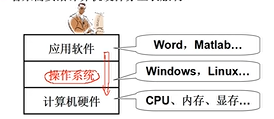
哈工大李志军操作系统：

操作系统要管理什么：

、

CPU管理、内存管理、终端管理、磁盘管理、文件管理

操作系统向上为上层提供接口，向下管理硬件

计算机是如何工作的：

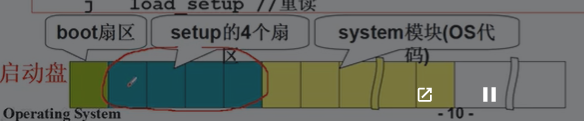
取指执行：存储器(代码）--> 控制器（执行）-->输出结果

打开电源，计算机首先做什么：初始情况下cpu是实模式（16位）

对于X86，内存中会固化一段代码（BIOS）,开机时CS（段寄存器）IP（偏移寄存器）指向BIOS地址，进行内存检测，

将磁盘0磁道0扇区的代码（操作系统的引导代码bootsect.s）读入内存0x07c0。

设置cs=0x07c0，ip=0x0000



Setup.s

完成os启动前的设置--内存，显卡等

将整个操作系统转移到0地址去

进入保护模式--增加寻址空间 GDT表  
system模块代码

Head.s

(栈向下增长)

各种初始化

Mmap 是内存的数组，指示哪些内存使用了未使用

IDT中断向量表

GDT

开机启动，将操作系统代码移动到内存0地址处，可以取指执行

操作系统接口：

系统提供的，连接应用程序和硬件的函数（让函数进入内核的函数）--系统调用system call

任务管理：fork，execl，pthread\_creat

文件系统：open,EACCES 等

统一的接口标准 POSIX

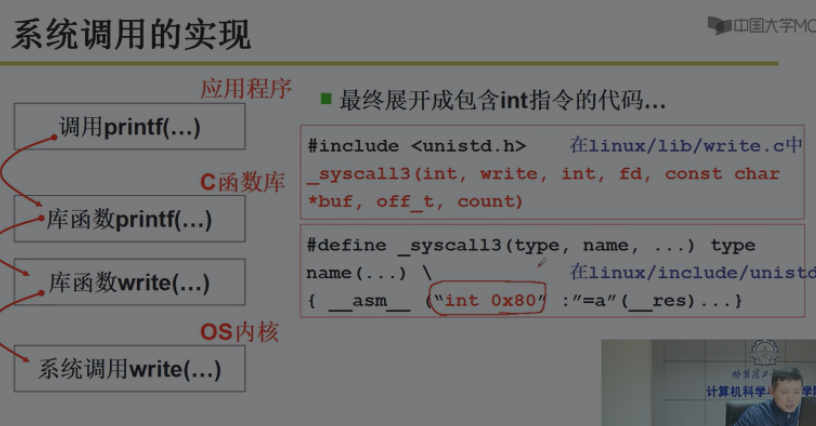
如何实现:

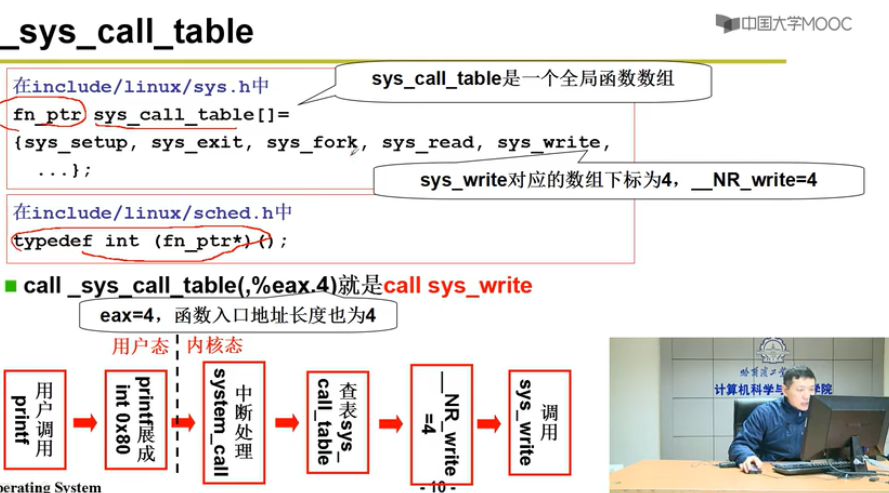
1 内核态 用户态



DPL 目标特权级 CPL 目前特权级 --实现用户态不可以直接访问内核态 -硬件实现

通过终端让系统调用进入内核：





操作系统：管理硬件的软件

进程view：CPU 内存

文件view：磁盘管理 终端管理

CPU管理1：

进程：运行着的程序

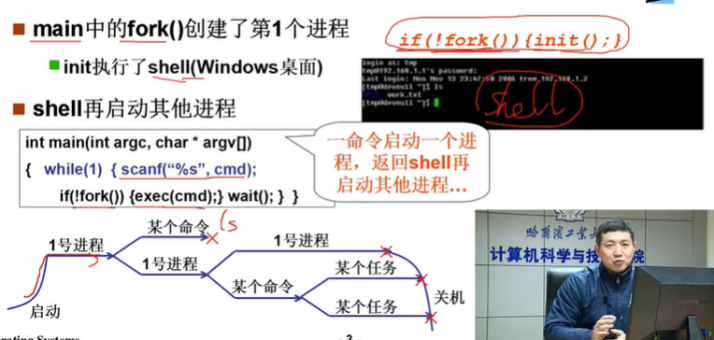
IO设备速度慢，单进程执行时CPU效率低

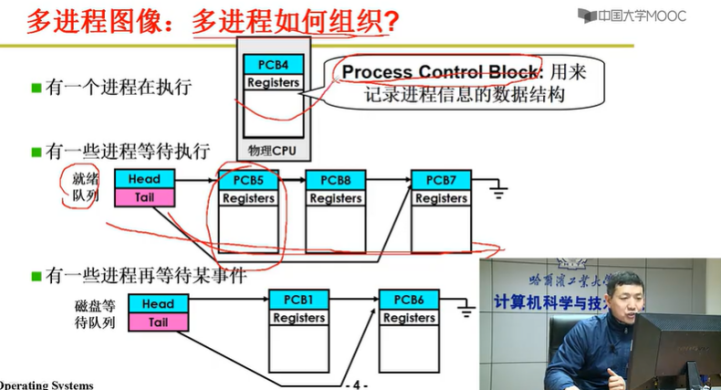
并发：一个CPU上交替执行多个程序--让CPU 高效的执行，减少IO等待

进程切换需要保存现场--PCB 进程控制快process control block

进程的创建：

Main函数中fork了第一个进程(1号进程)，启动了shell



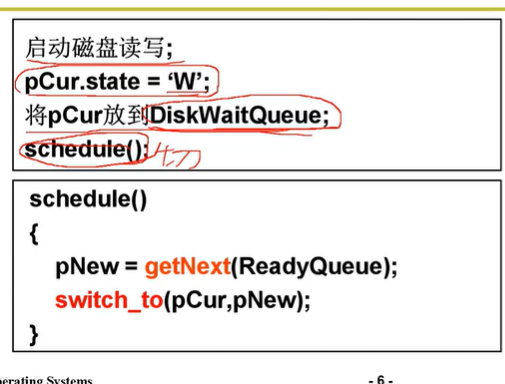


用PCB进行管理，每个进程都有一个PCB，保存现场

一个进程正在执行

就绪队列：

等待（阻塞）队列：



队列操作+调度+切换

进程同步：

加锁

线程：thread

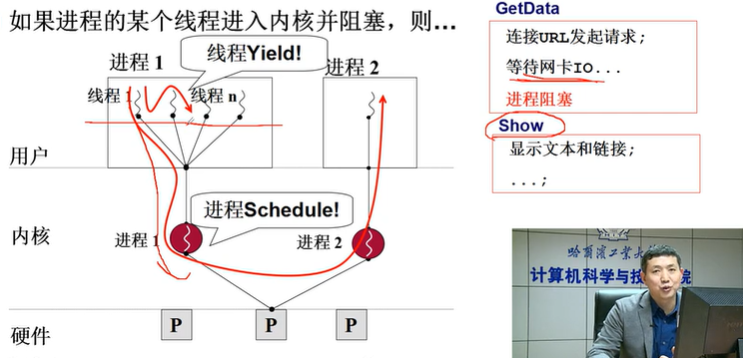
进程=资源+指令执行序列

进程切换时需要切换资源，消耗大，因此引入了线程，在切换线程时只需要切换指令，消耗小。Pthread\_create()

线程有自己独立栈空间，防止函数调用时在先线程之间乱切

TCB 线程控制块，存放栈空间的位置

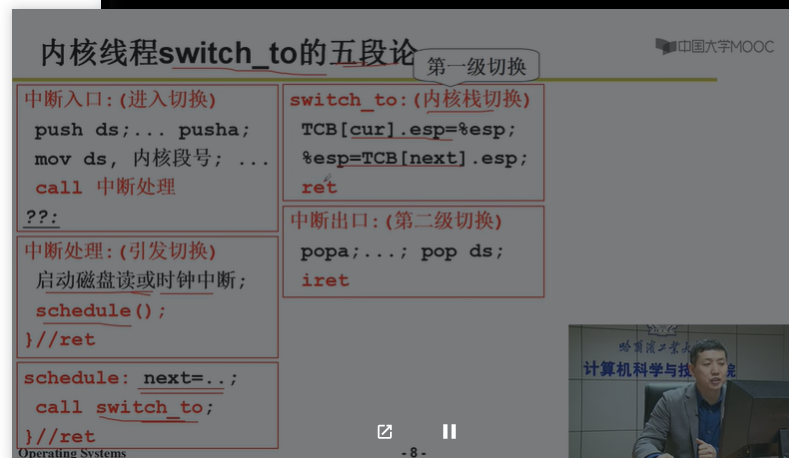
用户级线程通过yield和creat进行线程切换和创建--内核阻塞了，用户多线程没用



内核级线程：

多核--实现并行





CPU调度

响应时间小-->切换多-->内耗多-->吞吐量小

折中--响应时间和周转时间

FIFO 先来先服务（FCFS）

短作业优先（SJF）

时间片轮转调度（round robin）--保证了响应

优先级调度--导致饿死 优先级动态升高+时间片轮转

进程同步：

信号量

P V操作

P(SEM) 信号量-1，消耗一个资源 如果信号量《0 sleep；

V（sem） 信号量+1,欢笑一个进程，如果信号量<=0 wakeup



信号量和临界区

信号量操作不是原子操作，共享数据在竞争中发生错误

解决办法：加锁

