

第3章 进程描述和控制

3.1* 什么是进程

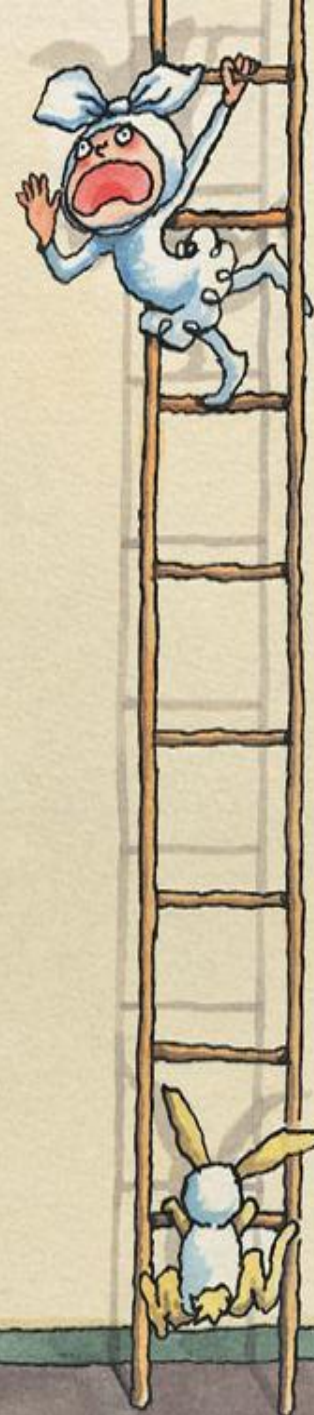
3.2* 进程状态

3.3* 进程描述

3.4* 进程控制

3.5 操作系统的执行

3.6 UNIX SVR4进程管理



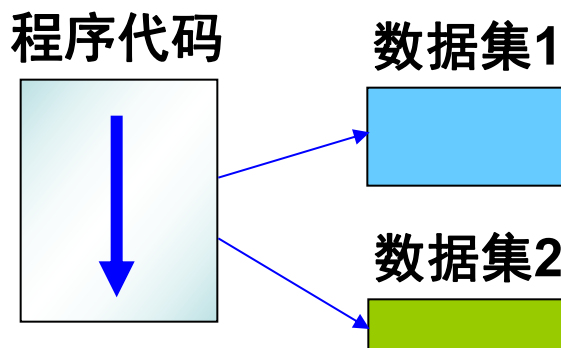
3.1 什么是进程 process



3.1.2 进程和进程控制块



- 进程：程序在一个数据集上的一次执行过程。
- 进程与程序的联系与区别：
 - 程序是静态的；进程是动态的（动态产生并消亡），且一个进程运行时可以创建其它进程。
 - 一个程序可对应一个进程，也可对应多个进程（只要进程所对应的数据集不同）。



