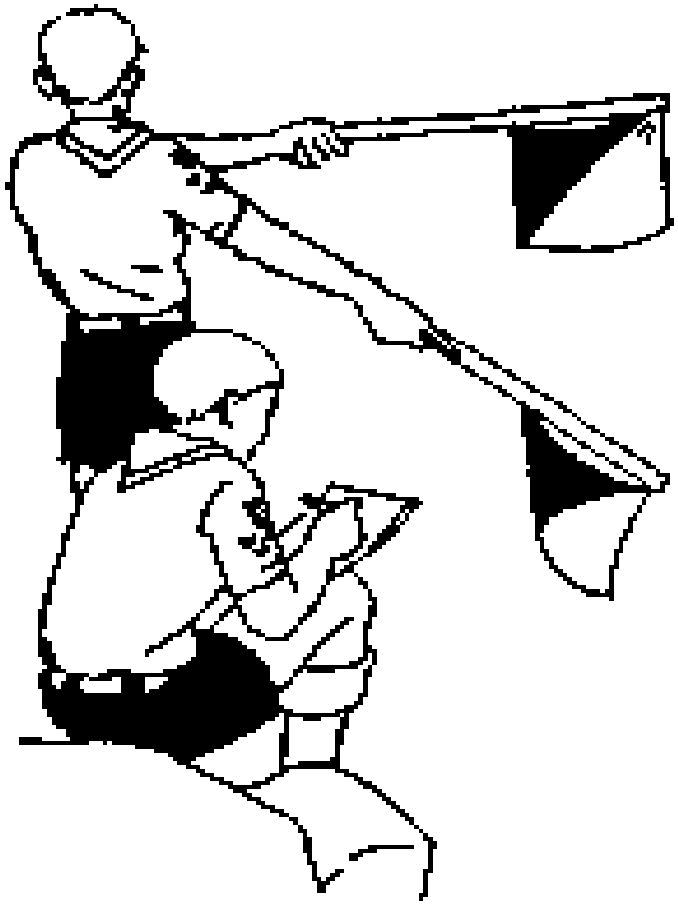
راهنما ها (Semaphore):

- راهنماها یک سیستم ارتباطی هستند که از پرچم ها استفاده میکنند. دو ایستگاه کاری فرستنده وگیرنده وجود دارد که این دو باید باشند به وضوح همدیگر را ببینند.

- معمولا برای مسافت های طولانی تعدادی ایستگاه تکرار کننده بین فرستنده و گیرنده وجود دارد.

راهنما ها در کاربرد :



- زمانی که یک پیام راهنما فرستاده میشود، دو ایستگاه وجود دارد : فرستنده و گیرنده.هر ایستگاه از دو نفر تشکیل شده : علامت دهند و دستیار. علامت دهنده پرچم ها را نگه میدارد و دستیار پیام ها را ضبط میکند و نتیجه را به علام دهنده میدهد.

راهنما ها :

- راهنما، مکانیسیمی است که از دسترسی دو یا چند فرایند به منابع مشترک به طور همزمان جلوگیری میکند. به عنوان مثال در یک راه آهن راهنما از برخورد قطار ها با هم در یک ریل مشترک جلوگیری میکند. در صورتیکه به راهنمها توجهی نشود تضمینی وجود ندارد که قطار ها با هم برخورد نکنند به طور مشابه درکامپیوتر در صورت عدم توجه به راهنما احتمال اغتشاش فرایندها وجود دارد.

راهنما ها :

- راهنما ها مانند پرچم ها عمل میکنند به همین دلیل به آنها راهنما گفته میشود. یک راهنما میتواند On یا Off باشد.یک فرایند میتواند یک پرچم را On یا Off کند. اگر پرچم On باشد و فرایندی بکوشد تا آنرا On کند، آن فرایند تا زمانی که پرچه Off شود منتظر می ماند.

راهنما ها :نرم افزار

- در سال ۱۹۶۵ دیجسترا راهنما ها را به عنوان راه حلی برای فرایند های همزمان در نظر گرفت.

- اصل بنیادی این بود: دو یا چند فرایند میتوانند با علامت های ساده با یکدیگر همکاری کنند. هنگامی که یک فرایند در انتظار یک علامت از طرف فرایند دیگر است، آن فرایند تا رسیدن آن علامت در حالت معلق است.

