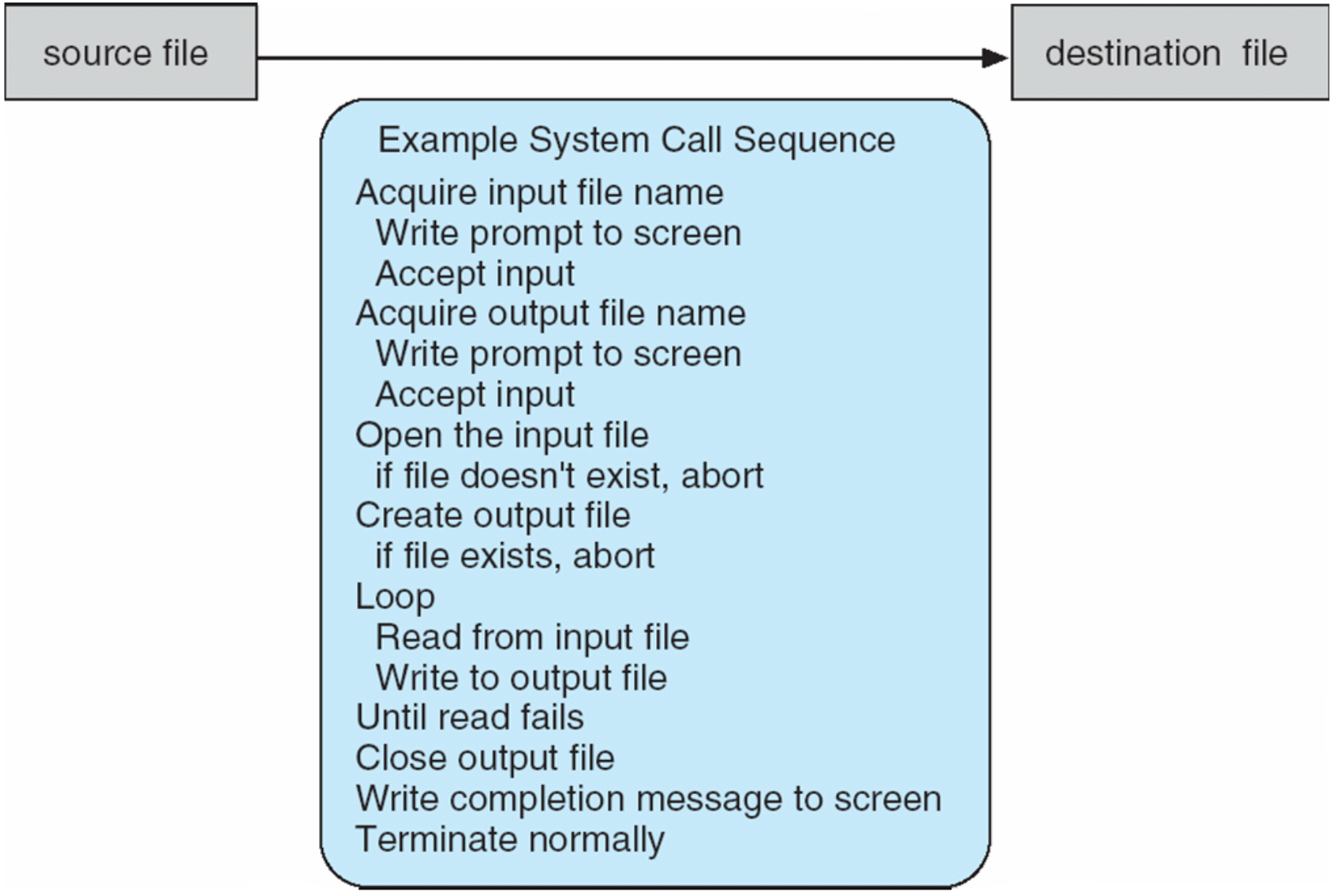
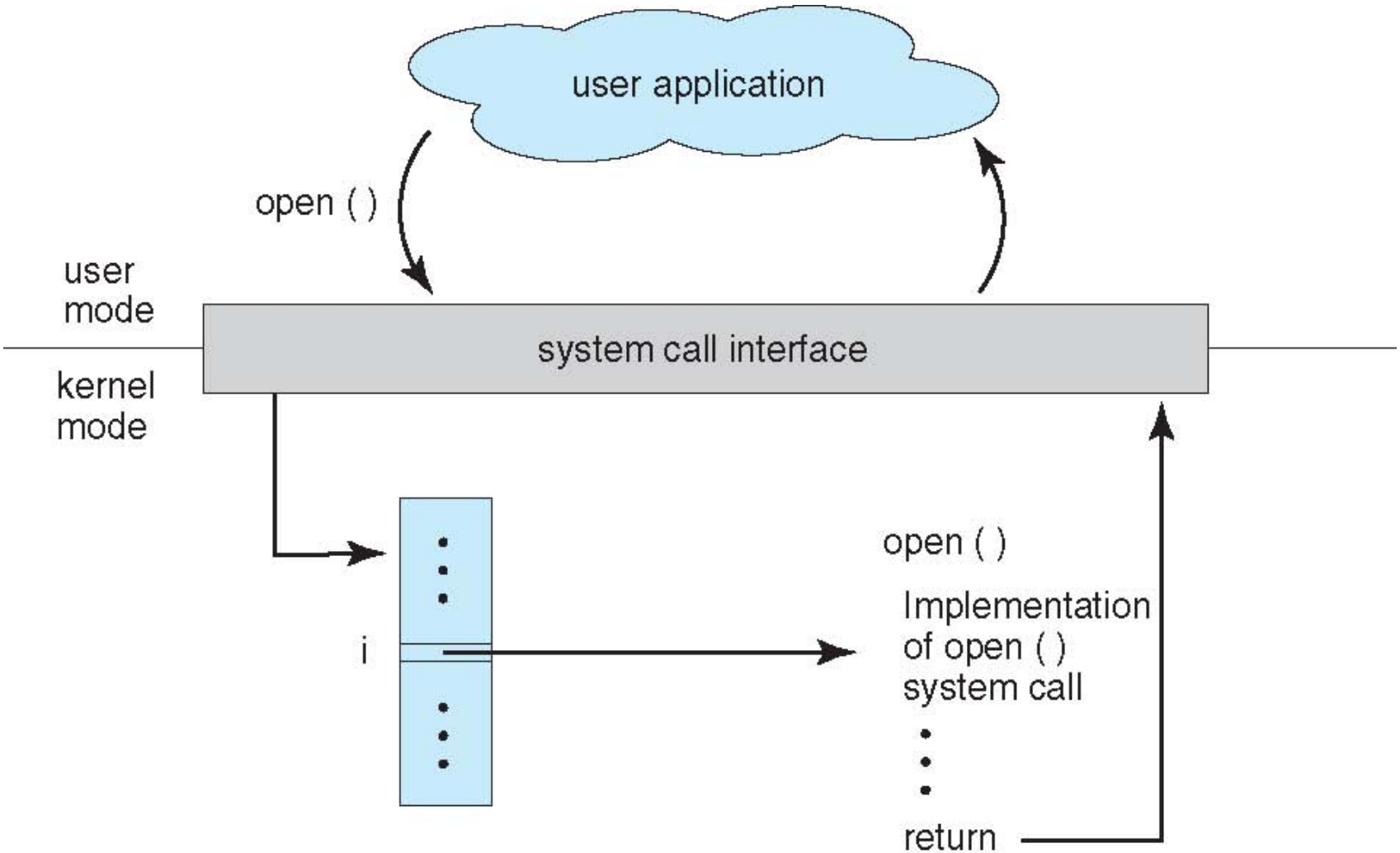
系统调用