3.1.2 进程和进程控制块



● 一个进程的组成（进程映像）：

- 程序代码

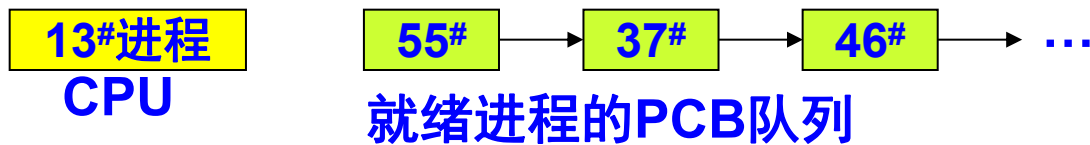
- 数据集、栈

- 进程控制块 **Process Control Block**

PCB是进程存在的唯一标识，
OS根据PCB中的属性控制进程

● **上下文**：进程运行时CPU的寄存器数据集(现场)。包括用户可见寄存器和控制/状态寄存器等。

● **分派器**(即进程调度程序)**调度**时发生**上下文切换**：保存旧进程的上下文到它的PCB，从新进程的PCB恢复它的上下文到寄存器。



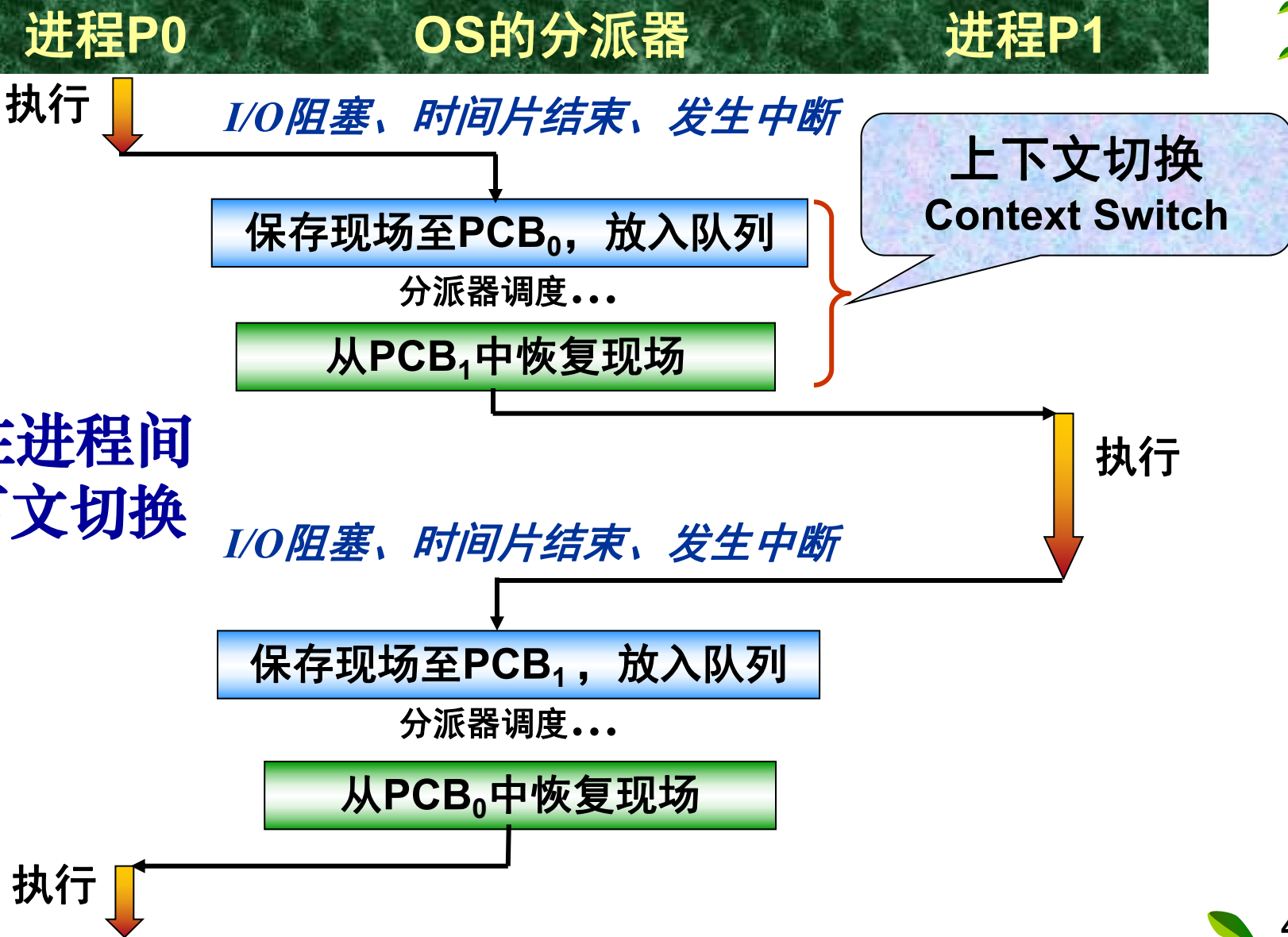
PCB结构将在3.3.2节描述



3.1.2 进程和进程控制块



CPU在进程间的上下文切换



Question:

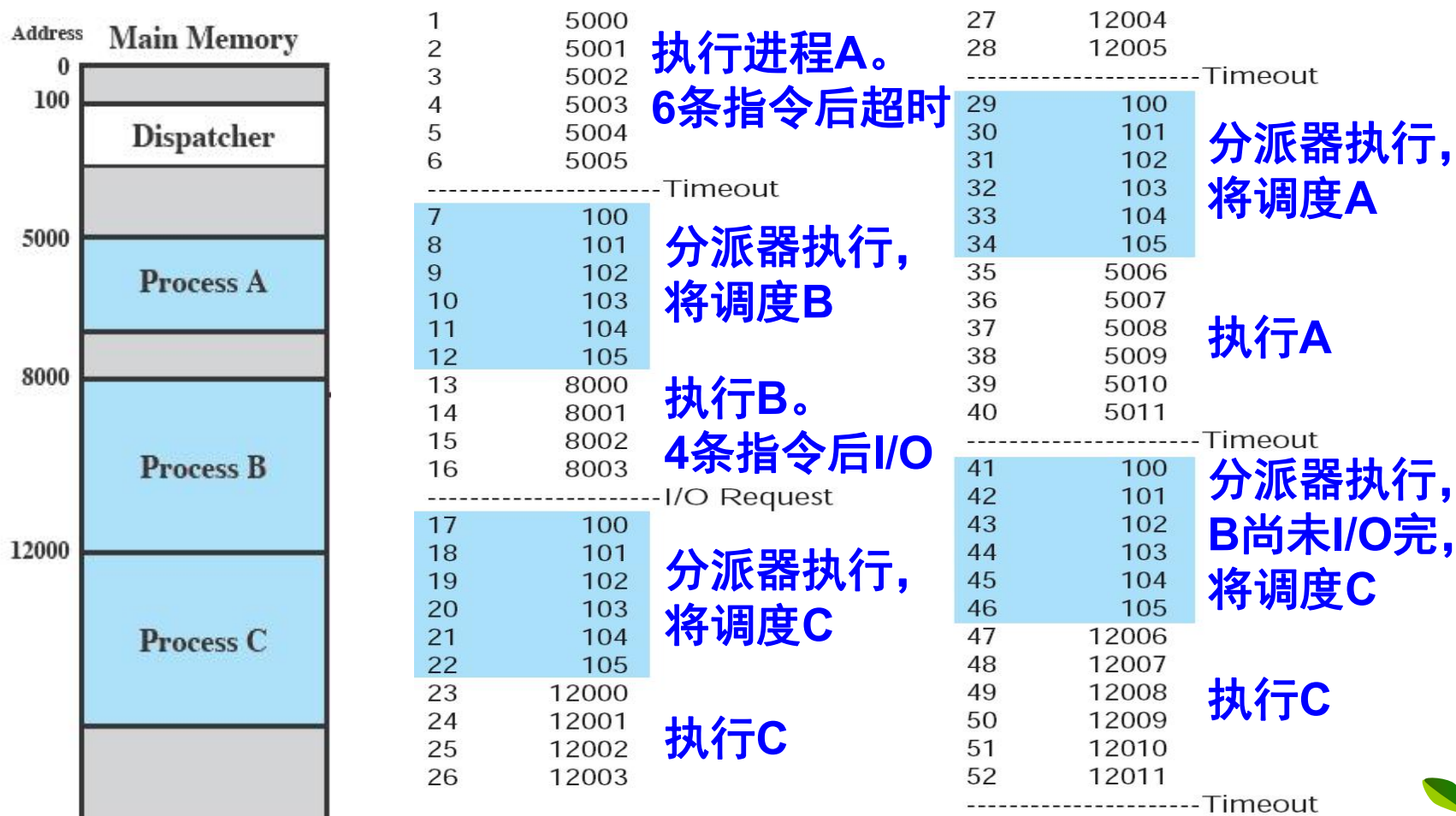


引入进程，由于实现了并发执行和资源共享，可带来提高资源利用率和吞吐率的好处，但却增加了系统的时间和空间开销。

3.2 进程状态



- **分派器**：选择并调度一个进程，使之占用CPU运行。
- 进程A, B, C及分派器的并发执行轨迹：假设每个进程连续执行6条指令后时间片结束（超时），让出CPU。