- برای علامت دهی از متغیر های ویژه ای به نام راهنما استفاده شد

عملیات روی راهنما:

- یک راهنما میتواند با یک مقدار غیر منفی صحیح مقدار دهی اولیه شود.
- عمل Wait مقدار راهنما را یک واحد کاهش میدهد. اگر مقدار منفی شود آنگاه فرایندی که دستور Wait را اجرا کرده مسدود میشود.
- عمل Signal مقدار راهنما را یک واحد افزایش میدهد. اگر مقدار مثبت نباشد آنگاه فرایندی که توسط دستور Wait مسدود شده بود آزاد میگردد.
  - غیر این ۳ عمل راه دیگری برای دستکاری سمافور وجود ندارد.

پیاده سازی راهنما:

- سمافورS یک متغیر صحیح است.
- تنها توسط P(s) و V(s) میتواند در دسترس قرار گیرد.

```
wait (S)
{           /* P operation */
            while (S≤ 0);       /* no-op */;
            S--;
}

signal (S)
{           /* V operation */
            S++;
}
```

تعریف اولیه های راهنما:

```
struct semaphore {
      int count;
      queueType queue;
}

void semWait(semaphore s)
{
      s.count--;
      if (s.count < 0)
      {
            place this process in s.queue;
            block this process
      }
}
void semSignal(semaphore s)
{
      s.count++;
      if (s.count <= 0)
      {
            remove a process P from s.queue;
            place process P on ready list;
      }
}
```

تعریف اولیه های راهنماهای دودویی

```
struct binary_semaphore {
      enum {zero, one} value;
      queueType queue;
};

void semWaitB(binary_semaphore s)
{
      if (s.value ==  1)
            s.value = 0;
      else
            {
                  place this process in s.queue;
                  block this process;
            }
}
void semSignalB(semaphore s)
{
      if (s.queue.is_empty())
            s.value = 1;
      else
      {
            remove a process P from s.queue;
            place process P on ready list;
      }
}
```

انحصار متقابل با استفاده از سمافور:

```
/* program mutualexclusion */
const int n = /* number of processes  */;
semaphore s = 1;
void P(int i)
{
     while (true)
     {
          semWait(s);
          /* critical section   */;
          semSignal(s);
          /* remainder    */;
     }
}
void main()
{
     parbegin (P(1), P(2), . . ., P(n));
}
```

مثالی از مکانیسم سمافور:

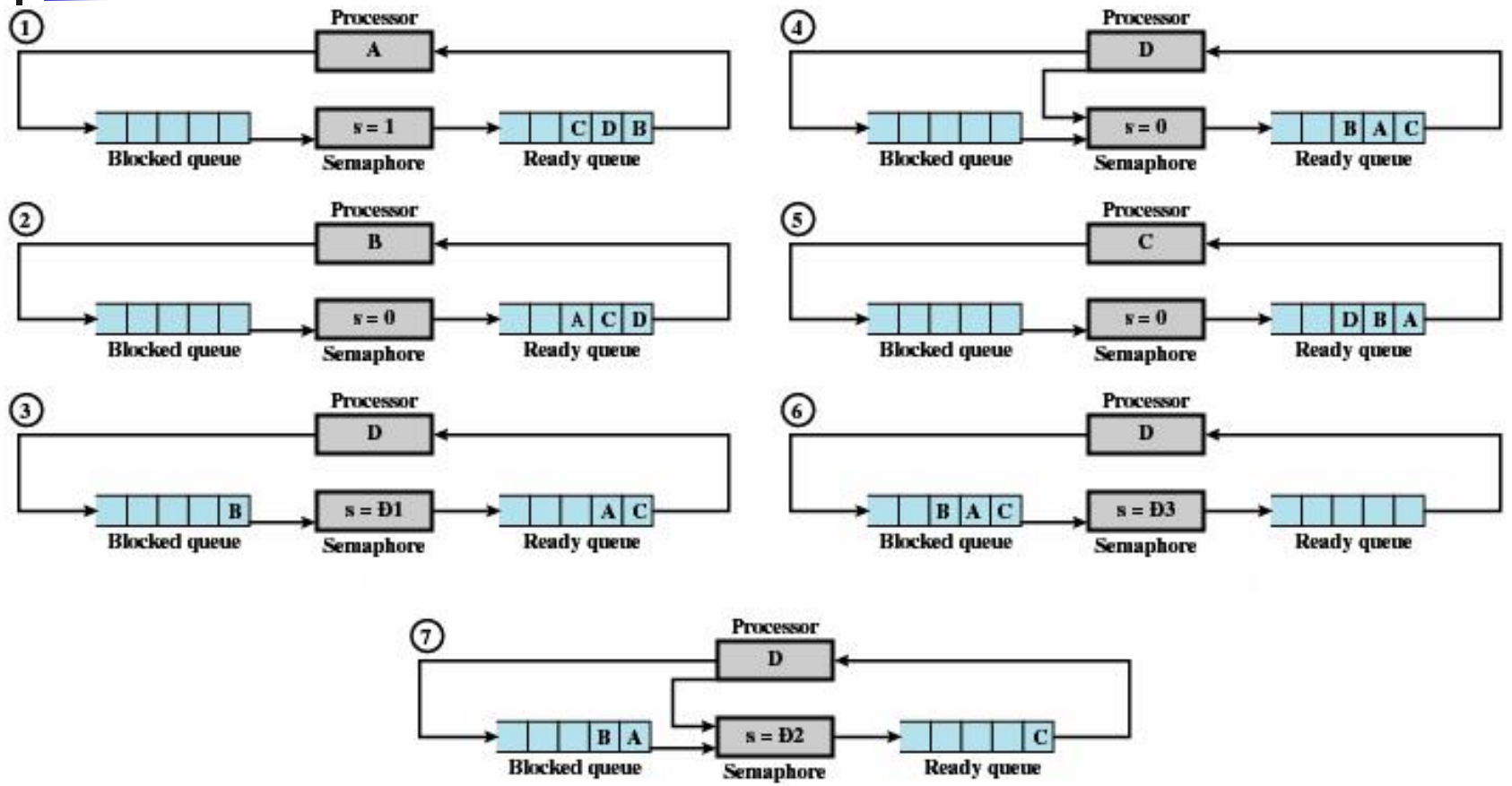


Figure 5.5 Example of Semaphore Mechanism

دسترسی فرایند ها به داده های مشترکی که با یک راهنما محافظت شده اند.(شکل)

