中国科学院大学
University of Chinese Academy of Sciences

操作系统课程 **B0911010Y-01**

进程

中国科学院大学计算机与控制学院
中国科学院计算技术研究所

2021-09-06

内容提要

- 进程的起源
- 进程的概念与表示
- 进程的状态
- 进程的操作原语（API）
- 进程、并发和并行

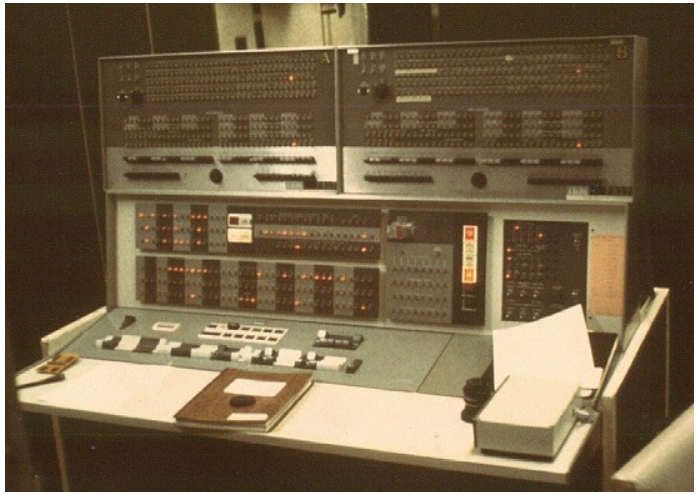
回顾：Intro01

- 操作系统的发展
  - IBM 701计算机 1954
    - OS是一个通用函数库、用户自行提交程序
    - Each user was allocated a minimum 15-minute slot, of which time he usually spent 10 minutes in setting up the equipment to do his computation . . . By the time he got his calculation going, he may have had only 5 minutes or less of actual computation completed-wasting two thirds of his time slot.

      [Hansen, The Evolution of Operating Systems, 2000]
  - IBM 709计算机 1959
    - 支持批处理
  - 1960s
    - 支持MultiProgramming，多个程序同时运行
    - CPU、内存、存储资源共享与隔离
  - Multics 1965、UNIX 1974
    - Time-sharing Operating System

IBM 7090/7094：分时共享

- Compatible Time Sharing System (CTSS)，在IBM 7090机器上第一次实现了多个程序同时运行，“进程”开始登上历史舞台
- Fernando J. Corbató
- 1990年图灵奖



内容提要

- 进程的起源
- 进程的概念与表示
- 进程的状态
- 进程的操作原语（API）
- 进程、并发和并行

程序视图

- 示例程序

```
#include <stdio.h>

void calculate() {
    int *array = malloc(5*sizeof(int));
    array[0] = 1;
    for(int i=1;i<5;i++) {
        array[i] = array[i-1] + i;
    }
}

int main(int argc, char *argv[]) {
    int fd = open("/tmp/examplefile", O_CREAT|O_WRONLY, S_IRWXU);
    calculate()
    write(fd, ”Finish\n", 7);
    close(fd);
    return 0;
}
```

