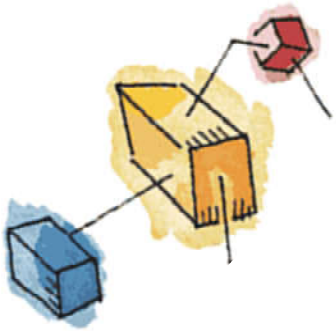
به نام خدا

فصل پنجم

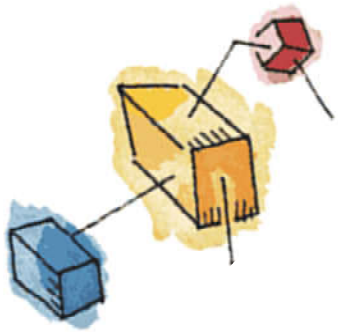
همروندی:
انحصار متقابل و همگام سازی
(بخش سوم)

Concurrency: Mutual Exclusion and Synchronization

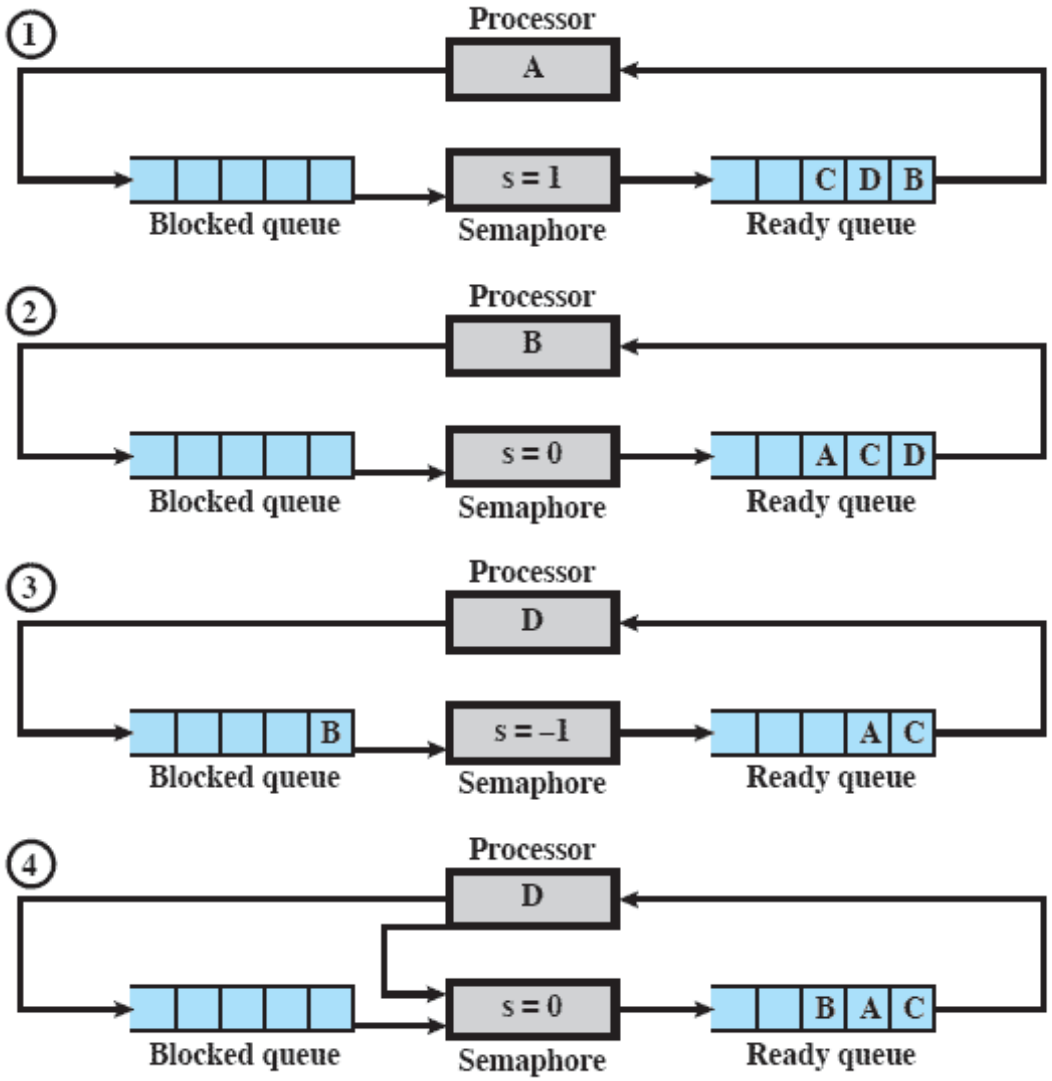


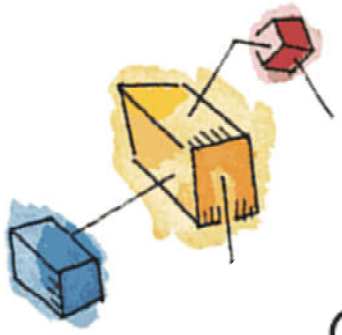
مثال دیگری از کاربرد سمافور

- مثالی از عملکرد یک سمافور قوی
- در این مثال، فرآیندهای A و B و C به نتایجی که توسط فرآیند D به دست می آید نیاز دارند.



