

# 操作系统概论

李中国

苏州大学计算机科学与技术学院

# 虚拟机软件及Ubuntu操作系统下载地址

虚拟机软件**VirtualBox**: [www.virtualbox.org](http://www.virtualbox.org)

**Ubuntu**操作系统: [www.ubuntu.com](http://www.ubuntu.com)

# 安装Ubuntu 11.10操作系统: ISO文件

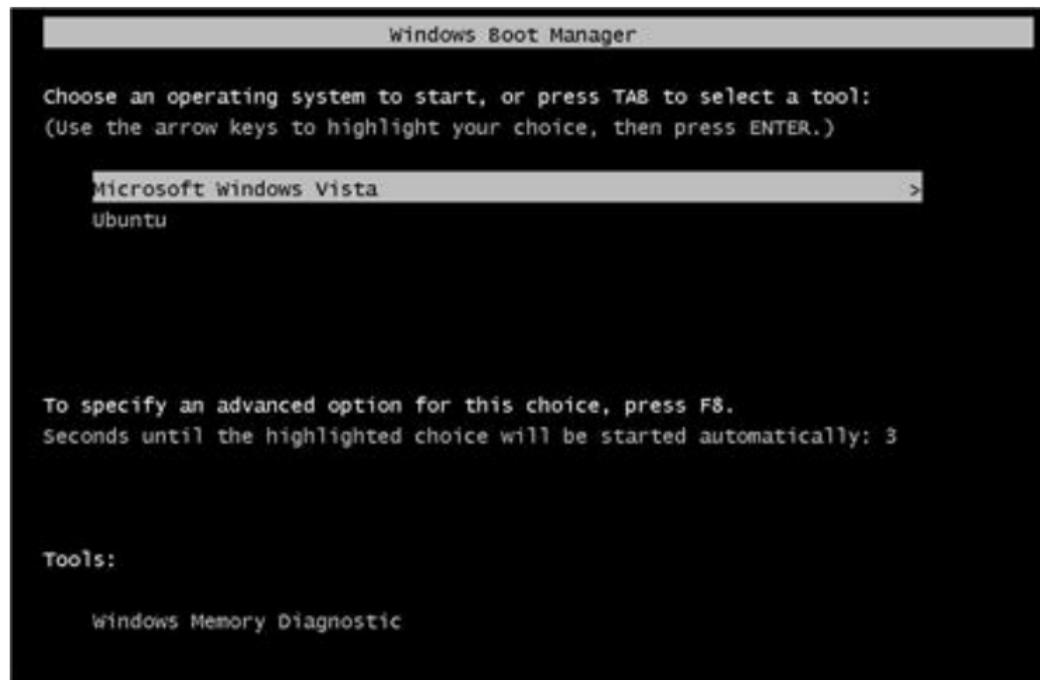
ubuntu-11.10-desktop-i386.iso [read only]			
Name	Size	Type	Date M
.disk	185 bytes	Folder	
boot	589 bytes	Folder	
casper	718.6 MB	Folder	
dists	11.7 kB	Folder	
install	1.7 MB	Folder	
isolinux	982.4 kB	Folder	
pics	14.9 kB	Folder	
pool	4.1 MB	Folder	
preseed	1.2 kB	Folder	
autorun.inf	143 bytes	unknown	12 Oct 2011
md5sum.txt	4.4 kB	plain text d...	12 Oct 2011
README.diskdefines	225 bytes	README d...	12 Oct 2011
wubi.exe	2.5 MB	DOS/Wind...	12 Oct 2011



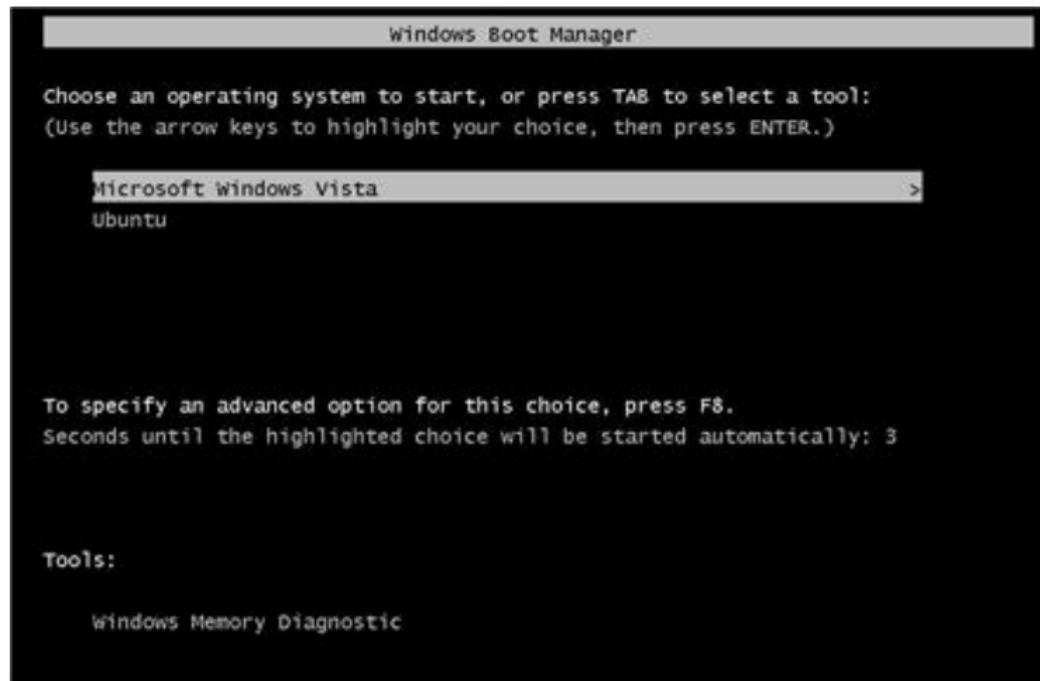
# 用Wubi安装Ubuntu 11.10操作系统



# 用Wubi安装Ubuntu以后的启动界面



# 用Wubi安装Ubuntu以后的启动界面

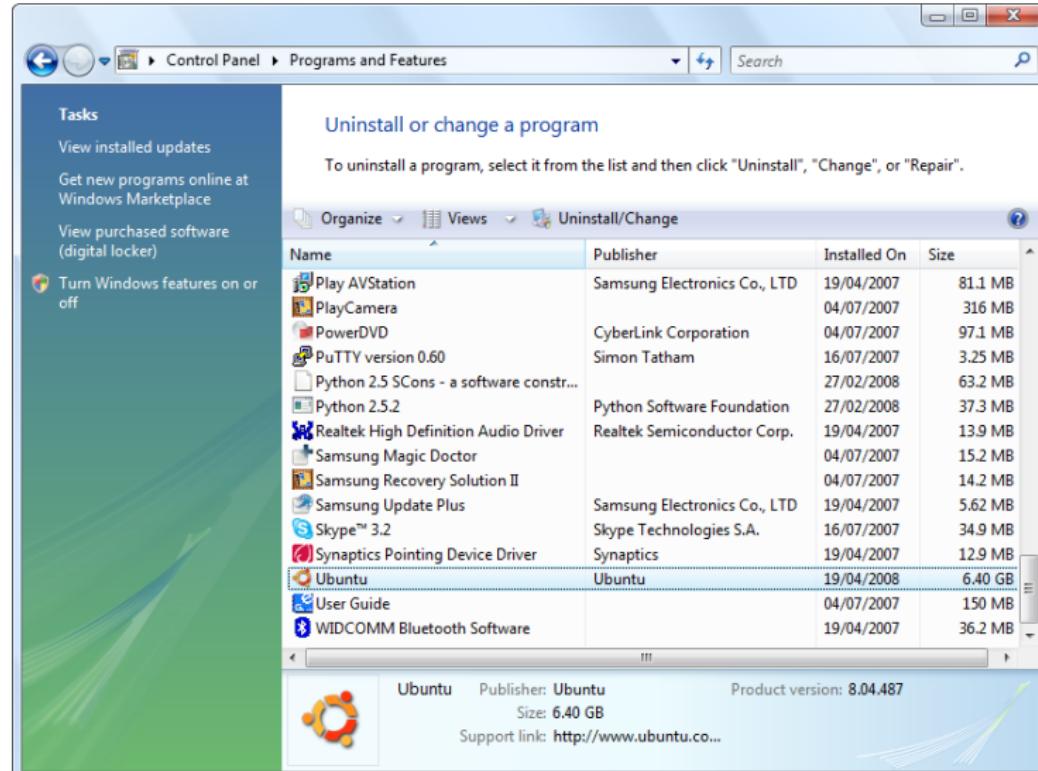


思考：这个界面截图是如何生成的？

# Ubuntu 11.10图形界面



# 卸载Wubi安装的Ubuntu 11.10



# 课程内容

- ▶ 操作系统概念及理论
- ▶ 阅读操作系统源代码并完成编程作业
- ▶ 学习本课程的知识要求：
  - ▶ C语言程序设计
  - ▶ 数据结构
  - ▶ 计算机组装原理
  - ▶ 计算机体系结构
  - ▶ 汇编语言程序设计