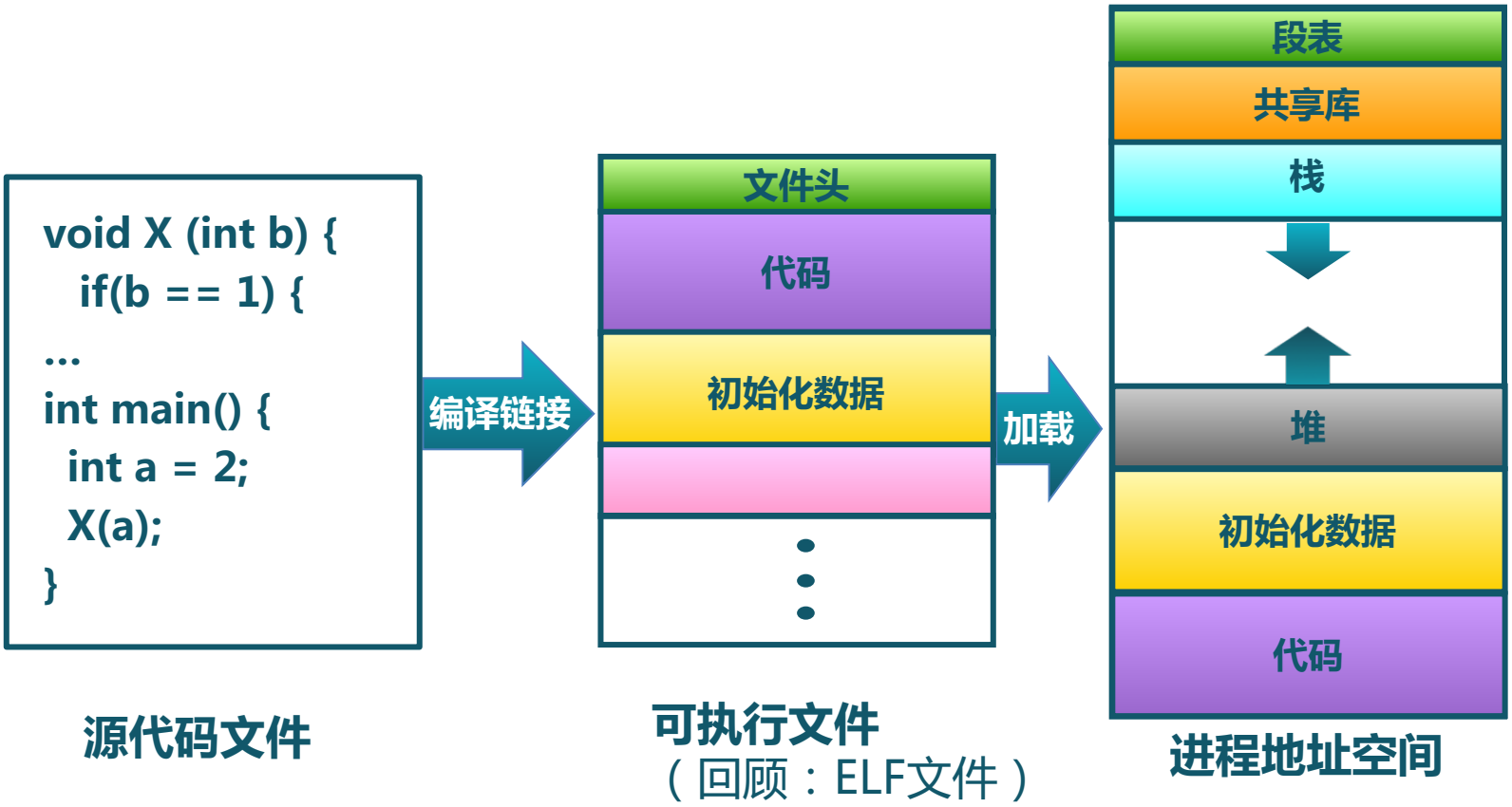
内存

CPU

IO设备

进程的概念

- 进程是指一个具有一定独立功能的程序在一个数据集合上的一次动态执行过程
- 进程刻画了一个程序运行所需要的资源和运行状态

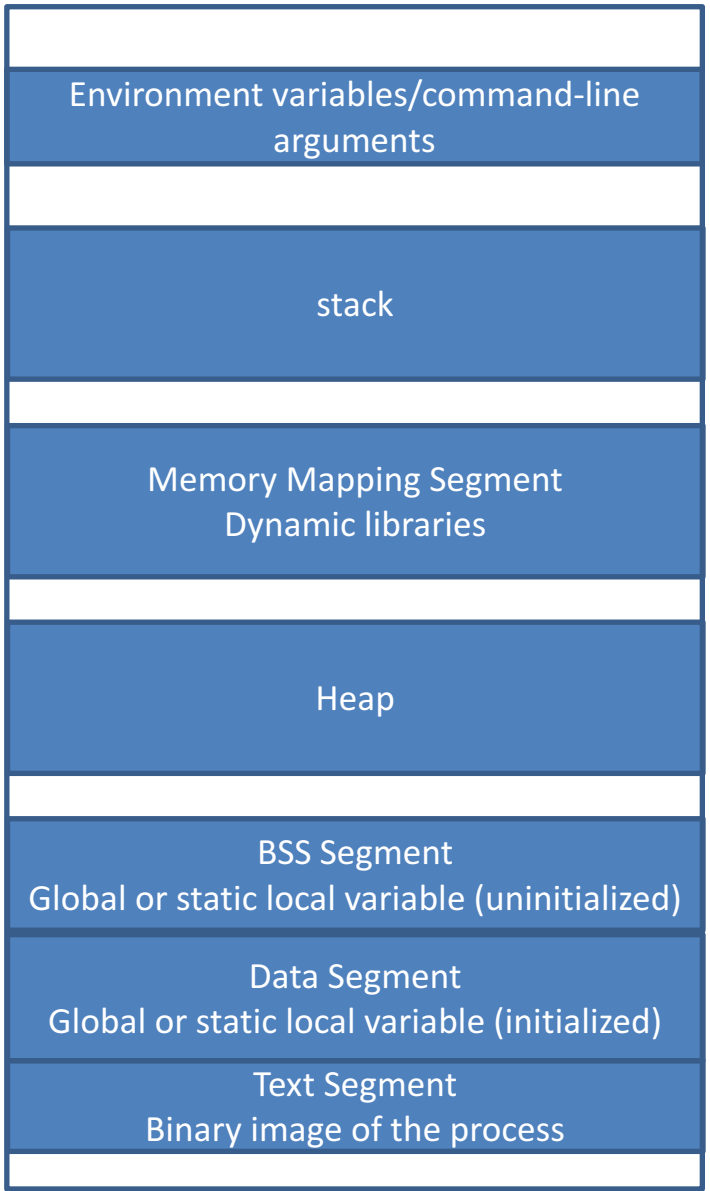


进程CPU资源

- 寄存器
- Program Counter/Instruction Pointer
  - 例如，x86 EIP寄存器
- 栈指针
  - 例如，x86 ESP，EBP寄存器
- 其他标识信息
  - 例如，运行状态，EFLAGS寄存器