ادامه مثال سمافور





ادامه مثال سمافور

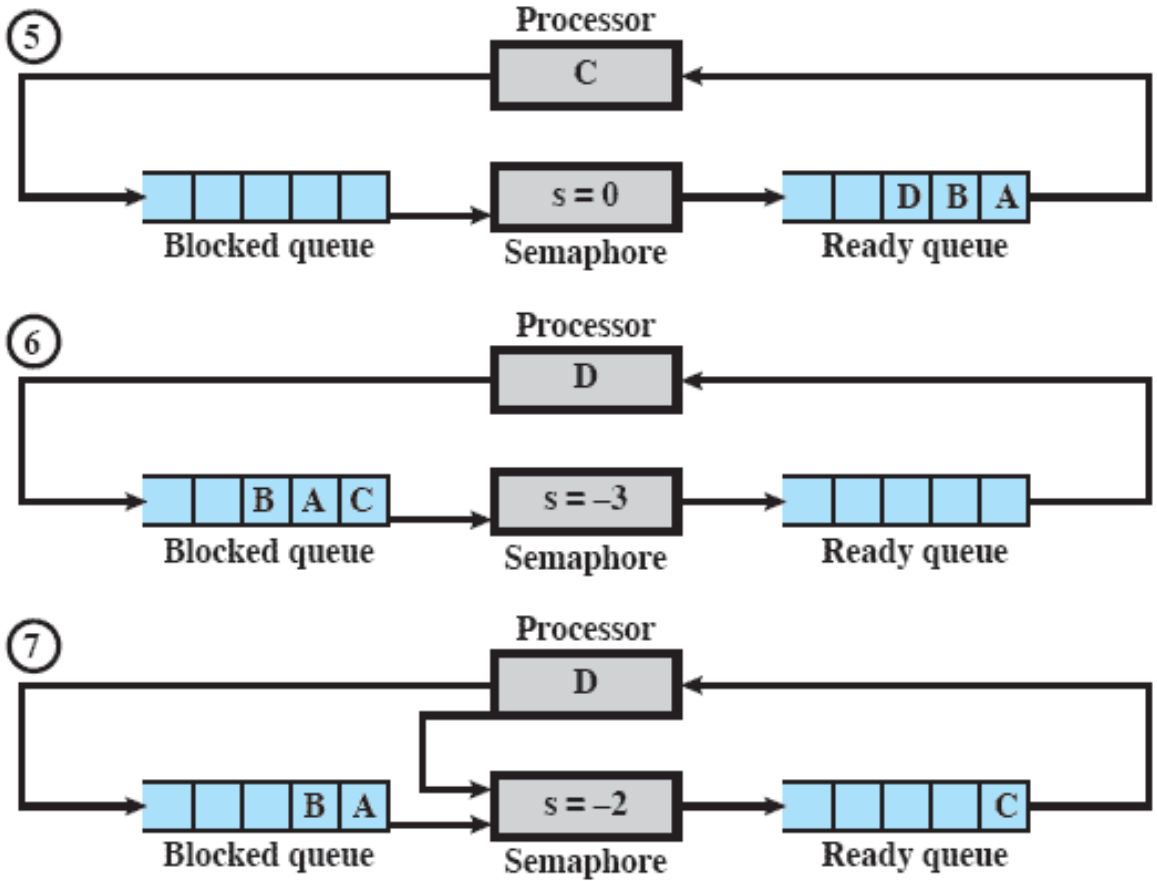
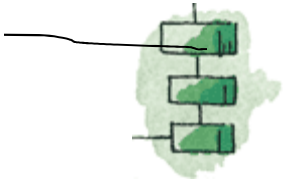
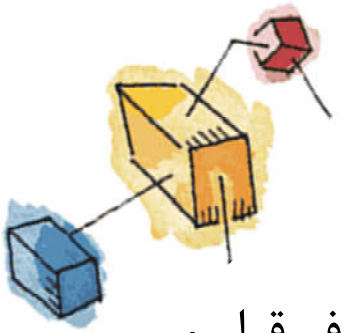


Figure 5.5 Example of Semaphore Mechanism





مساله تولید کننده و مصرف کننده
Producer-Consumer Problem

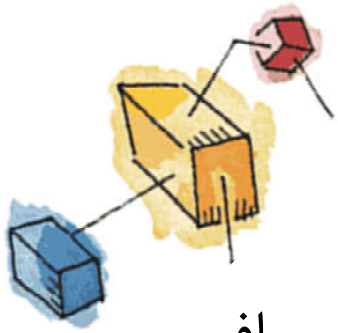
- یک یا چند تولید کننده نوعی داده را تولید می کنند و بر روی یک بافر قرار می دهند.
- یک مصرف کننده این اقلام را یک به یک از بافر بر می دارد و مصرف می کند.
- در هر زمان تنها یک تولیدکننده یا مصرف کننده می تواند به بافر دسترسی داشته باشد.

- مساله این است که تولید کننده نتواند داده را در بافر پر قرار دهد و مصرف کننده نیز نتواند داده را از بافر خالی بردارد.

```
/* program producerconsumer */
int n;
binary_semaphore s = 1, delay = 0;
void producer()
{
     while (true) {
          produce();
          semWaitB(s);
          append();
          n++;
          if (n==1) semSignalB(delay);
          semSignalB(s);
     }
}
void consumer()
{
     semWaitB(delay);
     while (true) {
          semWaitB(s);
          take();
          n--;
          semSignalB(s);
          consume();
          if (n==0) semWaitB(delay);
     }
}
void main()
{
     n = 0;
     parbegin (producer, consumer);
}
```

یک راه حل برای بافر نامحدود با استفاده از سمافورهای دودویی

این راه حل نادرست است.



عیب برنامه

- وقتی مصرف کننده بافر را تمام کرد، نیاز به مقداردهی مجدد سمافور delay دارد تا مجبور باشد تا زمان گذاشتن اقلام دیگر در بافر صبر کند.
  - در غیراینصورت ممکن است عنصری از بافر مصرف شود که وجود ندارد.

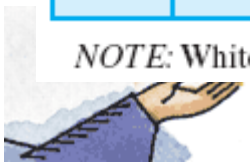
- ولی اگر اجرای دستورات تولید کننده و مصرف کننده، به صورت جدول اسلاید بعد رخ دهد، این انتظار مصرف کننده برروی سمافور delay تا پر شدن مجدد بافر، به صورت مناسبی انجام نمی شود.
  - باعث می شود که عنصری از بافر مصرف شود که وجود ندارد.

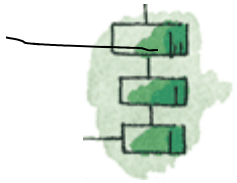
یک سناریوی ممکن

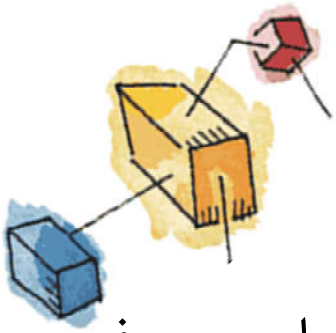
Table 5.4 Possible Scenario for the Program of Figure 5.9

	Producer	Consumer	s	n	Delay
1			1	0	0
2	semWaitB(s)		0	0	0
3	n++		0	1	0
4	**if** (n==1) (semSignalB(delay))		0	1	1
5	semSignalB(s)		1	1	1
6		semWaitB(delay)	1	1	0
7		semWaitB(s)	0	1	0
8		n--	0	0	0
9		semSignalB(s)	1	0	0
10	semWaitB(s)		0	0	0
11	n++		0	1	0
12	**if** (n==1) (semSignalB(delay))		0	1	1
13	semSignalB(s)		1	1	1
14		**if** (n==0) (semWaitB(delay))	1	1	1
15		semWaitB(s)	0	1	1
16		n--	0	0	1
17		semSignalB(s)	1	0	1
18		**if** (n==0) (semWaitB(delay))	1	0	0
19		semWaitB(s)	0	0	0
20		n--	0	–1	0
21		semiSignlaB(s)	1	–1	0

NOTE: White areas represent the critical section controlled by semaphore s.







راه حل برای عیب برنامه

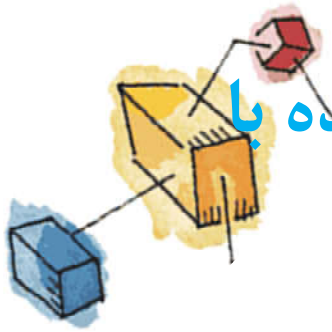
- به نظر می رسد که برای حل مشکل می توان دستور شرطی انتهای مصرف کننده را داخل ناحیه بحرانی آن برد. ولی این راه مناسبی نیست زیرا ممکن است منجر به بن بست شود.

- یک راه مناسب این است که **متغیر کمکی** معرفی کنیم که برای استفاده آتی در بخش بحرانی مصرف کننده مقدارگذاری شود.

