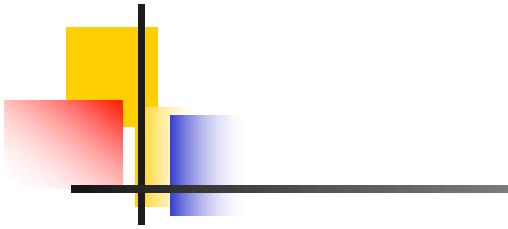


第2章 进程与线程

2.4 进程的定义与描述

2.4.1 程序的顺序执行

- 一个程序通常由若干个程序段所组成，它们必须按照某种先后次序来执行，仅当前一个操作执行完后才能执行后继操作，这类计算过程就是程序的顺序执行过程。
- 例如：先输入→再计算→最后打印输出，即：I→C →P。

程序顺序执行时的特征

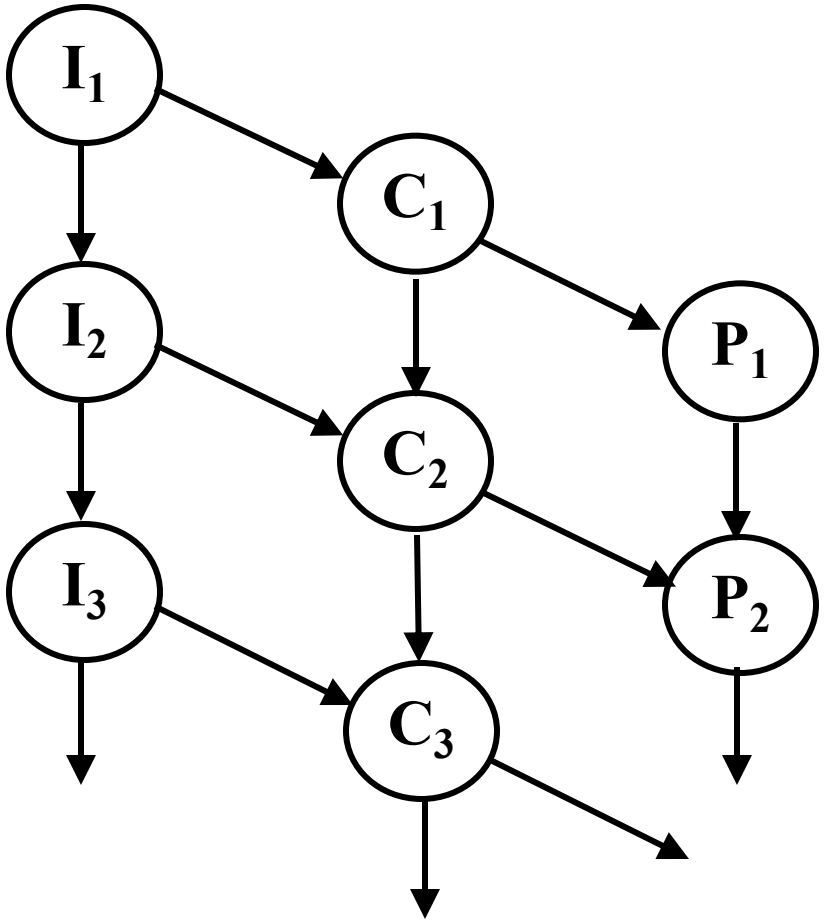
1. 顺序性：处理机的操作严格按照程序所规定的顺序执行，即每一个操作必须在下一个操作开始之前结束。
2. 封闭性：程序一旦开始运行，其执行结果不受外界因素影响。
3. 可再现性：只要程序执行时的初始条件和执行环境相同，当程序重复执行时，都将获得相同的结果。

2.4.2程序的并发执行及特点

- 程序的并发执行是指若干个程序（或程序段）同时在系统中运行，这些程序（或程序段）的执行在时间上是重叠的，一个程序（或程序段）的执行尚未结束，另一个程序（或程序段）的执行已经开始。

程序并发执行例子

- 进程1、2、3并发执行。对每个进程而言，其输入、计算和输出这三个操作必须顺序执行。它们之间存在如下先后关系：
- **I_1先于C_1和I_2， C_1先于P_1和C_2， P_1先于P_2**
- **I_2和C_1， I_3、 C_2和P_1可以并发。**



程序并发执行时的特征

1. 间断性：并发程序具有“执行---暂停---执行”这种间断性的活动规律。
2. 失去封闭性：多个程序共享系统中的资源，这些资源的状态将由多个程序来改变，致使程序之间相互影响。
3. 不可再现性：在初始条件相同的情况下，程序的执行结果依赖于执行的次序。