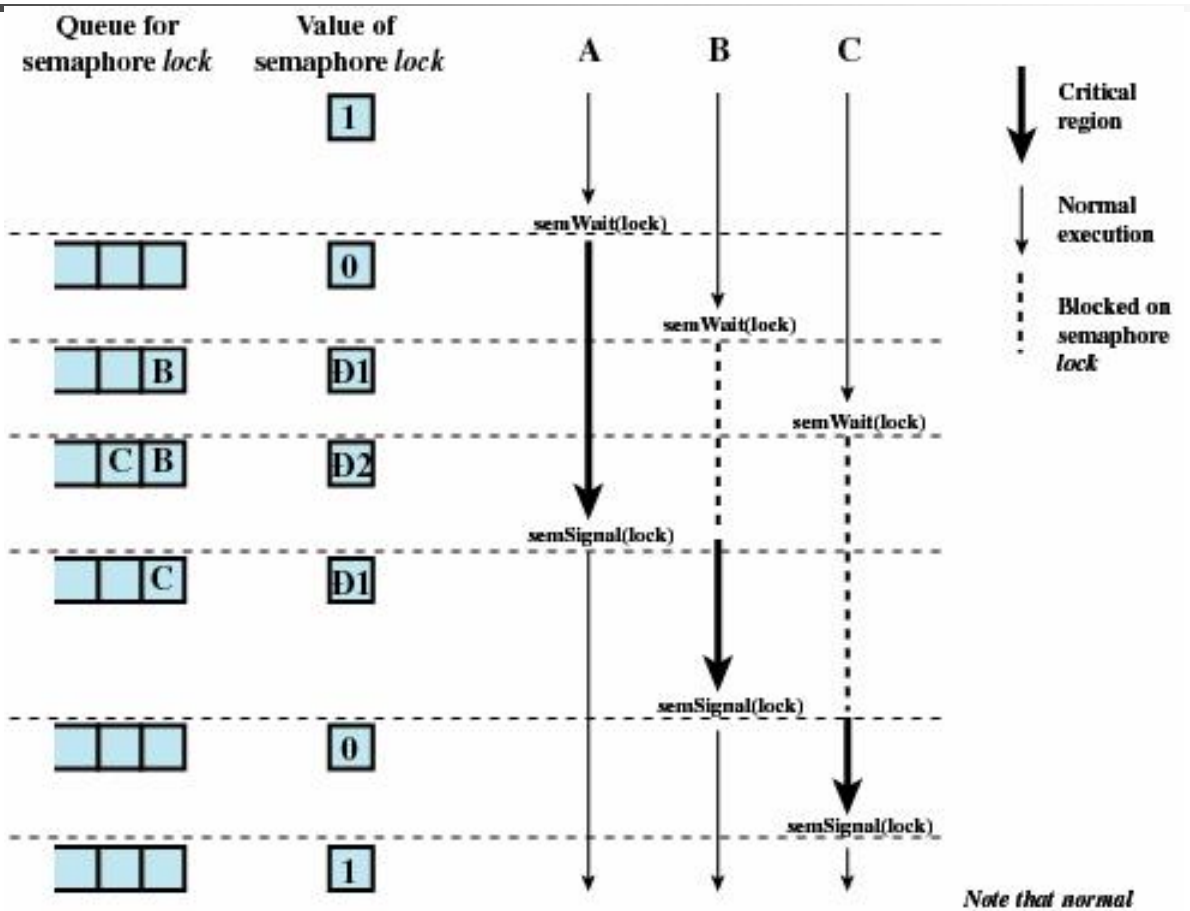


Figure 5.7 Processes Accessing Shared Data Protected by a Semaphore

مساله تولید کننده و مصرف کننده:

- یک تولید کننده یا بیشتر نوعی داده را تولید میکند و در میانگیری قرار میدهد.
- یک مصرف کننده این اقلام را یکی یکی از میانگیر برمیدارد.
- در هر زمان تنها یک تولید کننده یا مصرف کننده میتواند به میانگیر دسترسی داشته باشد.

مساله تولید کننده و مصرف کننده:

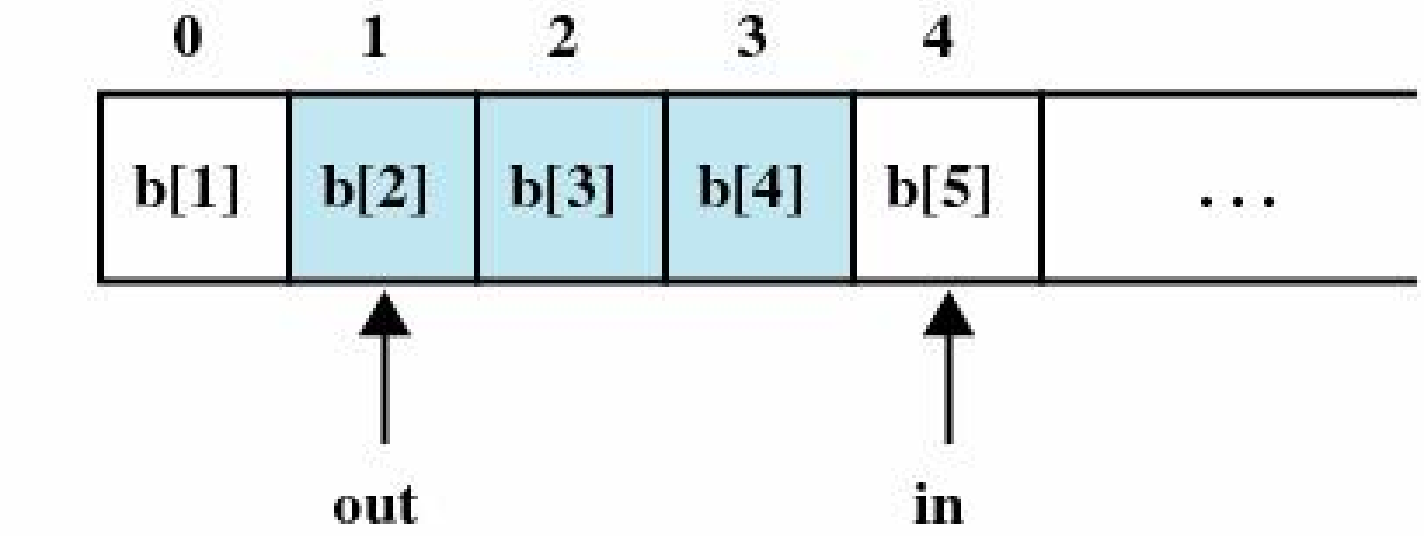
تولید کننده:

```
producer:
while (true) {
        /* produce item v */
        b[in] = v;
        in++;
}
```

مصرف کننده:

```
consumer:
while (true) {
    while (in <= out)
        /*do  nothing */;
    w = b[out];
    out++;
    /* consume item w */
}
```