- 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

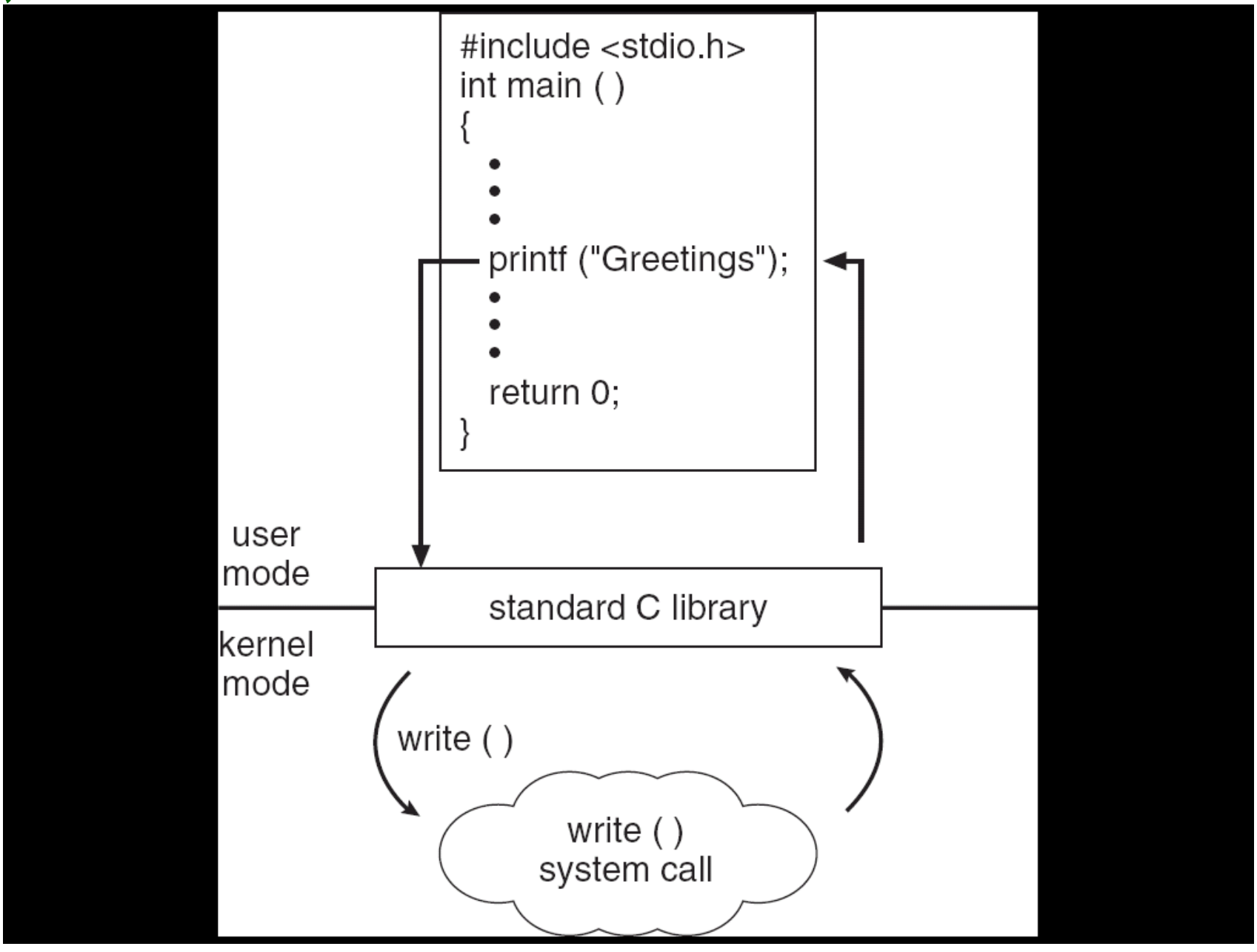
系统调用示例: 文件拷贝


source file
destination file
Example System Call Sequence
Acquire input file name
Write prompt to screen
Accept input
Acquire output file name
Write prompt to screen
Accept input
Open the input file
if file doesn't exist, abort
Create output file
if file exists, abort
Loop
Read from input file
Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally

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

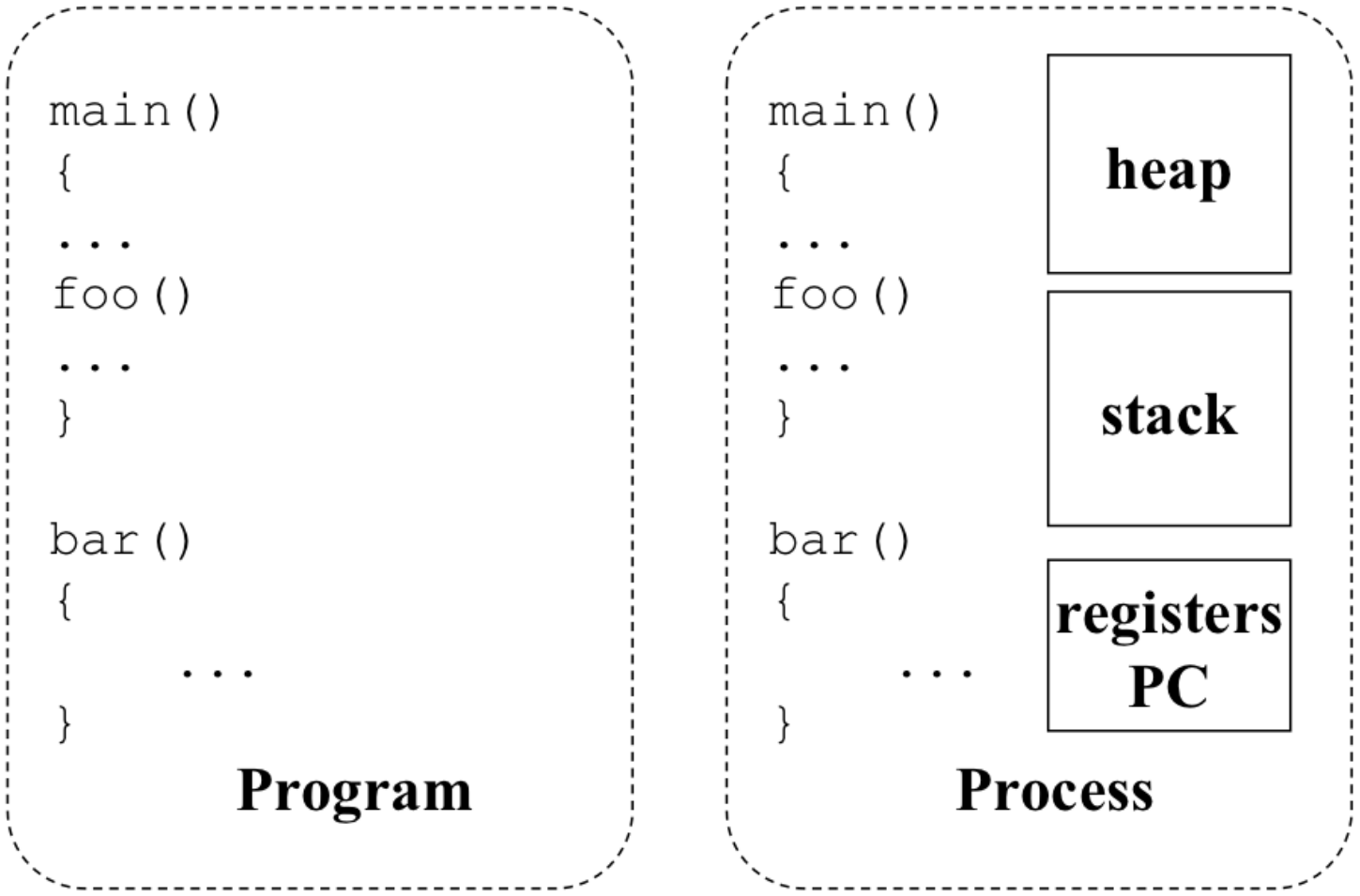
进程概念

- 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)

操作系统的抽象

- 进程
- 文件— 对Unix而言，除了进程，一切都是文件
  - 普通文件
  - Sockets
  - Pipes(管道)
  - Devices(设备)
- 地址空间、虚拟内存
- OS中最重要的三个概念