# 课程要求及考核方法

- ▶ 期末考试: 50%
- ▶ 平时作业(程序设计): 50%
- ▶ 两项均需及格方能通过本课程

Pass: 30, 30

Fail: 50, 10

Excellent: 40+, 40+

- ▶ 两点要求:

编程作业 需要独立完成

课堂考勤 要求

# 什么是操作系统

- ▶ 介于**用户**与计算机硬件之间的程序
- ▶ 思考: 操作系统的用户
- ▶ 操作系统的设计目标
  - ▶ 执行用户程序
  - ▶ 使计算机系统易于使用
  - ▶ 高效利用计算机硬件
- ▶ 对比: 应用软件的设计目标



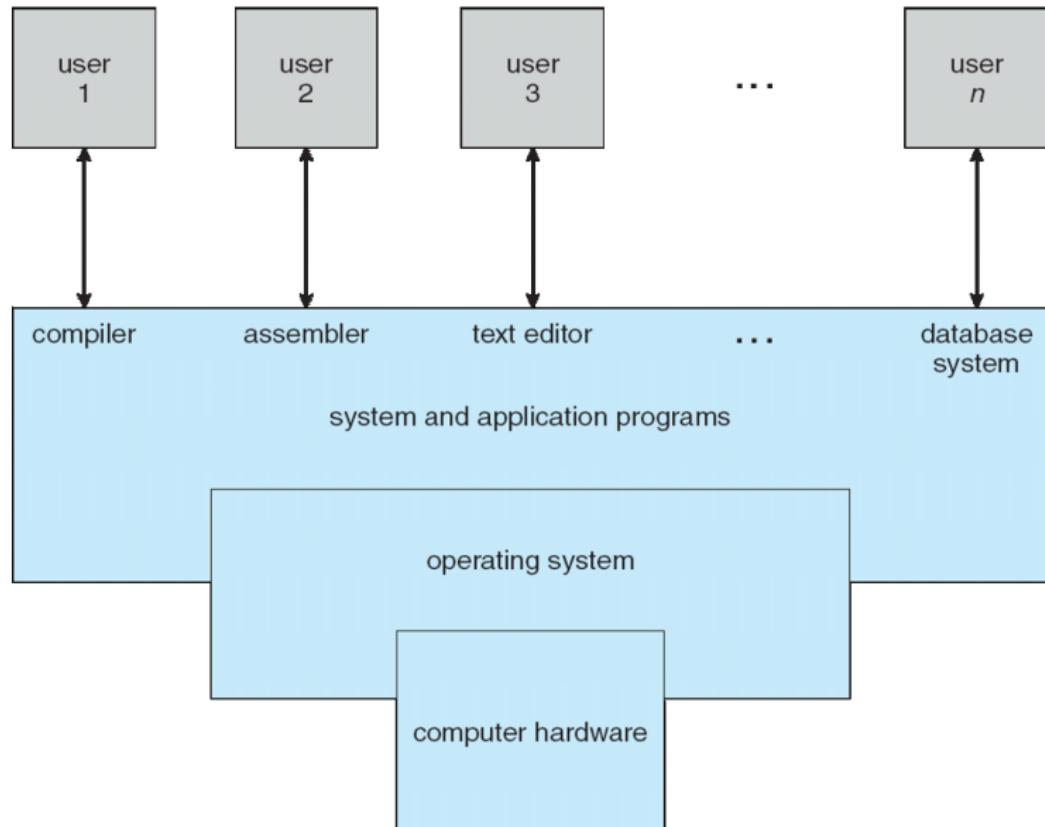
# 为什么学习操作系统

- ▶ 计算机系统的不可缺少的关键部分
- ▶ 非常复杂
  - ▶ Linux Kernel 2.6.35 — 13.5 M
  - ▶ FreeBSD — 8.8 M
  - ▶ Mac OS X 10.4 — 86 M
  - ▶ Windows Server 2003 – 50 M
- ▶ 真正理解计算机系统的工作原理
- ▶ 涉及硬件、编程语言、数据结构、算法等多个领域

# 计算机系统组成

- ▶ 计算机硬件: 提供基本计算资源
  - ▶ CPU, Memory, I/O devices...
- ▶ 操作系统
  - ▶ 控制、协调用户和程序对硬件资源的使用
- ▶ 应用程序: 解决用户的计算问题
  - ▶ Word processors, compilers, web browsers, database systems, video games
- ▶ 用户
  - ▶ People, machines (other computers)

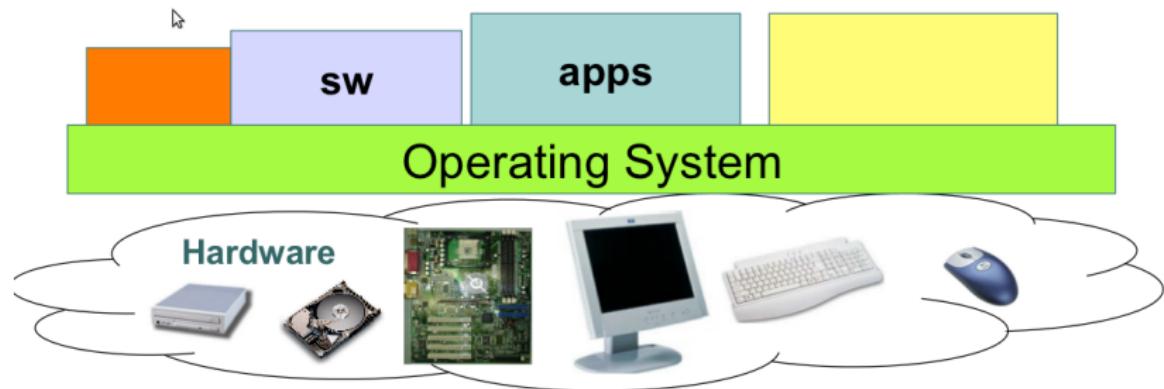
# 计算机系统组成



# 操作系统的作用

- ▶ 提供容易使用的界面(终端用户及程序员)
- ▶ 最大限度地提高资源利用率(CPU,内存)
- ▶ 为多用户提供分时服务(**time sharing system**)
- ▶ 在多用户多系统之间实现资源共享(存储、打印机)
- ▶ 嵌入式设备：界面问题、电池寿命问题(不只是OS的任务)

# 操作系统的角色：资源管理



# 操作系统的作用: 提供易于使用的界面

磁盘读写操作:

- ▶ 磁头、柱面、磁道、扇区
- ▶ 读写前需等待机械运动结束
- ▶ 数据存储可能不连续
- ▶ 磁盘大小、速度各不相同
- ▶ 对程序员而言, 编程读写磁盘数据是非常复杂的任务

操作系统提供的磁盘读写界面:

- ▶ `read()`
- ▶ `write()`
- ▶ `named files`(按名访问)

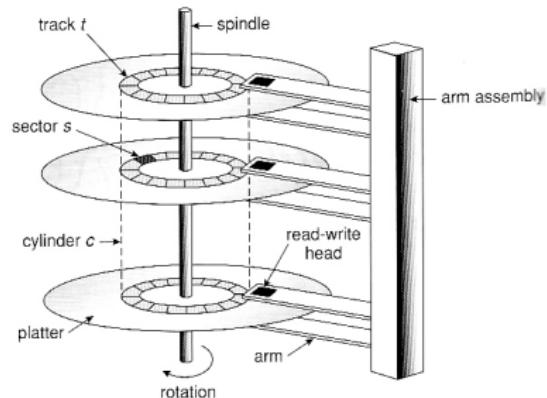
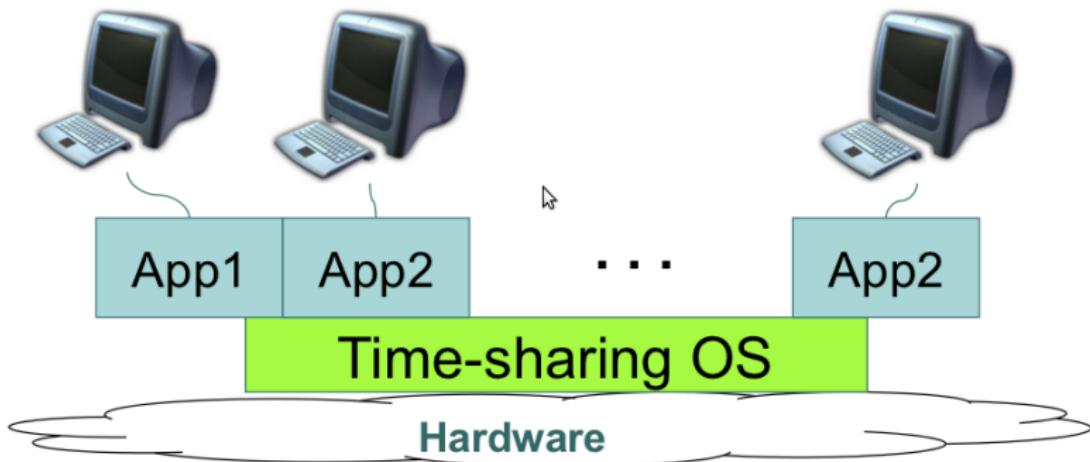


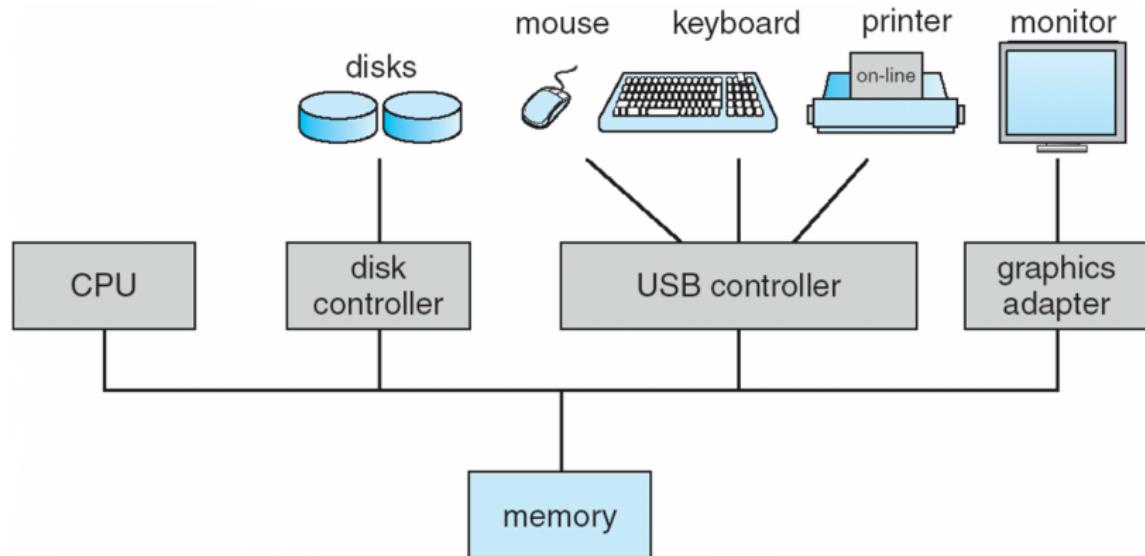
Figure 12.1 Moving-head disk mechanism.

# 操作系统的作用: 分时系统



# 计算机硬件系统

- ▶ CPUs, 设备控制器等通过总线与内存连接
- ▶ 竞争内存周期, 实现CPU、设备控制器间的并发执行

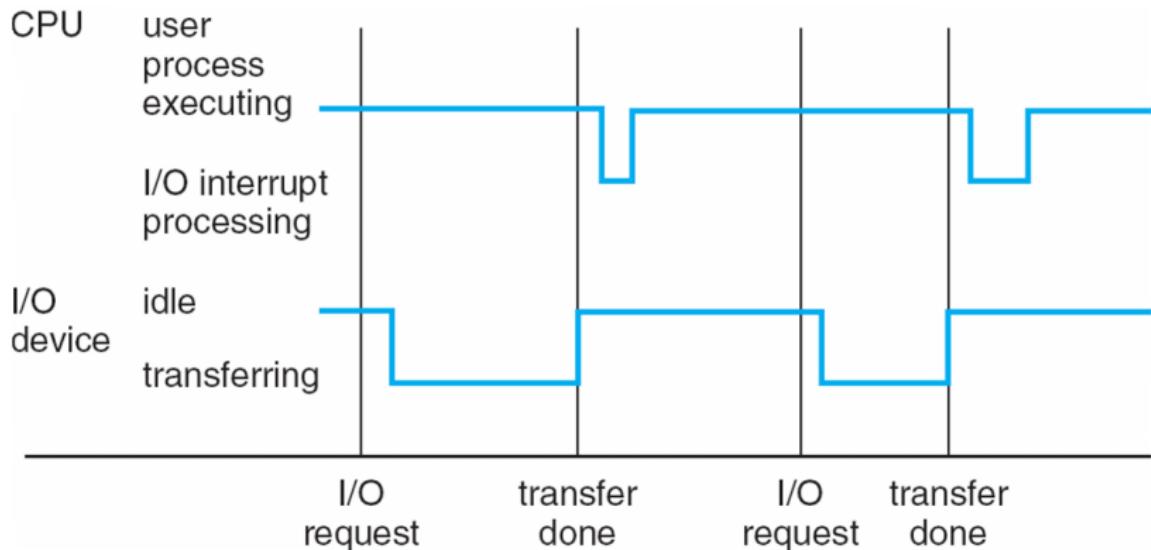


# 计算机硬件系统

- ▶ I/O设备与CPU可以并发运行
- ▶ 每个设备控制器控制一种I/O设备
- ▶ 设备控制器有自己的本地缓存
- ▶ CPU — 内存— 设备缓存— 设备
- ▶ I/O: 从设备到设备控制器的缓存
- ▶ I/O完成后，设备控制器通过中断通知OS