与时间有关的错误例

- 程序并发执行时可能出现与时间有关的错误。
- 例
  - 进程1：r1=x；　　进程2：r2=x；
  - 　　r1++；　　r2++；
  - 　　x=r1；　　x=r2；
- 设在两进程运行之前，x的值为0。则两进程运行结束后，x值可为多少？

2.4.3 程序并发执行的条件

如何解决时间有关的错误？

- 几个定义：
  - 读集：语句执行期间要引用的变量集合，记为
    $R(S_i)=\{a_1, \ldots, a_m\}$
  - 写集：语句执行期间要改变的变量集合，记为
    $W(S_i)=\{b_1, \ldots, b_n\}$

Bernstein条件

- Bernstein条件能保证两个程序段并发执行而不会产生与时间有关的错误：
  ① $R(S_i) \cap W(S_j) = \{\}$
  ② $R(S_j) \cap W(S_i) = \{\}$
  ③ $W(S_j) \cap W(S_i) = \{\}$
  - 条件1与条件2，保证了两个程序段之间不会引起读数据的干扰，两次读取数据之间，存储器数据不会发生改变
  - 条件3保证了两个程序段之间不会造成写数据的干扰，写数据结果不会丢失
- 满足以三个条件即可满足并发执行的程序可满足封闭性和可再现性

例：简单算一算

- 考虑下面是条语句：

  S1：a=x+y　　S2：b=z+1

  S3：c=a-b　　S4：d=c+1

- R(S1)={x,y}　R(S2)={z}　R(S3)={a,b}

  W(S1)={a}　W(S2)={b}　W(S3)={c}

- 因R(S1)∩ W(S2)∪R(S2)∩ W(S1)∪W(S1)∩W(S2)={ }，故S1和S2可以并发执行 。

- 因R(S2)∩ W(S3)∪R(S3)∩ W(S2)∪W(S3)∩W(S2)={b}，故S2和S3不能并发执行 。同理，S1和S3之间也不可以并发执行。

2.4.4 进程的引入

- 为了描述并发执行程序的动态特性，人们引入了一个新的概念——进程。
- 进程是一个可并发执行的具有独立功能的，是程序关于某个数据集合的一次执行过程，也是操作系统进行资源分配和保护的基本单位 。
- 进程是一个既能用来共享资源，又能描述程序并发执行过程的一个基本单位。

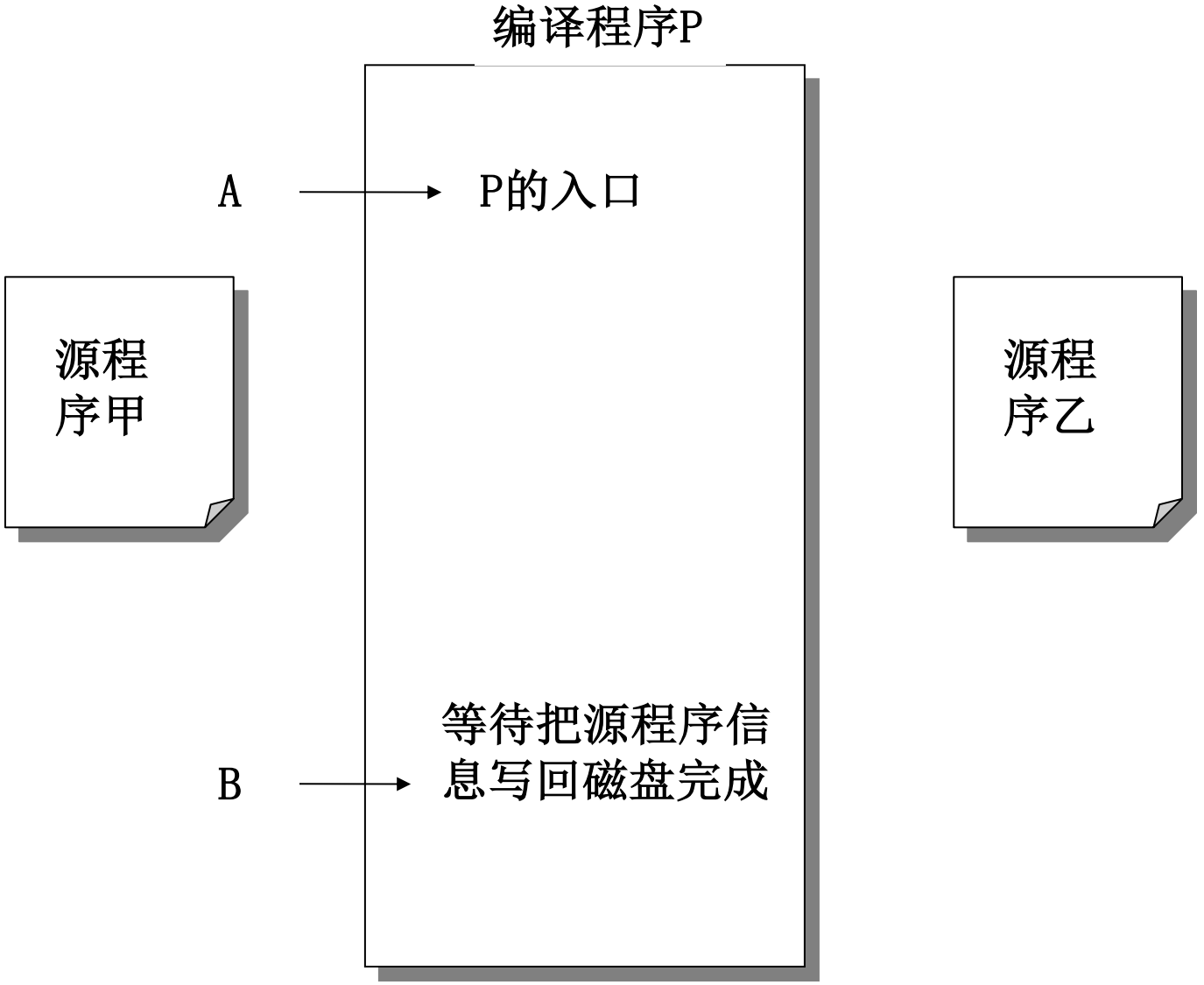
进程的引入

- 操作系统引入进程的概念
  - 从理论角度看，是对正在运行的程序过程的抽象；
  - 从实现角度看，是一种数据结构，目的在于清晰地刻划动态系统的内在规律，有效管理和调度进入计算机系统主存储器运行的程序。

进程的引入

- 操作系统为什么要引入进程概念?
  - 原因1-刻画系统的动态性，发挥系统的并发性，提高资源利用率。
  - 原因2-它能解决系统的“共享性”，正确描述程序的执行状态。
    - 可再入：可被多程序调用，执行过程中不被修改

可再入程序的一个例子



2.4.4 进程的定义

- 进程最初由20世纪60年代提出，由MULTICS和IBM TSS/360首先引入
- 进程有多种定义，下面列举一些有代表性的定义：
  ① 进程是程序在处理器上的一次执行过程。
  ② 进程是可以和别的计算并行执行的计算。
  ③ 进程是程序在一个数据集合上运行的过程，是系统进行资源分配和调度的一个独立单位。
  ④ 进程是一个具有一定功能的程序关于某个数据集合的一次运行活动。

2.4.5 进程的特征

① **动态性**：进程是程序的一次执行过程。进程是有“生命”的，动态性还表现为它因创建而产生，因调度而执行，因无资源而暂停，因撤消而消亡。而程序是静态实体。

② **并发性**：多个进程实体同时存在于内存中，能在一段时间内同时运行。

③ **独立性**：在传统OS中，进程是独立运行的基本单位，也是系统分配资源和调度的基本单位。凡是未建立进程的程序，很难作为独立单位参与调度和运行。

④ **异步性**：也叫制约性，进程以各自独立的不可预知的速度向前推进。

⑤ **结构性**：进程实体由程序段、数据段及进程控制块组成，又称为进程映像。

⑥ **共享性**：同一程序同时运行于不同的数据集合上时，将构成不同的进程。进程之间也可以共享公用变量，通过公用变量交换信号。

进程与程序的关系

① 进程是动态概念，程序是静态概念；进程是程序在处理机上的一次执行过程，而程序是指令的集合。

② 进程是暂时的，程序是永久的。进程是一个状态变化的过程；程序可以长久保存。

③ 进程与程序的组成不同。进程的组成包括代码段、数据段和进程控制块。

④ 进程与程序是密切相关的。一个程序可以对应多个进程；一个进程可以包括多个程序，进程和程序不是一一对应的。

⑤ 进程可以创建新进程，而程序不能形成新程序。

2.4.6 进程的描述

- 进程映象
  - 某时刻进程的内容及其状态的集合
- 组成
  - 进程控制块：每个进程捆绑一个控制块，用于存储进程的标识信息、现场信息、控制信息
  - 进程程序块：被执行的程序
  - 进程核心栈：每个进程捆绑一个核心栈，用于保存进程在核心态工作时的现场保护
  - 进程数据块：进程的私有地址空间，存放各种私有数据
- 控制块+程序块+核心栈+数据块

2.4.6 进程的描述

- 进程的上下文
  - 操作系统中把**进程物理实体和支持进程运行的环境**合称为进程上下文。
  - 当系统调度新进程占有处理器时，新老进程随之发生**上下文切换**。进程的运行被认为是在上下文中执行。

2.4.6 进程的描述

- 进程上下文组成
  - 用户级上下文：
    - 正文(程序)、数据和共享存储区、用户栈组成，占用进程的虚拟地址空间。
  - 寄存器级上下文
    - 程序状态寄存器、指令计数器、栈指针，控制寄存器、通用寄存器等组成
  - 系统级上下文
    - 进程控制块、主存管理信息、核心栈等组成

2.4.6 进程的描述

- 进程控制块
  - PCB是描述和管理进程的数据结构。它是进程实体的一部分
  - 操作系统创建PCB
  - 操作系统通过PCB感知进程的存在，PCB是进程存在的唯一标志
  - 操作系统通过PCB了解进程状态：执行情况、进程让出处理器后所处的状态、断点信息等
  - 操作系统通过PCB来调度、控制和管理进程
- 操作系统会执行如下操作：
  - 创建PCB、修改PCB、访问PCB、回收PCB

2.4.6 进程的描述

- PCB的组成：

1. 进程标识信息
   - ① 进程标识符：惟一标识进程的一个标识符或整数。
   - ② 家族关系：指明本进程与家族的关系，如父子进程标识。
   - ③ 用户标识符：表明进程的所有者
2. 进程现场信息
   - ① 保留进程在运行时存放在处理器中的各种信息
   - ② 包括：通用寄存器、控制寄存器、栈指针寄存器等

2.4.6 进程的描述

3. 进程控制信息

① 进程当前状态：说明进程当前所处状态。

② 进程队列指针：用于记录PCB队列中下一个PCB的地址。

③ 进程优先级：反映进程获得CPU的优先级别。

④ 通信信息：进程与其他进程所发生的信息交换。

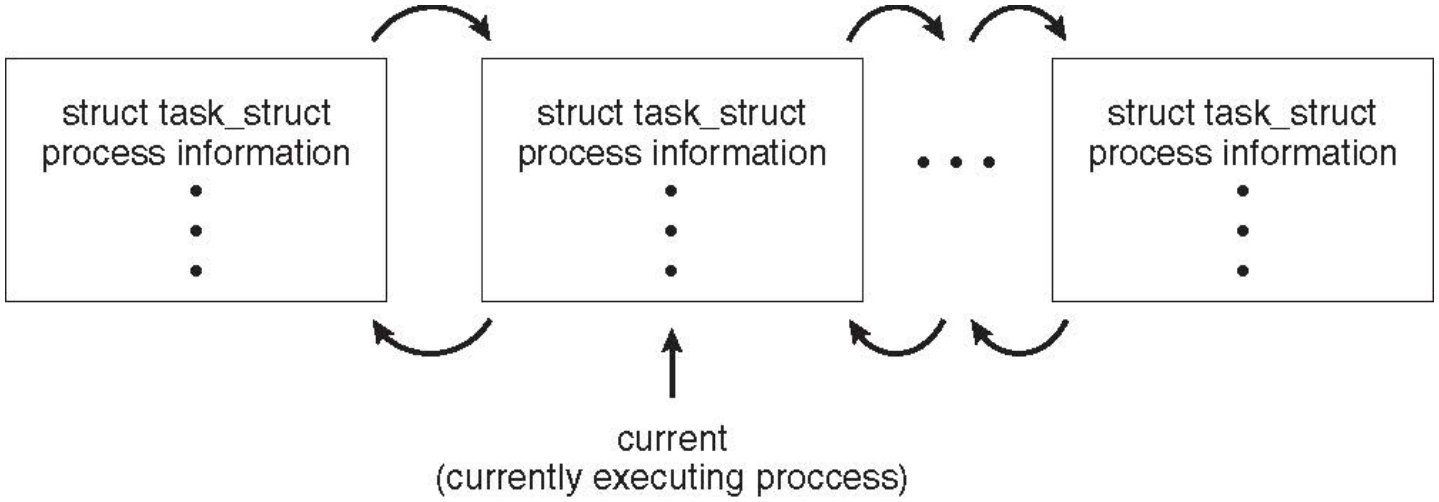
⑤ 程序和数据地址：定位进程的程序和数据在内存或外存中的存放地址。

⑥ 资源清单：列出进程所需资源及当前已分配资源，如打开的文件，使用CPU记录

Linux的进程表示

task_struct:

```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```



讨论

- 是否存在一个恶意代码或者攻击程序，没有自身独立的进程结构，但是却能够达到执行效果，为什么？