3.2.2 进程的创建和终止

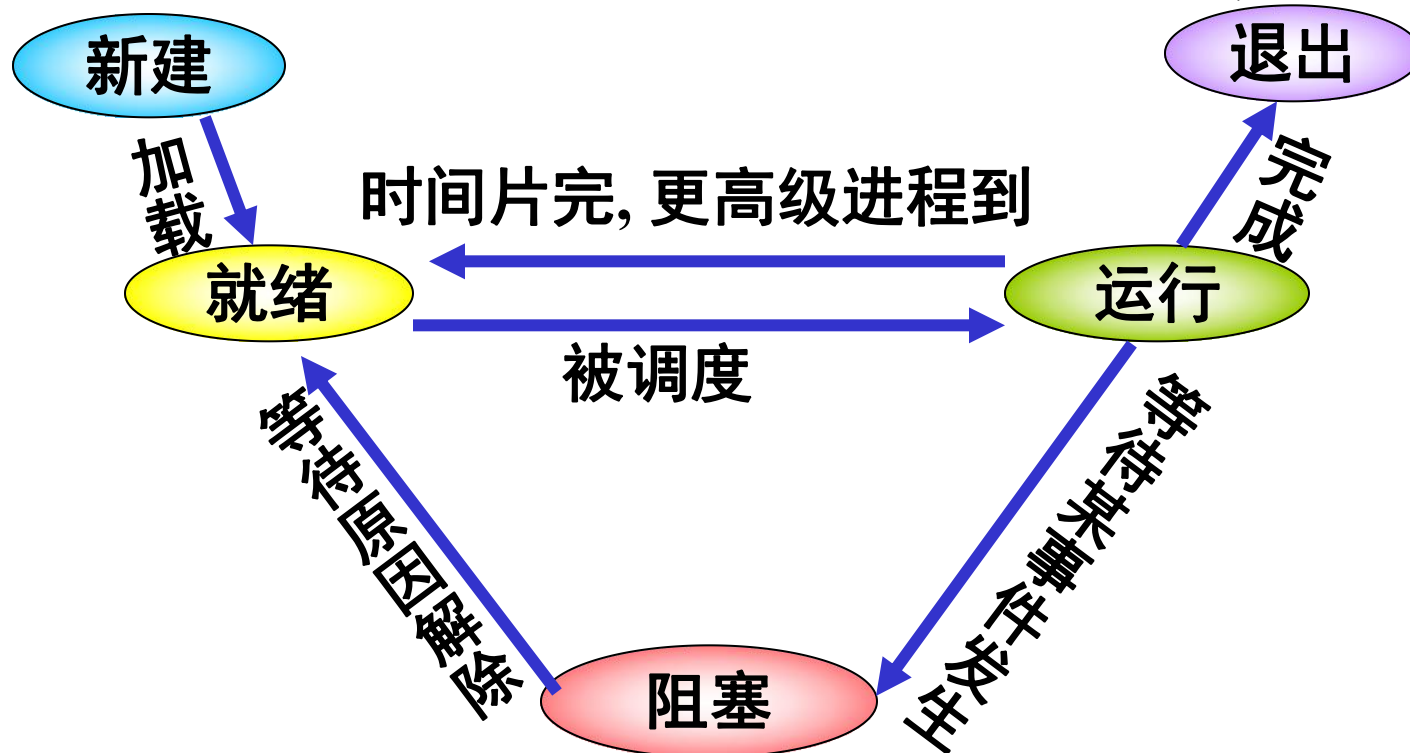


- **进程创建**：OS为该进程建立PCB，分配内存空间。
- **导致进程创建的原因**：
 - 新的批处理作业：作业提交后，开始运行时创建进程。
 - 交互登录：新用户登录和接收用户命令时创建进程。
 - OS提供服务：如控制打印、网络通信等的服务进程。
 - 父进程派生子进程：父进程请求创建子进程。
- **进程终止**：回收内存，释放资源，销毁PCB。
- **导致进程终止的原因**：
 - 进程正常运行完毕；用户或OS干预；父进程请求或父进程已终止；运行时发生的各种故障和错误。

3.2.1 两状态进程模型 (略)

3.2.3 五状态模型—五种进程状态

- **新建new**: 进程正被创建。分配内存后将被设为就绪态。
- **就绪ready**: 进程已得到除CPU以外的其它所需资源。
- **运行running**: 进程的指令正被执行。
- **阻塞(等待)blocked**: 进程正等待资源或某事件发生。
- **退出exit**: 进程已正常或异常结束。回收资源, 善后。



3.2.3 五状态模型—进程队列

- 处于同一状态的进程PCB组成一个进程队列。
 - 就绪队列：所有就绪进程按FCFS或优先级顺序排队。
 - 等待(阻塞)队列：每一种等待事件对应于一个队列。
例如：等待某一I/O设备的进程组成一个设备阻塞队列。

时间片结束（超时）



3.2.3 五状态模型—PCB数量

● OS中PCB的数量固定或不固定，但总是有限的：

- Solaris、Linux进程最大数量可调。默认=(内存大小/内核堆栈大小)/2，如 512M/8K/2=32768。

《莱昂氏UNIX源代码分析》
1975年UNIX版本6中PCB的定义

```
0358 struct      PROC
0359 {
0360     char      p_stat;
0361     char      p_flag;
0362     char      p_pri; /* priority, negative is high */
0363     char      p_sig; /* signal number sent to this process */
0364     char      p_uid; /* user id, used to direct tty signals */
0365     char      p_time; /* resident time for scheduling */
0366     char      p_cpu; /* cpu usage for scheduling */
0367     char      p_nice; /* nice for scheduling */
0368     int       p_tty; /* controlling tty */
0369     int       p_pid; /* unique process id */
0370     int       p_ppid; /* process id of parent */
0371     int       p_addr; /* address of swappable image */
0372     int       p_size; /* size of swappable image (*64 bytes) */
0373     int       p_wchan; /* event process is awaiting */
0374     int       *p_text; /* pointer to text structure */
0375
0376 } Proc[NPROC];
```

常量，=50，因此系统中同时存在的进程总数是50个。

Question:



1、一个多任务双核处理机系统，其操作系统是UNIX，PCB表的规模是100行，则：

最多有 2 个进程处于运行态；

最多有 98 个进程处于就绪态；

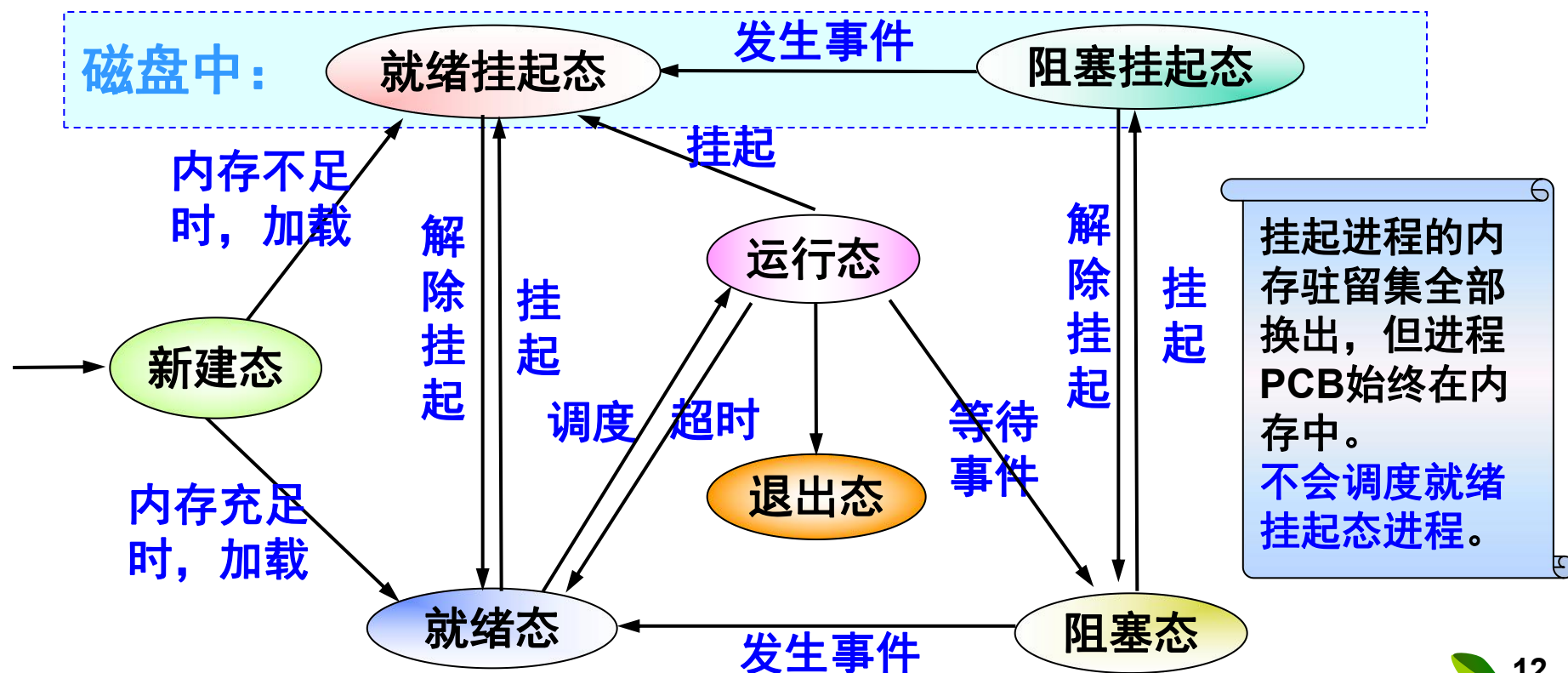
最多有 100 个进程处于阻塞态。

2、设某UNIX中每个用户创建进程数最大为50个，现有一用户执行某程序，该程序执行一死循环，循环创建新子进程。则当该进程创建了 49 个子进程后将不能再创建，该进程处于 阻塞 状态。

3.2.4 被挂起的进程—两种挂起态



- **挂起的根本原因**：内存不足，不得不把部分进程交换到磁盘。
- **就绪挂起态、阻塞挂起态**：外存就绪/阻塞态。由于内存有限，将原位于内存的就绪/阻塞进程(代码数据)换出到外存(磁盘)上。
- **解除挂起**：当挂起进程优先级高或内存空间足够时，把位于磁盘的挂起进程(代码数据)换入到内存。

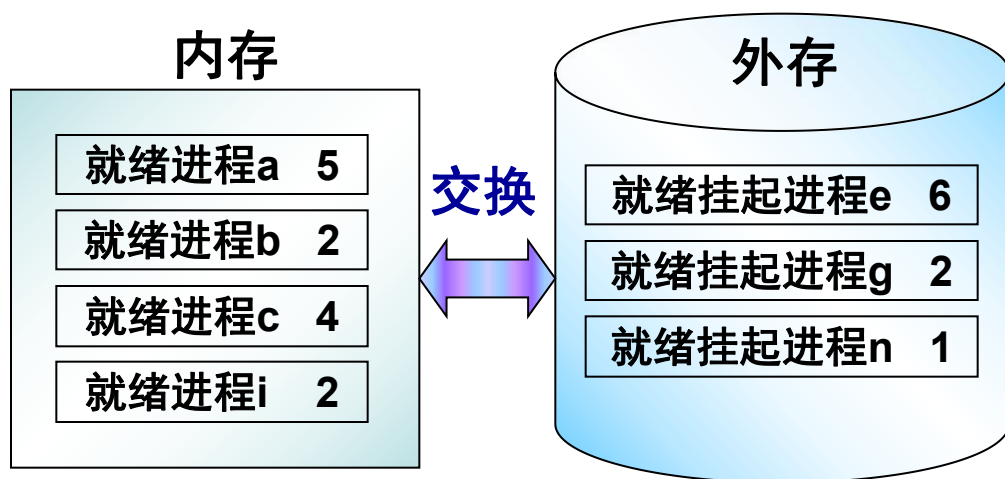


Question:



课后习题3.5：假设有些进程处于就绪态，有些处于就绪挂起态，且至少有一个就绪挂起态进程的优先级高于所有就绪态进程。有两种极端策略：

- 1、总是调度处于就绪态的进程，以减少交换；
 - 2、总是调度优先级最高的进程，会导致不必要的交换。
- 请给出一种均衡考虑优先级和交换性能的中间策略。



把就绪挂起进程降低
(1或2个) 优先级，
如果它仍然高于所有
就绪进程的最高优先
级，那么就将它换入
内存执行。

3.3 进程描述

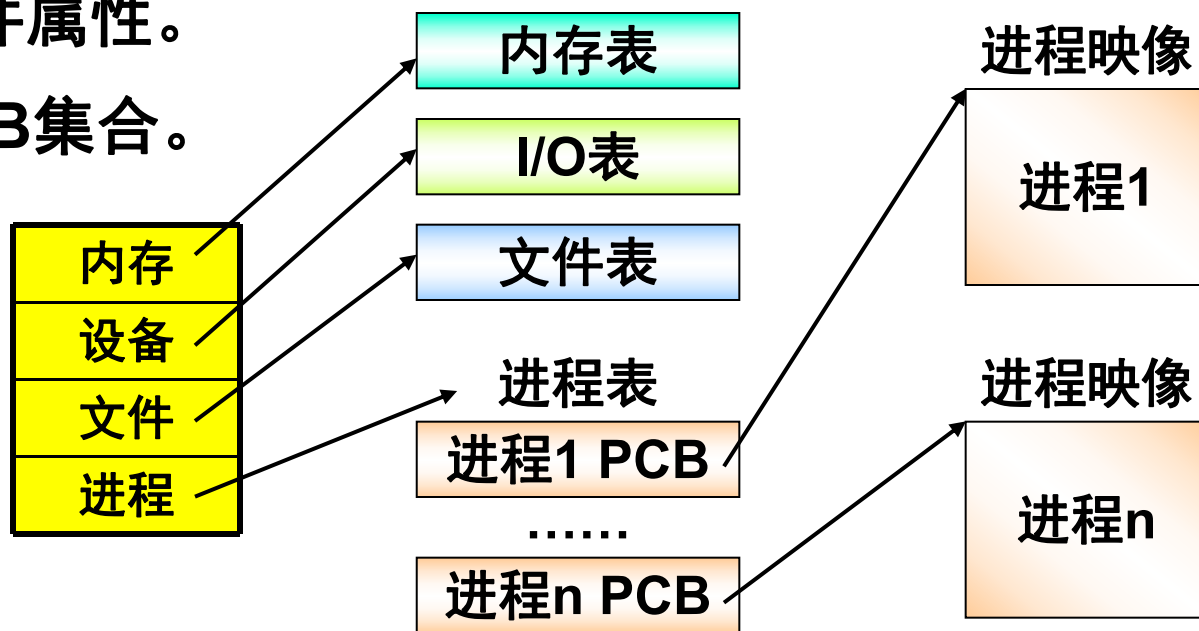


3.3.1 操作系统的控制结构



● OS构造并维护所管理的每个实体的信息表，4类：

- **内存表**：跟踪内存（实存）和外存（虚拟内存）。
- **I/O表**：管理I/O设备和通道。
- **文件表**：文件属性。
- **进程表**：PCB集合。



3.3.2 进程控制结构—PCB

- **PCB**: 进程属性的集合。
- OS进程表中包含所有进程的PCB。
- 创建新进程时, 分配进程表中的一个空闲PCB, 填写属性。
- 进程终止时回收PCB。

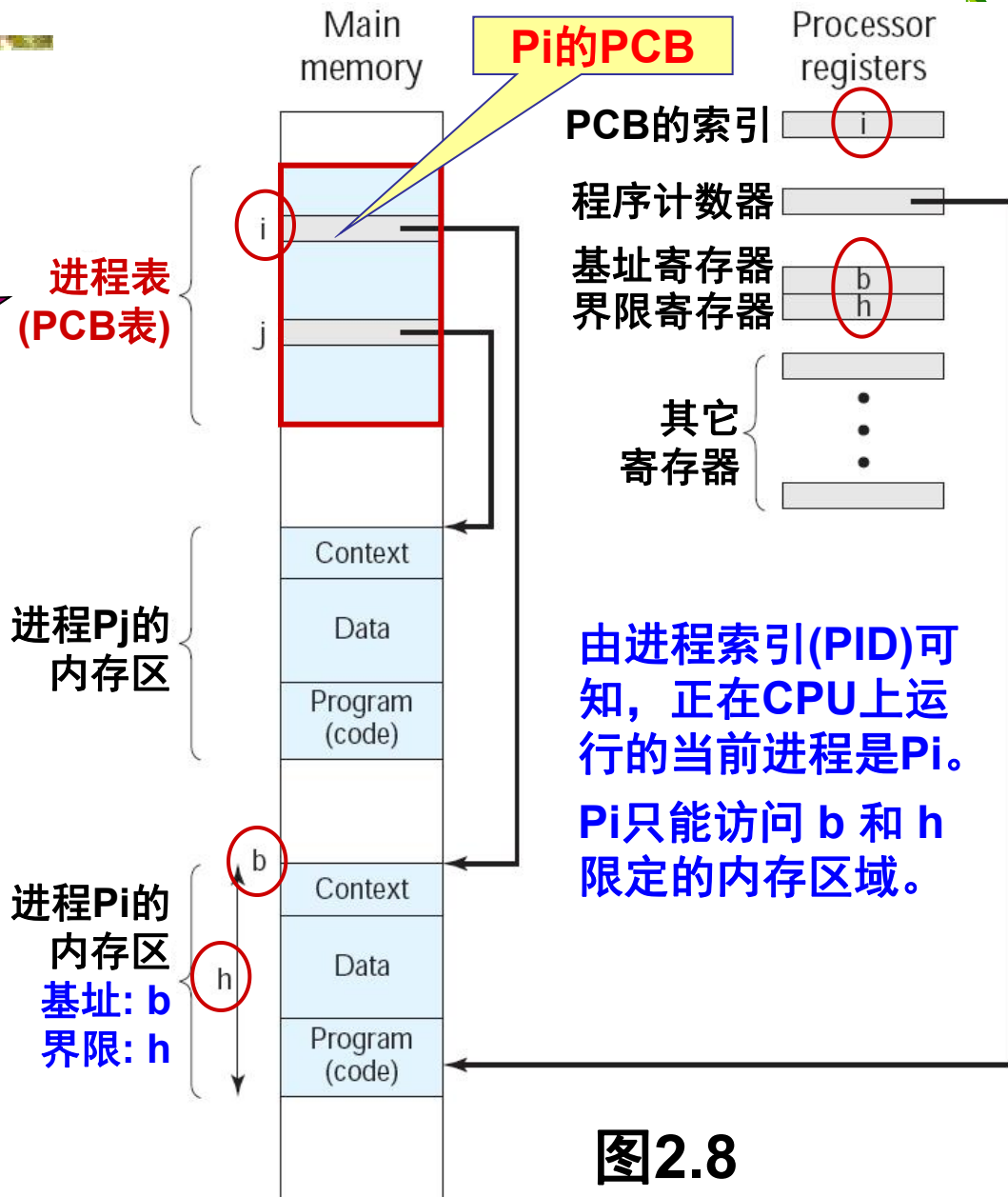


图2.8

3.3.2 进程控制结构—PCB的属性

● 现代操作系统中，PCB 的典型字段：

■ PCB的索引（**PID**）唯一地标识该进程。

中断时要
保存的上
下文现场

Process management

Registers
Program counter
Program status word
Stack pointer
Process state 进程状态

调度用

Priority
Scheduling parameters
Process ID 进程标识符
Parent process ID 父PID

通信

Process group
Signals

计帐

Time when process started
CPU time used
Children's CPU time
Time of next alarm

Memory management

Pointer to text segment
Pointer to data segment
Pointer to stack segment

正文(代码)段、
数据段、栈段
的内存指针

File management

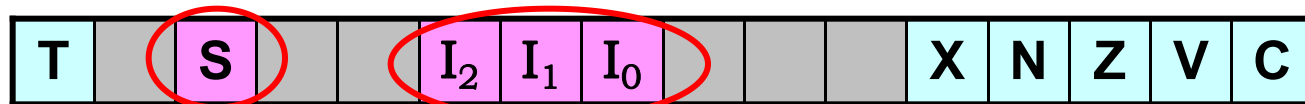
Root directory
Working directory
File descriptors
User ID
Group ID

打开的文件

3.3.2 进程控制结构—PSW

- **程序状态字** (Program Status Word, PSW) 寄存器：指明 CPU 当前特权级别、中断屏蔽码(中断优先级)等。

M68000(16位)
PSW结构



CPU特权状态(模式位)

中断屏蔽码

Pentium (32位) PSW

31 . . . 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0



Itanium (64位) 的PSW (称为PSR) 中用2b的CPL表示CPU特权状态 (CPU模式)。

3.4 进程控制



3.4.1 执行模式—CPU特权级别



- CPU有两种执行模式（CPU状态）：

由PSW中的
模式位指示

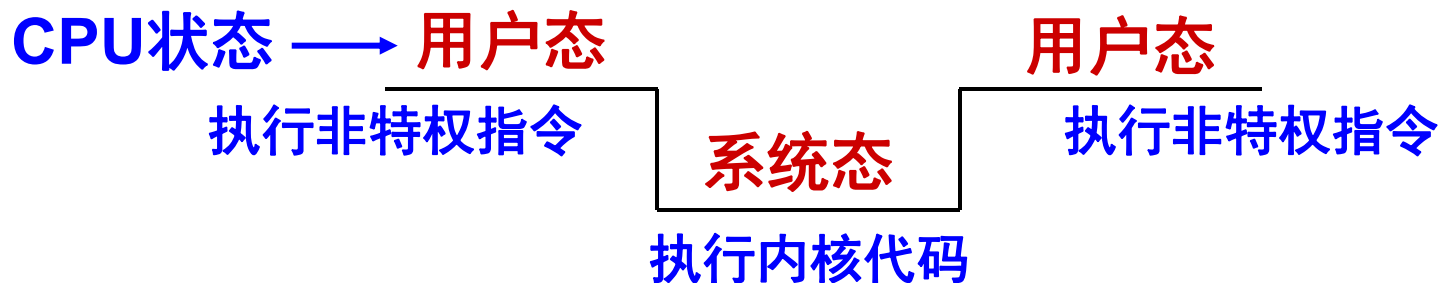
- **用户态**：只能执行非特权指令。
- **系统态**：也称 核心态、内核态、特权态、控制态、管态。可以执行所有指令（特权指令和非特权指令），使用所有资源以及改变CPU状态。

- 两类指令：

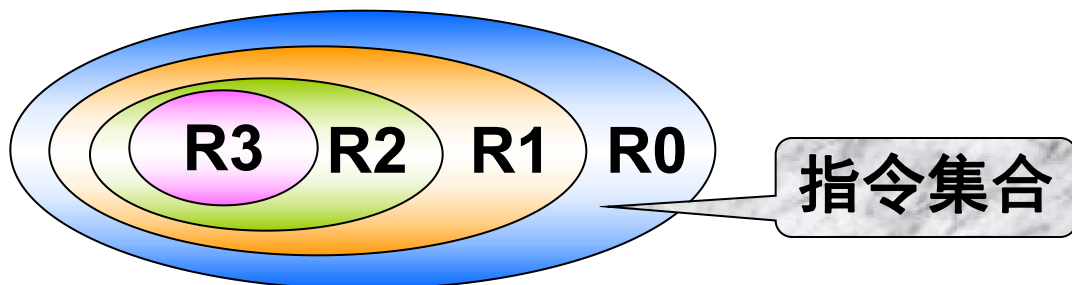
- **特权指令**：在系统态下执行的指令（OS内核使用）。
- **非特权指令**：用户态下执行（用户程序）。

3.4.1 执行模式—CPU特权级别

- 当CPU要执行特权指令时，会引起“陷入trap”，即CPU由用户态切换到系统态（称为模式切换），然后去执行OS内核中的一段（特权指令）代码。



- Intel x86 CPU 支持4种特权级别(R0~R3)。R0特权级最高，R3特权级最低。



- 但基于x86CPU的操作系统，如多数UNIX、Linux、Windows系列，只用了R0和R3两个特权级别。

3.4.1 执行模式—CPU特权级别

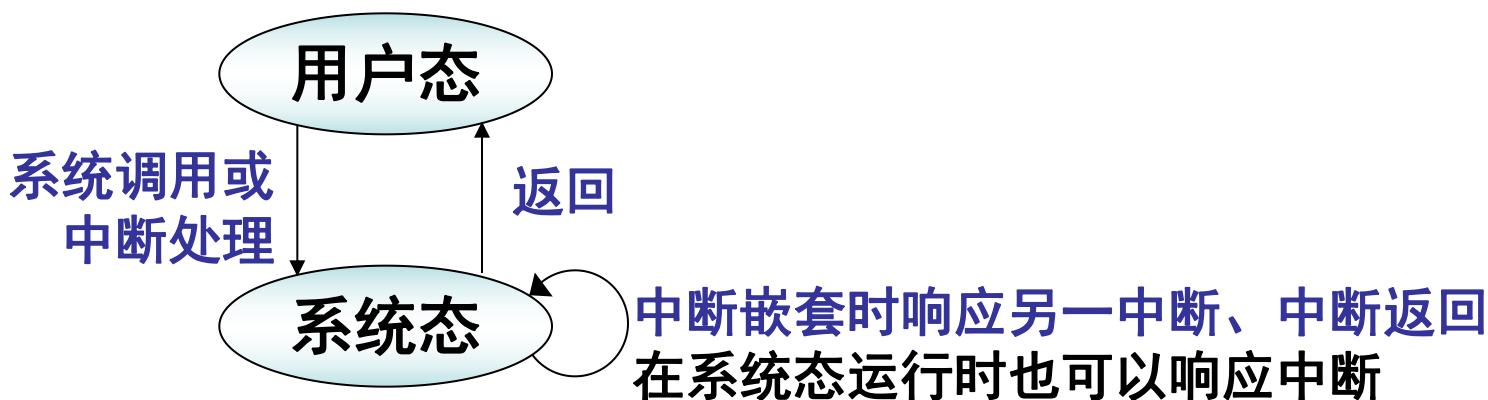


● 何时CPU从用户态到系统态？

- 执行系统调用（即内核代码，由OS提供服务）时。
- 发生中断或异常时，执行中断处理程序。

● 如何从系统态到用户态？

- 系统调用或中断处理完毕后，执行 IRET（中断返回）指令，恢复原进程的PSW，回到用户态。



Question:



1、下列选项中，不可能在用户态下发生的是_____。

- A. 系统调用 B. 外部中断 C. ☒ 进程切换 D. 缺页

2、可以在用户态下执行的指令是_____。

- A. 关中断 B. 输入/输出 C. 系统调用 D. ☒ 从内存取数

输入输出、系统调用和中断处理都属于OS内核

3、发生模式切换可以不改变当前运行态进程的状态？ ☒

例：进程P0正在CPU上运行以下代码，未超时前一直处于运行态：

.....

x++;

n=fork();

y--;

.....

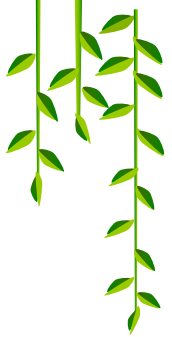
执行x++时，CPU模式是用户态；

执行fork()时，CPU模式切换为系统态；

执行完fork后，CPU切换回用户态，执行y--。



3.4.2 进程创建



● 创建新进程的步骤：

- 给新进程分配一个新PCB和唯一的PID；
- 给进程分配内存空间（代码、数据、栈）；
- 初始化PCB：各属性取默认值或所请求的值。
在UNIX中，子进程PCB基本复制自父进程PCB。
- 将PCB放置到就绪队列或就绪挂起队列。
- 创建其它数据结构：如记账、日志等。

3.4.3 进程切换—调度一个新进程



● 何时进行进程切换？

- **中断**发生时：如时钟中断。当前进程时间片结束，让出CPU，调度新进程，因此发生新旧进程的切换。
- **陷阱**发生时：当前进程的指令产生错误或异常，非致命时重试或报告，致命时将结束本进程，调度新进程。
- 当前进程执行**I/O型系统调用**时，会被阻塞，让出CPU。

● 进程切换的步骤：

- 保存当前进程的CPU上下文（寄存器现场）；更新其PCB；将其PCB转移到相应的就绪或阻塞状态队列；
- 选择（调度）另外一个就绪进程，准备执行；
- 更新该进程PCB，从就绪队列移出；更新内存管理的数据结构（如设置页表指针、基址/界限寄存器）；将该进程的上下文恢复到CPU寄存器。