# 中断(interrupt)

- ▶ 操作系统由中断驱动
- ▶ 硬件中断与软件中断

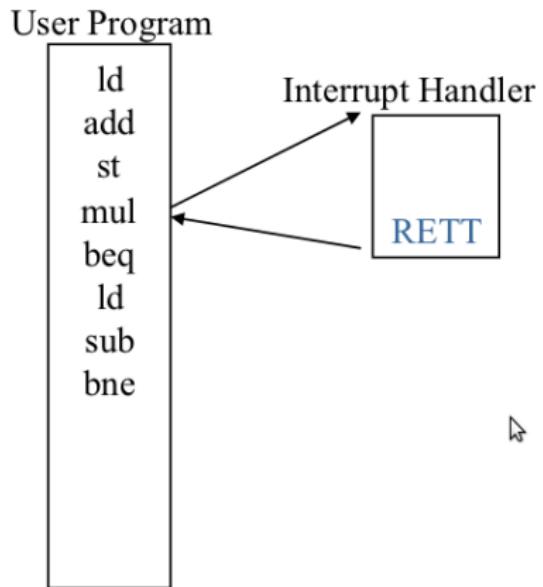


# 中断(interrupt)

思考：为什么I/O完成后，一定要中断CPU正在进行的操作，并转入操作系统的相应处理模块？

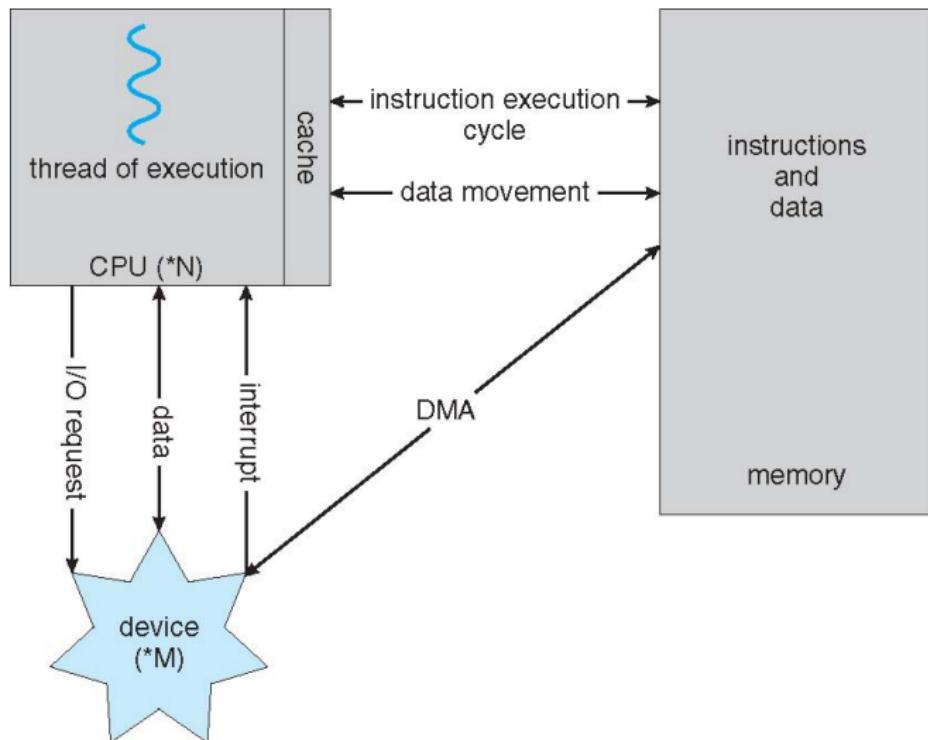
# 中断(interrupt)

- ▶ 根据事件类型确定中断处理程序(interrupt handler)
- ▶ 因此，需要先确定中断源，然后将PC设置为相应中断处理程序的入口



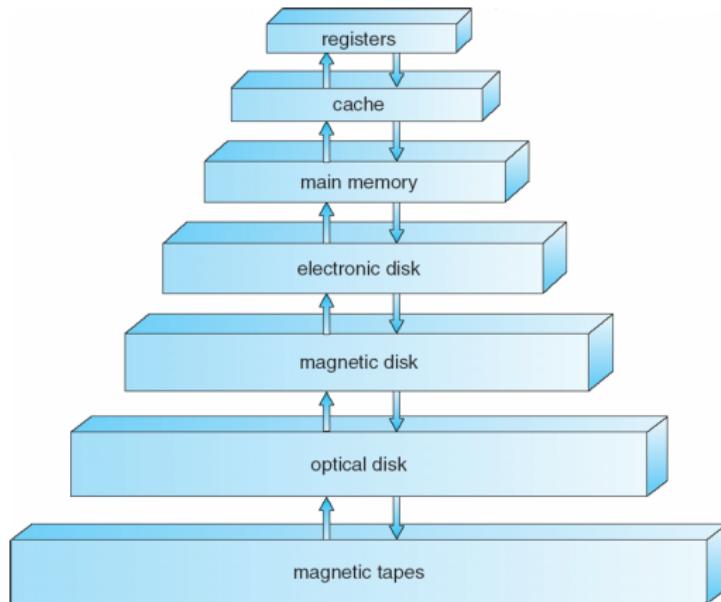
# 直接内存访问(DMA)

- ▶ 高速I/O设备与内存之间采用DMA传输数据
- ▶ 每传输1块数据产生一个中断

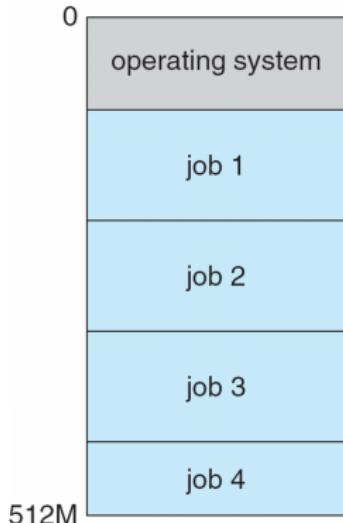


# 存储层级

- ▶ 存取速度
- ▶ 制造成本
- ▶ Caching技术



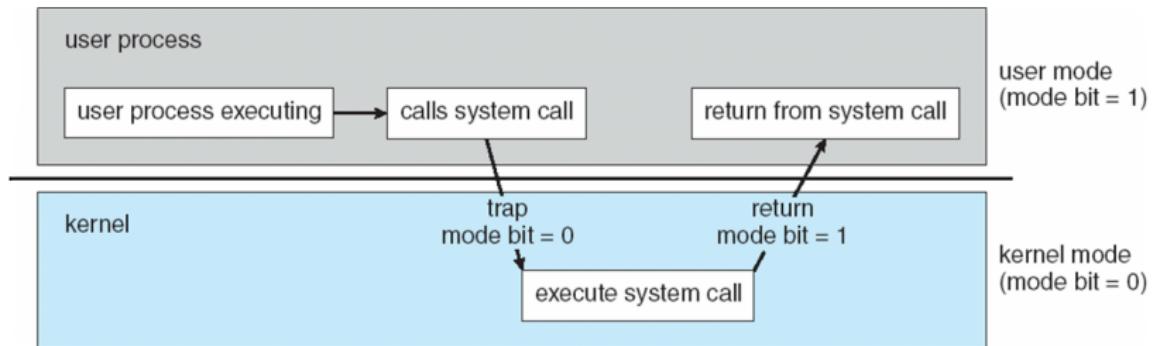
# 多道程序设计与分时系统



- ▶ 多道程序设计(multiprogramming)
  - ▶ 单个用户或程序无法使CPU或外设保持忙碌
  - ▶ 引入多道程序设计技术
  - ▶ 多个作业(jobs)驻留内存
  - ▶ 操作系统需要进行作业调度
  - ▶ 需要等待I/O时，调入另一作业运行
- ▶ 分时系统(timesharing system)与交互式计算(interactive computing)

# 核心态与用户态

- ▶ 核心态: 执行操作系统代码
- ▶ 用户态: 执行用户程序代码
- ▶ 思考: 哪些指令需要在核心态下执行?
- ▶ 系统调用导致从用户态转入核心态



# 核心态与用户态

	15	8 7	0
	AH	AL	
	BH	BL	
	CH	CL	
	DH	DL	
	BP		
	SI		
	DI		
	SP		

General-purpose registers

16-bit 32-bit

AX	EAX
BX	EBX
CX	ECX
DX	EDX
	EBP
	ESI
	EDI
	ESP



15 0


Segment registers

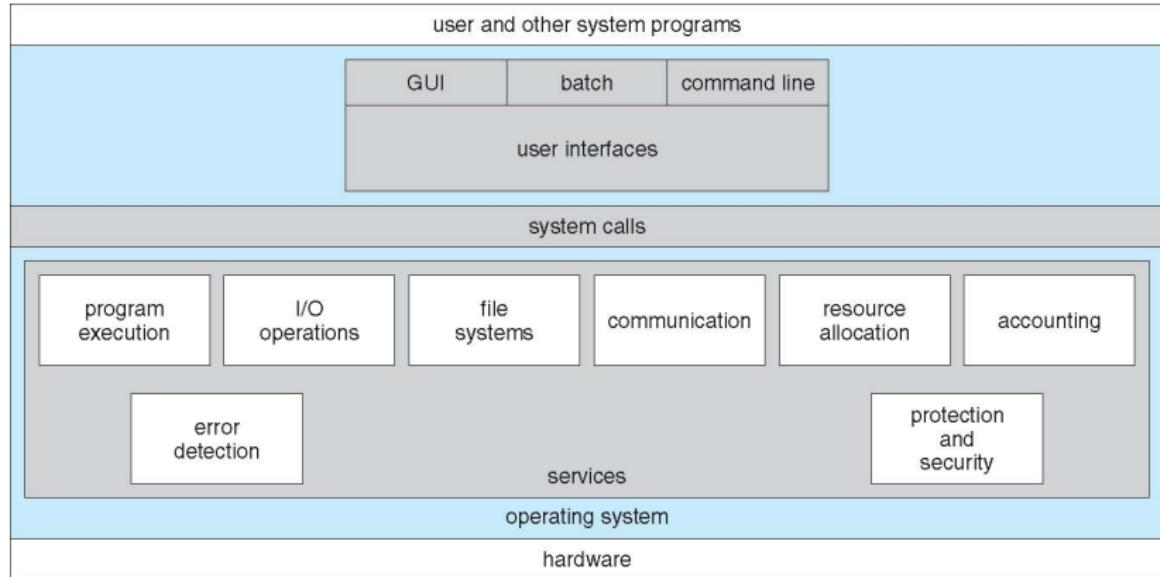


EFLAGS register

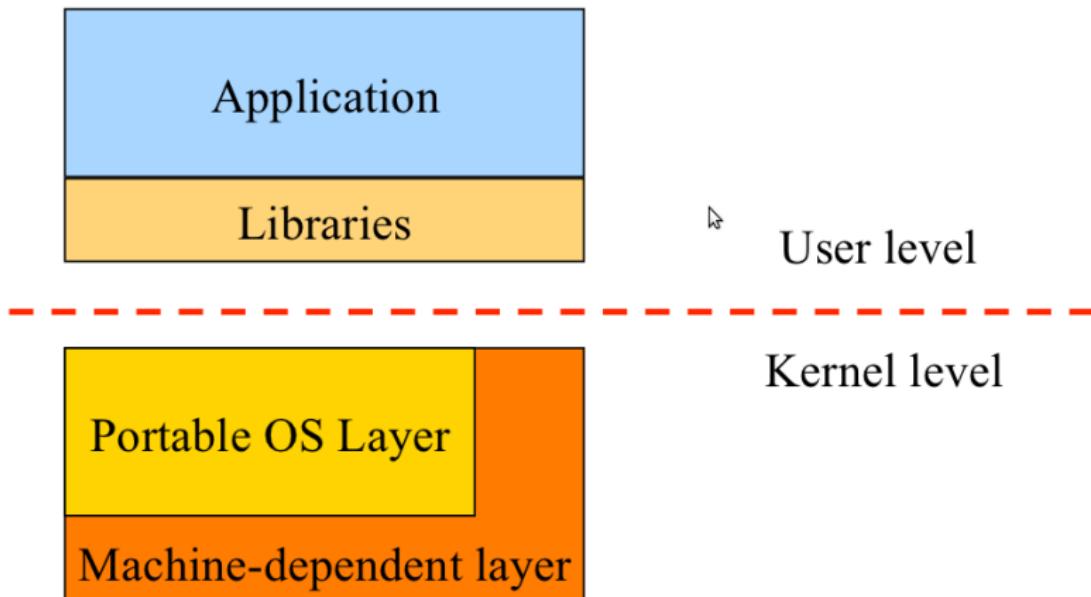


EIP (Instruction Pointer register)

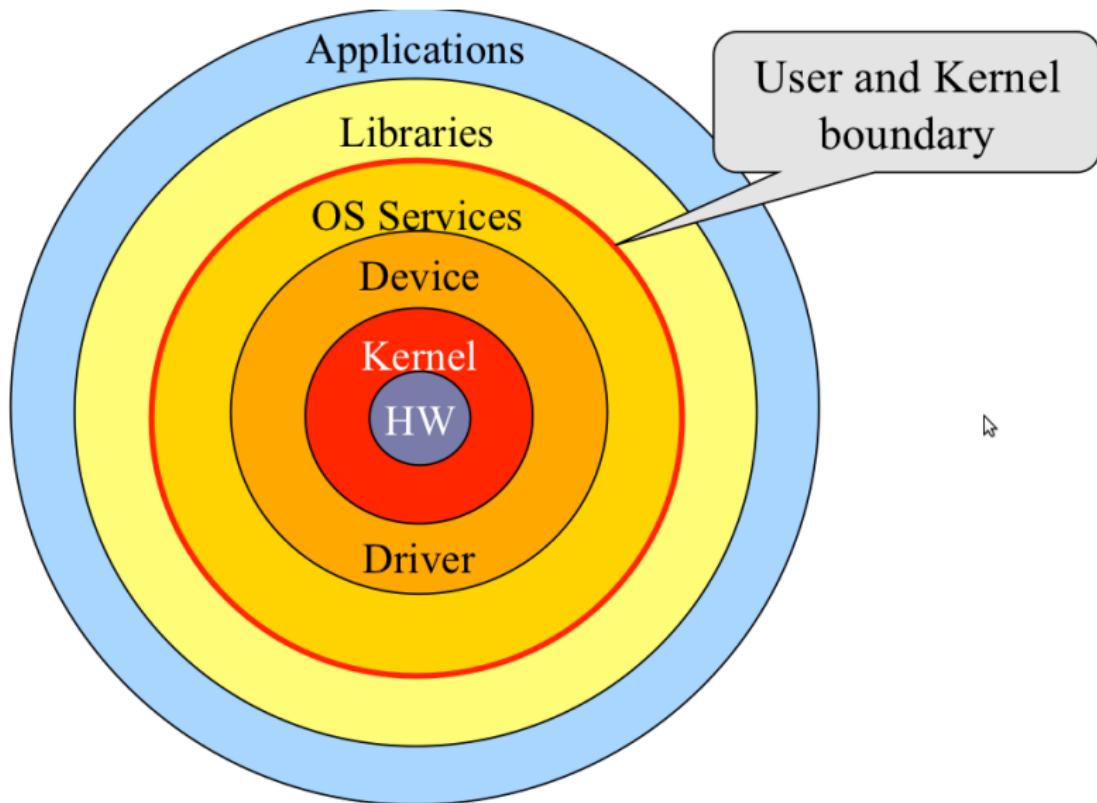
# 操作系统提供的服务



# 操作系统的结构： Unix为例



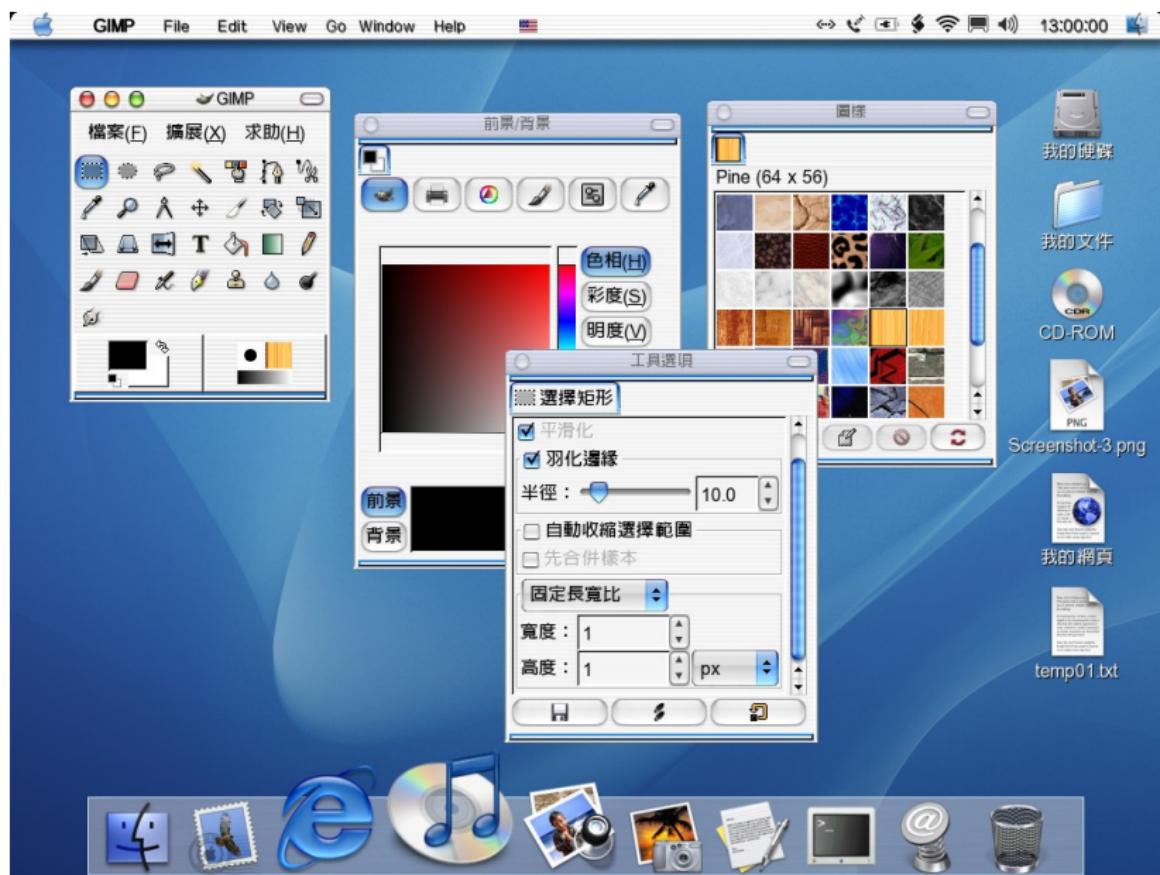
# 操作系统的结构: Unix为例



# 用户界面：命令行界面

```
hades:/Users/rob (70,23)
hades:/Users/rob
[1][rob.hades: /Users/rob]$ echo "Shorten paths > 18 chars"
Shorten paths > 18 chars
[2][rob.hades: /Users/rob]$ cd Desktop/
[3][rob.hades: Desktop]$ pwd|wc -c|tr -d ' '
19
[4][rob.hades: Desktop]$ echo "Show red when command fails"
Show red when command fails
[5][rob.hades: Desktop]$ ls -al dir_that_doesnt_exist_oh_noes
ls: dir_that_doesnt_exist_oh_noes: No such file or directory
[6][rob.hades: Desktop]$ mkdir !5
mkdir dir_that_doesnt_exist_oh_noes
[7][rob.hades: Desktop]$ !5
ls -al dir_that_doesnt_exist_oh_noes
total 0
drwxr-xr-x    2 rob  rob   68 Nov 15 00:47 .
drwx----- 12 rob  rob  408 Nov 15 00:47 ..
[8][rob.hades: Desktop]$ echo "Expand path back when < 18 chars"
Expand path back when < 18 chars
[9][rob.hades: Desktop]$ ..
[10][rob.hades: /Users/rob]$ !3
pwd|wc -c|tr -d ' '
11
[11][rob.hades: /Users/rob]$ Fin.
```

# 用户界面：图形用户界面



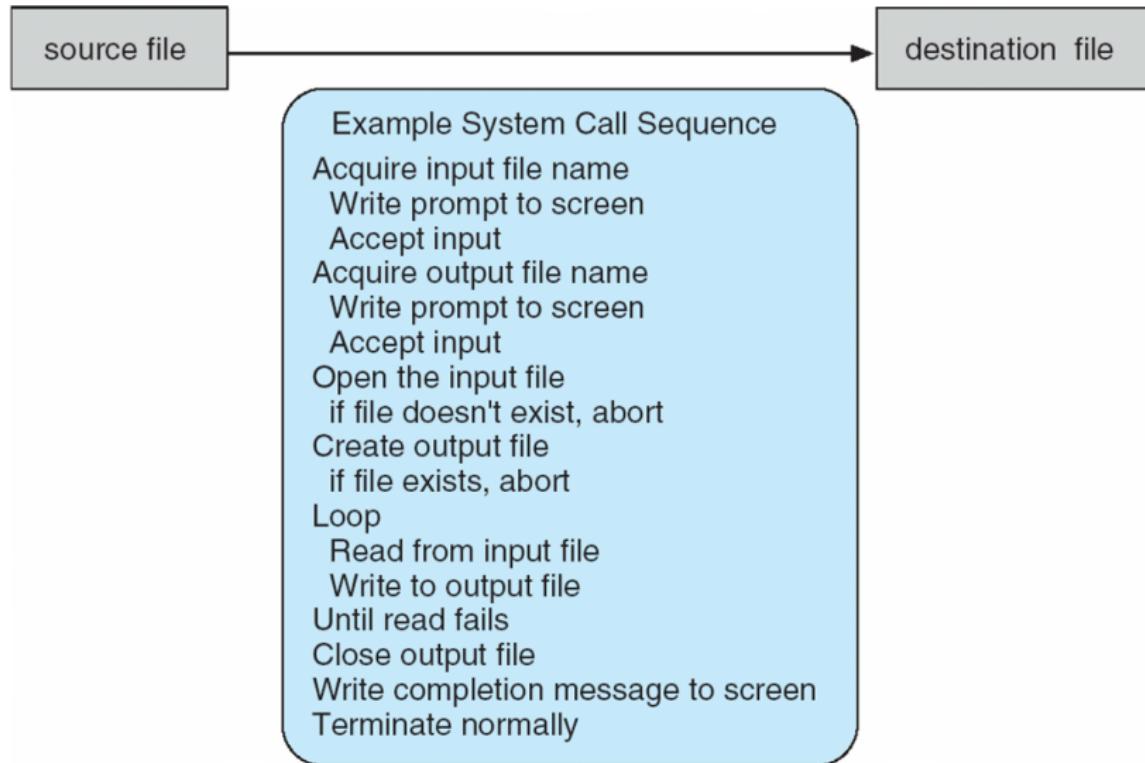
## 思考及演示：CLI与GUI的优缺点对比

- ▶ Demo 1 — word frequency counting
- ▶ Demo 2 — lines of C source code
- ▶ Unix/Linux命令行界面的哲学

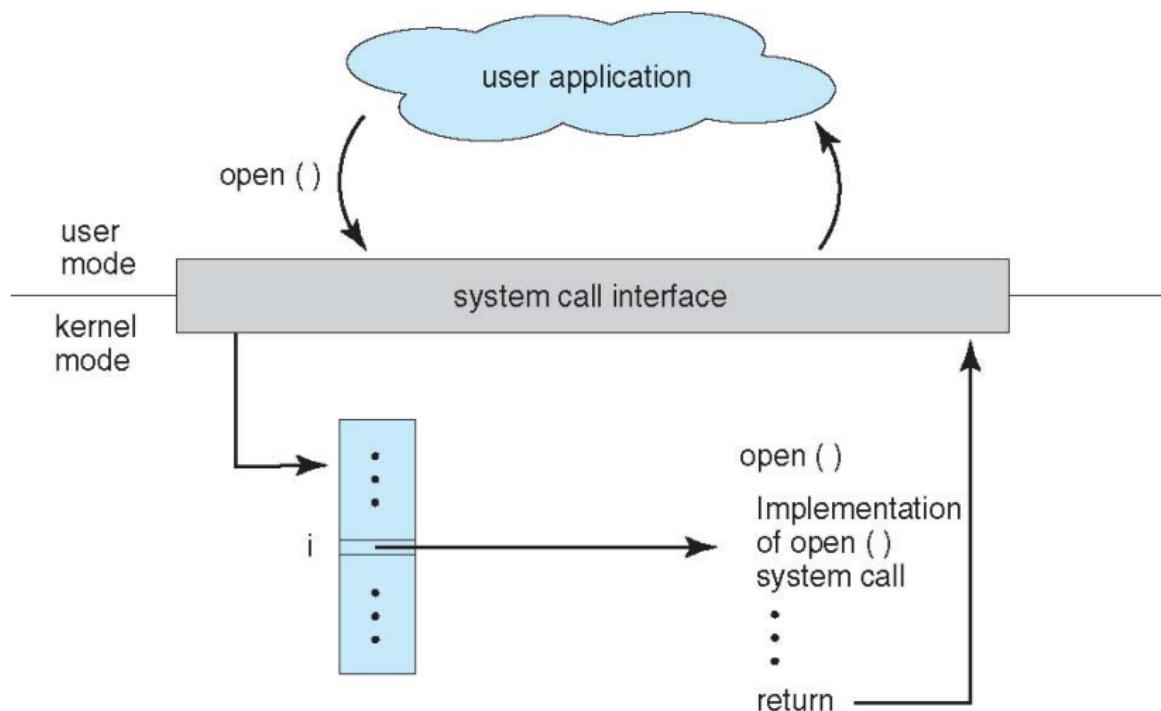
# 系统调用

- ▶ CLI/GUI是人使用操作系统时的界面
- ▶ 系统调用可看作获取操作系统服务的编程界面(接口)
- ▶ 一般可用C/C++编写
- ▶ 通常将系统调用封装成API方便使用
- ▶ 常见API:
  - ▶ Win32 API
  - ▶ POSIX API (Unix, Linux, Mac OS X)
  - ▶ Java API

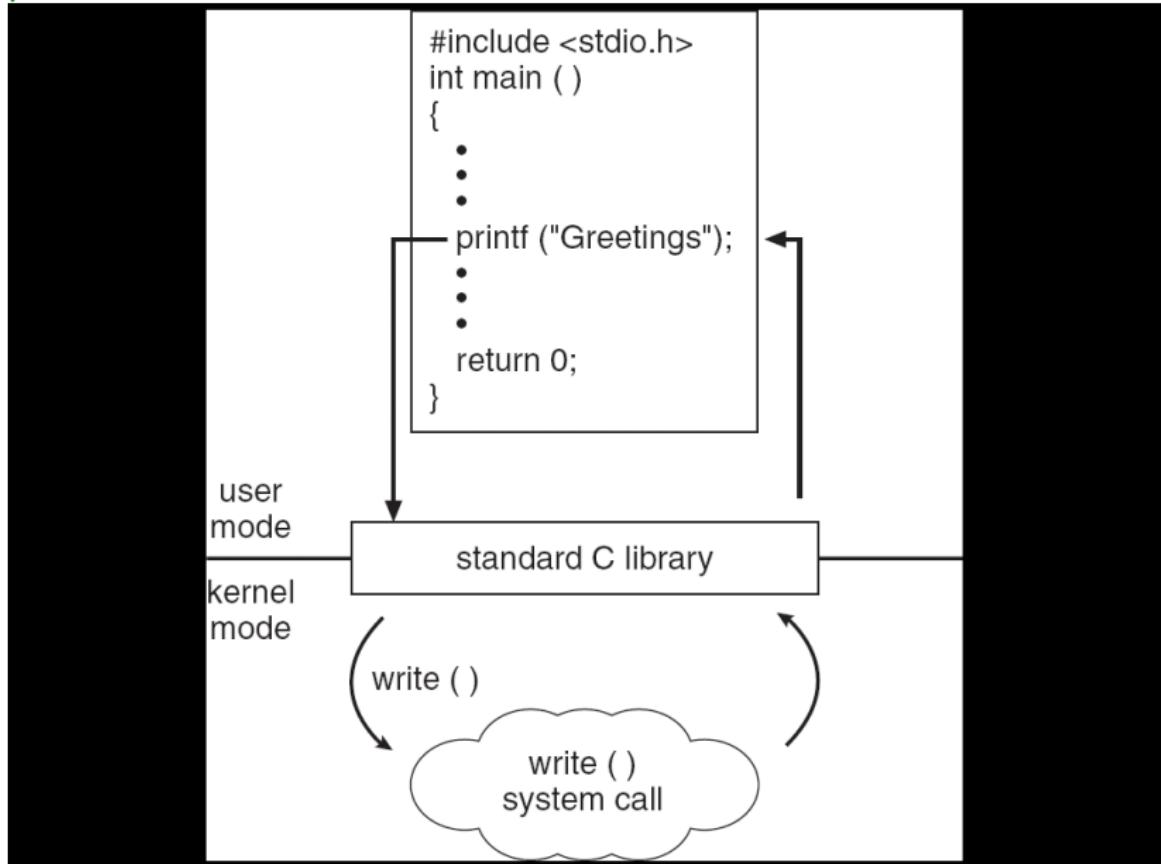
# 系统调用示例: 文件拷贝



# API – 系统调用– 操作系统之间的关系



# API – 系统调用 – 操作系统之间的关系: 标准C函数库的例子



# 系统调用的例子

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

# 系统程序(system programs)

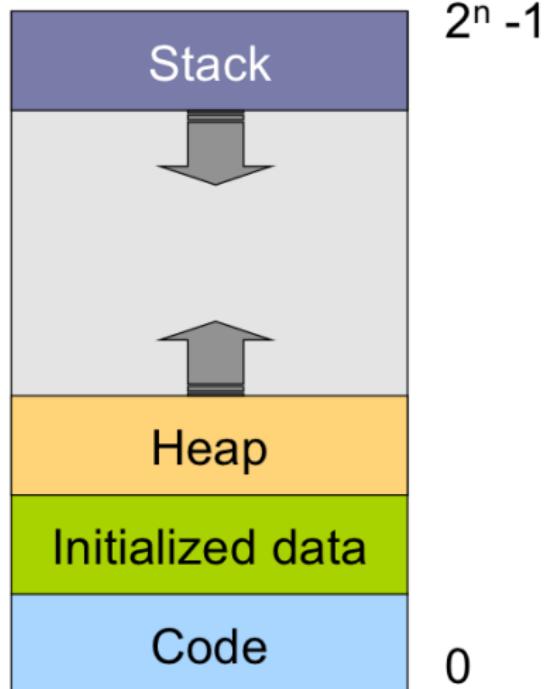
- ▶ 操作系统提供的用于系统管理等任务的程序
- ▶ 某些系统程序只是系统调用的接口程序(例如删除文件)
- ▶ 文件管理程序
  - ▶ touch
  - ▶ rm
  - ▶ cp
  - ▶ rename
  - ▶ mkdir
- ▶ 编程语言相关的程序
  - ▶ 编译器
  - ▶ 汇编器
  - ▶ 调试器
  - ▶ 解释器

# 进程概念

- ▶ 操作系统执行用户程序
  - ▶ 批处理系统— 作业
  - ▶ 分时系统— 用户程序、任务
- ▶ 进程: 运行中的程序
- ▶ 进程包含三部分内容:
  - ▶ 程序代码(**text section**)
  - ▶ 当前状态: 程序计数器(**program counter**)以及寄存器(**registers**)
  - ▶ 栈(函数参数, 返回地址, 局部变量)
  - ▶ 数据区(**data section**, 全局变量)
  - ▶ 堆(运行时动态分配的内存)

# 进程在内存中的结构

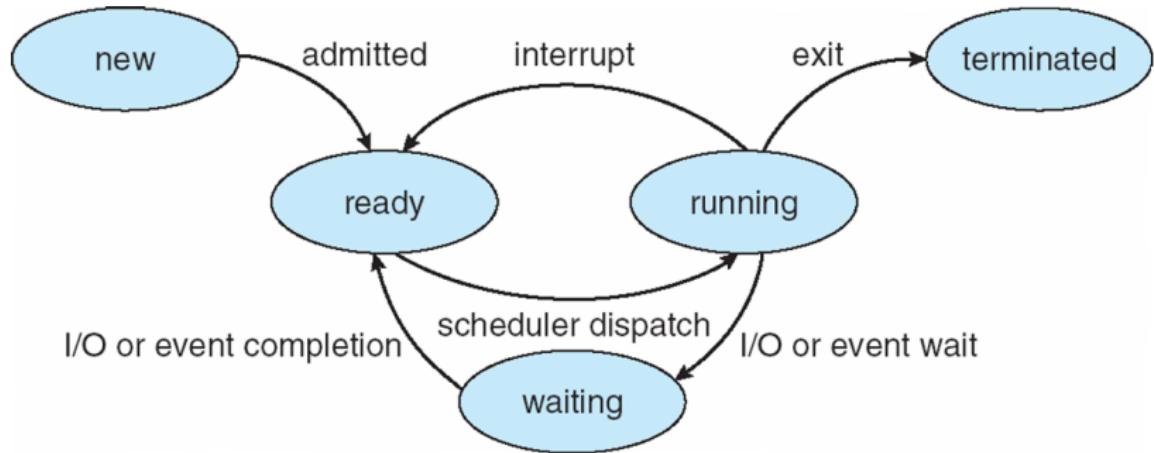
- ▶ Code/Text – 程序指令
- ▶ Data – 全局变量数据
- ▶ Stack – 栈
- ▶ Heap – 堆
- ▶ 目的: 将指令与数据分开(why?)
- ▶ 堆、栈朝相对的方向增长



# 进程的状态

- ▶ new: 进程正在被创建当中
- ▶ running: 进程的指令正在**CPU**上执行
- ▶ waiting: 等待某事件的发生(e.g, I/O)
- ▶ ready: 等待CPU
- ▶ terminated: 进程终止

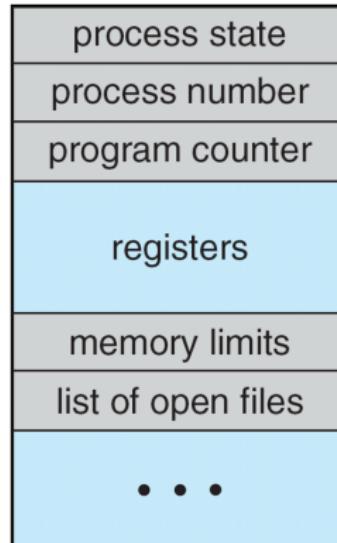
# 进程状态迁移



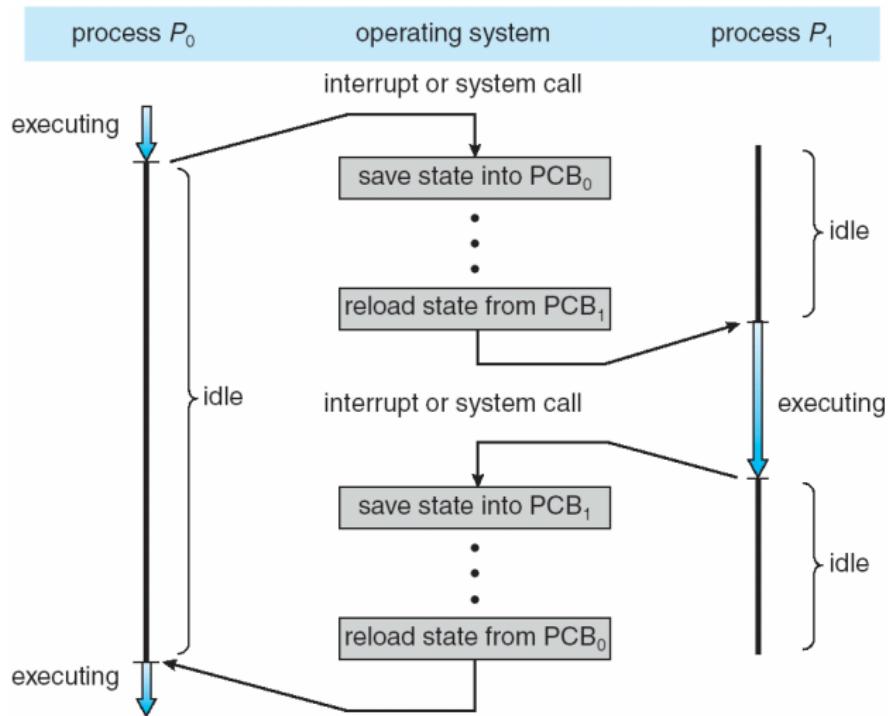
# 进程控制块(PCB)

用于存放与每个进程相关的信息

- ▶ 进程状态
- ▶ 程序计数器PC
- ▶ CPU寄存器
- ▶ CPU调度信息
- ▶ 内存管理信息
- ▶ I/O状态信息
- ▶ 记账信息



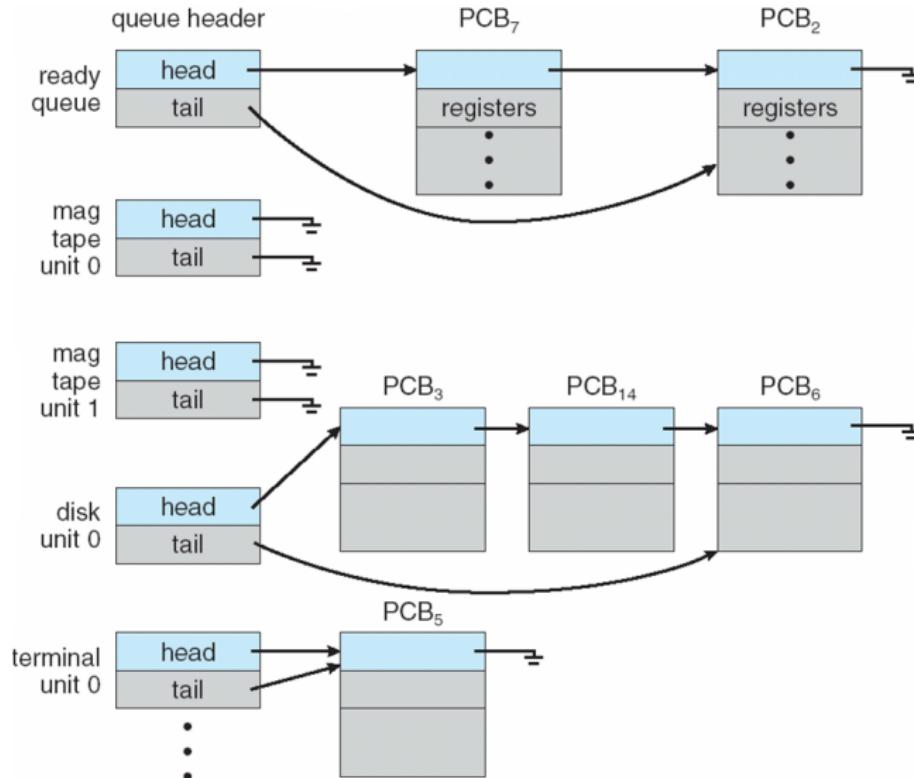
# CPU在进程间切换



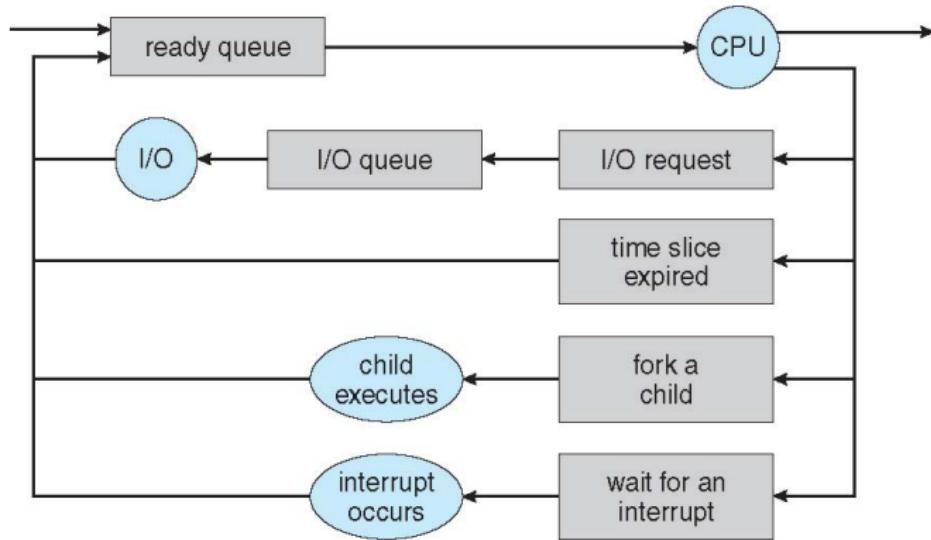
# 进程调度

- ▶ 最大化CPU利用率; 实现分时系统
- ▶ 进程调度器: 从**ready**状态的进程中选择一个运行
- ▶ 调度队列:
  - ▶ 作业队列(**Job queue**) – 系统中所有进程的集合
  - ▶ 就绪队列(**ready queue**) – 内存中所有处于**ready**状态的进程
  - ▶ 设备队列(**device queues**) – 等待某I/O设备的进程集合
  - ▶ 进程在上述队列之间来回迁移

# 就绪队列及设备队列示意



# 进程调度示意图



## 上下文切换(context switch)

- ▶ CPU分配给新进程时，原进程的状态需要保存，新进程的状态需要载入
- ▶ 进程的上下文(context)保存于进程控制块(PCB)中
- ▶ 上下文切换时间纯属无用开销，因此越快越好
- ▶ 切换时间取决于硬件支持(e.g, 具有多组寄存器的CPU)

# 进程创建, 子进程

- ▶ 进程可以创建子进程, 形成进程树
- ▶ process identifier (pid)
- ▶ 父进程、子进程之间的资源共享