مساله تولید کننده و مصرف کننده:



Note: shaded area indicates portion of buffer that is occupied

میانگیر نامحدود برای مساله تولید کننده و مصرف

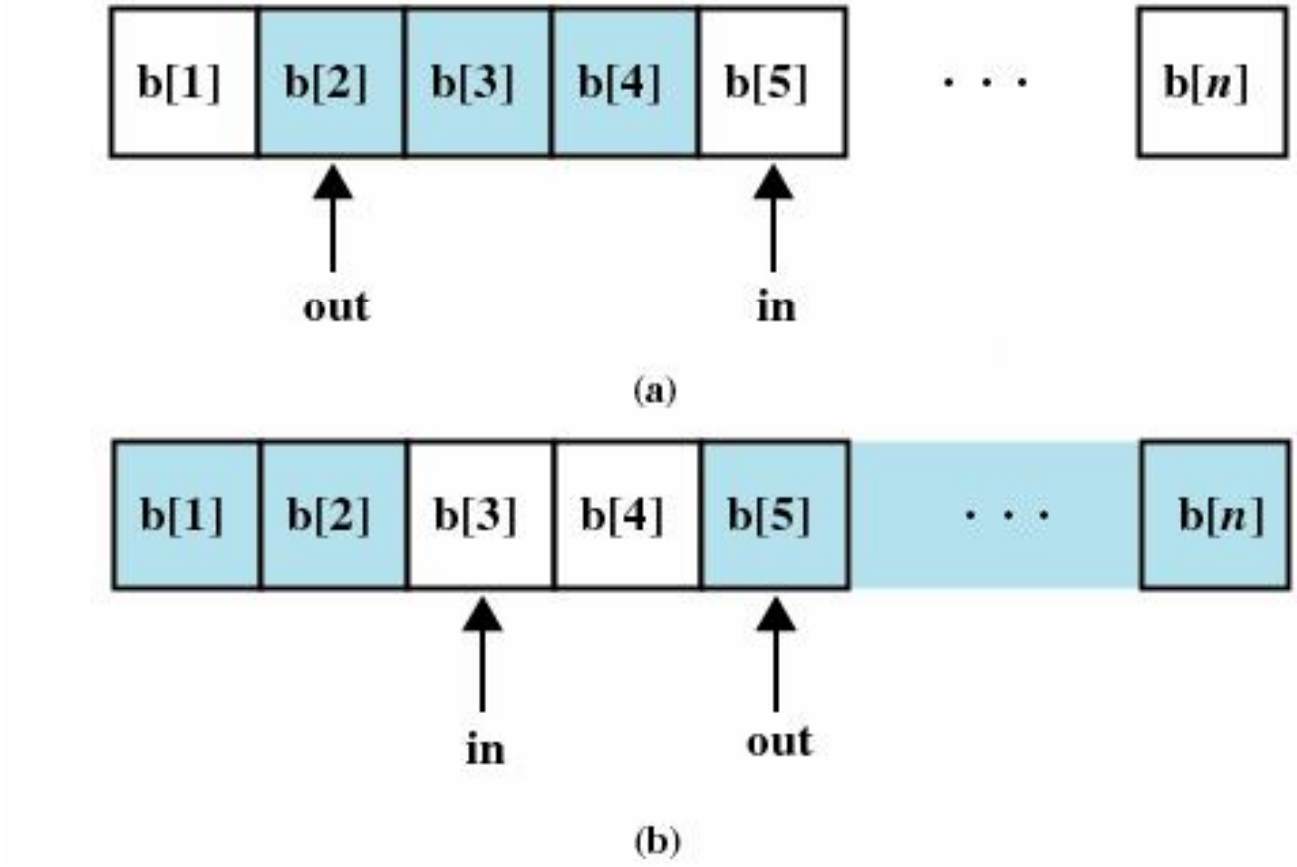
مساله تولید کننده و مصرف کننده با میانگیر نامحدود:

```
producer:
while (true) {
    /* produce item v */
    while ((in + 1) % n == out)

          /* do nothing */;
    b[in] = v;
    in = (in + 1) % n
}
```

```
consumer:
while (true) {
    while (in == out)
          /* do nothing */;
    w = b[out];
    out = (out + 1) % n;
    /* consume item w */
}
```

مساله تولید کننده و مصرف کننده با میانگیر نامحدود:



میانگیر محدود و مدور برای مساله تولید کننده و مصرف

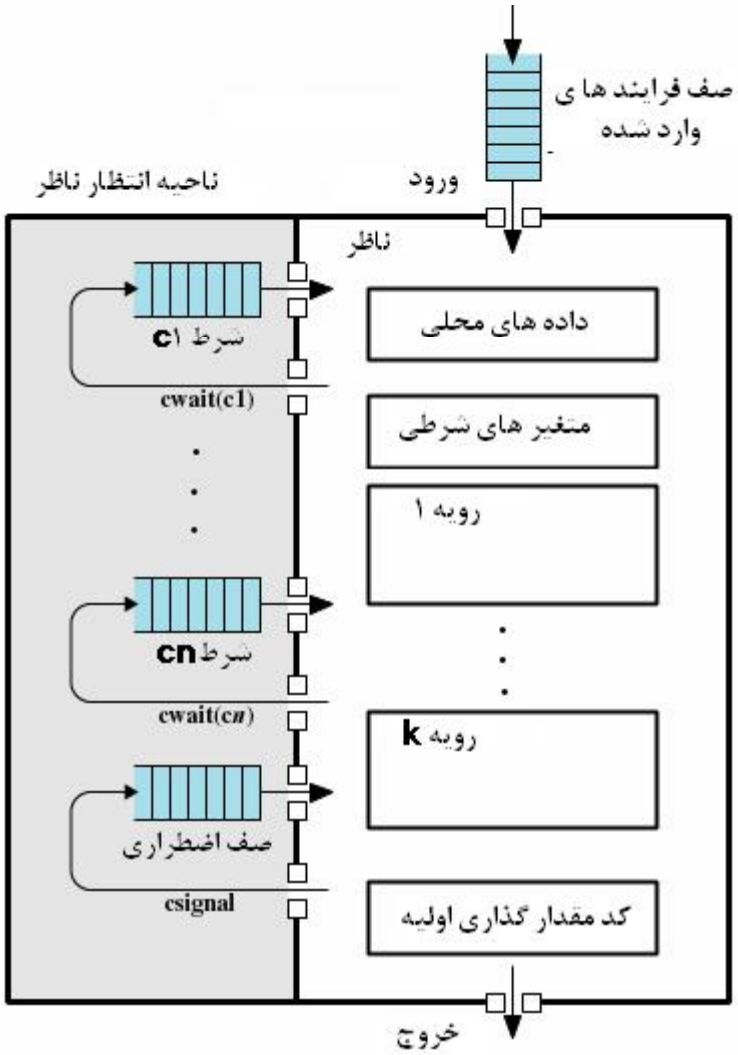
ناظر (Monitor):

- مجموعه ای از رویه ها، متغیر ها، و ساختمان داده ها میباشد.که همگی در یک ماژول نرم افزاری خاص قرار گرفته اند.
- پردازش ها در هر زمان میتوانند زیربرنامه های داخل مانیتور را فراخوانی کرده و اجرا کنند.

ویژگیهای مهم ناظر :

- متغیر ها و داده های محلی ناظر تنها برای رویه های خود ناظر قابل دسترس است و هیچ رویه دیگری به آنها دسترسی ندارد.
- یک فرایند با احضار یکی از رویه های ناظر وارد آن میشود.
- در هر زمان تنها یک فرایند میتوان در ناظر در حال اجرا باشد، فرایند های دیگری که ناظر را احضار کرده اند تا فراهم شدن ناظر معلق خواهند ماند.

ساختار ناظر:


صف فرایند ها ی
وارد شده
ناحیه انتظار ناظر
ورود
ناظر
شرط c۱
cwait(c1)
داده های محلی
متغیر های شرطی
رویه ۱
شرط cn
cwait(cn)
رویه k
صف اضطراری
csignal
کد مقدار گذاری اولیه
خروج

تبادل پیام:

- در یک نتیجه گیری کلی میتوان گفت سمافور ها سطح پایین هستند، و کار کردن با آنها مشکل است.مانیتور ها نیز به جز چند زبان برنامه نویسی غیر معروف غیر قابل استفاده اند.علاوه بر این هیچ کدام از این دو، امکانات لازم برای تبادل اطلاعات بین کامپیوتر ها در کامپیوتر های توزیع شده را ندارند بنابرین تکنیک دیگری به نام تبادل پیام ابداع شد.
- پیام ها یک مکانیزم ساده و مناسب جهت همگام سازی و ارتباط دهی بین فرایندها در یم محیط غیرمتمرکز و توزیع شده اند.
- بسیاری از سیستم عامل های چند برنامگی از نوعی پیام های بین فرایند ها پشتیبانی میکنند.