进程与程序

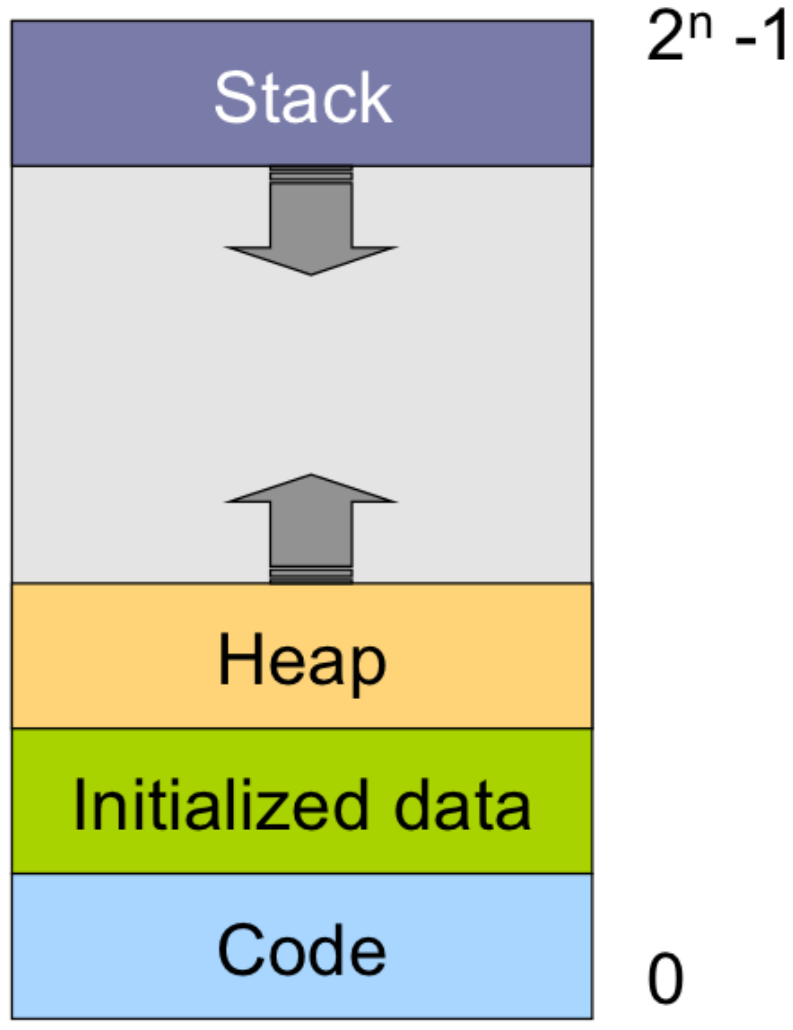

main()
{
...
foo()
...
}
bar()
{
...
}
Program
main()
{
...
foo()
...
}
bar()
{
...
}
heap
stack
registers
PC
Process

进程与程序

- 进程大于程序
  - 程序代码是进程的一部分
  - 例如：多个用户同时使用一个程序(代码相同，进程状态可能不同)
- 进程小于程序
  - 程序中可以多个创建新进程（后续编程作业）

进程在内存中的结构

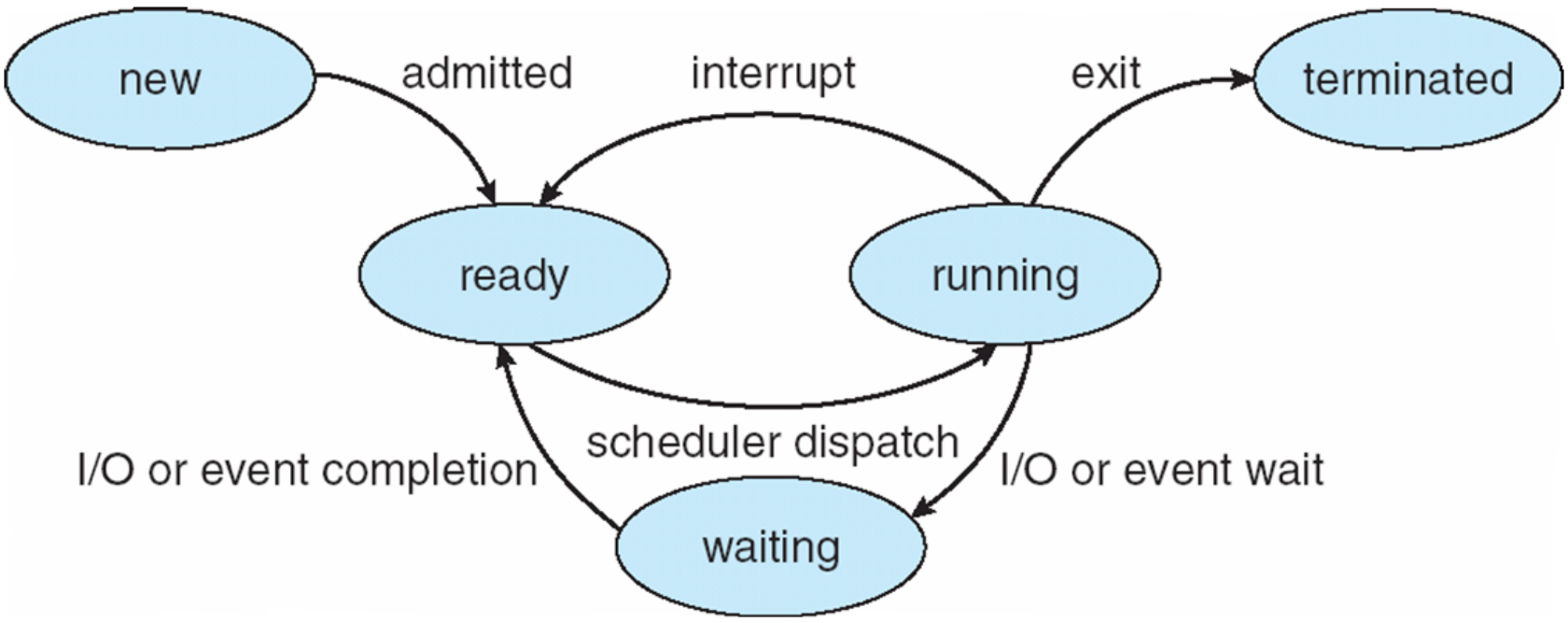
- Code/Text – 程序指令
- Data – 全局变量数据
- Stack – 栈
- Heap – 堆
- 目的：将指令与数据分开(why?)
- 堆、栈朝相对的方向增长