- راه مناسب دیگر، استفاده از **سمافورهای شمارشی (سمافورهای عمومی)** است.

```
/* program producerconsumer */
int n;
binary_semaphore s = 1, delay = 0;
void producer()
{
     while (true) {
          produce();
          semWaitB(s);
          append();
          n++;
          if (n==1) semSignalB(delay);
          semSignalB(s);
     }
}
void consumer()
{
     int m; /* a local variable */
     semWaitB(delay);
     while (true)  {
          semWaitB(s);
          take();
          n--;
          m = n;
          semSignalB(s);
          consume();
          if (m==0) semWaitB(delay);
     }
}
void main()
{
     n = 0;
     parbegin (producer, consumer);
}
```

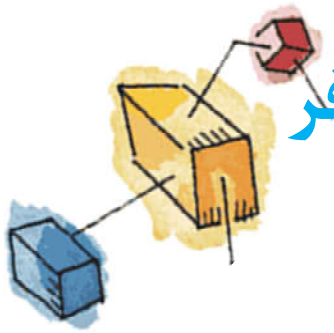
یک راه حل درست
برای بافر نامحدود
با استفاده از
سمافورهای
دودویی
(استفاده از
متغیر کمکی)



راه حلی دیگر برای مساله تولیدکننده و مصرف کننده با بافر نامحدود (استفاده از سمافورهای شمارشی)

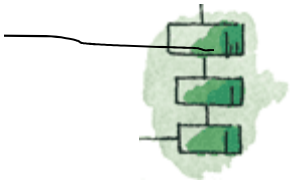
```
/* program  producerconsumer */
semaphore n = 0, s = 1;
void producer()
{
     while (true) {
          produce();
          semWait(s);
          append();
          semSignal(s);
          semSignal(n);
     }
}
void consumer()
{
     while (true) {
          semWait(n);
          semWait(s);
          take();
          semSignal(s);
          consume();
     }
}
void main()
{
     parbegin (producer, consumer);
}
```

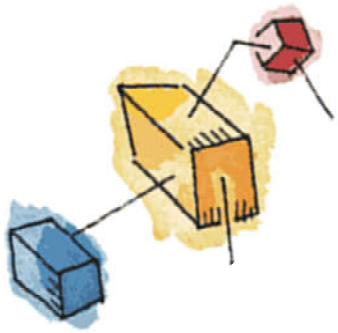

Figure 5.11 A Solution to the Infinite-Buffer Producer/Consumer Problem Using Semaphores



راه حلی برای مساله تولیدکننده و مصرف کننده با بافر محدود (استفاده از سمافورهای شمارشی)

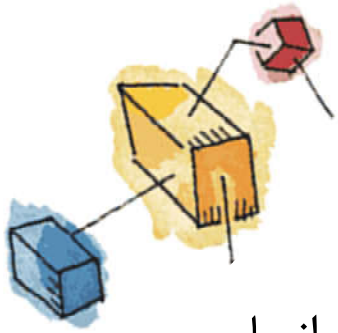
```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1, n= 0, e= sizeofbuffer;
void producer()
{
     while (true) {
          produce();
          semWait(e);
          semWait(s);
          append();
          semSignal(s);
          semSignal(n);
     }
}
void consumer()
{
     while (true) {
          semWait(n);
          semWait(s);
          take();
          semSignal(s);
          semSignal(e);
          consume();
     }
}
void main()
{
     parbegin (producer, consumer);
}
```





سرفصل مطالب

- اصول همروندی (همزمانی)
- انحصار متقابل: حمایت سخت افزار
- راهنماها (سمافورها)
- ناظرها (مانیتورها)
- تبادل پیام
- مسائل کلاسیک همزمانی

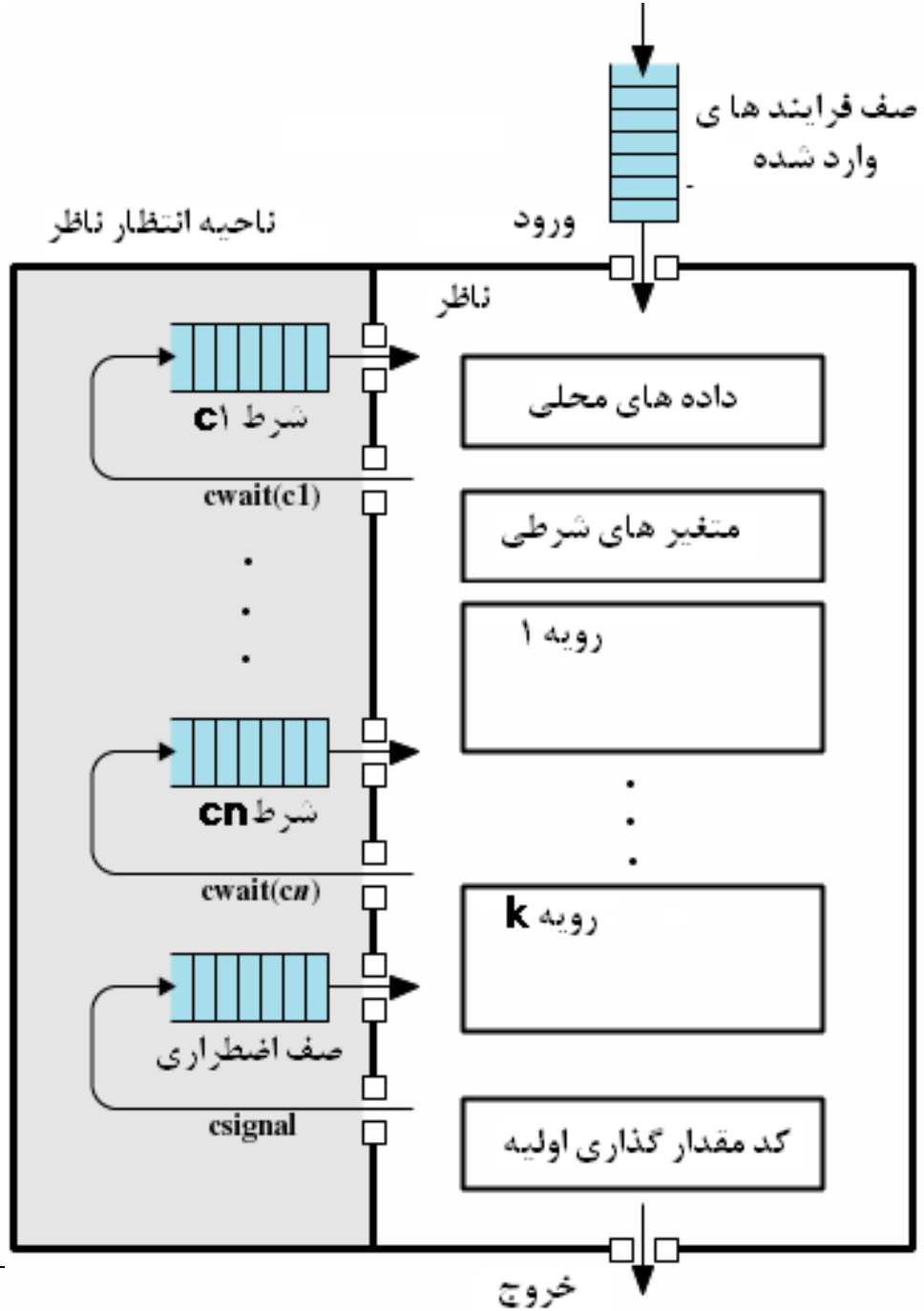


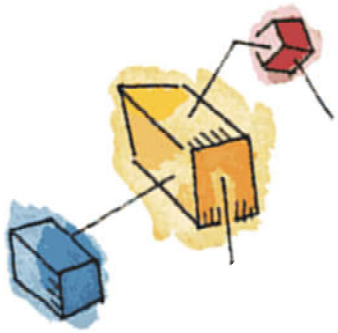
ناظرها (Monitors)

- ناظر ساختاری از زبان برنامه نویسی است که همان کار سمافورها را انجام می دهد ولی کنترل آن ساده تر است.
- راه حلی در سطح زبان برنامه نویسی برای تامین انحصار متقابل.

- مجموعه ای از رویه ها، متغیرها، و ساختمان داده ها می باشد که همگی در یک ماژول نرم افزاری خاص قرار گرفته اند.
- مهم ترین خصوصیات ناظر:
  - متغیرهای محلی تنها توسط ناظر قابل دسترسی می باشند.
  - یک فرآیند با احضار یکی از رویه های ناظر، وارد آن می شود.
  - در هر زمان تنها یک فرآیند می تواند در ناظر در حال اجرا باشد.

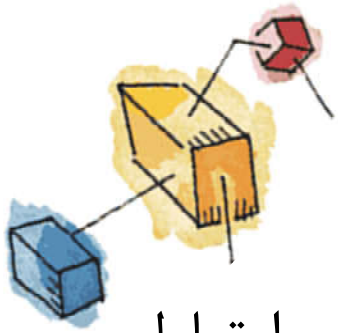
ساختار ناظر


صف فرایند ها ی
وارد شده
ورود
ناحیه انتظار ناظر
ناظر
داده های محلی
شرط c1
cwait(c1)
متغیر های شرطی
رویه ۱
شرط cn
cwait(cn)
رویه k
صف اضطراری
csignal
کد مقدار گذاری اولیه
خروج



سرفصل مطالب

- اصول همروندی (همزمانی)
- انحصار متقابل: حمایت سخت افزار
- راهنماها (سمافورها)
- ناظرها (مانیتورها)
- تبادل پیام
- مسائل کلاسیک همزمانی

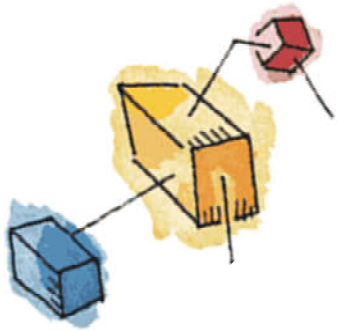


تبادل پیام

- پیام ها یک مکانیزم ساده و مناسب جهت همگام سازی و ارتباط دهی بین فرآیندها در یک محیط غیرمتمرکز و توزیع شده اند.
- بسیاری از سیستم عامل های چند برنامگی از نوعی پیام های بین فرآیندها پشتیبانی می کنند.
- پیام مجموعه ای از اطلاعات است که بین فرآیندهای ارسال کننده و دریافت کننده مبادله می شود.

```
send (destination, message)

receive (source, message)
```

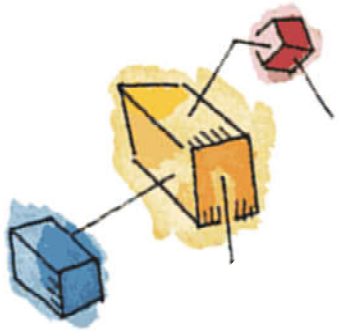



همگام سازی

- تبادل پیام بین دو فرآیند نیازمند سطحی از همگام سازی است:
- تا زمانی که پیامی توسط فرآیندی فرستاده نشده باشد، دریافت کننده نمی تواند آن را دریافت کند.

- بعد از اینکه فرآیندی send یا receive را صادر کرد چه اتفاقی برایش می افتد؟
  - ارسال کننده و دریافت کننده ممکن است **مسدود شونده** (منتظر پیام) باشند یا نباشند.

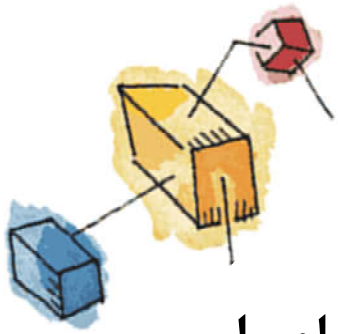
- **سه ترکیب ممکن وجود دارد:**
- **۱. ارسال مسدود شونده، دریافت مسدود شونده**
  - هم فرستنده و هم گیرنده تا زمانی که پیام تحویل داده شود مسدودند.
  - گاهی به آن **قرار ملاقات** (rendezvous) هم گفته می شود.
  - این ترکیب همگام سازی محکم بین فرآیندها را میسر می کند.



همگام سازی

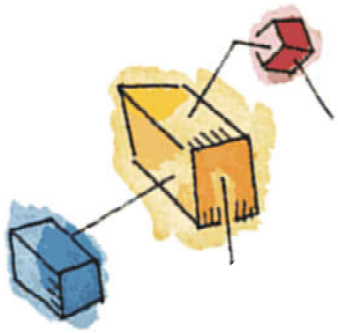
- **۲. ارسال غیر مسدود شونده، دریافت مسدود شونده**
  - فرستنده به کار خود ادامه می دهد.
  - گیرنده تا زمان رسیدن پیام درخواستی مسدود است.
  - این مفیدترین ترکیب است، چرا که اجازه می دهد یک یا چند پیام در اسرع وقت به مقصدهای متنوع ارسال شود.

- **۳. ارسال غیر مسدود شونده، دریافت غیر مسدود شونده**
  - هیچ یک از دو طرف منتظر تحویل پیام نمی مانند.



آدرس دهی

- در فرآیند ارسال نیاز است مشخص شود که کدام فرآیند باید پیام را دریافت کند.
  - آدرس دهی مستقیم
  - آدرس دهی غیر مستقیم



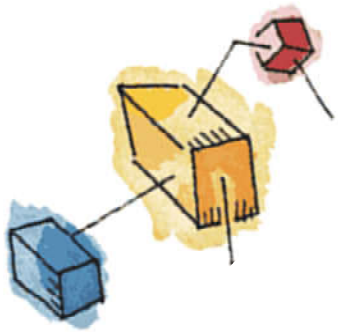
آدرس دهی

- **آدرس دهی مستقیم**

– هر دو طرف یکدیگر را شناخته و آدرس (شناسه) فرآیند دریافت کننده یا ارسال کننده بطور صریح مشخص می شود.

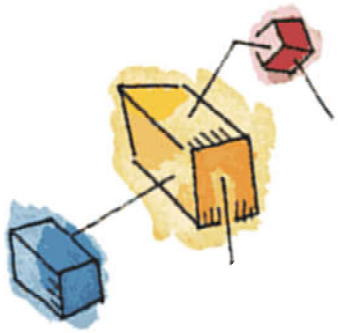
– ارسال کننده، شناسه (آدرس) فرآیند دریافت کننده را دارا بوده و در هنگام ارسال، دریافت کننده را بطور صریح مشخص می کند.

– دریافت کننده از فرآیند مشخصی انتظار دریافت پیام را دارد (شناسه آن فرآیند را صریحا مشخص می کند).



آدرس دهی

- **آدرس دهی غیرمستقیم**
  - پیام ها مستقیماً از فرستنده به گیرنده ارسال نمی شوند. بلکه به یک ساختمان داده اشتراکی ارسال می شوند که شامل صف هایی برای نگهداری پیام ها است.
  - این صف ها معمولا **صندوق پستی** نامیده می شوند.
  - یک فرآیند پیام را به صندوق پستی ارسال نموده و فرآیند دیگر پیام را از صندوق پستی بر می دارد.

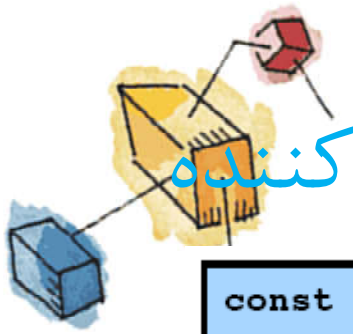


انحصار متقابل با استفاده از پیام ها

- استفاده از صندوق پستی (آدرس دهی غیرمستقیم)
- فرآیند گیرنده مسدود شونده است.

```
/* program mutualexclusion */
const int n = /* number of processes  */;
void P(int i)
{
    message msg;
    while (true) {
     receive (box, msg);
     /* critical section   */;
     send (box, msg);
     /* remainder    */;
    }
}
void main()
{
    create mailbox (box);
    send (box, null);
    parbegin (P(1), P(2), . . ., P(n));
}
```

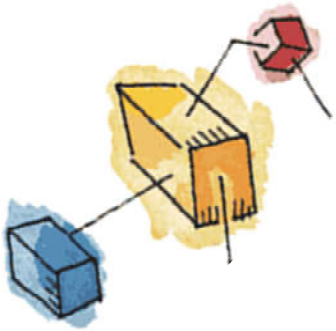

Figure 5.20 Mutual Exclusion Using Messages



پیام های مربوط به مساله تولیدکننده/مصرف کننده

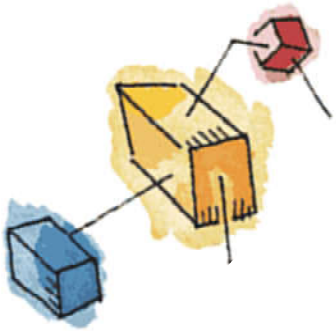
```
const int
    capacity = /* buffering capacity */ ;
    null =/* empty message */ ;
int i;
void producer()
{   message pmsg;
    while (true) {
      receive (mayproduce, pmsg);
      pmsg = produce();
      send (mayconsume, pmsg);
    }
}
void consumer()
{   message cmsg;
    while (true) {
      receive (mayconsume, cmsg);
      consume (cmsg);
      send (mayproduce, null);
    }
}

void main()
{
    create_mailbox (mayproduce);
    create_mailbox (mayconsume);
    for (int i = 1; i <= capacity; i++) send (mayproduce, null);
    parbegin (producer, consumer);
}
```



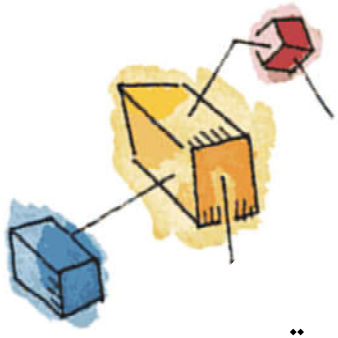
سرفصل مطالب

- اصول همروندی (همزمانی)
- انحصار متقابل: حمایت سخت افزار
- راهنماها (سمافورها)
- ناظرها (مانیتورها)
- تبادل پیام
- مسائل کلاسیک همزمانی



مسائل کلاسیک همزمانی

- مساله خوانندگان و نویسندگان
- مساله شام خوردن فیلسوف ها



مسالهٔ خوانندگان و نویسندگان (Readers/Writers Problem)

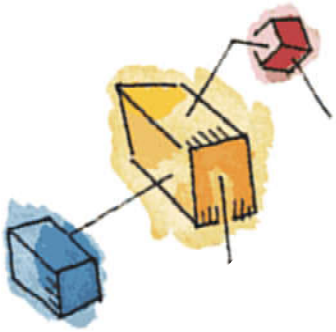
- یک ناحیه داده در بین تعدادی فرآیند به اشتراک گذاشته شده است.
- بعضی از فرآیندها داده ها را فقط می خوانند (خوانندگان) و برخی دیگر در ناحیه داده فقط می نویسند (نویسندگان).

- شرایط زیر باید برقرار باشد:
  - هر تعدادی از خوانندگان می توانند همزمان پرونده را بخوانند.
  - در هر زمان فقط یک نویسنده می تواند در پرونده بنویسد.
  - هرگاه یک نویسنده در حال نوشتن بر روی پرونده است، هیچ خواننده ای اجازه خواندن آن را ندارد.

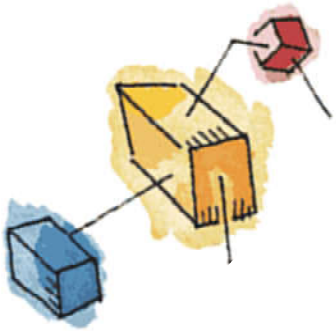
```
/* program readersandwriters */
int readcount;
semaphore x = 1, wsem = 1;
void reader()
{
    while (true) {
      semWait (x);
      readcount++;
      if (readcount == 1) semWait (wsem);
      semSignal (x);
      READUNIT();
      semWait (x);
      readcount--;
      if (readcount == 0) semSignal (wsem);
      semSignal (x);
    }
 }
void writer()
{
    while (true) {
      semWait (wsem);
      WRITEUNIT();
      semSignal (wsem);
    }
}

void main()
{
    readcount = 0;
    parbegin (reader, writer);
}
```

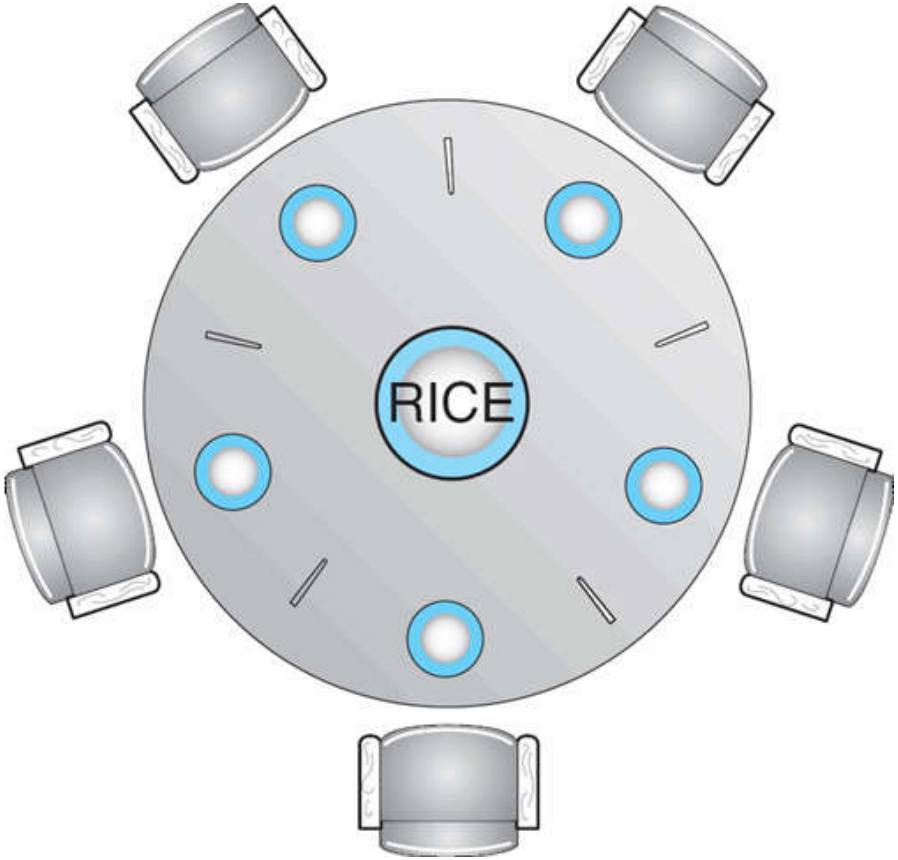
راه حلی برای مساله
خوانندگان و نویسندگان
با استفاده از سمافورها،
در شرایطی که
خوانندگان اولویت دارند



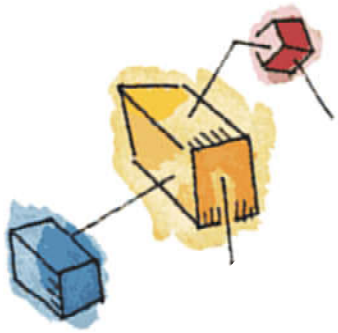
- یک مساله کلاسیک دیگر



مساله غذا خوردن فیلسوفان



- سمافور:
- آرایه chopstick[5] که با مقدارهای ۱ مقداردهی اولیه شده است.



مساله غذا خوردن فیلسوفان

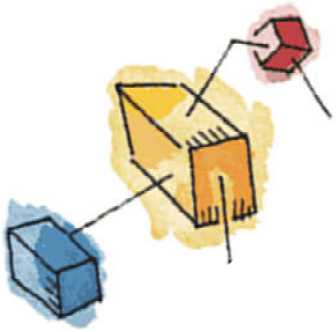
- ساختار فیلسوف i ام

```
do  {
        wait ( chopstick[i] );
        wait ( chopstick[ (i + 1) % 5] );

            //  eat

        signal ( chopstick[i] );
        signal (chopstick[ (i + 1) % 5] );

            //  think
} while (TRUE);
```



پایان فصل پنجم