تبادل پیام:

- پیام مجموعه ای از اطلاعات است که بین فرایندهای ارسال کننده و دریافت کننده مبادله میشود.
- قالب پیام قابل انعطاف و قابل تبادل توسط هر زوج گیرنده فرستنده است.

```
send (destination, message)
receive (source, message)
```

همگام سازی:

- فرستنده و گیرنده میتوانند مسدود باشند یا نباشند(منتظر برای پیام)
- مسدود شدن فرستنده، مسدود شدن گیرنده:
  - هم فرستنده و هم گیرنده تا زمانی که پیام تحویل داده شود مسدودند.
  - گاهی به آن قرار ملاقات هم گفته میشود.
  - این ترکیب همگام سازی محکم بین فرایندها را میسر میکند.

همگام سازی:

- مسدود نشدن فرستنده، مسدود شدن گیرنده:
  - فرستنده به کار خود ادامه میدهد.
  - گیرنده تا زمانی که پیام تحویل داده شود، مسدود است.
  - این مفید ترین ترکیب است، چرا که اجازه میدهد یک پیام یا بیشتر در اسرع وقت به مقصد های متنوع ارسال شود.
- مسدود نشدن فرستنده، مسدود نشدن گیرنده
  - انتظار هیچ یک از دو طرف ضروری نیست

آدرس دهی:

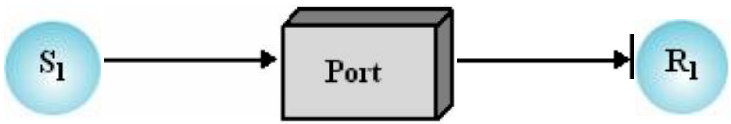
- آدرس دهی مستقیم:
  - اولیه Send شامل شناسه مشخص فرایند مقصد است
  - اولیه Receive میتوان به یکی از دو صورت زیر داده شود:
    - فرایند گیرنده صراحتاً فرایند فرستنده را تعیین کند بنابراین فرایند باید از قبل بداند از کدام فرایند باید انتظار پیام داشته باشد.
    - مشخص کردن فرایند مبدأ مورد انتظار غیر ممکن است در این حالت پارامتر مبدأ از اولیه Receive دارای مقداریست که با انجام عمل دریافت برگشت داده میشود.

آدرس دهی:

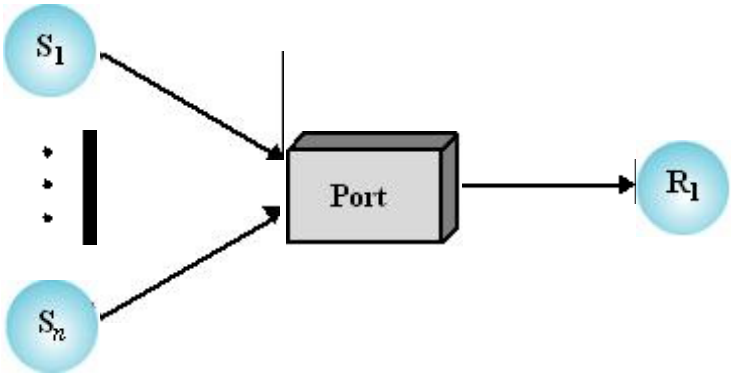
- آدرس دهی غیر مستقیم:
  - پیامها مستقیماً از فرستنده به گیرنده ارسال نمیشوند.بلکه به یک ساختمان داده مشترک که شامل صفهایی است که میتوانند پیامها را به طور دائمی نگه دارند ارسال میشود.
  - صف ها معمولاً صندوقهای پستی (Mail box) نامیده میشوند.
  - یک فرایند پیام را به صندوق پستی ارسال میکند و فرایند دیگر آن را از صندوق پستی بر میدارد.