Question:



1、进程创建时，不需要为该进程做的是_____。

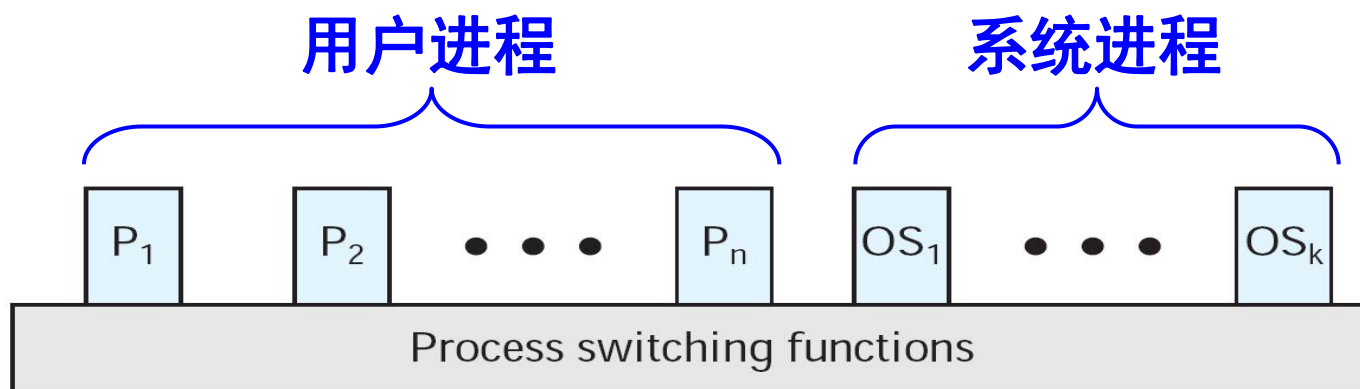
- A. 填写进程表（PCB）项
- B. 分配内存
- C. 将它插入就绪队列
- D. ☒ 分配CPU

2、_____一定会引起进程切换。

- A. 一个进程创建后，进入就绪状态
- B. ☒ 一个进程从运行状态变为就绪状态
- C. 一个进程从阻塞状态变为就绪状态
- D. 以上答案都不对


3.5 操作系统的执行—OS本身是进程吗?

- **无进程**：早期操作系统用。
- **在用户进程中执行OS**：小型操作系统用。
- **基于进程的OS**：如UNIX。
在模块化结构的OS中，主要的内核函数作为一组独立的**系统进程**（特权状态下执行）。



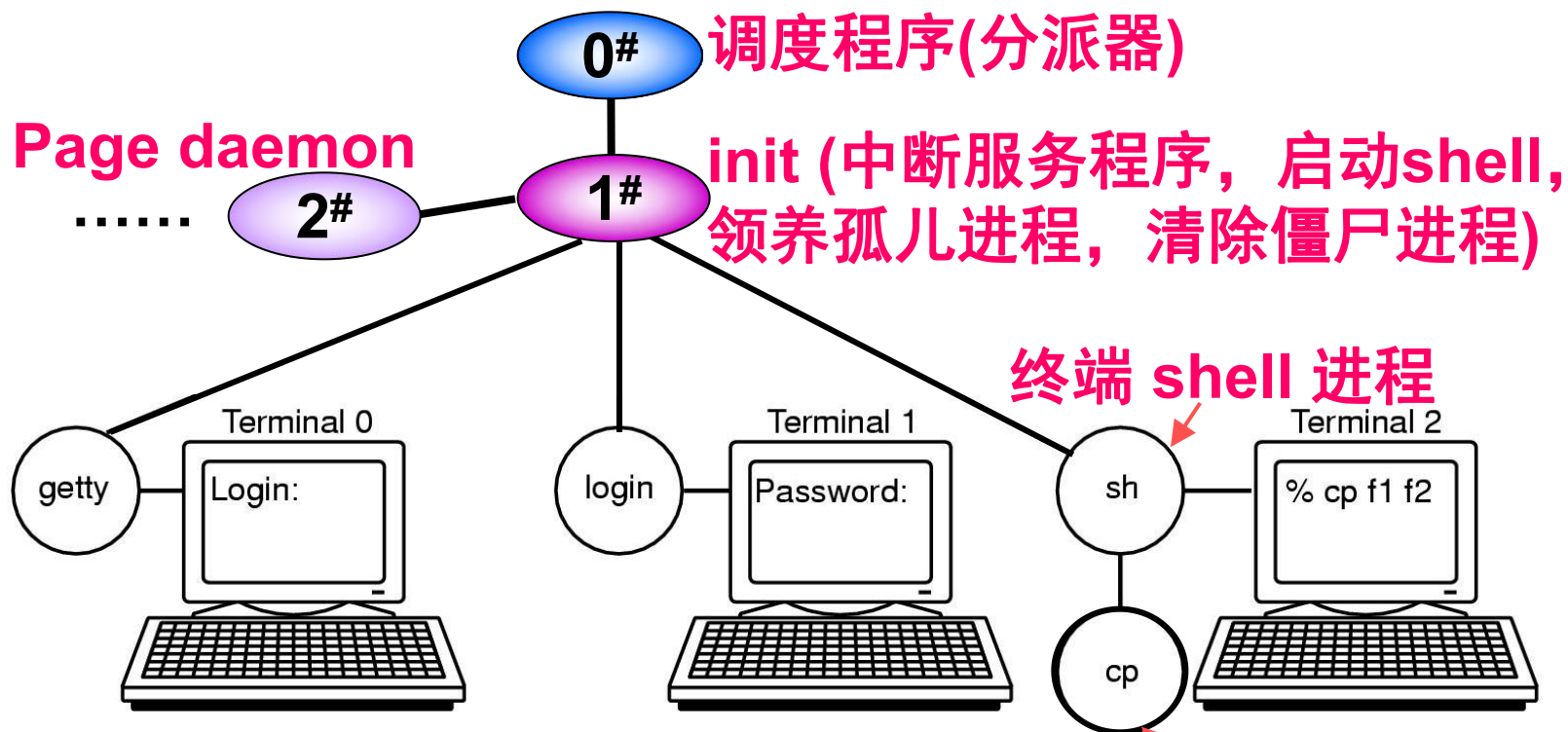
3.6 UNIX SVR4进程管理



- SVR4: System V Release 4。UNIX内核标准。
- UNIX SV将进程分为两类：
 - **系统进程**：在内核态下运行，执行OS代码；
 - **用户进程**：在用户态下执行用户程序中的非特权指令，在内核态下执行内核代码（如当要执行系统调用时）。
- 3.6.1 进程状态（略）
 - UNIX系统V的进程有9种状态。
- 3.6.2 进程描述（略）

3.6.3 进程控制—进程树

- 父进程创建子进程，形成进程树。



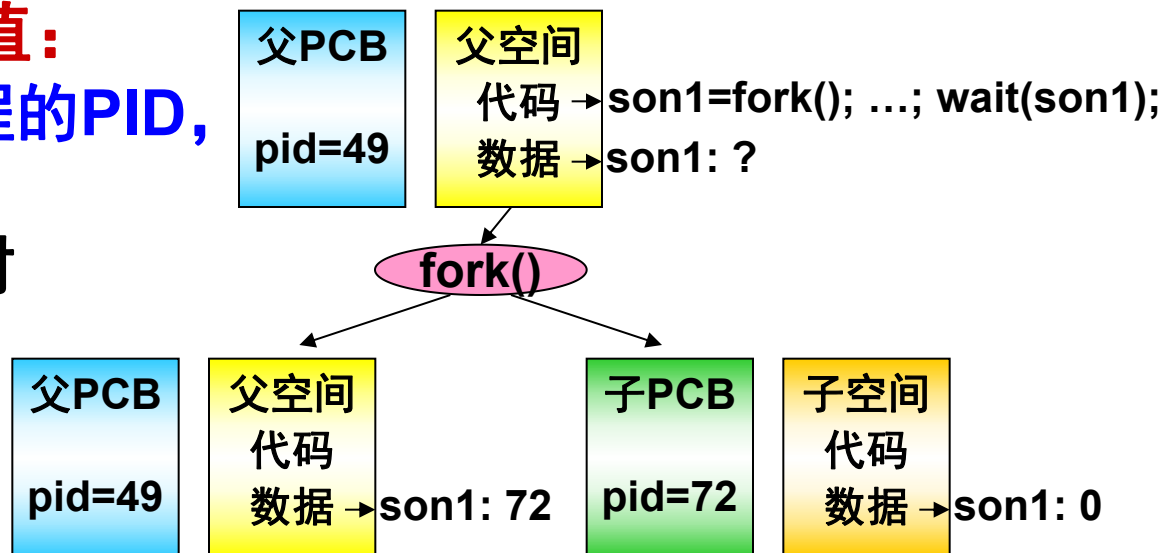
孤儿进程：其父进程已结束的进程；

僵尸进程：已exit()，其PCB仍存在，但其父未用wait()等其结束，也未忽略SIGCHLD信号，则该进程处于Zombie状态。

3.6.3 进程控制—fork()

- UNIX中，父进程通过系统调用**fork()**创建子进程：
 - 父、子进程并发执行，且父子共享父进程的所有资源。
 - 子进程复制父进程的地址空间，甚至有相同的正文段和程序计数器PC值。
 - 写时复制**Copy_on_Write**技术可减少不必要的复制：
fork时父子共用父空间，当一方试图修改时才复制。

- **fork()有两个返回值：**
向父进程返回子进程的PID，
向子进程返回0。
因此父应在创建子时
用变量保存子PID。



Question:



下面这个UNIX平台下运行的C程序结果是什么？

```
#include <stdio.h>
#include <sys/types.h> /* pid_t类型,实际上是int类型*/
#include <unistd.h> /* 调用fork */
main()
{ pid_t pid;
  printf("ONE\n");
  pid=fork();
  printf("TWO\n");
}
```

子进程复制父进程的地址空间，甚至有相同的正文段和程序计数器PC值

父 `printf("ONE\n");` ← 程序计数器PC
`pid=fork();`
`printf("TWO\n");`

fork

之前

父 `printf("ONE\n");`
`pid=fork();`
`printf("TWO\n");` ← PC

子 `printf("ONE\n");`
`pid=fork();`
`printf("TWO\n");` ← PC

之后

Question:



课后习题3.12：有如下C程序，假设fork函数运行成功，程序可能的输出是什么？

```
main()
{ int pid;
  pid=fork();
  printf("%d \n", pid);
}
```

fork()有两个返回值：
向父进程返回子进程的PID，
向子进程返回0。

设子进程的PID为xxxx，
则由于父子并发运行，可能的输出为：

xxxx
0

或

0
xxxx

作业:

● 复习题: 3, 14

● 习题: 2

3.2 假设在时刻 5 仅使用了系统资源中的处理器和内存。考虑如下事件:

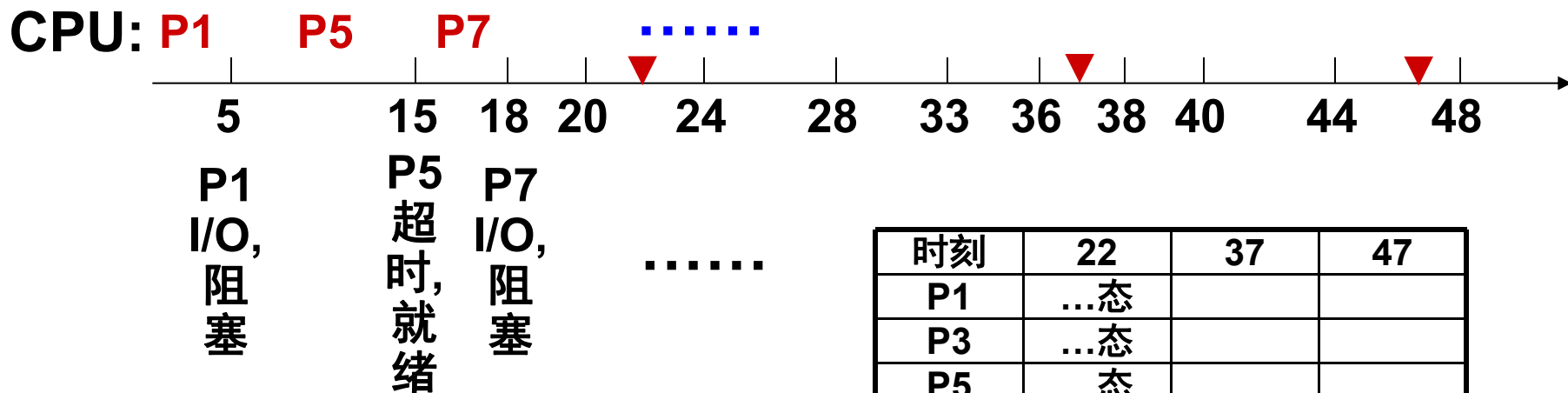
时刻 5: P1 执行一个命令读磁盘单元 3。

时刻 15: P5 的时间片结束。

时刻 18: P7 执行一个命令写磁盘单元 3。

时刻 20: P3 执行一个命令读磁盘单元 2。

提示: 某进程在执行读/写某磁盘单元的命令之前, 处于“运行态”。读/写磁盘(无论哪个磁盘单元)都算作I/O, 该进程将被阻塞。



第8版更正:

时刻36: 应为“P1读磁盘单元3完成, 产生中断”。

时刻	22	37	47
P1	...态		
P3	...态		
P5	...态		
P7	...态		
P8	...态		