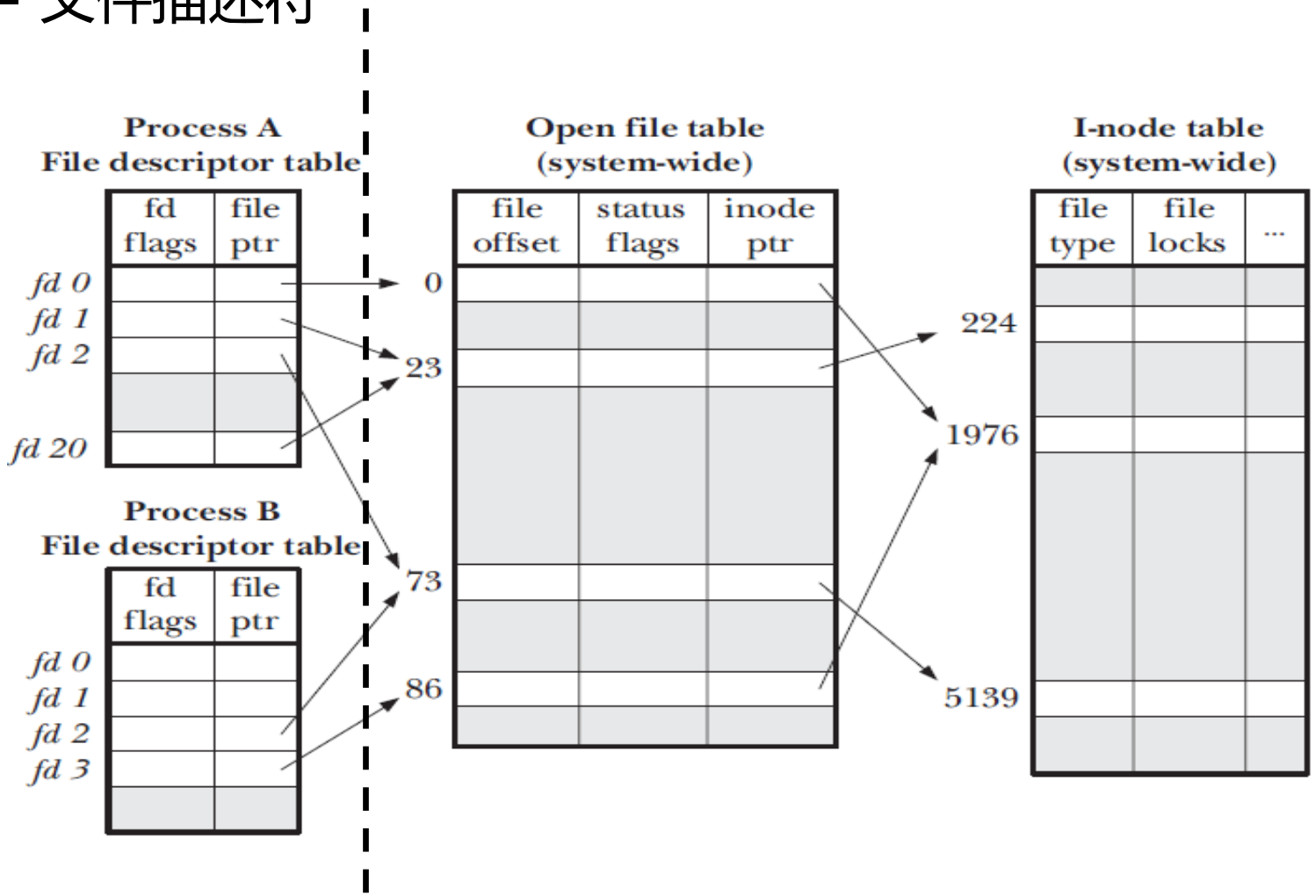
进程内存资源

- 抽象：进程地址空间
  - Address Space



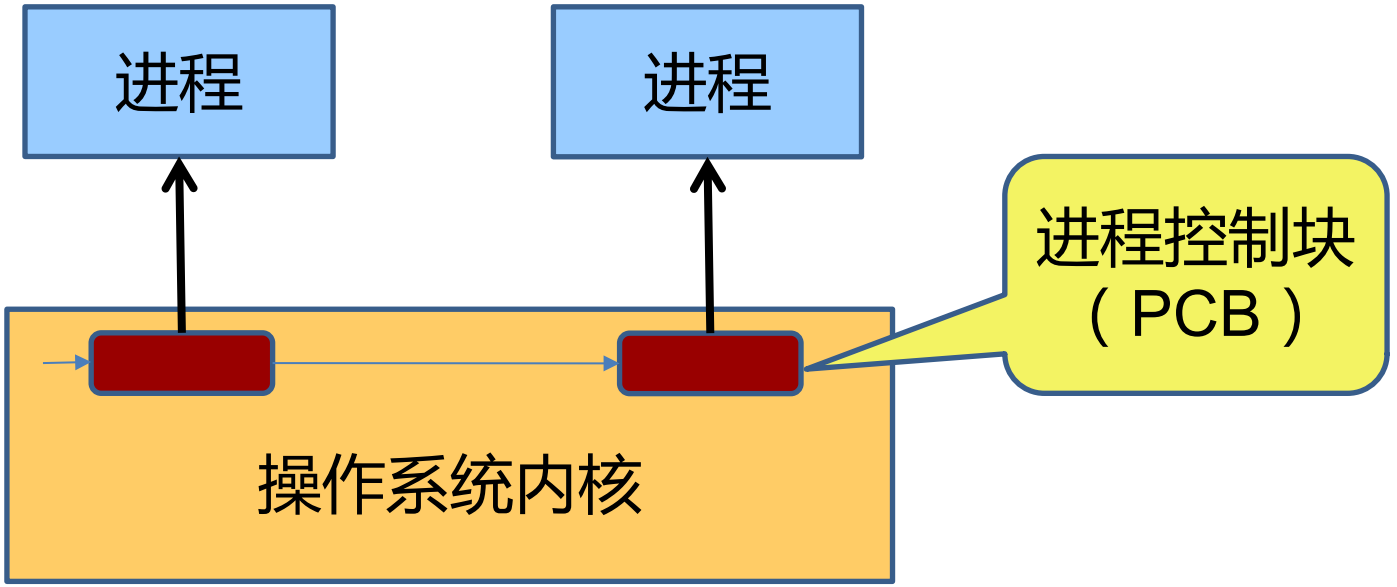
进程IO设备资源

- 抽象：文件
  - 文件描述符



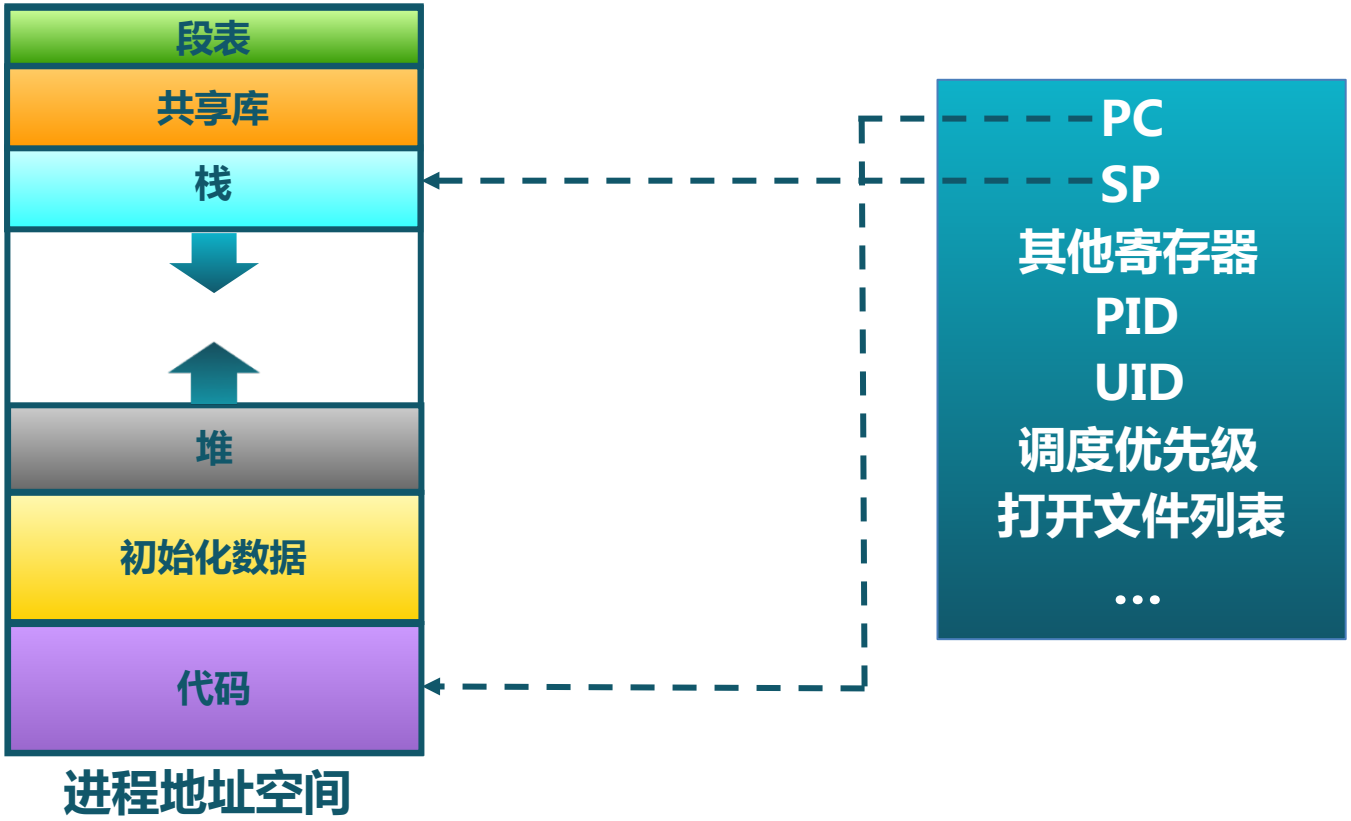
进程在内核中的表示

- 每个进程的创建与销毁都由内核负责，每个进程都需要在内核登记信息
- 内核用进程控制块（Process Control Block，PCB）来保存进程的信息
- PCB是进程在内核中的表示，也是一种索引



进程控制块包含的信息

- 进程标识信息：进程ID、进程名称
- 与各种资源相关的信息
  - CPU、内存、文件、通信、……



进程控制块包含的信息

- **CPU相关的进程管理信息**
  - 状态
    - 就绪态：准备运行
    - 运行态：正在运行
    - 阻塞态：等待资源
  - 寄存器，EFLAGS，以及其他的CPU状态
  - 优先级
- **内存管理信息**
  - 栈（用户栈、内核栈），代码段和数据段
  - 页表，统计信息等
- **I/O和文件管理**
  - 通信端口，目录，文件描述符等

扩展了解：xv6代码

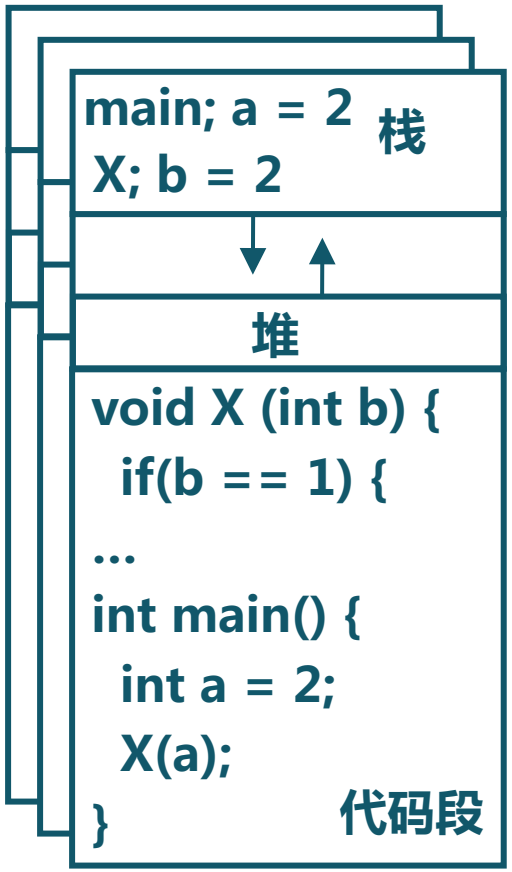
进程 vs. 程序

- 关联
  - 程序是进程的一部分（e.g. 代码段）
  - 进程是程序的动态执行
- 区别
  - 一个程序可以创建多个进程，例如一个程序同时运行多次
  - 进程还包括运行时状态和使用的资源信息

程序

```
void X (int b) {
  if(b == 1) {
…
int main() {
  int a = 2;
  X(a);
}
```

进程



Linux进程（top命令）

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
642	hadoop	20	0	4620m	236m	7412	S	0.3	3.0	31:26.21	java
6454	hadoop	20	0	2776m	312m	7064	S	0.3	3.9	170:35.51	java
22033	hadoop	20	0	2989m	277m	7516	S	0.3	3.5	209:24.10	java
26712	hadoop	20	0	15132	1312	944	R	0.3	0.0	0:01.24	top
30468	hadoop	20	0	5849m	1.1g	7668	S	0.3	14.1	74:01.21	java
1	root	20	0	19356	792	568	S	0.0	0.0	0:24.26	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.02	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:38.49	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:15.07	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
6	root	RT	0	0	0	0	S	0.0	0.0	2:06.15	watchdog/0
7	root	RT	0	0	0	0	S	0.0	0.0	0:53.46	migration/1
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/1

扩展了解：Linux的init进程是如何工作的？

Windows进程（任务管理器）

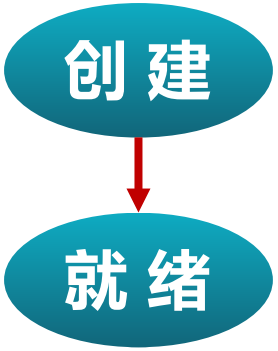
名称	CPU	内存	磁盘	网络
Bonjour Service	0%	1.0 MB	0 MB/秒	0 Mbps
Camera Mute Control Servic...	0%	0.1 MB	0 MB/秒	0 Mbps
COM Surrogate	0%	1.0 MB	0 MB/秒	0 Mbps
Communications Utility laun...	0%	0.8 MB	0 MB/秒	0 Mbps
Cortana	0%	101.6 MB	0 MB/秒	0 Mbps
Device Association Framewo...	0%	1.1 MB	0 MB/秒	0 Mbps
Dropbox (32 位)	0%	18.7 MB	0.1 MB/秒	0 Mbps
Evernote Clipper (32 位)	0%	0.2 MB	0 MB/秒	0 Mbps
Evernote Tray Application (3...	0%	0.2 MB	0 MB/秒	0 Mbps
Google Chrome (32 位)	0.7%	58.7 MB	0 MB/秒	0 Mbps
Google Chrome (32 位)	0%	17.0 MB	0 MB/秒	0 Mbps
Google Chrome (32 位)	0%	32.3 MB	0 MB/秒	0 Mbps
Google Chrome (32 位)	0%	42.5 MB	0 MB/秒	0 Mbps

内容提要

- 进程的起源
- 进程的概念与表示
- 进程的状态
- 进程的操作原语（API）
- 进程、并发和并行

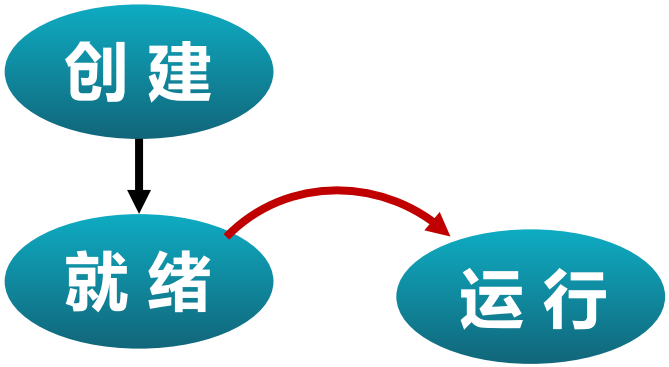
进程创建

- 什么时候会创建进程？
  - 系统初始化时
  - 用户请求创建一个新进程
    - 例如，在shell执行命令
  - 正在运行的进程执行了创建进程的系统调用
    - 例如，调用fork
- 创建进程
  - 创建与初始化PCB
  - 创建地址空间、调用栈、堆
  - 将数据和代码加载至内存
  - 初始化进程的状态，把进程标志为就绪态



进程运行

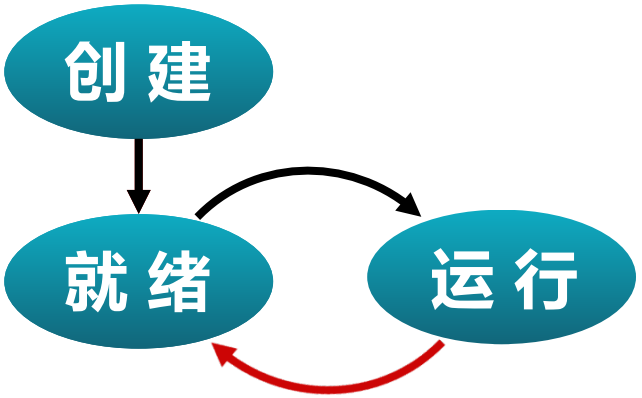
- 内核选择一个就绪的进程，为它分配一个处理器的时间片，并开始执行（时间片倒计时）



- 如何选择就绪的进程？
  - 进程调度算法
  - 进程优先级
  - 思考：之前没有运行完的进程如何处理？

进程抢占

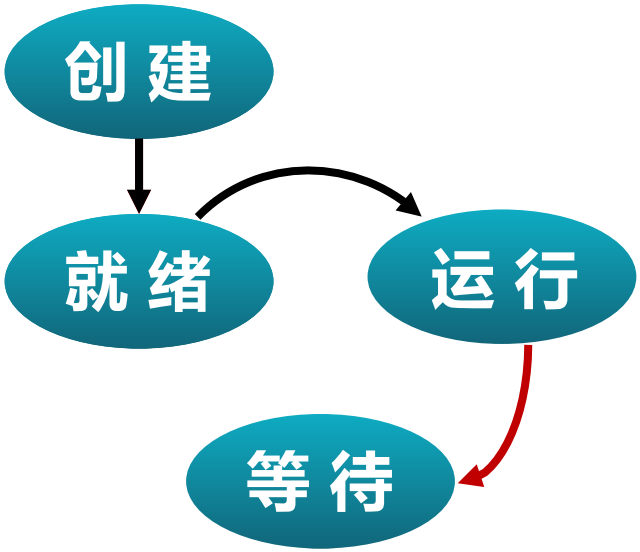
- 进程时间片用完
  - 设计选择1
    - 进程若未执行完，则继续执行
  - 设计选择2：
    - 允许有高优先级进程，在就绪时抢占CPU并运行
    - 低优先级进程回到就绪状态
- 系统设计结果
  - 非抢占式内核
    - 一个进程运行直到其主动退出，让出CPU资源
  - 抢占式内核
    - 时间片用完时，调度高优先级进程
- 问题
  - 时间片未用完时，高优先级进程就绪，是否可以执行？



扩展了解：实时系统

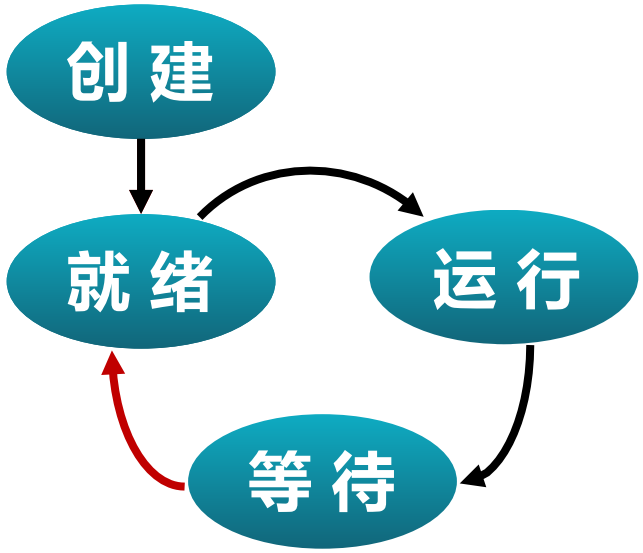
进程等待

- 进程进入等待(阻塞)的情况
  - 需要的数据没有到达/需要的资源没有获得
    - 例如，等待用户输入，等待锁
  - 进程主动进入等待
    - 例如，sleep
  - 请求并等待系统服务，无法马上完成
    - 例如，内存页分配，磁盘读、同步写



进程唤醒

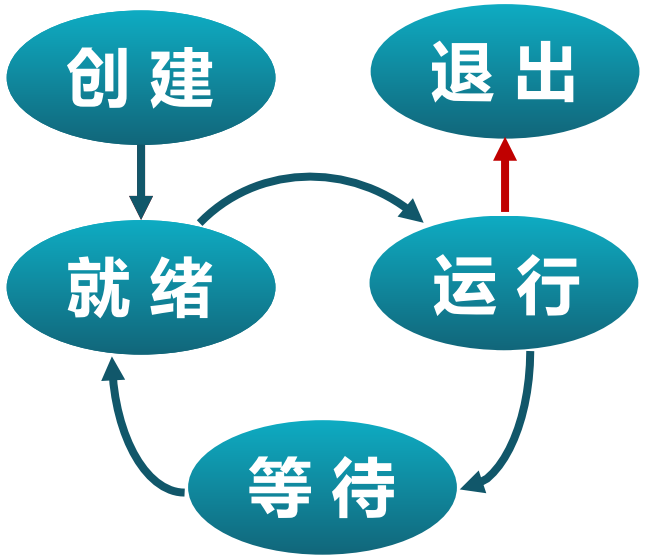
- 唤醒进程的情况
  - 被阻塞进程需要的资源被满足
  - 被阻塞进程等待的事件到达
- 进程只能被别的进程或操作系统唤醒



进程结束

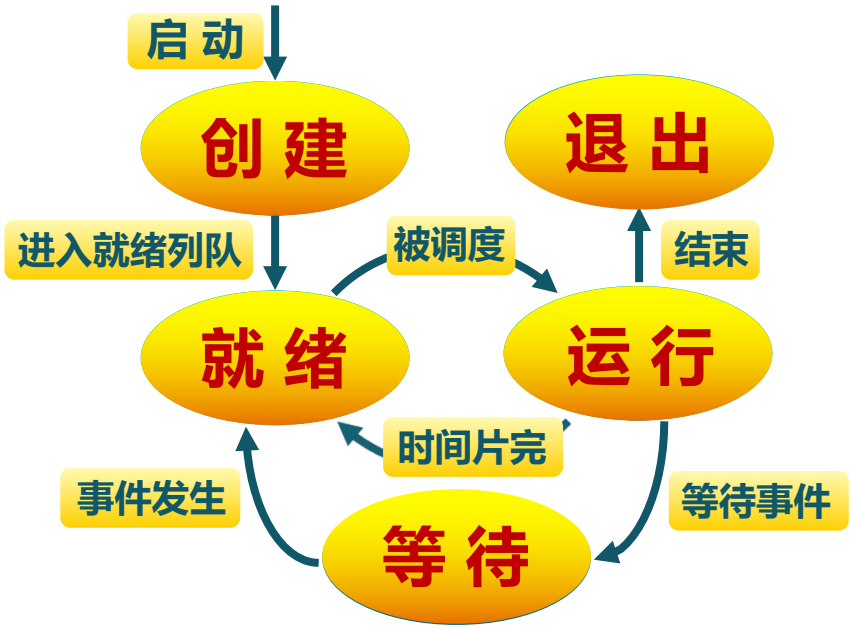
进程结束的情况：

- 正常退出(自愿的)
- 错误退出(自愿的)
- 致命错误(强制性的)
- 被其他进程所杀(强制性的)



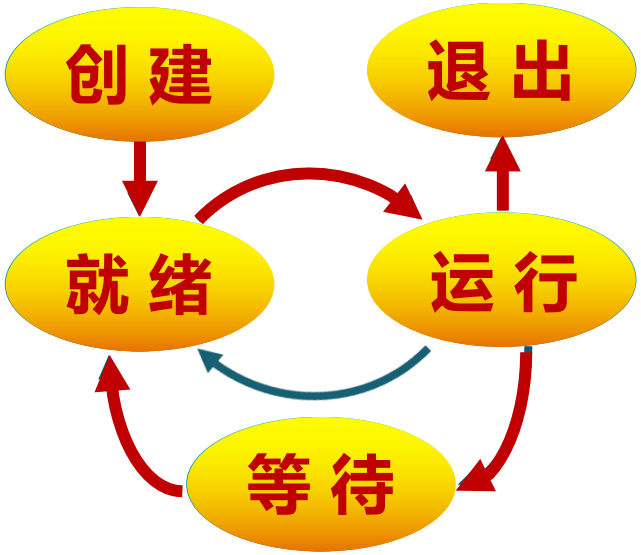
进程状态

- 进程状态转换图



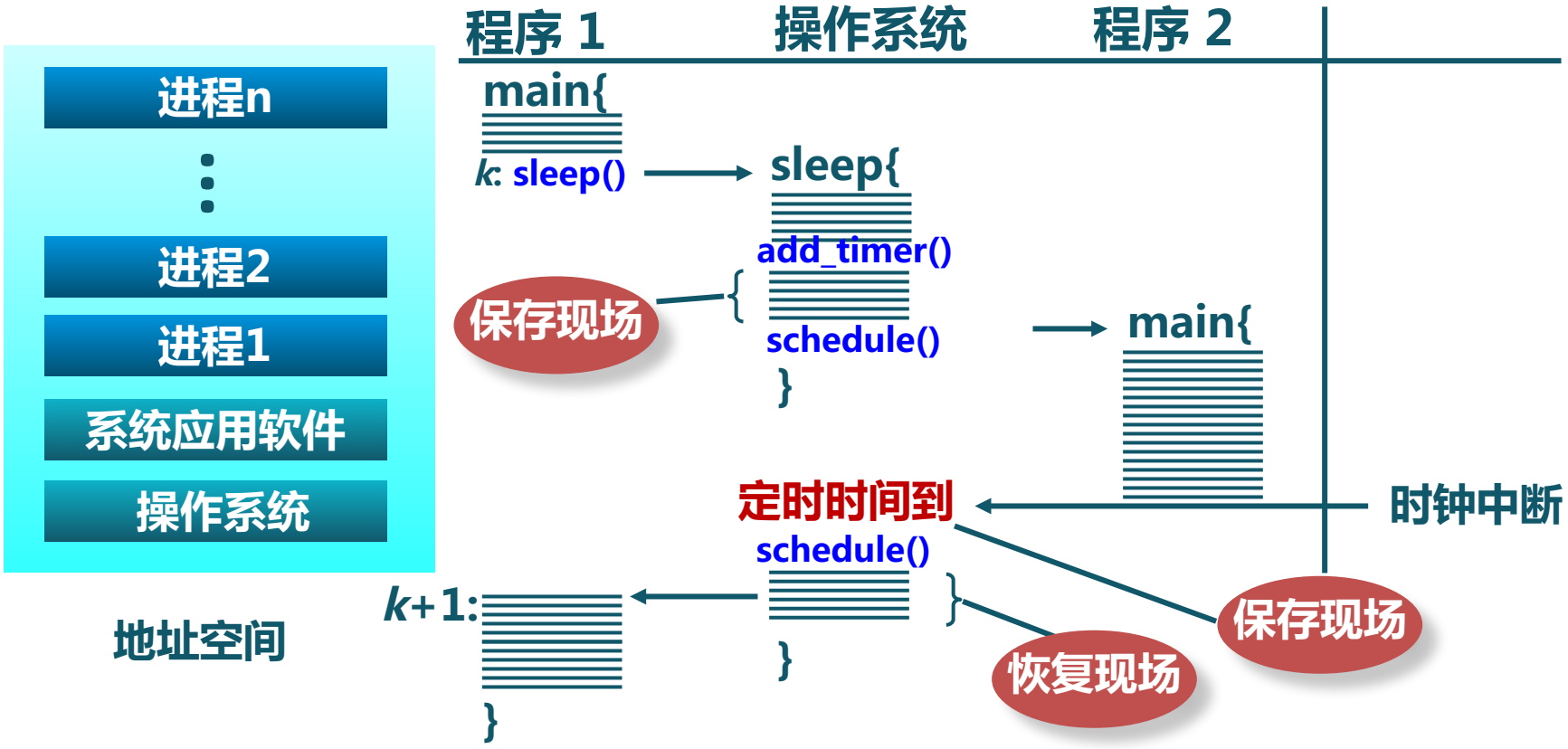
进程状态

- 示例
  - sleep()系统调用时的进程状态变化



进程切换

- 示例
  - schedule负责从就绪队列中选择进程执行



进程上下文切换

- 保存上下文
  - 保存所有寄存器（通用寄存器和浮点寄存器）及标志位状态，例如EFLAGS，EIP，CS等
  - 保存所有协同处理器的状态
  - 需要将内存保存到磁盘吗？
  - CPU Cache和TLB该怎么办？
- 恢复上下文
  - 相反的操作过程
- 实现
  - 硬件自动保存/恢复：例如x86 Task Register寄存器，修改其值触发硬件保存所有寄存器值
  - 软件保存/恢复：汇编指令保存相关寄存器，例如Linux SAVE_ALL宏

进程生命周期

- 非抢占式内核
  - 进程创建
  - 进程执行
  - 进程等待
  - 进程结束

- 抢占式内核
  - 进程创建
  - 进程执行
  - 进程抢占
  - 进程等待
  - 进程唤醒
  - 进程结束

内容提要

- 进程的起源
- 进程的概念与表示
- 进程的状态
- 进程的操作原语（API）
- 进程、并发和并行

进程的操作原语（API）

- 创建和终止
  - Fork， Exec ，Wait，Kill
- 操作
  - 阻塞，放弃CPU控制权（yield）
- 信号
  - 信号处理函数
- 同步
  - 锁，信号量，屏障

- 问题：fork, exec, wait, kill原语对PCB进行了什么操作？

fork和exec

- fork
  - 克隆出一个进程
  - 复制当前代码段、数据段、堆栈、页表等
  - 采用写时复制机制
- exec
  - 替换掉当前进程的代码段、数据段、堆栈等

```
If ((pid = fork()) == 0) {
        /* child process */
        exec("foo");    /* does not return */
else
        /* parent */
        wait(pid);      /* wait for child to die */
```

wait和kill

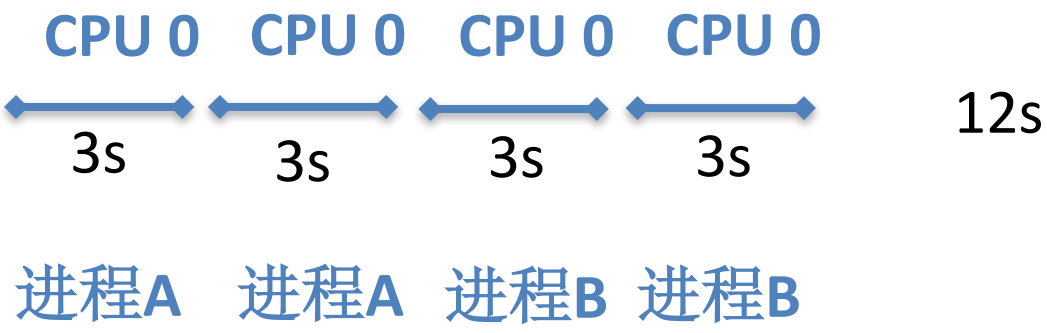
- wait
  - 等待子进程结束
- kill
  - 向进程发送信号。
  - 通常用于结束进程，释放资源
  - 如何处理PCB？

内容提要

- 进程的起源
- 进程的概念与表示
- 进程的状态
- 进程的操作原语（API）
- 进程、并发和并行

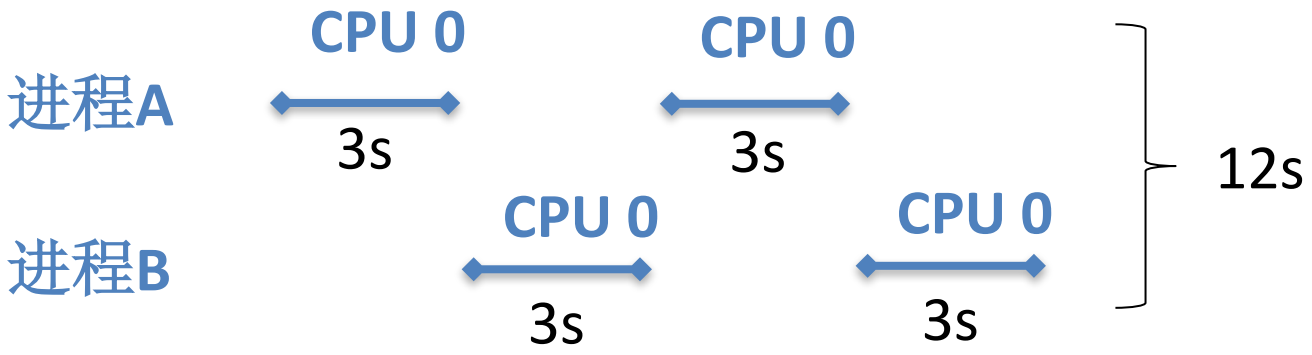
进程的顺序执行

- 示例
  - 进程A和进程B都需要6s完成任务执行
  - 时间片是3s，进程A和B依次执行，没有并发性



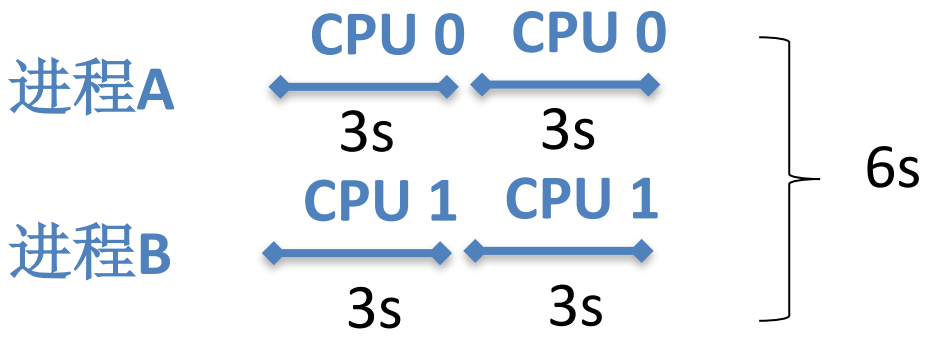
进程和并发性

- 并发性
  - 一个系统中有多个进程“同时”运行（逻辑上）
  - CPU、DRAM和I/O设备是共享的
  - 每一个进程都希望能拥有自己的计算机资源
- 虚拟化（分时复用）
  - 每个进程都运行一段时间（时间片）
  - 使得一个CPU变成“多个”，每一个进程就好像拥有了自己的CPU

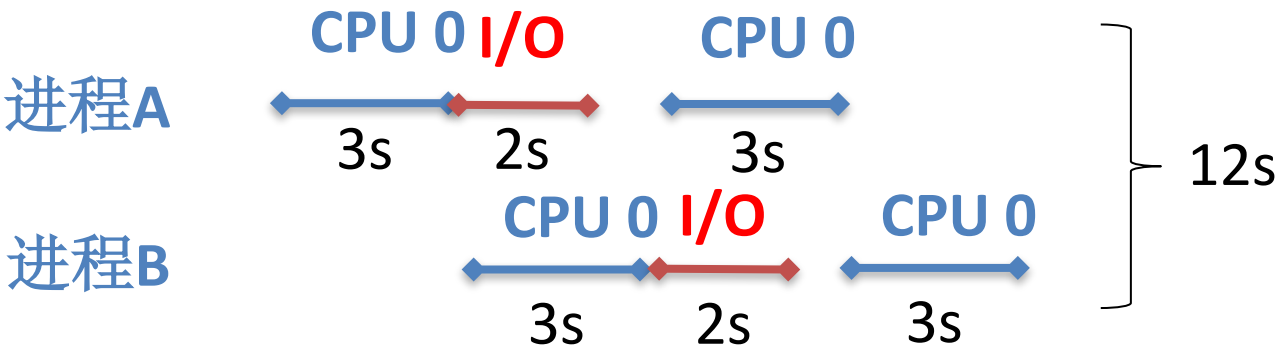


进程和并行性

- CPU并行性
  - 多个CPU（如SMP）
  - 进程在物理上同时运行
  - 用途：加速



- I/O并行性
  - CPU计算与I/O操作交叠
  - 减少总共所需完成时间
  - 如果没有I/O并行，需要多长时间？



并发和并行

- 并发
  - 一个系统能同时处理多个任务的能力，但同一时刻可能只有一个任务在运行
- 并行
  - 一个系统在同一时刻支持多个任务同时运行
- 并行处理收益
  - 将一个复杂问题分解成多个子问题
  - 每个子问题由一个进程处理
  - 多个进程同时处理，减少处理时间

并行性收益

- 并行性在日常生活中很普遍
  - 1个销售员一年的销售额是10万
  - 雇佣10个销售员就可以售出100万
- 加速比
  - 理想情况下可以获得N倍加速比
  - 现实：各种瓶颈 + 协调开销
- 问题
  - 你和1个伙伴合作可以加速完成任务吗？
  - 你和20个伙伴合作可以加速完成任务吗？
  - 你可以获得超线性的加速比吗（大于N倍）？

总结

- 进程的起源
- 进程的概念与表示
  - CPU、内存和IO等资源的抽象
  - PCB
- 进程的状态
  - 非抢占式内核和抢占式内核
  - 等待和唤醒
  - 进程切换
- 进程的操作原语（API）
  - fork，exec，wait，kill
- 进程、并发和并行
  - 并发和并行的区别
  - 并行的收益