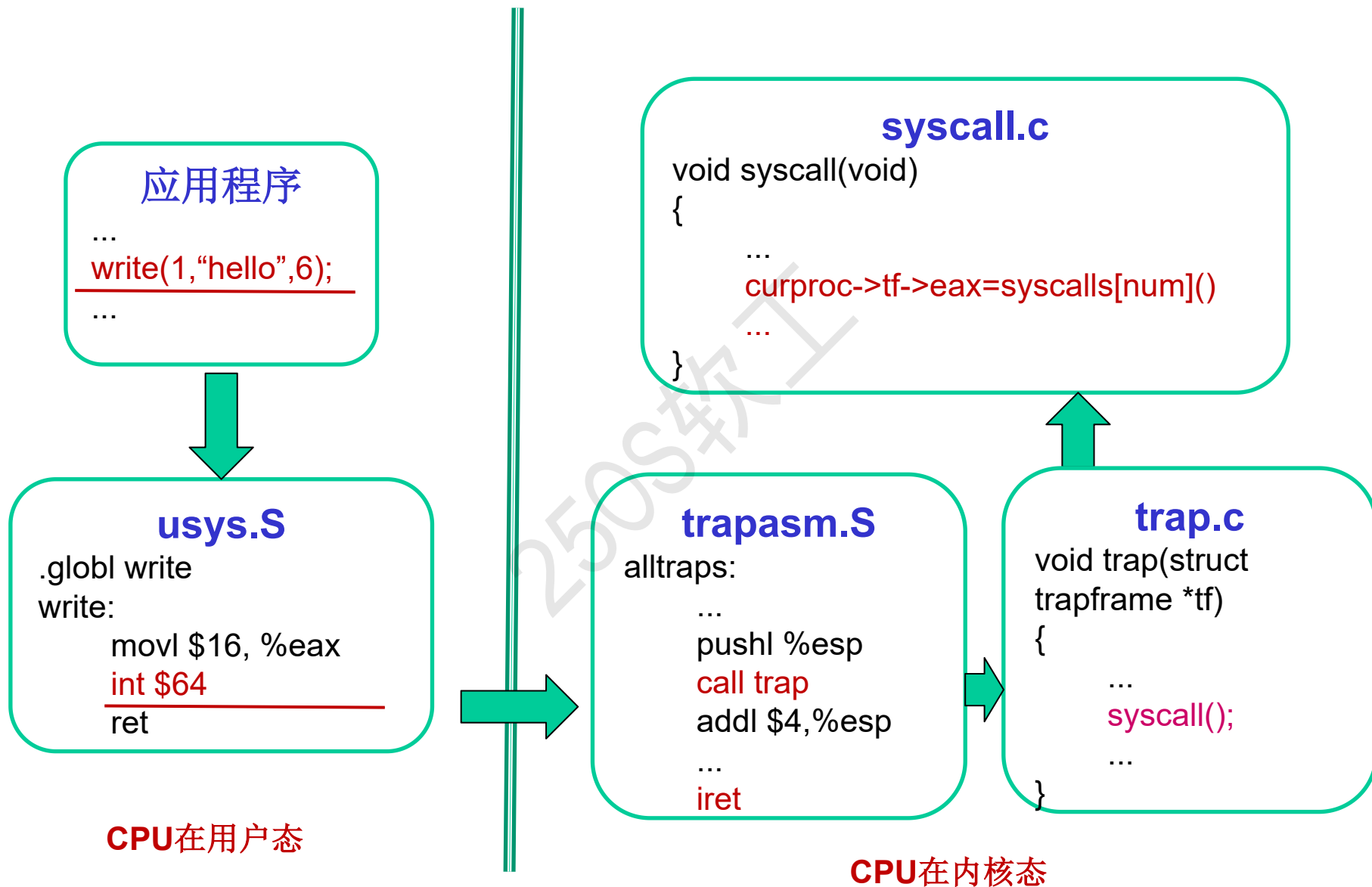
xv6操作系统提供的系统调用

System call	Description
fork()	Create process
exit()	Terminate current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return current process's id
sleep(n)	Sleep for n seconds
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process's memory by n bytes
open(filename, flags)	Open a file; flags indicate read/write
read(fd, buf, n)	Read n bytes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd's in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file

xv6中的系统调用



xv6中的系统调用案例



特权2：处理中断

- 中断发生时，由操作系统先处理，再转发给相应的用户程序，例如：键盘上按下一个键
 - 回忆：中断控制器、中断处理程序（设备驱动程序提供）、应用程序
 - 上述中断为**硬件中断**（interrupt）
 - 软件中断，又称为**异常**、陷入（trap），机制与硬件中断类似，可用于实现系统调用
- 观看B站视频演示 <https://b23.tv/az8Pxqo>

中断演示视频

nomanker bilibili

当你在键盘敲下一字符时

特权2：处理中断

- 如果操作系统要“剥夺”一个程序的CPU，让CPU去执行另外一个程序，如何做到？
 - 让定时器周期性产生时钟中断

内容回顾&学生提问

- 操作系统的起源（库函数，追求效率）
- 操作系统是什么（给用户接口，管理硬件、软件）
- 操作系统的特权（特权指令、管理中断）
- 学生提问：
 - 例子中，user space是指用户态吗？ 是
 - 中断视频中，都是CPU在执行，操作系统起了什么作用？ CPU在执行操作系统代码

操作系统的运行

- 计算机启动时(xv6为例)
 - 引导程序 (bootloader)
 - 硬件部分 (ROM固件, BIOS/UEFI) : 磁盘第0扇区加载到内存
 - 软件部分 (bootasm.S, bootmain.c) : 加载内核到内存, 作用类似于U-Boot, GRUB
 - 提问: 为何要有硬件部分? 为何要有软件部分?
 - 内核初始化 (main.c) : 控制台、中断初始化 (定时器)、进程管理、创建第1个用户进程、调度器
 - 调度器选择用户进程, CPU切换到用户态, 运行用户程序

启动后的操作系统在做什么？

OS的核心： `while(1) { select a task; execute it; }`

以
x
v
6
为
例

main.c

```
int main(void)
{
    ...
    userinit();
    mpmain();
}
```

main.c

```
void mpmain(void)
{
    ...
    scheduler();
}
```

proc.c

```
void scheduler(void)
{
    ...
    for(;;){
        //select a process
        //switch context to run it
        //switch back (when?)
    }
}
```

提问：上面代码中，**CPU**在用户态还是在内核态？

操作系统的运行

- 中断（硬中断或软中断）来临时

- CPU从用户态升级为内核态（硬件）

- 中断处理程序（软）

- OS重新掌握CPU

- 处理完毕后CPU降级为用户态

- 操作系统。。。

- 是否类似于事件驱动的编程？

```
trapasm.S
1      #include "mmu.h"
2
3      # vectors.S sends all traps here.
4      .globl alltraps
5      alltraps:
6      # Build trap frame.
7      pushl %ds
8      pushl %es
9      pushl %fs
10     pushl %gs
11     pushal
12
13     # Set up data segments.
14     movw $(SEG_KDATA<<3), %ax
15     movw %ax, %ds
16     movw %ax, %es
17
18     # Call trap(tf), where tf=%esp
19     pushl %esp
20     call trap
21     addl $4, %esp
22
23     # Return falls through to trapret...
24     .globl trapret
25     trapret:
26     popal
27     popl %gs
28     popl %fs
29     popl %es
30     popl %ds
31     addl $0x8, %esp # trapno and errcode
32     iret
```

类似的模式：python的GUI编程

```
import tkinter as tk
from tkinter import ttk

# Create the application window
window = tk.Tk()

# Create the user interface
my_label = ttk.Label(window, text="Hello World!")
my_label.grid(row=1, column=1)

# Start the GUI event loop
window.mainloop()
```

类似的模式：网络仿真

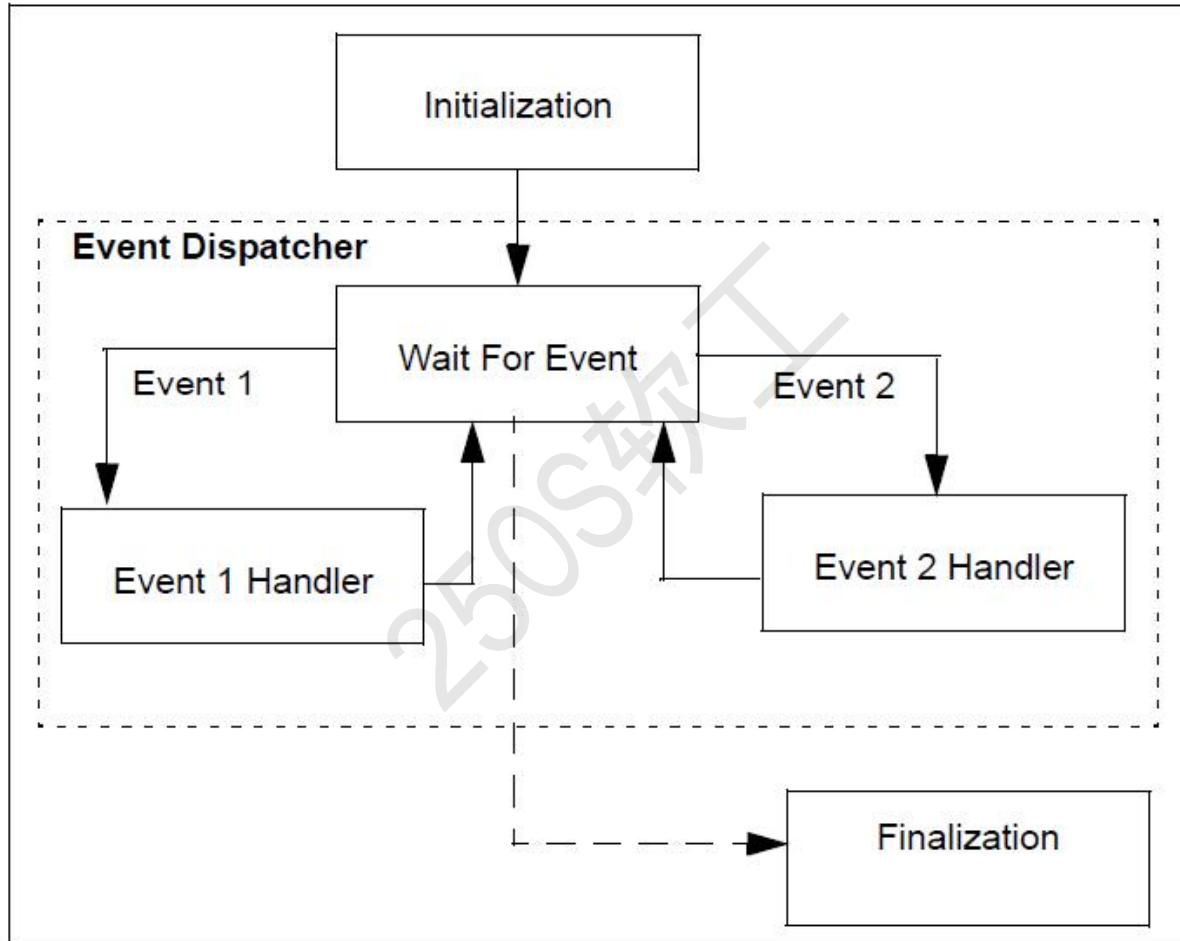


FIGURE 3-1. Protocol Model in EXata

xv6操作系统

- 目录结构
- 系统调用
- 演示

250S软工

操作系统的结构

操作系统控制复杂性的方法

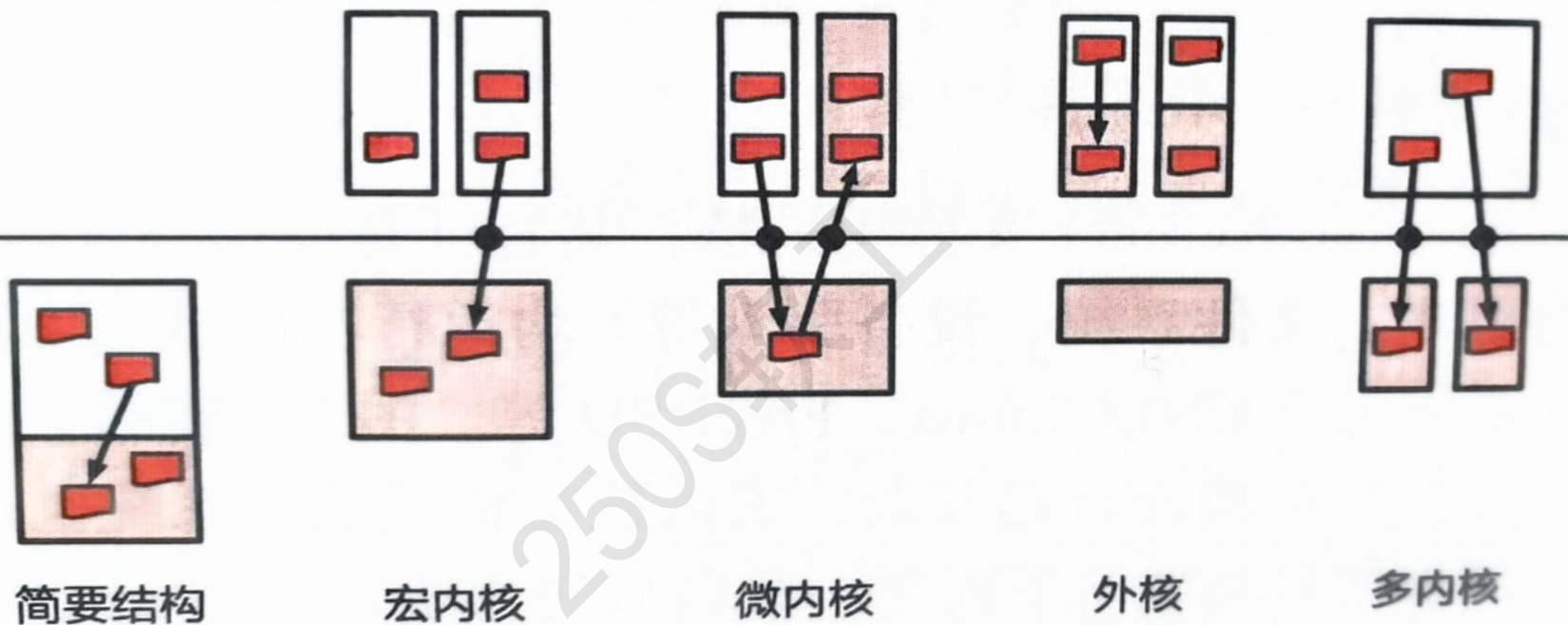
- M.A.L.H方法: Modularity, Abstraction, Layering, Hierarchy
- 模块化: 分而治之, 高内聚低耦合
 - 如进程管理、内存管理、网络协议栈, 设备驱动
- 抽象: 对模块的接口和内部实现分离
 - 如虚拟内存对物理内存提供了抽象, 文件系统对物理介质的抽象
- 分层: 将模块按层次划分, 同层和相邻层交互
 - 如, 内核层、系统服务层、应用框架层
- 层级: 同一层中模块进一步分级
 - 例如, 内核层进一步细分

常见的OS内核架构

→ 调用 ●→ 模式切换 ■ 代码 ■ OS □ 应用程序

用户态

内核态



MS-DOS,
FreeRTOS,
μC/OS

Linux、Unix、
xv6

Mach, Minix

LibOS

现代操作系统的基本特征

并发 (Concurrency)

并行性：两个或多个事件在同一时刻发生

并发性：两个或多个事件在**同一时间间隔**内发生

共享 (Sharing)

系统中的资源可供内存中多个并发执行的进程共同使用

虚拟 (Virtual)

虚拟是一种特殊的抽象：**如果抽象提供的接口和原接口一致可称为虚拟**，比如，虚拟内存和内存都使用mov指令操作

异步 (Asynchronism)

进程之间互相不等待（不同步）

导致：进程是以人们不可预知的速度向前推进的

案例类比：云优盘、优盘；
虚拟机

Linux编程

- **Linux**下编写和运行程序

`gcc test.c`

– 默认生成的可执行文件是**a.out**

- 如果要求可执行程序是**32**位，名为**test**？

`gcc test.c -m32 -o test`

- 如果**test.c**需要调用**A.c**中的函数？

`gcc test.c A.c -m32 -o test`

- 如果。。。。

`gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall
-MD -ggdb -m32 -Werror -fno-omit-frame-pointer ...`

- 记不住

使用make

- 将命令写到一个名为**Makefile**的文件中

test: test.c A.c

gcc test.c A.c -m32 -o test

- 目标、依赖文件、产生目标需要执行的命令
- 不明确指明目标时，执行第一个
- 只编译修改过的文件

课程内容安排

内容安排

- 管理CPU
 - 进程管理，第2、3、4章
- 管理内存
 - 内存管理，第5、6章
- 管理外存
 - 存储管理，第7、8、9章

本章要求（2025考研大纲）

- 操作系统的基本概念
- 操作系统的发展历程
- 程序运行环境
 - CPU运行模式：内核态、用户态
 - 中断与异常的处理
 - 系统调用
- 操作系统的结构
 - 分层，模块化，宏内核，微内核，外核
- 操作系统引导

向你最熟悉的大模型提问：用一句话总结操作系统内核和内核态的区别。