操作系统

Xia Tian

Renmin University of China

March 7, 2019

CH2 进程管理

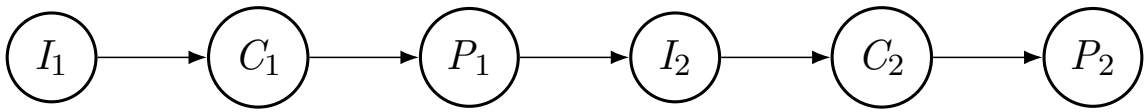
- 2.1 进程的基本概念
- 2.2 进程控制
- 2.3 进程同步
- 2.4 经典进程同步问题
- 2.5 管程机制
- 2.6 进程通信
- 2.7 线程

2.1 进程的基本概念

- 未配置 OS 的系统：程序顺序执行
  - 程序的顺序执行及其特征
- 现代多道程序环境下：程序并发执行
  - 程序的并发执行及其特征

2.1 进程的基本概念

- 2.1.1 程序的顺序执行及特征
  - 1. 程序执行有固定的时序



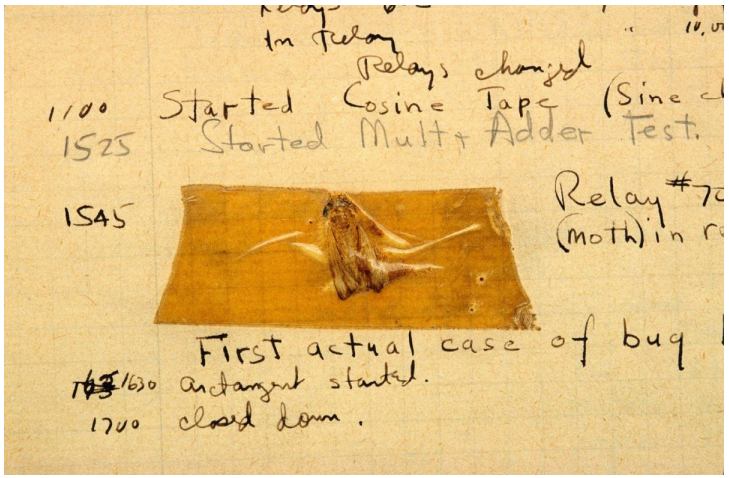
  - 2. 程序顺序执行时的特征
    - 顺序性：操作的前后依赖性
    - 封闭型：独占资源，资源状态只有本程序更改
    - 可再现性：初始环境和条件，结果相同

程序顺序执行的优点

- 符合人的直觉
- 有利于错误调试
  - DEMO
  - BUG vs DEBUG

格蕾丝 · 赫柏 (Grace Murray Hopper)

赫柏是一位为美国海军工作的电脑专家。1945 年的一天，赫柏对 Harvard Mark II 设置好 17000 个继电器进行编程后，技术人员在进行整机运行时，它突然停止了工作。于是他们爬上去找原因，发现这台巨大的计算机内部一组继电器的触点之间有一只飞蛾，这显然是由于飞蛾受光和热的吸引，飞到了触点上，然后被高电压击死。所以在报告中，赫柏用胶条贴上飞蛾，并把“bug”来表示”一个在电脑程序里的错误”。



2.1.2 程序的并发执行

- 特征
  - 间断性:
    - 如打印程序等待计算程序完成之后方可继续
    - 执行—暂停—执行……
  - 失去封闭性
    - 主要由共享资源引起，资源的状态将由多个程序改变;
  - 不可再现性
    - 计算结果与并发程序的执行速度有关

例子 — 课堂讨论

- 有 2 个循环程序 A 和 B，共享一个变量 N（设 N 的初值为 n ）
- 程序 A 每执行一次时，都要做 $N := N + 1$
- 程序 B 每次要执行 $Print(N)$, 然后再做 $N := 0$

- 若程序 A,B 以不同的速度运行，其结果将会是?
- 注意，代码采用了类 Pascal 语言

例子

- N:=N+1 在 print(N) 和 N:=0 之前，则 N 值分别为 n+1, n+1, 0.
- N:=N+1 在 print(N) 和 N:=0 之后，则 N 值分别为 n, 0, 1.
- N:=N+1 在 print(N) 和 N:=0 之间，则 N 值分别为 n, n+1, 0.

Python 代码示例

```
from multiprocessing import Process
def f1(i):
    while i<10:
        i = i+1
    print 'f1 finished  ‘

def f2(i):
    while i>-10:
        i=i-1
    print 'f2 finished!'


if __name__=='__main__':
    Process(target=f1, args=(0,)).start()
    Process(target=f2, args=(0,)).start()
```

多次运行结果

- Run 1:
  - f1 finished
  - f2 finished!
- Run 2:
  - f1 finished
  - f2 finished!
- Run 3:
  - f2 finished!
  - f1 finished

思考

- 在多道程序环境下，程序执行属于并发执行，具有 3 个典型特性（哪 3 个?）
- 结果的不可再现性的问题
- 要保证结果的再现性，就需要对并发执行的程序加以描述和控制，其结果就是引入了“进程”概念
- 进程 = 程序 + 执行
- 在 Multics OS 之前，主要采用 IBM 的“作业（job）”概念，之后，改为进程（Process）

2.1.3 进程的特征和状态

- 进程的定义
  - 程序的一次执行过程
  - 进程是程序实体的执行过程，是系统进行资源分配与调度的独立单位。

进程的特征

- 1. 结构特征
  - 进程：由程序段、数据段及进程控制块三部分构成，总称“进程映像（Unix 中）”。
- 2. 动态性：进程实体的一次执行过程
  - 由“创建”而产生，由“调度”而执行；由得不到资源而阻塞；由撤消而消亡。（而程序是静态的）。
- 3. 并发性: 只有建立了进程，才能并发执行
  - 如同时浏览多个网页
- 4. 独立性: 独立运行，独立获得资源。资源分配与调度的基本单位
  - 浏览器邮箱登陆实例
- 5. 异步性：
  - 各进程以不可预知的速度向前推进
  - 间断性

进程的状态

- 进程执行的间断性，使得进程具有多种不同的状态
- 进程的三种基本状态
  - 就绪
  - 执行
  - 阻塞

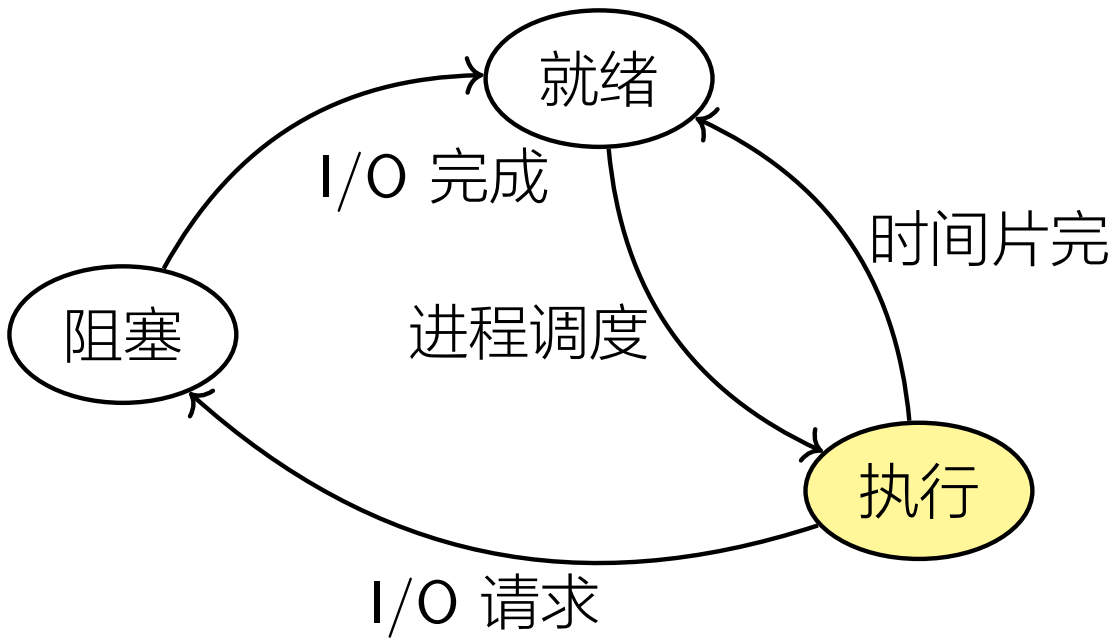


Figure: 进程的三种基本状态及其转换

挂起状态（被换出内存的状态）

- 引入原因
  - 终端用户请求
  - 父进程请求
  - 负荷调节需要
  - 操作系统需要
- 挂起演示
  - vi，CTRL+z; debugging
- 进程状态的转换
  - 活动就绪 $\Rightarrow$ 静止就绪
  - 活动阻塞 $\Rightarrow$ 静止阻塞
  - 静止就绪 $\Rightarrow$ 活动就绪
  - 静止阻塞 $\Rightarrow$ 活动阻塞

具有挂起状态的进程状态图

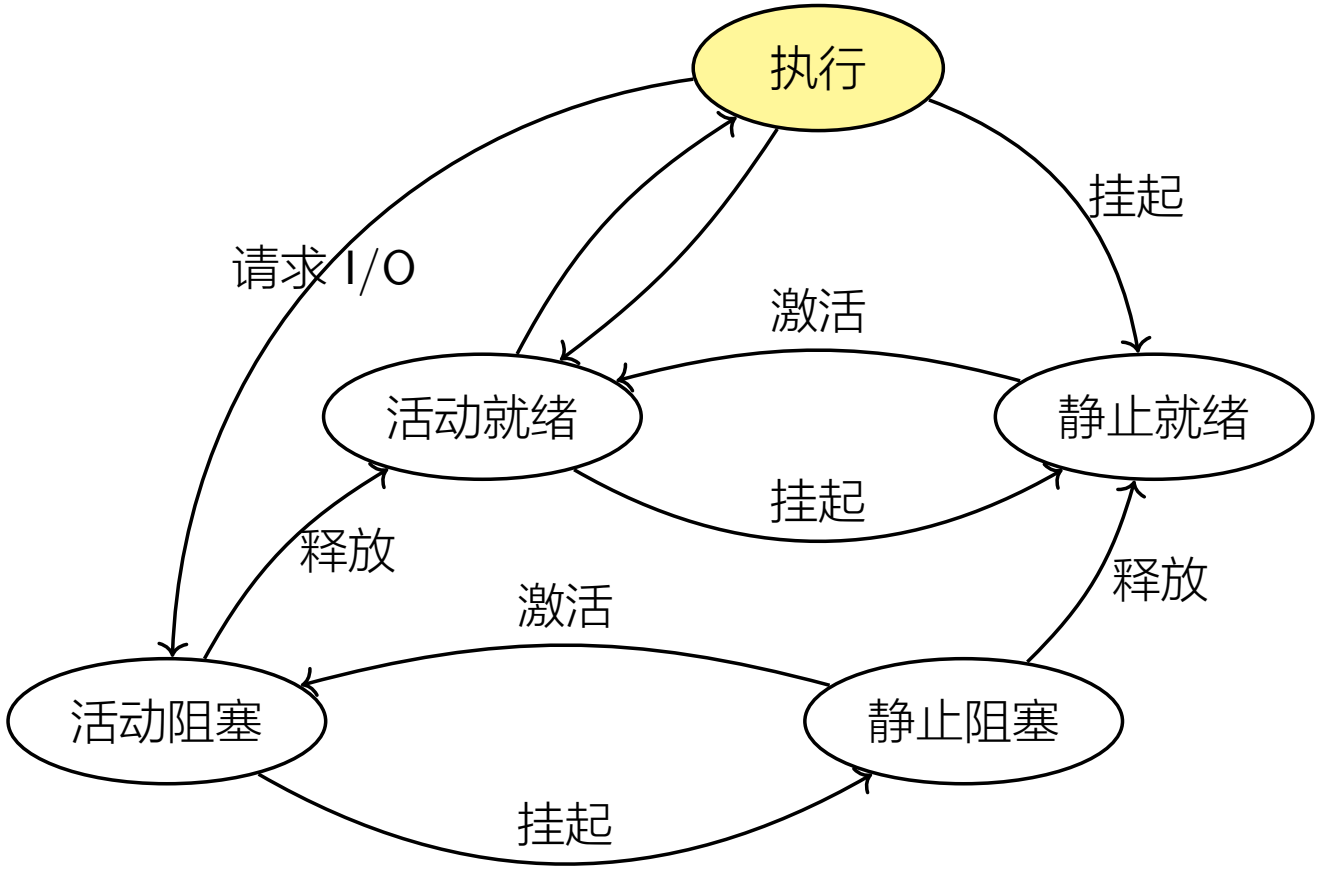


Figure: 具有挂起状态的进程状态及其转换

实验

- 写一个程序描述进程状态迁移过程。
- 要求：
  - 提供导致进程状态变化的调用接口，包括创建、删除、调度、阻塞、时间到、挂起、激活等。
  - 实现进程列表显示的接口。
  - 注：这里设计的进程是一个假设的对象实体，是由程序自己创建和删除，不是系统维护的进程。

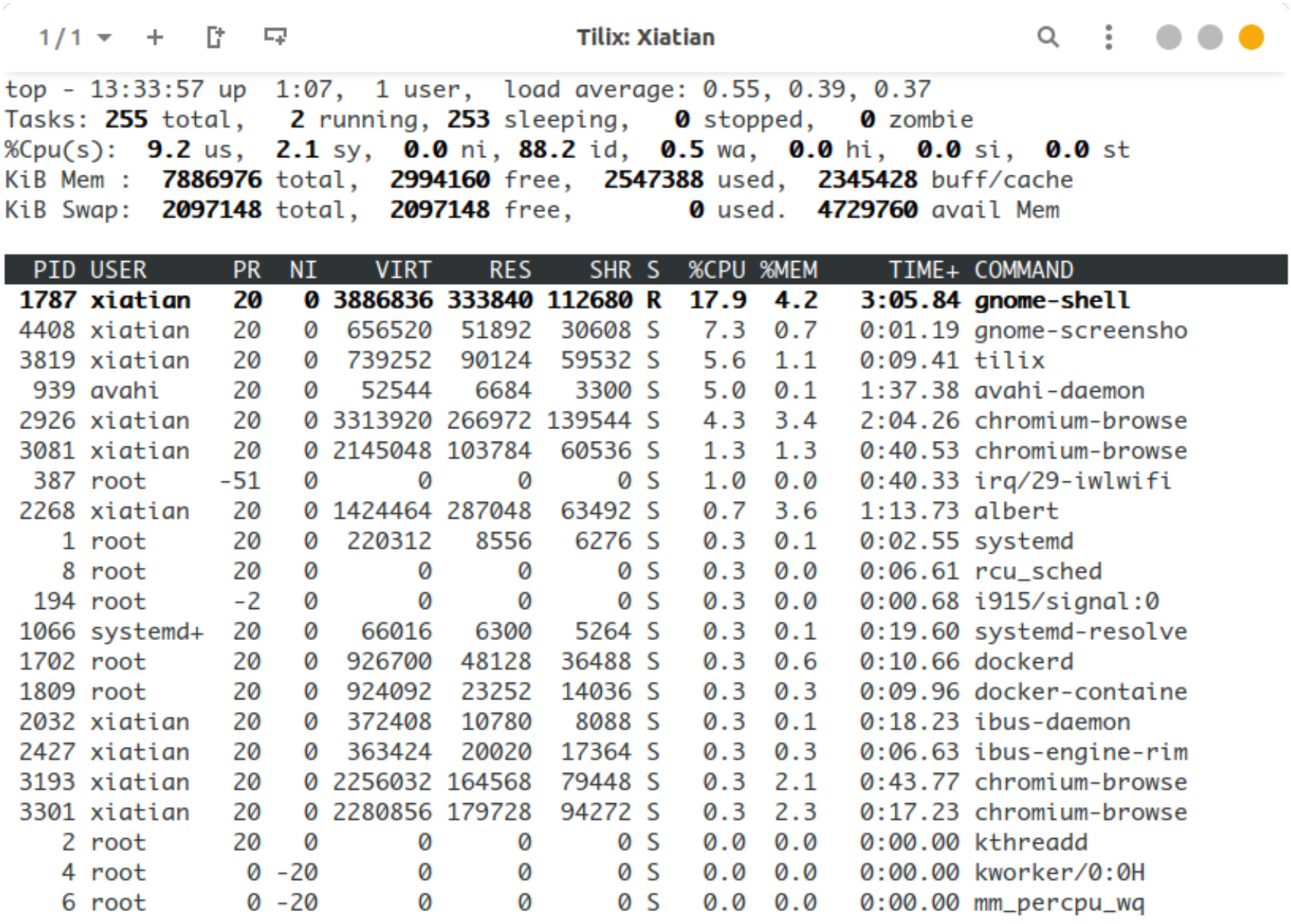
2.1.4 进程控制块

1. 进程控制块的作用
   - 使不能独立运行的程序变为能独立运行的基本单位
   - 是进程存在的唯一标志
   - PCB (Process Control Block) 常驻内存
2. 进程控制块中的信息
   - 标识、处理机状态，进程调度信息，进程控制信息

PCB

- 进程标识符
  - 内部标识符与外部标识符 (下页 top 示例)
- 处理机状态
  - 能在断点恢复运行
  - 通用寄存器、指令计数器、程序状态字 PSW、用户栈指针
- 进程调度信息
  - 进程状态、优先级、与调度有关的其他信息（如等待时间）、事件（阻塞事件）
- 进程控制信息
  - 程序和数据的地址
  - 进程同步和通信机制：消息队列指针、信号量
  - 资源清单
  - 在 PCB 队列中的链接指针

Linux top command



PCB 的组织：链接方式

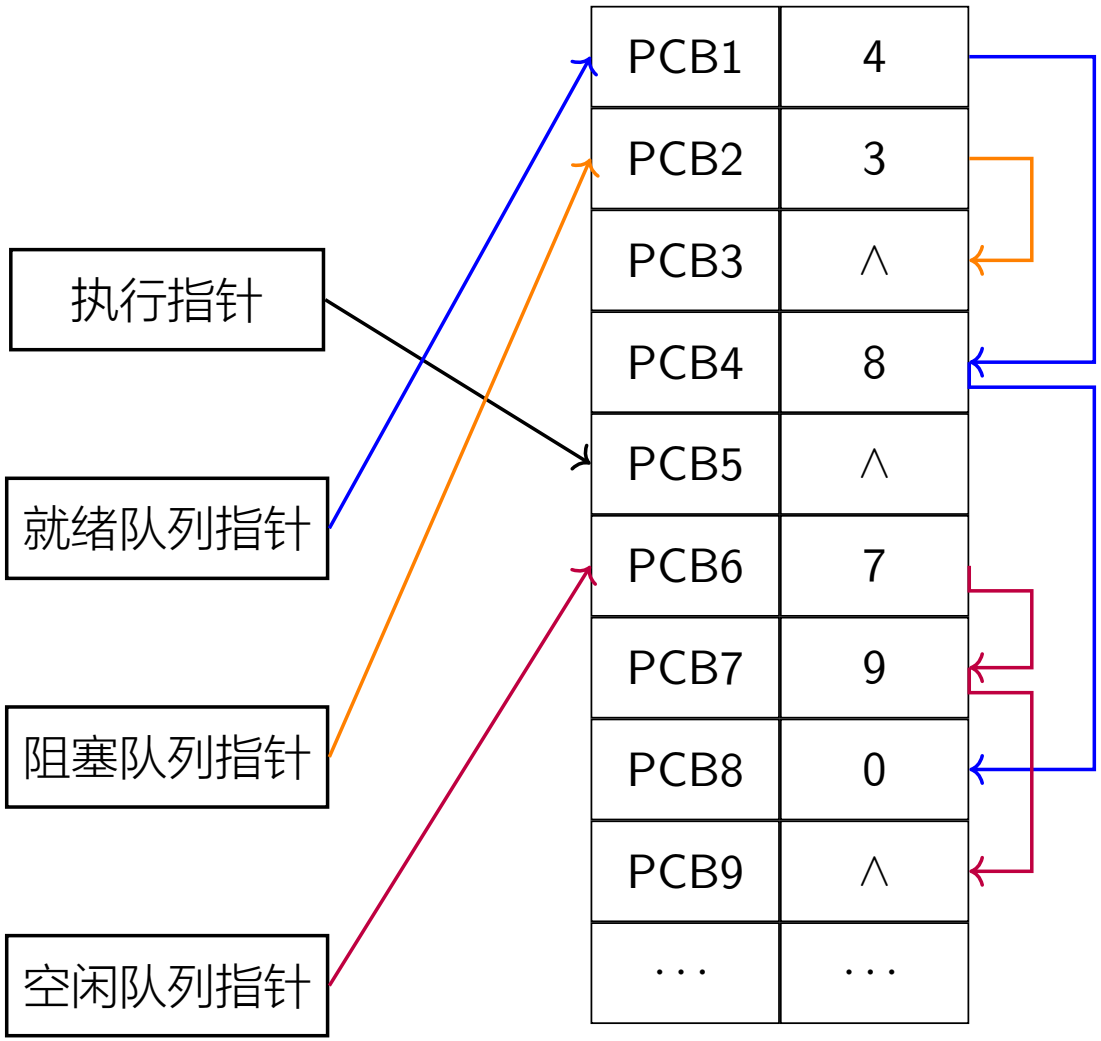


Figure: PCB 链接组织方式

PCB 的组织: 索引方式

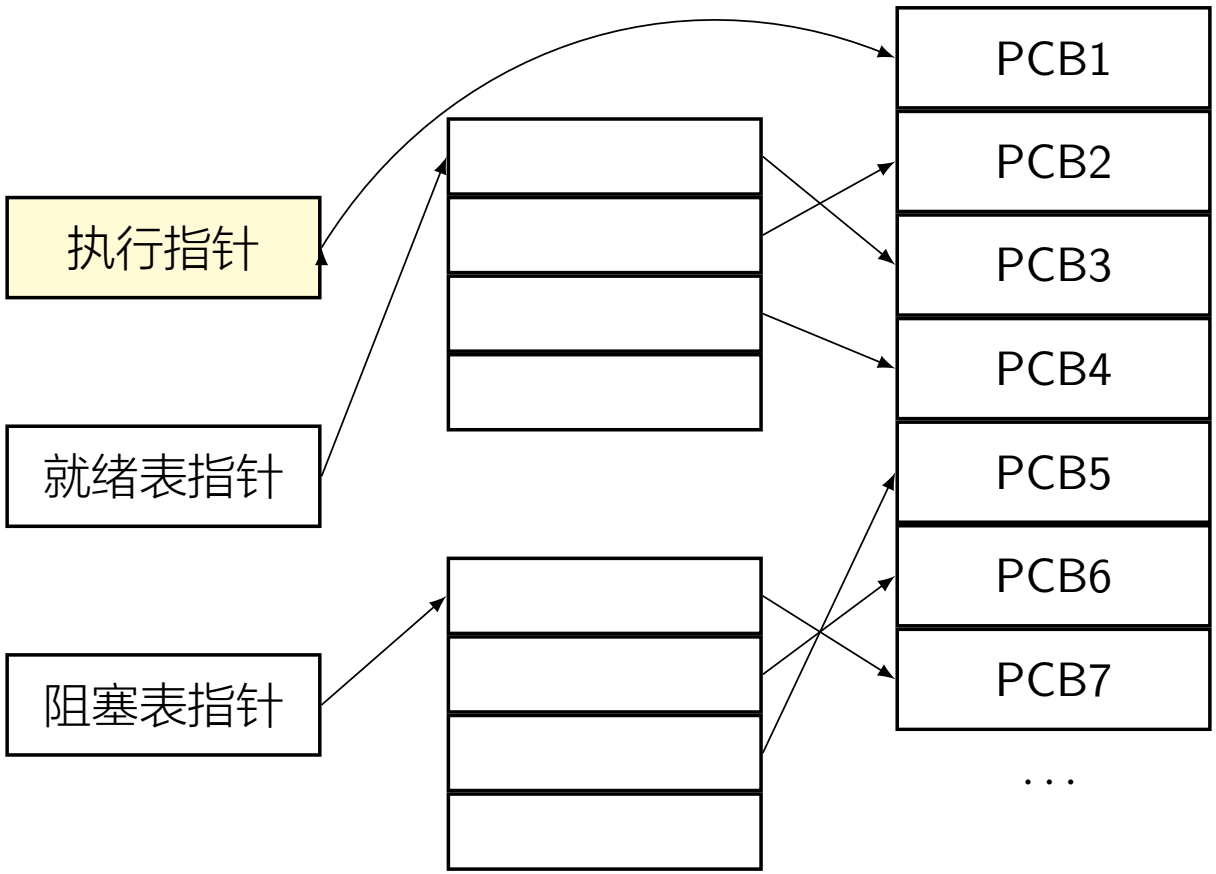


Figure: PCB 索引组织方式

补充

- 指针和链表的概念
  - E.g. 从链表中移除我的下一个：
    - 我的下一个是（变成）我的下一个的下一个
  - E.g. 从链表中移除我本身
    - 我的上一个的下一个是我的下一个
- PCB 和进程的代码数据放在一起吗?
  - 系统态和用户态
  - 系统空间和用户空间
- 系统调用和普通调用的区别?
  - 系统调用会引起从用户态进入核心态

Review last lesson

- 什么是 PCB，操作系统采用哪些方式对 PCB 进行组织和管理的?

2.2 进程控制

- 2.2.1 进程的创建
- 2.2.2 进程的终止
- 2.2.3 进程的阻塞与唤醒
- 2.2.4 进程的挂起与激活

2.2.1 进程的创建

- 进程图：
- 引起创建进程的事件：
- 进程的创建

进程图

- 描述了进程的家族关系
- 子进程可继承父的资源，撤消时应归还给父进程，父的撤消会撤消全部子进程。

引起创建进程的事件

- 1. 用户登录:
  - 为终端用户建立一进程
- 2. 作业调度:
  - 为被调度的作业建立进程
- 3. 提供服务:
  - 如要打印时建立打印进程
- 4. 应用请求:
  - 由应用程序建立多个进程

进程的创建

- (create 原语)
  - 1. 申请空白 PCB（一个系统的 PCB 是有限的）
  - 2. 为新进程分配资源（不同于一般的分配，PCB-LIST 在一个特殊区域）
  - 3. 初始化 PCB
  - 4. 将新进程插入就绪队列。

2.2.2 进程的终止

- 引起进程终止的事件
- 进程的终止过程

引起进程终止的事件

- 1. 正常结束：如 Halt、logoff
- 2. 异常结束：如 Protect error、overtime 等
- 3. 外界干预：
  - a. 系统员 kill 进程；（ Linux 演示 ）
  - b. 父进程终止；(impressive 打开 pdf 文件演示)
  - c. 父进程请求。

进程的终止过程

- 1. 检查进程状态；
- 2. 执行态 $\rightarrow$ 中止，且置调度标志为真。
- 3. 有无子孙需终止。
- 4. 归还资源给其父进程或系统。
- 5. 从 PCB 队列中移出 PCB.

2.2.3 进程的阻塞与唤醒

- Content:
  - 引起进程阻塞和唤醒的事件
  - 进程阻塞过程
  - 进程唤醒过程

引起进程阻塞和唤醒的事件

- 1. 请求系统服务而得不到满足时，如向系统请求打印。
- 2. 启动某种操作而需同步时：如该操作和请求该操作的进程需同步运行（即非异步操作）。
- 3. 新数据尚未到达：如进程 A 写，进程 B 读，则 A 未写完 B 不能读。
- 4. 无新工作可做。

阻塞过程

- 是进程自身的一种主动行为
  - a. 调 block 原语
  - b. 停止执行，修改 PCB 入阻塞队列（一个或多个），并转调度。

唤醒过程

- 其它相关进程完成。
  - a.wakeup 原语
  - b. 修改 PCB，入就绪队列
  - 可见，有 block 原语，在其它进程中就应有 wakeup 原语。

2.2.4 进程的挂起与激活

- 进程的挂起过程
  - 由进程自己或其父进程调 suspend 原语完成，将该进程 PCB 移到指定区域，注意状态的改变，有可能要重新调度。
- 进程的激活过程。
  - active 原语（如在外存，调入内存，改变状态，根据情况看是否调度，如抢先或非抢先）。

- 阻塞、唤醒一般由 OS 实现，而挂起与激活可由用户干预。

2.3 进程同步

- 并发提高了资源利用率和系统吞吐量，但也会给系统造成混乱。
- 同步：
  - 并发进程在执行次序上的协调，以达到有效的资源共享和相互合作，使程序执行有可再现性。

2.3.1 进程同步的基本概念

- 1. 两种形式的制约关系
  - 资源共享关系：（进程间接制约）
    - 如争用一台打印机
    - 需互斥地访问临界资源。
  - 相互合作关系：（进程直接制约）
    - 如 A 的输出作为 B 的输入
    - 需要同步解决
- 2. 临界资源：（一次仅允许一个进程访问的资源）
  - 引起不可再现性是因为临界资源没有互斥访问。

生产者－消费者问题

var n, integer; //变量定义
Type item=⋯;
var buffer:array[0,1,⋯,n−1] of item;
in, out: 0,1, ⋯, n−1;
counter: 0,1,⋯,n;

生产者－消费者问题


```
producer:
repeat
    produce an item in nextp;
    ...
    while counter=n do no−op;
    buffer[in]:=nextp;
    in:=(in+1)mod n;
    counter:=counter+1;
until false;
```

```
consumer:
repeat
    while counter=0 do no−op;
    nextc:=buffer[out];
    out:=(out+1) mod n;
    counter:=counter−1;
    consumer the item in nextc;
until false;
```

Question

- 两个进程共享变量 counter
- counter 会导致结果不确定

生产者 – 消费者问题 (2)

- 设 counter 的初值为 5

```
register1:=counter;
register1 :=register1+1;
counter :=register1;
```

```
register2:=counter;
register2:=register2−1;
counter :=register2;
```

```
register1:=counter;          (register1:=5)
register1 :=register1+1;     (register1:=6)
register2:=counter;          (register2:=5)
register2 :=register2-1;     (register2:=4)
counter :=register1;         (counter:=6)
counter :=register2;         (counter:=4)
```

3. 临界区

- 定义：进程访问临界资源的那段代码称为临界区
- 访问临界资源的描述：
  - 进入区：检查有无进程进入
  - 临界区：
  - 退出区：将访问标志复位

Repeat

Entry section

Critical section

Exit section

Until false

4. 同步机制应遵循的准则

- 1. 空闲让进
- 2. 忙则等待
- 3. 有限等待
  - 应保证为有限等待，避免“死等”。
- 4. 让权等待
  - 不能进入临界区的执行进程应放弃 CPU 执行权。避免“忙等”

2.3.2 信号量机制

- Edsger Wybe Dijkstra(1930 年 5 月 11 日–2002 年 8 月 6 日)
  - 毕业于 Leiden 大学
  - 1972 年获得图灵奖
  - 1989 年计算机科学教育杰出贡献奖
  - 2002 年 ACM PODC 最具影响力论文奖
- 与 Knuth 并称为我们这个时代最伟大的计算机科学家的人。
  - 提出“goto 有害论”；
  - 1965 年，提出信号量和 PV 原语;
  - 解决了有趣的“哲学家聚餐”问题;
  - 最短路径算法 (SPF) 和银行家算法的创造者;
  - 第一个 Algol 60 编译器的设计者和实现者;
  - THE 操作系统的设计者和开发者;

2.3.2 信号量机制

- 1 整型信号量
  - 是一个整型量，通过 2 个原子操作wait(s) 和 signal(s) 来访问。
  - Wait(s):

```
while s<= 0 do no−op;
s:=s−1;
```

  - Signal(s):

```
s:=s+1;
```

2 记录型信号量

```
type  semaphore=record
        value:integer;
        L: list of process;
end
procedure wait(s)
  var s: semaphore
  begin
        s.value:=s.value − 1;
        if s.value < 0 then block (s, L)
  end
procedure signal (s)
  var s:semaphone
  begin
        s.value:=s.vaule + 1
        if s.value<=0 then wakeup(s.L)
  end
```

- L: 为进程链表，用于链接所有等待该类资源进程。
- 用 wait(s) 和 signal(s) 实现同步与互斥。
- 在记录型信号量机制中：
- s.value 初值：表示系统中某类资源的数目。
- s.value<0: 表该信号量链表中已阻塞进程的数目。
- Qestion: 为什么叫“记录型”信号量?

PV 操作

- Wait(s): 也用 $P(s)$ 或者 $Down(s)$ 表示，相当于申请资源
- Signal(s): 也用 $V(s)$ 或者 $Up(s)$ 表示，相当于释放资源
- 例如:
  - 在公共电话厅打电话

3 AND 型信号量

- 当不用它时，有可能发生系统死锁。
- 死锁：在无外力作用下的一种僵持状态。
- 特点：要么全分配，要么一个也不分配。

3 AND 型信号量

process A:	process B:
wait(Dmutex);	wait(Emutex);
wait(Emutex);	wait(Dmutex);

若两个进程交替执行，则死锁

3 AND 型信号量

Swait(s1,s2,$\cdots$,sn)

if s1 >= 1 and $\cdots$ and sn >= 1 then

for i:=1 to n do si:=si−1; endfor

else

place the process in the waiting queue with the first si found with si<1, and set the program count of this process to the beginning of swait operation

end if

Ssignal(s1,s2,$\cdots$,sn)

for i:=1 to n do si:=si+1;

remove all the process waiting in the queue associated with si into the ready queue

endfor

4 信号量集

- 某进程需要 100 个临界资源 X 时：
  - wait(x);
  - ……
  - wait(x)
- 有些情况下，只有系统空闲资源数量大于等于一定数值，才予以分配。

4 信号量集

- 为提高效率而对 AND 信号的扩充。
  - Swait(S, t, d): t 为下限制，d 为需求值
- 三种特例:
  - （1）Swait(S,d,d)：允许每次申请 d 个资源。
    - 当资源数少于 d 时，不予分配。
  - （2）Swait (S,1,1)：$S > 1$，记录型信号量。
    - S=1 时，互斥型信号量。
  - （3）Swait(S,1,0)，可控开关，当 $S \geq 1$ 时，允许进入，$S < 1$ 时，不能进入。

利用信号量实现互斥

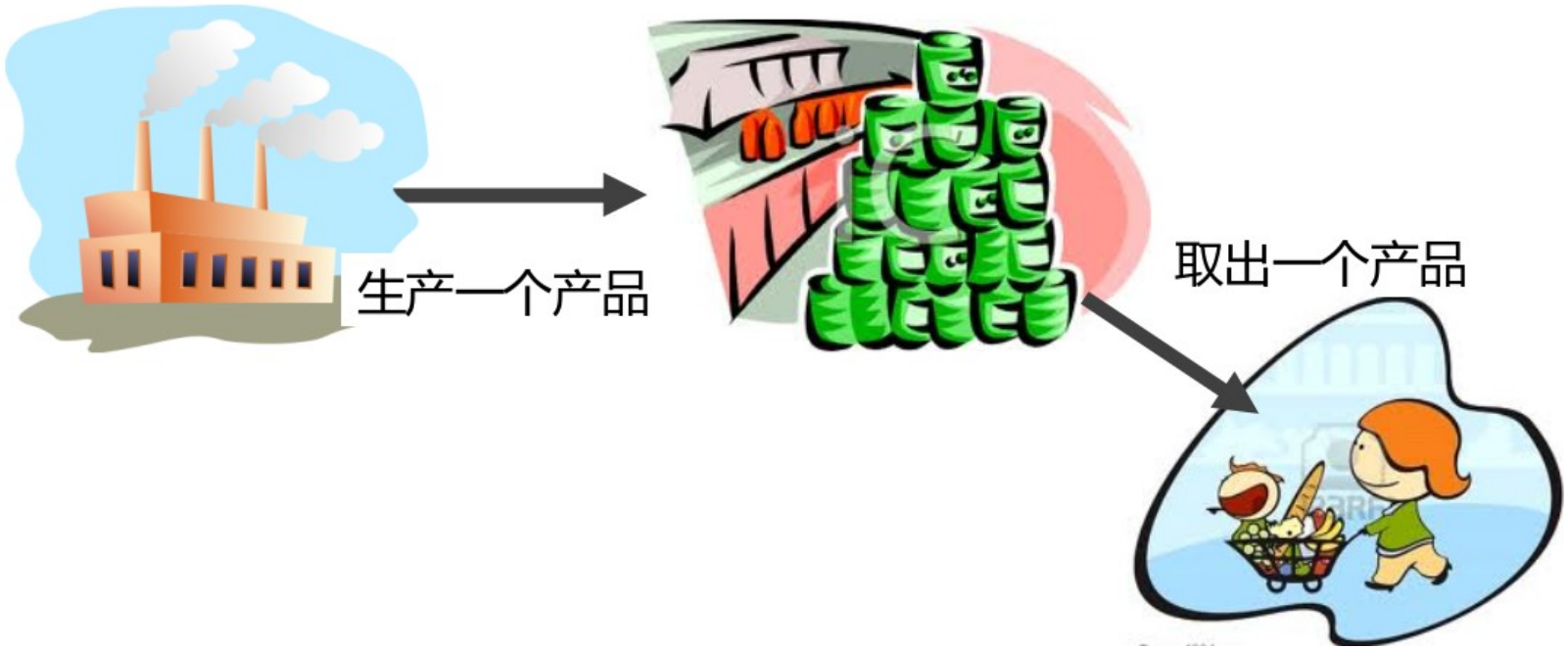
```
var mutex: semaphore:=1

parbegin
    process1:begin
        repeat
            wait(mutex);
            critical setion
            signal(mutex);
            remainder section
        until false;
    end
    process2: begin
        repeat
            wait(mutex);
            critical setion
            signal(mutex);
            remainder section
        until false;
    end
parend
```

2.4 经典进程同步问题

- 2.4.1 生产者—消费者问题
- 2.4.2 哲学家进餐问题
- 2.4.3 读者—写者问题

生产者—消费者问题



生产者—消费者问题

- 定义两个同步信号量：
  - empty—表示缓冲区中空缓冲区的数量，初值为 n(n 个缓冲区)。
  - full—表示缓冲区中满的数量，初值为 0。
  - mutex—使诸进程互斥地访问缓冲区

记录型信号量解决生产者—消费者问题

```
var mutex,empty,full:Semaphore:=1,n,0;
    buffer:array[0,1,⋯,n−1] of item;
    in, out: integer: =0,0;
```

```
producer: begin
    repeat
        ⋯
        Produce an item in nextp;
        wait(empty);
        wait(mutex);
        buffer(in):=nextp;
        in:=(in+1) mod n;
        signal(mutex);
        signal(full);
    until false;
end
```

```
consumer:begin
    repeat
        wait(full);
        wait(mutex);
        nextc:=buffer(out);
        out:=(out+1) mod n;
        signal(mutex);
        signal(empty);
        Consumer the item in nextc;
    until false;
end
```

利用 AND 信号量解决生产者—消费者问题

```
var  mutex, empty, full: semaphore:=1,n,0;
     buffer:array[0,⋯,n−1] of item;
     in out: integer :=0,0;
```

```
producer: begin
    repeat
        ...
        Produce an item in nextp;
        Swait(empty, mutex);
        buffer(in):=nextp;
        in:=(in+1) mod n;
        Ssingal(mutex, full);
    until false;
end
```

```
consumer:begin
    repeat
        Swait(full, mutex);
        nextc:=buffer(out);
        out:=(out+1) mod n;
        Ssignal(mutex, empty);
        consumer the item in nextc;
    until false;
end
```

2.4.2 哲学家进餐问题

有五个哲学家围坐在一圆桌旁，桌中央有一盘通心面，每人面前有一只空盘子，每两人之间放一把叉子。每个哲学家思考、饥饿、然后吃通心面。为了吃面，每个哲学家必须获得两把叉子，且每人只能直接从自己左边或右边去取叉子。

哲学家进餐问题解决思路

- 1、每一把叉子都是必须互斥使用的，所以，必须为每一把叉子设置一个互斥信号量 S_i （ $i = 0, 1, 2, 3, 4$);
- 2、初值都为 1;
- 3、当一个哲学家吃面时必须获得自己左边和右边的两把叉子，即执行两个 P 操作 $(wait)$；吃完面后，必须放下两个叉子，即执行两个 V 操作 $(signal)$。

利用记录型信号量解决哲学家进餐问题

第 i 个哲学家:

```
var chopstick: array[0, ···, 4] of semaphore;
repeat
    wait(chopstick[i]);
    wait(chopstick[(i+1)mod 5]);
    ...
   eat
    ...
   signal(chopstick[i]);
   signal(chopstick[(i+1)mod 5]);
    ...
   think;
until false
```

利用 AND 信号量解决哲学家进餐问题

```
var chopstick: array[0, ···, 4] of semaphore:=(1,1,1,1,1);
processi
    repeat
        think;
        Sswait(chopstick[(i+1) mod 5],chopstick[i]);
        eat
        Ssignal(chopstick[(i+1) mod 5],chopstick[i]);
    until false
```

课堂练习

- 请利用记录型信号量写出一个不会死锁的哲学家进餐问题的算法。

2.4.3 读者—写者问题

- 读者写者问题
  - 有两组并发进程:
    - 读者和写者, 共享一组数据区
  - 要求:
    - 允许多个读者同时执行读操作
    - 不允许读者、写者同时操作
    - 不允许多个写者同时操作
- 特点:
  - 读进程可共享同一对象。
  - 写进程不可共享同一对象。

利用记录型信号量解决读者—写者问题 I


```
var rmutex, wmutex: semaphore: =1,1;
    readcount:integer: =0;
begin
    parbegin
```

利用记录型信号量解决读者—写者问题 II

```
reader: begin
  repeat
      wait(rmutex);
      if readcount=0 then wait(wmutex);
      readcount:=readcount+1;
      signal(rmutex);

      ...
      perform read operation
      ...

      wait(rmutex);
      readcount:=readcount−1;
      if readcount=0 then signal(wmutex);
      signal(rmutex);
  until false;
end
```

利用记录型信号量解决读者—写者问题 III


```
    writer: begin
      repeat
          wait(wmutex)
          perform write operation;
          signal(wmutex)
       until false;
     end
   parend
end
```

信号量集解决读者—写者问题 (略) I

```
var RN integer;
        L, mx: semaphore: =RN, 1;
 begin
   parbegin
     reader: begin
       repeat
              swait(L,1,1);
              swait(mx,1,0);
                ...
              perform read operation;
                ...
              ssignal(L,1);
        until false;
      end
      writer: begin
        repeat
```

信号量集解决读者—写者问题 (略) II

```
            swait(mx,1,1; L,RN,0);
            perform write operation;
            ssignal(mx, 1);
        until flase;
    end
  parend
end
```

上述方法为读者优先

- 如果读者来：
  - (1) 无读者、写者，新读者可以读
  - (2) 有写者等，但有其它读者正在读，则新读者也可以读
  - (3) 有写者写，新读者等
- 如果写者来：
  - (1) 无读者，新写者可以写
  - (2) 有读者，新写者等待
  - (3) 有其它写者，新写者等待

读者优先分析

- 问题：
  - 读者源源不断，readCount 不归 0，写者会被饿死。
- 策略：
  - 一旦有写者等待，新到达读者等待，正在读的读者都结束后，写者进入。

写者优先

- 条件：
  - (1) 多个读者可以同时进行读
  - (2) 写者必须互斥（只允许一个写者写，也不能读者写者同时进行）
  - (3) 写者优先于读者（一旦有写者，则后续读者必须等待，唤醒时优先考虑写者）

- 练习尝试

写者优先 (采用类 Java/C 伪代码) I

```
semaphore fmutex=1, rdcntmutex=1, wtcntmutex=1, queue=1;
//fmutex -->access to file;
// rdcntmutex -->access to readcount
//wtcntmutex -->access to writecount
int readcount = 0, writecount = 0;

void reader(){
    while(true){
        wait(queue);
        wait(rdcntmutex);
        if(0 == readcount)wait(fmutex);
        readcount = readcount + 1;
        signal(rdcntmutex);
        signal(queue);

        //Do read operation ...
```

写者优先 (采用类 Java/C 伪代码) II

```
        wait(rdcntmutex);
        readcount = readcount - 1;
        if(0 == readcount) signal(fmutex);
        signal(rdcntmutex);
    }
}

void writer(){
    while(true){
        wait(wtcntmutex);
        if(0 == writecount)wait(queue);
        writecount = writecount + 1;
        signal(wtcntmutex);

        wait(fmutex);
```

写者优先 (采用类 Java/C 伪代码) III

```
        //Do write operation ...
        signal(fmutex);

        wait(wtcntmutex);
        writecount = writecount − 1;
        if(0 == writecount)signal(queue);
        signal(wtcntmutex);
    }
}
```

练习

- a,b 两点间是一段东西向的单行车道，现要设计一个自动管理系统，管理规则如下：
  - 当 ab 间有车辆在行驶时，同方向的车可以继续驶入 ab 段，但另一方向的车必须在 ab 段外等待；
  - 当 ab 之间无车时，到达 a（或 b）的车辆可以进入 ab 段，但不能从 a，b 点同时驶入；
  - 当某方向在 ab 段行驶的车辆使出了 ab 段且无车辆进入 ab 段时，应让另一方向等待的车辆进入 ab 段行驶。

- 请用 wait,signal 工具对 ab 段实现正确管理。

练习答案 I

```
semaphore s, mutexab,mutexba
integer countab = 0, countba = 0
pab:
    wait(mutexab);
    countab++;
    If countab=1 then wait(s);
    signal(mutexab);
    ...
    wait(mutexab);
    countab−−;
    if countab=0 then signal(s);
    signal(mutexab);
```

练习答案 II

```
pba:
    wait(mutexba);
    countba=countba+1;
    if countba=1 then wait(s);
    signal(mutexba);
    enter;
    ... ...
    wait(mutexba);
    countba−−;
    if countba=0 then signal(s);
    signal(mutexba);
```

作业讨论

- 无死锁的哲学家进餐问题
- 写者优先

练习

- 设有两个生产者进程 A、B 和一个销售者进程 C，他们共享一个无限大的仓库
  - 生产者每次循环生产一个产品，然后入库供销售者销售；
  - 销售者每次循环从仓库中取出一个产品进行销售。
  - 不允许同时入库，也不允许边入库边出库；
  - 要求生产和销售 A 产品和 B 产品的件数都满足以下关系：
    - $-n \leqslant$ A 生产的件数 $-$ B 生产的件数 $\leqslant m$,
    - $-n \leqslant$ A 销售的件数 $-$ B 销售的件数 $\leqslant m$,
    - 其中 m, n 是正整数。

- 使用信号量机制写出 A、B 和 C 三个进程的工作流程

分析

- 设置信号量 mutext，互斥访问仓库
- 为满足 $-n \leq A$ 的件数 $-B$ 的件数 $\leq m$，设置两个同步信号量
  - SAB：允许 A 当前生产的数量，初值为 m
  - SBA：允许 B 当前生产的数量，初值为 n
- 为实现生产者和销售者的同步，
  - 设置变量 Difference：表示 A 与 B 所销售的数量之差，初值为 0
  - 设置三个信号量：S、SA、SB，分别表示仓库中总的产品量、仓库中 A 的产品量和 B 的产品量，初值为 0

生产者

Process A:

```
repeat
    wait(SAB)
    produce a product A
    signal(SBA)

    //加入仓库
    wait(mutex)
    add A to storehouse
    signal(mutex)

    siganl(SA)
    signal(S)
until false
```

Process B

```
repeat
    wait(SBA)
    produce a product B
    signal(SAB)

    //加入仓库
    wait(mutex)
    add B to storehouse
    signal(mutex)

    siganl(SB)
    signal(S)
until false
```

销售者 C

```
Repeat
    wait(S)
    if (difference<=−n) {
      //B卖的太多了
        wait(SA)
        wait(mutex)
        take a product A
        signal(mutex)
        difference += 1
    } else if(difference >=m) {
      //A卖的太多了
        wait(SB)
        wait(mutex)
        take B
        signal(mutex)
        difference −= 1
    } else{
        wait(mutex)
        take product A or B
        signal(mutex)
        if(type==A){
            wait(SA)
            difference+=1
        } else {
            wait(SB)
            difference −=1
        }
    }
    Sell product
until false
```

练习（嗜睡的理发师问题）

- 一个理发店由一个有 N 张沙发的等候室和一个放有一张理发椅的理发室组成。
- 没有顾客时，理发师便去睡觉。
- 当一个顾客走进理发店时，如果所有的沙发都已被占用，便离开理发店；否则，如果理发师正在为其他顾客理发，该顾客就找一张空沙发坐下等待；如果理发师因无顾客正在睡觉，则由新到的顾客唤醒理发师为其理发。在理发完成时，顾客必须付费，直到理发师收费后才能离开理发店。

- 试用信号量实现这一同步问题

分析

- 设置整型变量 count，记录顾客数；
- 设置 mutex 信号量，保证对 count 的互斥，初值为 1
- 对等候室中的 N 张沙发，设置信号量 sofa，初值为 N
- empty 信号量表示是否有空闲的理发椅，初值为 1
- full 表示理发椅上是否有等待理发的顾客，初值为 0
- cut 信号量表示理发是否完成，初值为 0
- payment 表示等待付费，初值为 0
- receipt 表示等待收费，初值为 0

顾客

```
wait(mutex)
if(count>N){
    signal(mutex)
    exit
} else {
    count++;
    signal(mutex)
    if(count>1) {
        wait(sofa)
        sit on
        wait(empty)
        get up from sofa
        signal(sofa)
    } else {
        wait(empty)
    }
    ...
}
```

顾客

```
wait(mutex)
if(count>N){
    signal(mutex)
    exit
} else {
    count++;
    signal(mutex)
    if(count>1) {
        wait(sofa)
        sit on
        wait(empty)
        get up from sofa
        signal(sofa)
    } else {
        wait(empty)
    }
    ...
}
```

```
sit on baber_chair
signal(full)
wait(cut)
pay
signal(payment)
wait(receipt)
get up from baber_chair
signal(empty)
wait(mutex)
count−−
siganl(mutex)
exit shop
```

理发师

```
while(true){
    wait(full)
    cut hair
    signal(cut )
    wait(payment)
    accept payment
    signal(recipt)
}
```

2.5 管程机制

- 70 年代初, By
  - E.W.Dijkstra, C.A.R.Hoare, P.B.Hansen.
  - 背景: Structured programming
- 引入原因:
  - 为了避免凡要使用临界资源的进程都自备同步操作 wait(s) 和 signal(s). 将同步操作的机制和临界资源结合到一起，形成管程。
  - 信号量程序编写困难
  - 让困于人：将信号量的组织工作交给一个专门的机构负责，解脱程序员。

2.5.1 管程的基本概念

- 一、定义：一个数据结构和能为并发进程所执行的一组操作。
  - 局部于管程的共享变量。
  - 对该数据结构进程操作的一组过程。
  - 对局部管程数据设置初值。
- 二、条件变量：
  - x.y: x.wait; x.signal; x.queue

2.5.2 利用管程解决生产者—消费者问题 I

- 一、建立管程：PC
  - 包括：二过程：
    - （1）put(item) 过程；
    - （2）get(item) 过程
  - 一变量：$count \geqslant n$ 时满；$\leqslant 0$ 时空
  - 初始：in=out=count=0

```
type producer−consumer=monitor
    var in,out,count: integer;
    buffer: array [0,···,n−1] of item;
    notfull, notempty: condition;
    procedure entry put (item)
    procedure entry get (item)
```

2.5.2 利用管程解决生产者—消费者问题 II

```
Procedure entry put(item)
~~|begin
~~|  if count >= n then notfull.wait;
~~|    buffer(in):=nextp;
~~|    in:=(in+1)mod n
~~|    count:=count+1;
~~|    if notempty.queue then notempty.signal;
~~|end
Procedure entry get(item)
~~|begin
~~|  if count <= 0 then notempty.wait;
~~|    nextc:=buffer(out);
~~|    out:=(out+1)mod n
~~|    count:=count−1;
~~|    if notfull.queue then notfull.signal;
~~|end
```

2.5.2 利用管程解决生产者—消费者问题 III

```
Begin in:=out:=0;   count:=0 end

producer: begin
    repeat
        produce an item in nextp
        PC. put (item);
    until false.
end
consumer: begin
    repeat
        PC.get(item);
        consume the item in nextc;
    until false
end
```

PV 操作与管程对比

PV 操作：

(1) 分散式同步机制：共享变量操作，PV 操作，分散在整个系统中或各个进程中。

(2) 缺点：

(a) 可读性差；

(b) 正确性不易保证；

(c) 不易修改。

(3) 优点：高效，灵活。

管程：

(1) 集中式同步工具：共享变量及其所有相关操作集中在一个摸块中。

(2) 优点：

(a) 可读性好；

(b) 正确性易于保证；

(c) 易于修改。

(3) 缺点：不甚灵活，效率略低。

2.6 进程通信

- 概念：进程间的信息交换。
- 实例：
  - 信号量机制（一种低级通信）
    - 缺点：
      - （1）效率低
      - （2）通信对用户不透明
  - 高级通信特点：
    - 效率高，通信实现细节对用户透明

2.6.1 进程通信的类型 I

- 一、共享存贮器系统
  - 1. 基于共享数据结构的通信方式：
    - produce-consume 中的缓冲区，低效，不透明。
    - 系统只提供了一共享存贮器，适于少量通信。
  - 2. 基于共享存储区的通信方式：
    - 系统提供：共享存储区。
    - 通信过程：
      - （1）向系统申请一个或多个分区
      - （2）获得分区获后即可读/写.
    - 特点：高效，速度快。

2.6.1 进程通信的类型 II

- 二、消息传递系统（可用于异种机）
  - 信息单位：消息（报文）
  - 是目前的主要通信方式，分为直接通信方式、间接通信方式
  - 实现：一组通信命令（原语），具有透明性　同步的实现。
- 三、管道通信
  - 管道：连接一个读进程和一个写进程之间通信的共享文件。
  - 功能：大量的数据发收。
  - 注意：
    - （1）互斥
    - （2）同步
    - （3）对方是否存在

2.6.2 消息传递通信的实现方法 I

- 一、直接传递方式
  - send(Receiver, message)
  - receive(Sender, message)
  - 例：解决生产—消费问题

```
repeat
    produce an item in nextp;
    ...
    send(consumer, nextp);
until false;
repeat
      receive( producer, nextc);
      ...
      consumer the item in nextc;
 until false;
```

2.6.2 消息传递通信的实现方法 II

- 二、间接（可以实现非实时通信）
  - 优点：在读/写时间上的随机性
  - 写进程 → 信箱（中间实体）→ 读进程
  - 原语
    - （1）信箱的创建与撤消：
      - 信箱名属性（公用、私用、共享）（共享者名字）
    - （2）消息的发送和接收
      - Send (mailbox, message)
      - Receive (mailbox, message)

2.6.2 消息传递通信的实现方法 III

- 信箱类型
  - （1）私用：拥有者有读/写数，其它只有写权，（单向）存在期 = 进程存在期。
  - （2）公用：系统创建, 双向, 存在期 = 系统存在期。
  - （3）共享信箱：一般进程创建，并指明其共享者，是双向。
- 发送—接收进程之间的关系：
  - （1）一对一关系；
  - （2）多对一关系；（客户-服务器方式）
  - （3）一对多关系；（适用于广播方式）
  - （4）多对多关系：公用信箱

2.6.3 消息传递系统中的几个问题 I

- 一、通信链路：
  - （1）显式建立：（进程完成、网络中）
  - （2）隐式建立：（系统完成、单机中）
  - 链路类型：
    - （1）由连接方法分：点—点链路，多点链路。
    - （2）由通信方式分：单向、双向。
    - （3）由容量分：无容量（无缓冲区）、有（有缓冲区）。

2.6.3 消息传递系统中的几个问题 II

- 二、消息格式：
  - 格式组成
    - 消息头：含控制信息如：收/发进程名，消息长度、类型、编号
    - 消息内容：
  - 格式类型
    - 定长消息：系统开销小，用户不便（特别是传长消息用户）
    - 变长消息：开销大，用户方便。

2.6.3 消息传递系统中的几个问题 III

- 三、进程同步方式
  - 1. 发送和接收进程阻塞（汇合）
    - 用于紧密同步，无缓冲区时。
  - 2. 发送进程不阻塞，接收进程阻塞（多个）
    - 相当于接收进程（可能是多个）一直等待发送进程，如：打印进程等待打印任务。
  - 3. 发送/接收进程均不阻塞
    - 一般在发、收进程间有多个缓冲区时。

2.7 线程

2.7.1 线程的基本概念 (1)

- 1. 线程的引入
  - 减少并发执行时的时空开销，进程的创建、撤消、切换较费时空，因它既是调度单位，又是资源拥有者。
  - 线程是系统独立调度和分派的基本单位，其基本上不拥有系统资源，只有少量资源（寄存器，栈…），但共享其所属进程所拥有的全部资源。

2.7.1 线程的基本概念 (2)

- 2. 线程的属性
  - 轻型实体
  - 独立调度和分派的基本单位
  - 可并发实体
  - 共享进程资源
- 3. 线程的状态
  - 状态参数
    - 寄存器状态、堆栈、运行状态、优先级、线程专有存储器、
    - 信号屏蔽
  - 线程的运行状态
    - 就绪、执行、阻塞

2.7.1 线程的基本概念 (3)

- 4. 线程的创建和终止
  - 初始化线程
- 5. 多线程中的进程
  - 进程是拥有系统资源的基本单位，但不再是一个可执行的实体。

2.7.2 线程的同步和通信

- 1. 互斥锁
  - 阻塞方式

    lock(mutex)

    访问

    unlock(mutex)
  - 非阻塞方式

    if（ trylock ）then

    else

2.7.2 线程的同步和通信

- 2. 条件变量
  - 用于线程的长期等待
- 3. 信号量机制
  - 私用信号量（private semaphore)
    - 作用域在一个进程中
  - 公用信号量（public semaphore)
    - 作用于多个进程间

练习 I

1. 进程的并发执行是指若干个进程 (　　)
   A、同时执行
   B、在执行的时间上是重叠的
   C、在执行的时间上是不可重叠的
   D、共享系统资源
2. 若 PV 操作的信号量 S 初值为 2, 当前值为 − 1, 则表示有 (　　) 个等待进程。
   A、0 个　　B、1 个　　C、2 个　　D、3 个
3. 用 PV 操作管理临界区时，信号量的初值应定义为 (　　)
   A、 − 1　　B、0　　C、1　　D、任意值
4. 用 V 操作唤醒一个等待进程时，被唤醒进程的状态变为 (　　)
   A、等待　　B、就绪　　C、运行　　D、完成

练习 II

5. (　　) 是一种只能进行 P 操作和 V 操作的特殊变量。(PV 操作只能在 (　　) 上操作)

   A、调度　　B、进程　　C、同步　　D、信号量

6. 对于两个并发进程，设互斥信号量为 mutex，若 mutex = 0, 则 (　　)

   A、表示没有进程进入临界区

   B、表示有一个进程进入临界区

   C、表示有一个进程进入临界区，另一个进程等待进入

   D、表示有两个进程进入临界区

7. 临界区是 (　　)

   A、一个缓冲区　　B、一段共享数据区

   C、一段程序代码　　D、一个互斥资源

8. 信号量的物理意义是当信号量大于零时表示 (　　)；当信号量小于零时，其绝对值为 (　　)。

练习 III

9. 临界资源的概念是 (　　)，而临界区是指 (　　)。
10. 若一个进程已进入临界区，其它欲进入临界区的进程必须 (　　)。
11. 用 PV 操作管理临界区时，任何一个进程在进入临界区之前应调用 (　) 操作，退出临界区时应调用 (　) 操作。
12. 有 m 个进程共享同一临界资源，若使用信号量机制实现对临界资源的互斥访问，则信号量的变化范围是 (　　)。
13. 操作系统中，对信号量 S 的 P 原语的定义中，使进程进入相应等待队列等待的条件是 (　　)。
14. 如果信号量的当前值为 − 4，则表示系统中在该信号量上有 (　) 个等待进程。
15. 并发进程之间的基本关系是 (　　) 或 (　　)。其中 (　　) 是指进程之间的一种间接的关系。

本章作业及延伸阅读

- 作业：操作系统同步之信号量机制

  根据文章介绍，实现并进行体验。https://zhuanlan.zhihu.com/p/34410587

  要求：以 Markdown 格式交作业，包括但不限于以下部分：

  * 整个测试流程及代码
  * 心得体会
  * 个人信息
- 阅读（不用交）：关于现代 CPU，程序员应当更新的知识：

  http://www.iteye.com/news/30978

— END —

展示作业 I

1. 协程与进程
2. CPU 与 GPU
3. 中国芯片发展状况
4. 开源软件的发展历史和重要开源项目介绍
5. 计算机存储系统的现状与发展趋势
6. Linux 常用命令
X. 区块链
7. Java 编程语言的发展趋势
X. 著名 IT 公司介绍
8. 著名 IT 人物介绍
9. 信息的度量
10. 操作系统扩展学习资料汇编

致谢

本讲义的内容来自于参考教材及互联网上的部分讲义和公开资料，包括但不限于：

计算机操作系统、深入理解计算机系统、操作系统精髓、操作系统之哲学原理、带你逛西雅图活电脑博物馆...

遗漏之处，请留言联系以便补充。