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

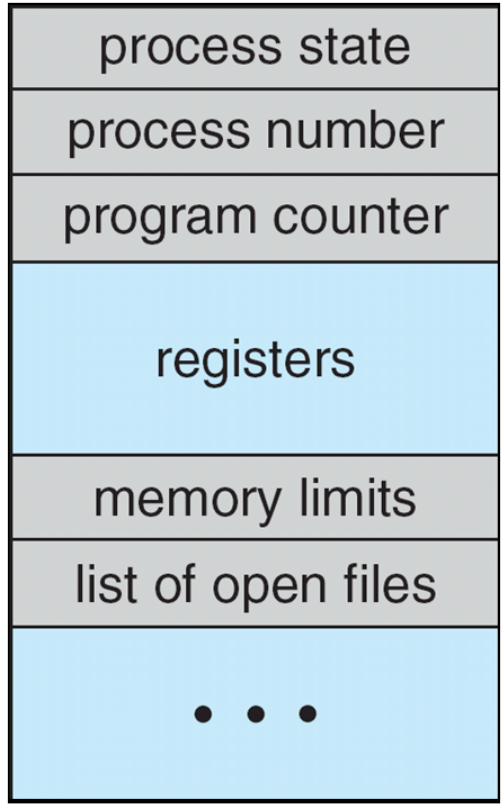
进程状态迁移


new
admitted
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

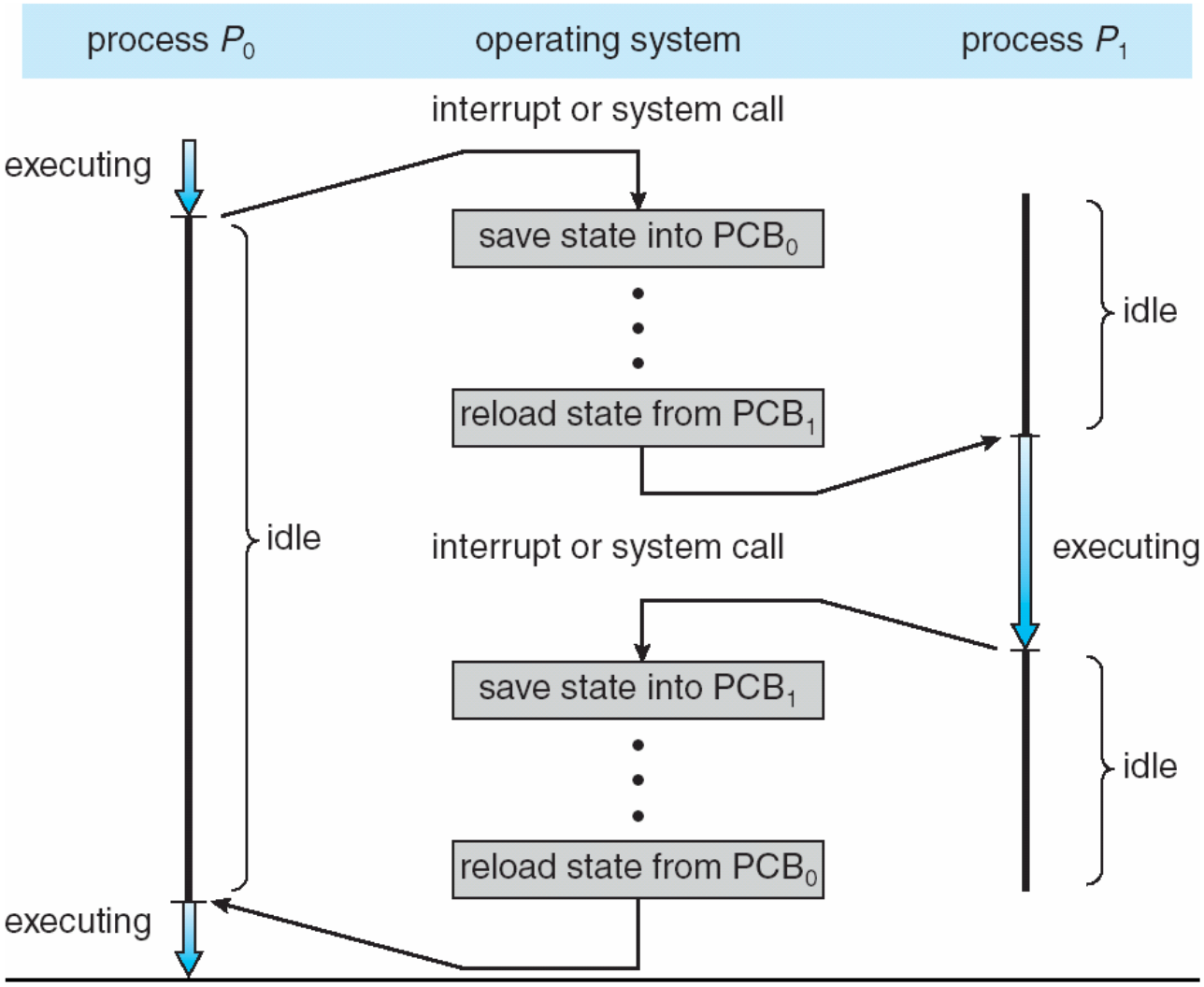
进程控制块(PCB)

用于存放与每个进程相关的信息

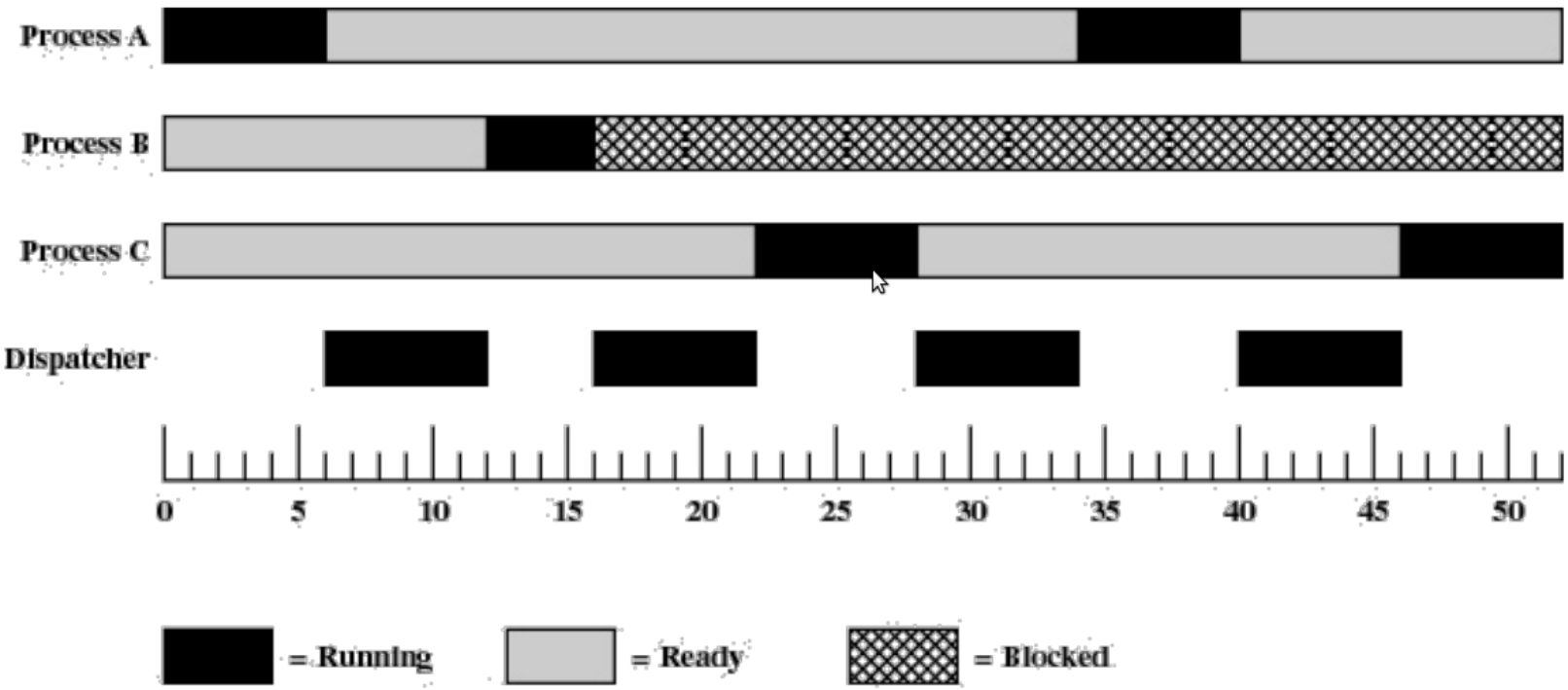
- 进程状态
- 程序计数器PC
- CPU寄存器
- CPU调度信息
- 内存管理信息
- I/O状态信息
- 记账信息