ارتباط غیر مستقیم فرایند ها:

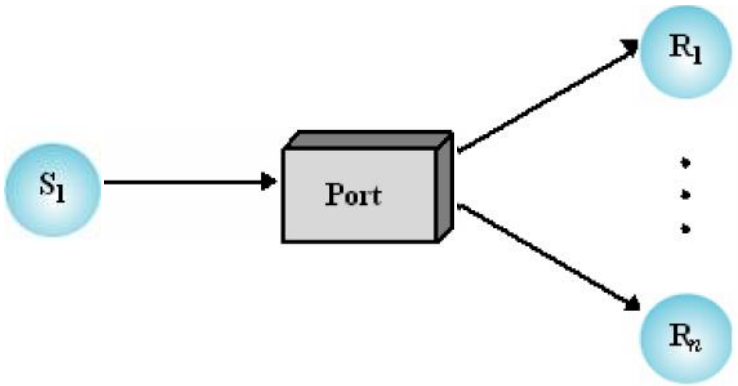
یک به یک



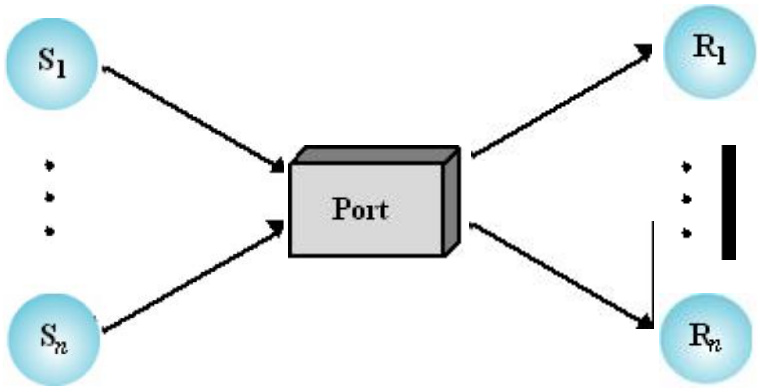
چند به یک



یک به چند



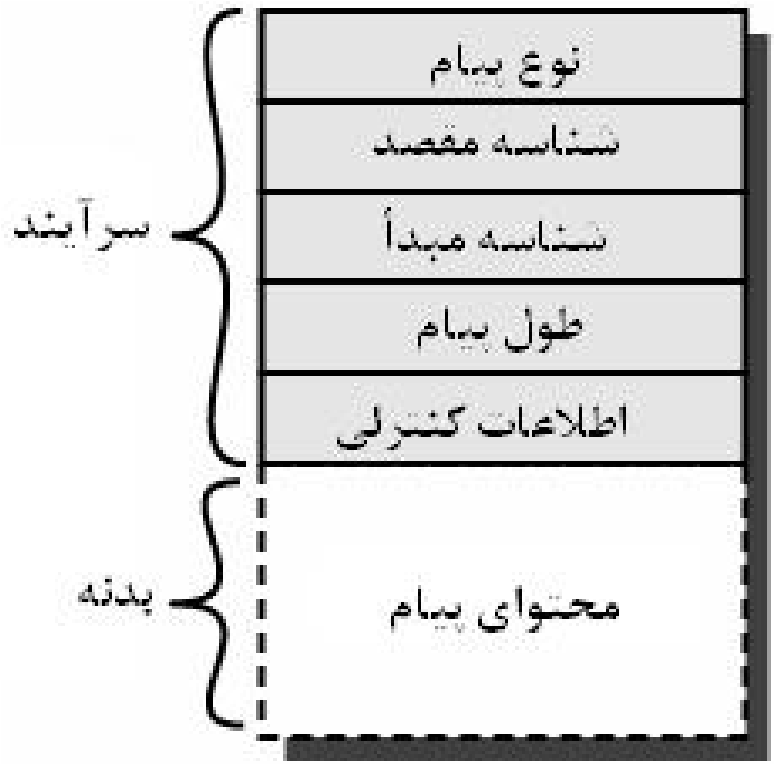
چند به چند



قالب پیام:

- قالب پیام به اهداف امکانات پیام دهی و اینکه این امکانات روی یک کامپیوتر یا یک سیستم توزیعی اجرا میشوند بستگی دارد.
- بعضی سیستم عاملها برای حداقل کردن سربار پردازش و حافظه پیامهای کوچک با طول ثابت را ترجیح داده اند.
- قالب کلی پیام های طول متغیربه دو بخش تقسیم میشود:
  - سرآمد: حاوی اطلاعات مربوط به پیام
  - بدنه: حاوی خود پیام

قالب پیام:



- سرآیند:
  - اطلاعات مبدأ و مقصد مورد نظر پیام
  - طول پیام
  - اطلاعات کنترلی
    - اشاره گر برای ایجاد لیستی از پیام ها
    - شماره ترتیب برای پیگیری تعداد و ترتیب انتقال پیامها
- بدنه: