

ch7 进程同步

ch7 进程同步

背景

临界区问题/进程互斥

临界区概念

临界区使用原则

进程互斥方法

软件方法

硬件方法

进程同步

同步机制

信号量

管程

典型同步互斥问题

生产者-消费者问题

读者-写者问题

哲学家就餐问题

信号量使用注意事项

举例

寄信

吃水果

背景

- 对共享数据的并发访问（**并行存取**）可能导致数据不一致。
- 维护**数据一致性**需要机制来确保协作过程的有序执行。
- **竞争条件**：
 - 多个进程同时访问和操作共享数据的情况。
 - 共享数据的最终值取决于最后完成的进程。
- 为了防止竞争条件，必须同步并发进程。

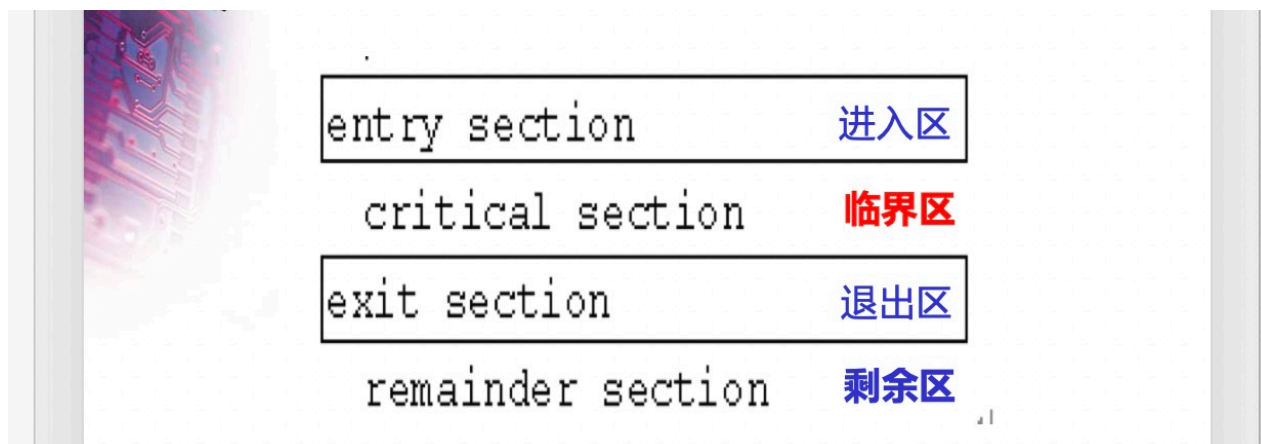
临界区问题/进程互斥

临界区概念

临界资源（互斥资源）：一次只允许一个进程使用的资源

临界区（互斥区）：在进程中涉及到临界资源的程序段。

进入区 退出区 剩余区（代码中的其余部分）



临界区使用原则

- 互斥访问
- 有空让进
- 无空等待
- 多中择一
- 有限等待
- 让权等待

处于等待状态的进程应放弃占用CPU，以使其他进程有机会得到CPU的使用权。

进程互斥方法

软件方法

只讨论了两个进程

- 基本思路
 - 在进入区检查和设置一些标志，如果已有进程在临界区，则在进入区通过循环检查进行等待；在退出区修改标志
 - 主要问题是设置什么标志和如何检查标志

- 为了简单说明，算法中只涉及两个协作进程

- 算法1:单标志位

turn

轮流进入临界区

- 算法2:双标志位（先检查）

flag[]

不用交替进入

Pi 和 Pj 可能同时进入临界区

- 算法3:双标志位（先修改）

flag[]


放置两个进程同时进入临界区

有可能都进不了临界区

算法2, 3 的问题在于，检查和修改操作不能连续进行

- 算法4:结合算法1, 3（先修改、后检查、后修改者等待）

Turn + flag[]



```
flag[i] = TRUE; turn = j;  
while (flag[j] && turn == j);
```

critical section

```
flag[i] = FALSE;
```

remainder section

终止，软件算法的缺点：

1. 忙等待
2. 实现复杂，需要高的编程技巧

硬件方法

原子操作

- 硬件解法1

“测试并设置”指令 TS (test and set)

- 硬件解法2

“交换”指令 Swap/Exchange

- 硬件解法3

“开关中断”指令

进入临界区前：关中断

离开临界区后：开中断

总结，硬件方法的优点：

1. 适用于任意数目的进程，在单处理器或多处理器上均适用
2. 简单，容易验证其正确性
3. 可以支持进程内存的多个临界区，只需为每个临界区设立一个布尔变量

缺点：

1. 等待要耗费CPU时间，不能实现“让权等待”
2. 可能“饥饿”：从等待进程中随机选择一个进入临界区，有的进程可能一直选不上
3. 可能死锁

进程同步

同步机制

OS可从进程管理者的角度来处理互斥同步问题，这就是同步机制。

同步机制：

- 信号量P、V操作；
- 管程（资源管理模块）；
- 条件临界域等

信号量

信号量表示资源的实体，是一个与队列有关的整型变量。数据结构定义如下：

```
struct semaphore
{
    int value;
    pointer_PCB queue; //阻塞在该信号量的各个进程的标识
}
```

信号量只能通过**初始化**和**P/V**原语来访问，不受进程调度的打断。

信号量分类：

1. 公用信号量（互斥信号量）

初值为1，允许一组进程对它操作

2. 私用信号量（同步信号量）

初值为0或某个正整数，只允许拥有进程对其操作

互斥信号量用于申请或释放资源的**使用权**，常初始化为1。

资源信号量用于申请或归还**资源**，可以初始化为大于1的正整数，表示系统中某类资源的可用数。

信号量的值：

- 非负数：空闲资源数
- 负数：等待临界区的进程数

信号量的P/V操作：

• P (wait)

P (S)

① $S=S-1$

② 如果 $s \geq 0$,则调用P(S)的进程继续运行

③ 如果 $s < 0$,则调用P(S)的进程被阻塞，并把它插入到等待信号量S的阻塞队列中。

• V (signal)

V(S)

① $S=S+1$

② 如果 $s > 0$,则调用V(S)的进程继续运行

③ 如果 $s \leq 0$,则从等待信号量s的阻塞队列中 唤醒头一个进程到就绪队列中，然后调用v(s)的进程继续运行。

P.V操作必须成对出现：

- n当为互斥操作时，它们同处于同一进程
- n当为同步操作时，则不在同一进程中出现

如果P(S1)和P(S2)两个操作在一起，那么P操作的顺序至关重要。

P/V优缺点：

- 优点
简单，而且表达能力强
- 缺点
不够安全，使用不当会出现死锁；
遇到复杂同步互斥问题时实现复杂

P/V操作的应用：

1. 实现互斥
2. 实现同步

管程

对信号量的二次封装

管程的引入

信号量机制的引入解决了进程同步的描述问题，但信号量的大量同步操作分散在各个进程中不便于管理，还可能导致系统死锁。

把资源集中起来，构成秘书进程。

管程概念

管程定义了一个数据结构和能为并发进程所执行的一组操作，这组操作能同步进程和改变管程中的数据。

管程的构成

- 局部于管程的共享数据结构
- 对共享数据结构进行操作的一组函数
- 对局部于管程的数据设置初始值的语句

管程的基本特性

局部性

保护性

互斥性

系统级

条件变量

利用管程实现同步时，还应设置条件变量和在条件变量上进行操作的两个同步原语

- 条件变量用于区别各种不同的等待原因。
- 同步原语(wait和signal)。

典型同步互斥问题

生产者-消费者问题

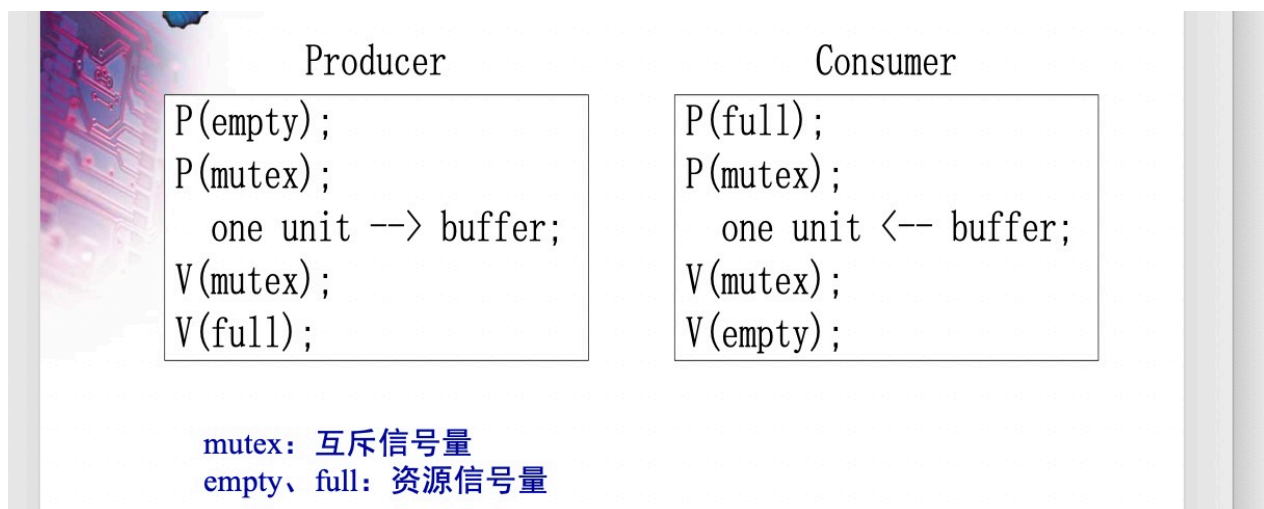
互斥 + 同步

full = 0

empty = N

mutex = 1

先检查资源数量，再检查是否互斥。（写反会导致死锁）



读者-写者问题

不同步



读者间不互斥，其他情况互斥

- 第一类：读者优先（简单）

当读者在读，即使有写者在等，后续读者现在也能进来读

所有读者读完，才唤醒一个写者

可能导致写者饥饿



读者：

```
while (true) {
    P(mutex);
    readcount++;
    if (readcount==1)
        P(w);
    V(mutex);
    读
    P(mutex);
    readcount--;
    if (readcount==0)
        V(w);
    V(mutex);
};
```

写者：

```
while (true) {
    P(w);
    写
    V(w);
};
```

同濟大學軟件學院

82

mutex、w初值为1，readcount初值为0

readcount用来记录当前有多少个读者在访问数据



mutex用来保证读者之间互斥地修改readcount。

w是读者和写者公用的互斥变量，用来互斥读写或写写同时进行
(两个互斥信号量)

- 第二类：写者优先

主要有两点考虑：

1. 一旦有写者在等待，则后续读者必须等待
2. 写者结束后，唤醒时优先考虑写者



读者：

```
while (true) {
    P(z);
    P(r);
    P(x);
    readcount++;
    if(readcount==1) P(w);
    V(x);
    V(r);
    V(z);
    读
    P(x);
    readcount--;
    if(readcount==0) V(w);
    V(x);
};
```

写者：

```
while (true) {
    P(y);
    writecount++;
    if(writecount==1) P(r);
    V(y);
    P(w);
    写
    V(w)
    P(y);
    writecount--;
    if(writecount==0) V(r);
    V(y);
};
```

readcount,writecount用来记录当前有多少个读者和写者在访问数据

w是读者和写者公用的互斥变量，用来互斥读写或者写写同时进行

y用来保证写者之间互斥地修改writecount

x用来保证读者之间互斥地修改readcount

r用来保证当有写者等待时，后续的读者必须也等待

z用来保证读者之间互斥地修改信号量r -> 写者写完之后，唤醒时优先唤醒写者

x,y,z,r,w初值为1

r只能被减到-1

z可以被减到很小

如果没有信号量z，则当写者运行时，后面的读者被r阻塞，后面的写者被y阻塞。当现在的写者结束时，首先释放r，再释放y，导致读者优先被唤醒。

哲学家就餐问题

- 为每根筷单独设立一个信号量，哲学家取筷子执行P操作，放下筷子执行V操作。

var chopstick : array[0..4] of semaphore;初值为1

各个哲学家执行的程序为：

P(chopstick[i]); //取左边筷子

P(chopstick[(i+1) mod 5]); //取右边筷子

进食

V(chopstick[(i+1) mod 5]); //放下右边筷子

V(chopstick[i]); //放下左边筷子

思考

死锁

改进方案1：

✓ 算法改进

- 当哲学家获得左边筷子后，查看右边筷子是否可用，如不可用，则“谦让”，放下已获得的左边筷子，等一段时间再重复之前的过程。
- 该方法可能导致各哲学家都“谦让”，都处于“忙等”，并都进入“饥饿”状态。

改进方案2：

增加一个服务员，让他安排其中的四位哲学家先入座，四位竞争5只筷子，必有一位会同时获得两双筷子，他用餐完毕，再安排剩下那位就坐，重复竞争、用餐过程。

资源信号量 room = 4

✓ 算法改进

var chopstick : array[0..4] of semaphore;初值为1

room : semaphore ;初值为4

各个哲学家执行的程序为：

P(room);

P(chopstick[i]); //取左边筷子

P(chopstick[(i+1) mod 5]); //取右边筷子

进食

V(chopstick[(i+1) mod 5]); //放下右边筷子

V(chopstick[i]); //放下左边筷子

V(room)

思考

改进方案3:

使用非对称解决：奇数哲学家先拿起左边筷子，再拿右边筷子；偶数哲学家先拿起右边筷子，再拿左边筷子；

信号量使用注意事项

- 进程应该先申请资源信号量，再申请互斥信号量，顺序不能颠倒。
- 对任何信号量的P/V操作必须配对使用。同一进程中的多对P/V语句只能嵌套，不能交叉。
- 对同一信号量的P/V操作可以不在同一进程中。
- P/V语句不能颠倒顺序

举例

寄信

信箱问题：

一次只能放一封信

信箱被多对人使用

3 个信号量

吃水果

信箱不只可以放一封信的信箱问题