CPU在进程间切换

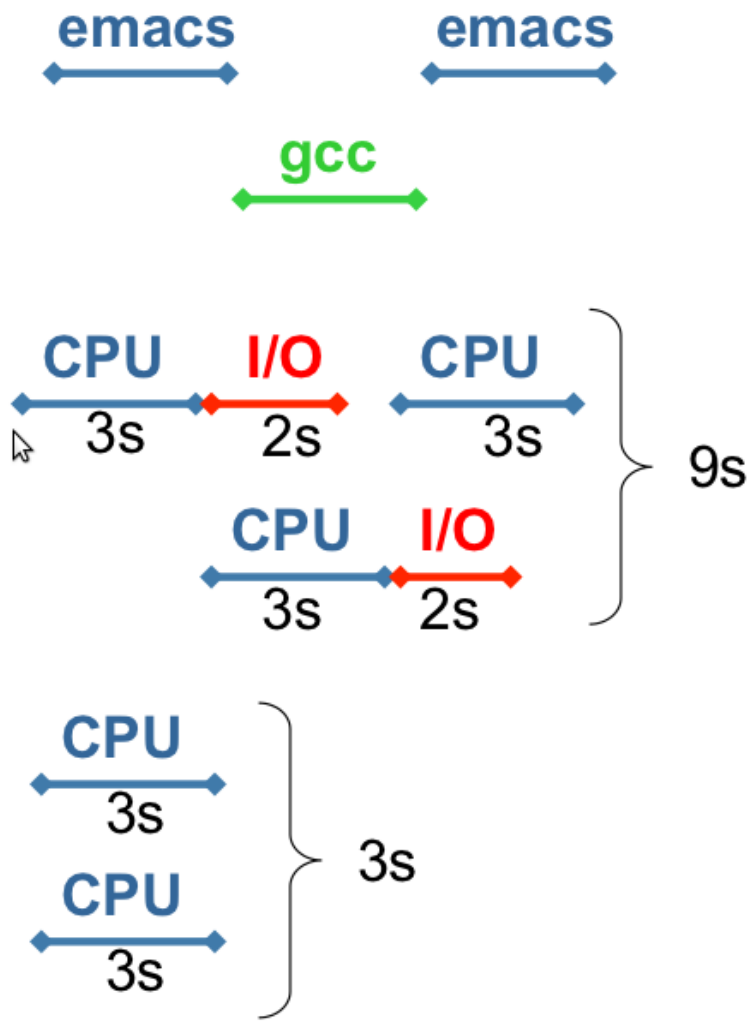

process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
reload state from PCB_1
idle
idle
interrupt or system call
executing
save state into PCB_1
reload state from PCB_0
idle
executing

CPU在进程间切换


Process A
Process B
Process C
Dispatcher
0
5
10
15
20
25
30
35
40
45
50
= Running
= Ready
= Blocked

