第22 - 23讲 信号量及生产者/消费者、读者/写者问题

Approaches of Mutual Exclusion

- *Software Approaches*
- Hardware Support
- ***Semaphores***
- Monitors
- Message Passing

Semaphores

- Special variable called a semaphore is used for signaling.
- If a process is waiting for a signal, it is blocked until that signal is sent.
- Wait and Signal operations cannot be interrupted.
- Queue is used to hold processes waiting on the semaphore.

Semaphores

Semaphore is a variable that has an integer value.

- **May be initialized to a nonnegative number .**
- **Wait and Signal are primitives (atomic, cannot be interrupted and each routine can be treated as an indivisible step).**

Semaphores

- **Wait operation decrements the semaphore value**
  - *wait(s)： s － 1*
  - wait 操作：*申请资源且可能阻塞自己* (s ＜ 0)
- **Signal operation increments semaphore value**
  - *signal(s)： s ＋ 1*
  - signal 操作：*释放资源并唤醒阻塞进程* (s≤0)

Types of Semaphores

- General Semaphore (通用信号量)
  - 通用信号量是*记录型*，其中一个域为*整型*，另一个域为*队列*，其元素为等待该信号量的阻塞进程（ FIFO ）
- Binary Semaphore (二进制信号量)

General Semaphore

```
type semaphore = record
count : integer;
queue : list of process
end;

var s: semaphore;
```

wait/signal Operations

```
type semaphore = record
count : integer;
queue : list of process
end;
var s: semaphore;
```

```
wait(s):
s.count := s.count – 1;
if s.count < 0
      then begin
      block P;
      insert P into s.queue
;
       end;
```

```
signal(s):
s.count:= s.count + 1;
if s.count ≤ 0
     then begin
     wakeup the first P ;
     remove the P from
    s.queue;
     end ;
```

```
program mutualexclusion;
const n=…;      /*  进程数  */
var  s: semaphore(:= 1); /*  定义信号量 s ，  s.count 初始化为 1 */
procedure P(i:integer);
begin
  repeat
      wait(s);
      < 临界区 >;
      signal(s);
      < 其余部分 >
   forever
end;
```

```
begin    /*  主程序  */
parbegin
P(1);    P(2);    …    P(n)
 parend
end.
```

图 2.34　利用信号量实现互斥的通用模式

信号量的类型

- 信号量分为：互斥信号量和资源信号量。
- 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个数。
- wait 操作用于申请资源（或使用权），进程执行 wait 原语时，可能会阻塞自己；
- signal 操作用于释放资源（或归还资源使用权），进程执行 signal 原语时，有责任唤醒一个阻塞进程。

Mutual Exclusion: Semaphores

信号量的意义

1. ***互斥信号量***：申请 / 释放使用权，常初始化为 1
2. ***资源信号量***：申请 / 归还资源，资源信号量可以初始化为一个正整数（表示系统中某类资源的可用个数）， s.count 的意义为：

   s.count≥0 ： 表示还可执行 wait(s) 而不会阻塞的进程数（可用资源数）

   s.count<0 ： 表示 s.queue 队列中阻塞进程的个数（被阻塞进程数）

s.count 的取值范围

- 当仅有两个并发进程共享临界资源时，互斥信号量仅能取值 0 、 1 、 － 1 。

  其中，

  - s.count=1, 表示无进程进入临界区
  - s.count=0 ， 表示已有一个进程进入临界区
  - s.count= - 1, 则表示已有一进程正在等待进入临界区

- 当用 s 来实现 n 个进程的互斥时， s.count 的取值范围为 1 ～ －（ n-1 ）

- 操作系统内核以系统调用形式提供 wait 和 signal 原语，应用程序通过该系统调用实现进程间的互斥。

- 工程实践证明，利用信号量方法实现进程互斥是高效的，一直被广泛采用。

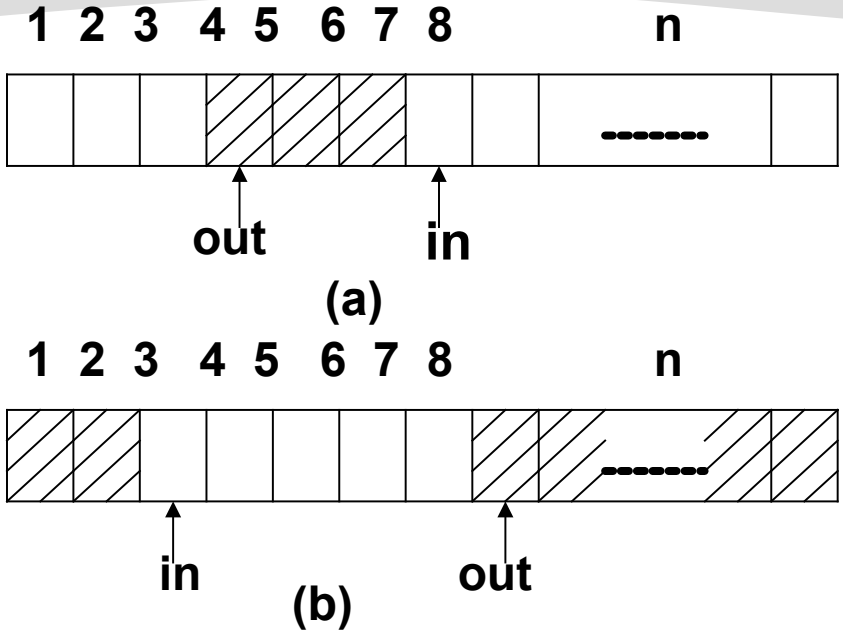
Producer/Consumer Problem

- One or more producers are generating data and placing data in a buffer.
- A single consumer is taking items out of the buffer one at time.
- Only one producer or consumer may access the buffer at any one time.

Producer/Consumer Problem

任务及要求

1. buffer 不能并行操作（互斥），即某时刻只允许一个实体（ producer or consumer ）访问 buffer.
2. 控制 producer and consumer 同步读 / 写 buffer，即不能向满 buffer 写数据；不能在空 buffer 中取数据 .



其中， in 表示存数据位置， out 表示取数据位置

单元 ：被占用单元 ， ：空存储

图 2.35　生产者 / 消费者循环使用缓冲区

？ 不控制生产者 / 消费者

- 指针 in 和 out 初始化指向缓冲区的第一个存储单元。
- 生产者通过 in 指针向存储单元存放数据，一次存放一条数据，且 in 指针向后移一个位置。
- 消费者从缓冲区中逐条取走数据，一次取一条数据，相应的存储单元变为“空”。每取走一条数据， out 指针向后移一个存储单元位置。
- 试想，如果不控制生产者与消费者，将会产生什么结果？

生产者/消费者必须互斥

- 生产者和消费者可能同时进入缓冲区，甚至可能同时读/写一个存储单元，将导致执行结果不确定。

- 这显然是不允许的。必须使生产者和消费者***互斥***进入缓冲区。即，某时刻只允许一个实体（生产者或消费者）访问缓冲区，生产者互斥消费者和其它任何生产者。

生产者/消费者必须同步

生产者不能向满缓冲区写数据，消费者也不能在空缓冲区中取数据，即生产者与消费者必须*同步*。

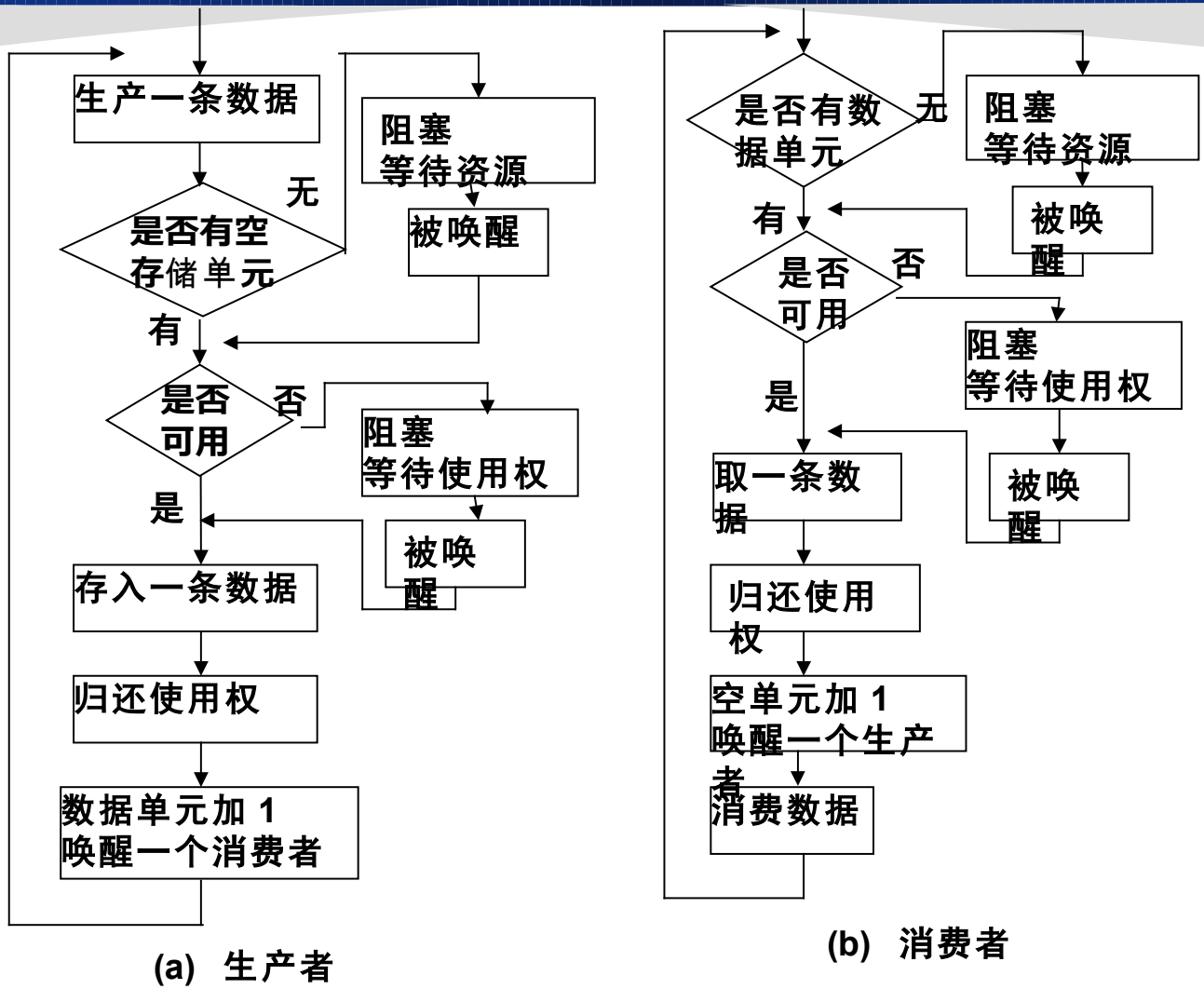


图 2.36 生产者 / 消费者执行流程图

利用信号量实现生产者 / 消费者同步与互斥

```
program producer_consumer;
const sizeofbuffer =…;  /*  缓冲区大小    */
var s: semaphore(:= 1); /*  互斥信号量 s，初始化为 1 */
    n : semaphore(:= 0); /*  资源信号量 n，数据单元，初始化为 0 */
    e : semaphore(:= sizeofbuffer); /*  资源信号量 e，空存储单元   */
procedure producer ;
begin
  repeat
        生产一条数据；
    wait(e);
    wait(s);
        存入一条数据；
    signal(s);
    signal(n) ；
消费数据；
  forever
 end;
```

```
procedure consumer ;
        begin
 repeat
            wait(n);
   wait(s);
    取一条数据；
            signal(s);
   signal(e);

  forever
end;
```

利用信号量实现生产者 / 消费者同步与互斥（续）

```
begin    /*  主程序  */
          parbegin
           producer ; consumer ;
          parend
end.
```

图 2.37　利用信号量实现生产者 / 消费者同步与互斥

注意

1. 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。
2. 对任何信号量的 wait 与 signal 操作必须配对。同一进程中的多对 wait 与 signal 语句只能嵌套，不能交叉。
3. 对同一个信号量的 wait 与 signal 可以不在同一个进程中。
4. wait 与 signal 语句不能颠倒顺序， wait 语句一定先于 signal 语句。

Readers/Writers Problem

- Any number of readers may simultaneously read the file.
- Only one writer at a time may write to the file.
- If a writer is writing to the file, no reader may read it.

Readers/Writers Problem

可用于解决多个进程共享一个数据区（文件、内存区、一组寄存器等），其中若干读进程只能读数据，若干写进程只能写数据等实际问题。

Readers/Writers Problem （Readers have priority）

- **Readers have priority:** 指一旦有读者正在读数据，允许多个读者同时进入读数据，只有当全部读者退出，才允许写者进入写数据。
  - **Writers are subject to starvation**

Readers/Writers Problem （Readers have priority）

```
program  readers_writers;
const readcount: integer;   /*  统计读者个数    */
var x,wsem: semaphore(:= 1); /*  互斥信号量，初始化为 1 */
```

```
procedure reader ;
begin
   repeat
    wait(x);
    readcount : = readcount + 1;

    if readcount = 1 then wait (wsem);

   signal(x);

       读数据；

   wait(x);
    readcount : = readcount - 1;
    if readcount = 0 then signal(wsem);
```

```
procedure writer;
begin
    repeat
     wait(wsem);
          写数据；
     signal(wsem);
    forever
end;
```

```
begin    /*  主程序  */
          readcount := 0;
          parbegin
          reader; writer ;
          parend
end.
```

Readers/Writers Problem （Writers have priority）

Writers have priority: 指只要有一个 writer 申请写数据，则不再允许新的 reader 进入读数据

```
program  readers_writers;
const readcount ,writecount: integer;
var x,y,z,rsem,wsem: semaphore(:= 1);
procedure reader ;
begin
  repeat
    wait(z);
    wait(rsem);
    wait(x);
    readcount : = readcount + 1;
    if readcount = 1 then wait (wsem);
   signal(x);
   signal(rsem);
   signal(z);
        读数据；

   wait(x);
    readcount : = readcount - 1;
    if readcount = 0 then signal(wsem);
    signal(x);
 forever
 end;
```

```
procedure writer;
begin
   repeat
      wait(y);
       writecount := writecount + 1;
        if writecount = 1 then wait (rsem);
        signal(y);
        wait(wsem);
                写数据；
        signal(wsem);
        wait(y);
        writecount := writecount - 1;
         if writecount = 0 then signal (rsem);
        signal(y);
      forever
end;
```

Readers/Writers Problem （Writers have priority）

```
begin    /* 主程序  */
          readcount := 0;  writecount := 0;
          parbegin
          reader; writer ;
          parend
end.
```

Summary of Mutual Exclusion

1. 利用 wait 、 signal 原语对 Semaphore 互斥操作实现 ,powerful and flexible
2. 可用软件方法实现互斥，如 Dekker 算法、 Peterson 算法等（但增加处理负荷）
3. 可用硬件或固件方法实现互斥，如屏蔽中断、 Test and Set 指令等 （属于可接受的忙等）