进程与程序的并发执行

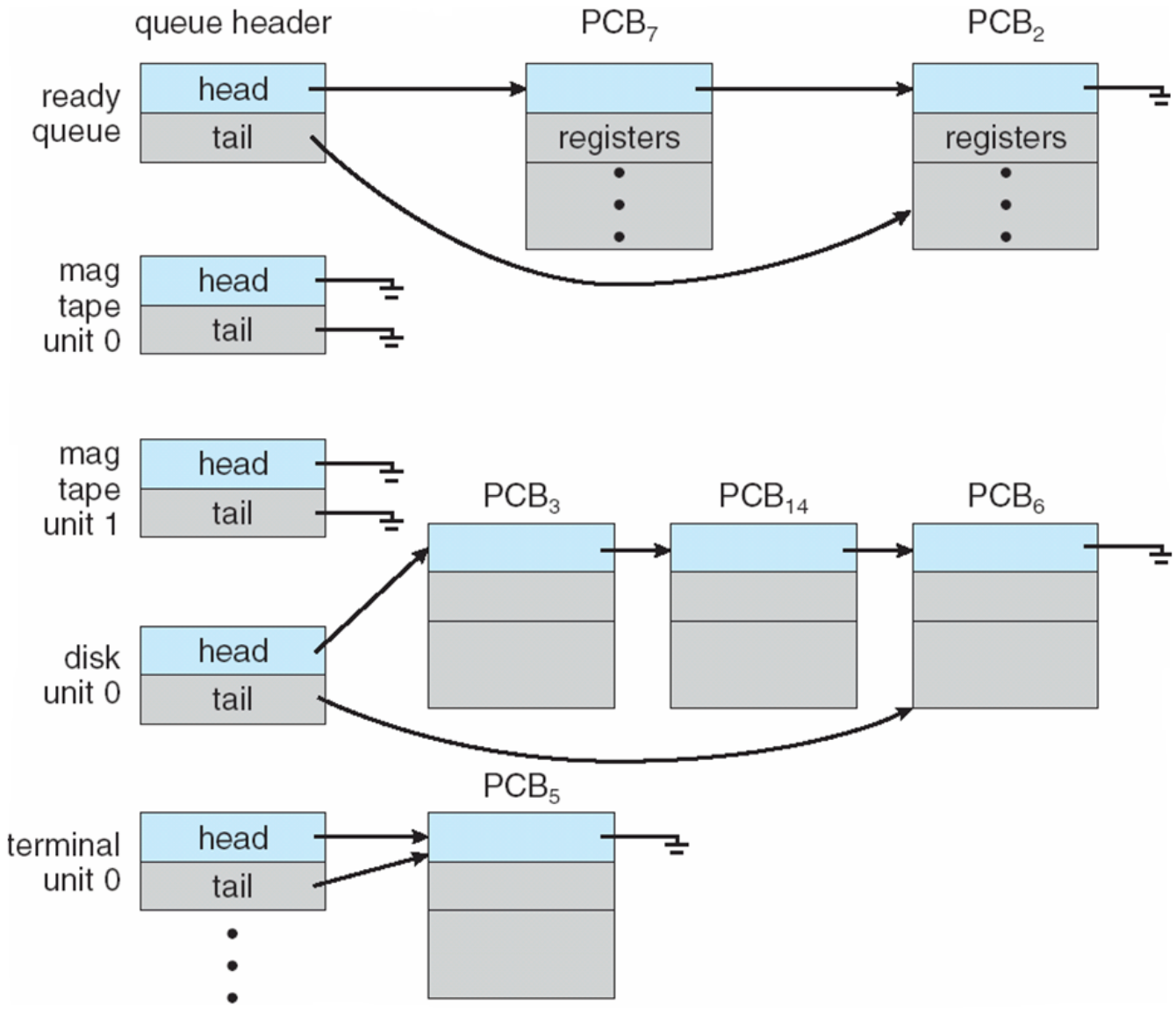
- CPU虚拟化，多个程序“同时”运行
- CPU与I/O同时运行
- 多CPU系统中，同一时刻不同CPU运行不同程序



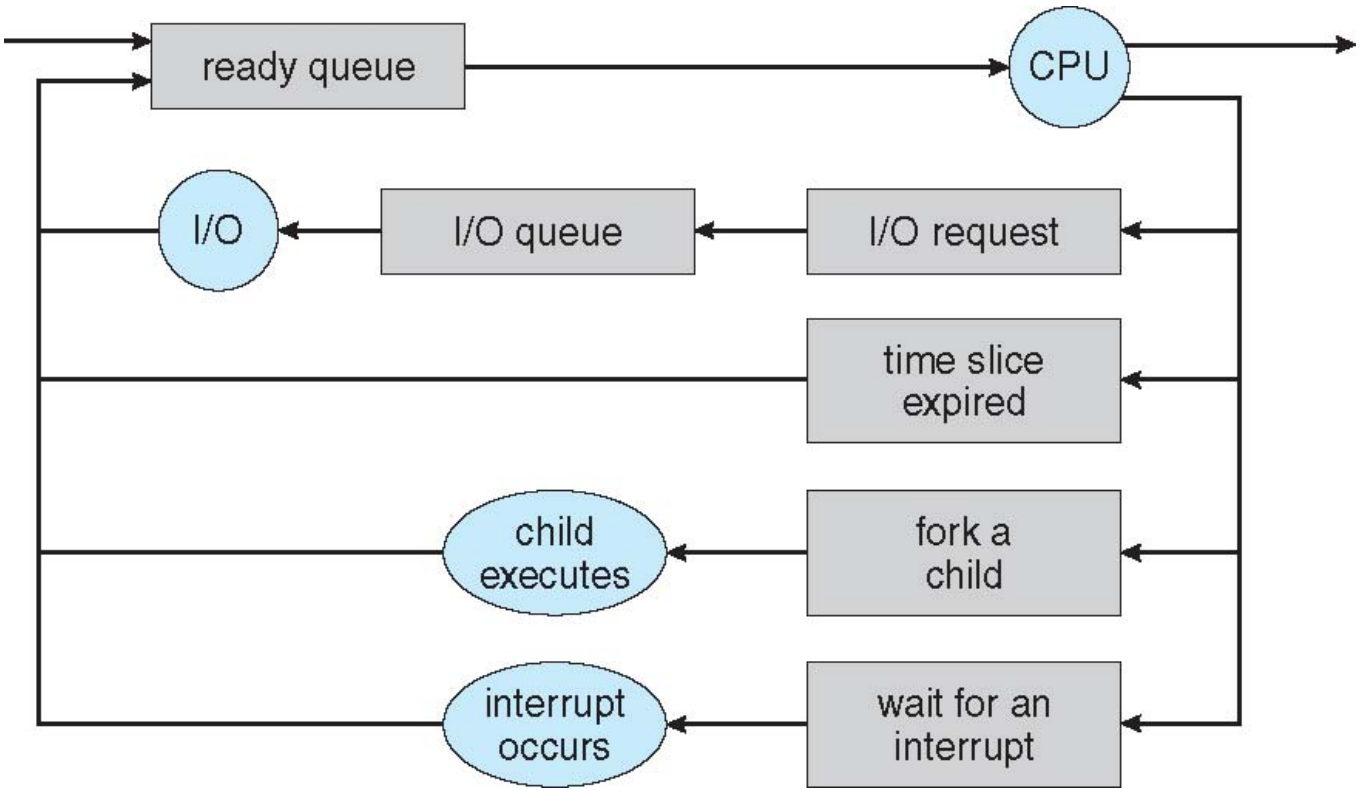
进程调度

- 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

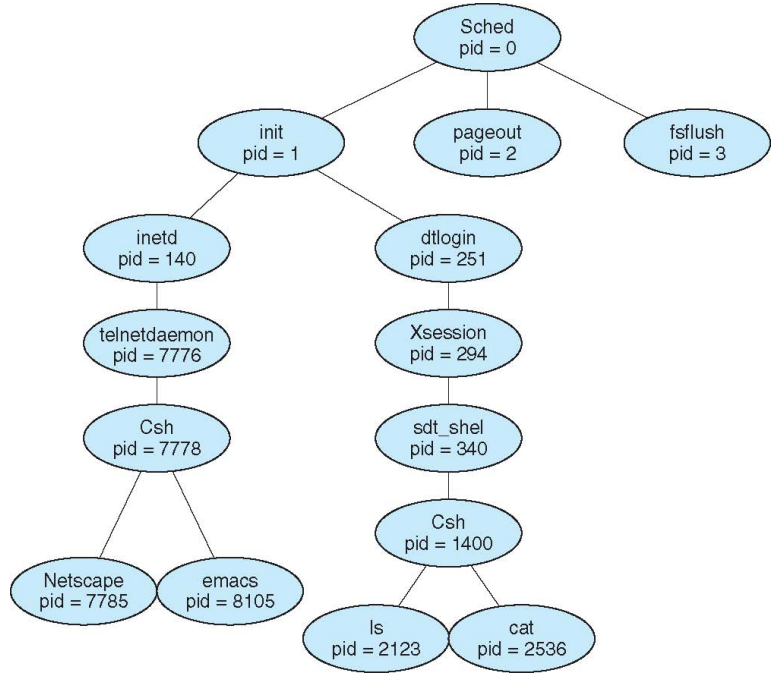

ready queue
CPU
I/O
I/O queue
I/O request
time slice expired
child executes
fork a child
interrupt occurs
wait for an interrupt

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



线程的概念

- （回顾）进程包含:
  - 程序（代码）
  - 数据
  - 栈
  - PCB(进程控制块)
- 所有这些都需要驻留内存相应位置
- 进程（上下文）切换纯属额外开销

线程的概念

可以发现，进程概念可以分割成两块：

- 资源分配单位
  - 内存空间
  - I/O设备，文件等
- 执行单位
  - 单一执行路径
  - 状态：寄存器、栈等

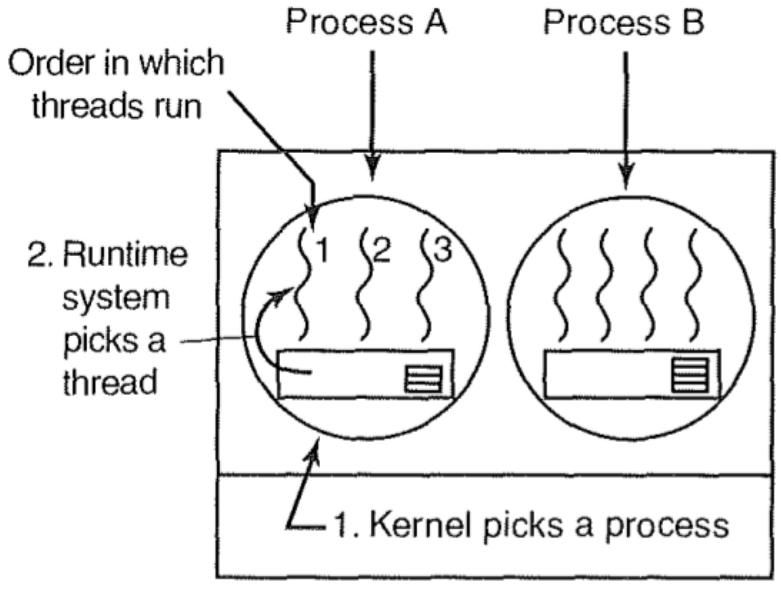
线程的概念

- 重新定义术语：
  - 进程：资源分配单位
  - 线程：执行单位（轻量级进程）
- 多线程技术：支持一个进程中同时运行多个线程
- Why? (I/O阻塞？多核？用户界面？)

为什么引入多线程技术?

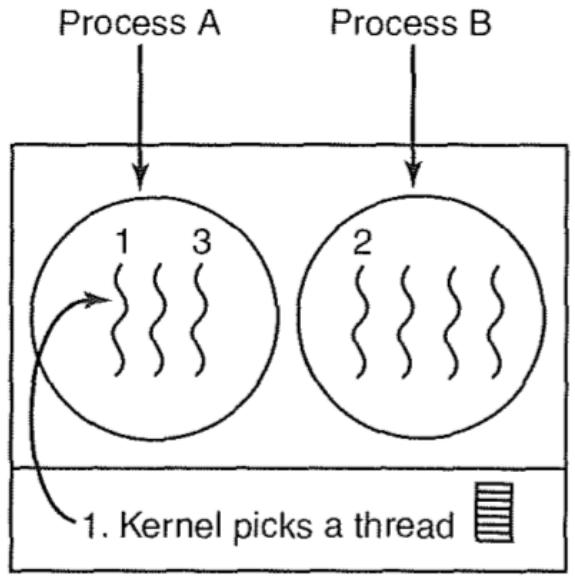
- 应用程序可能同时执行多个动作(例如文字编辑器)
- 线程比进程更容易创建和销毁
- 如果程序部分因I/O阻塞，其余线程可以运行
  - CPU密集型与I/O密集型并行
  - 加快系统速度

多线程技术的应用举例



Possible: A1, A2, A3, A1, A2, A3
Not possible: A1, B1, A2, B2, A3, B3

(a)



Possible: A1, A2, A3, A1, A2, A3
Also possible: A1, B1, A2, B2, A3, B3

(b)