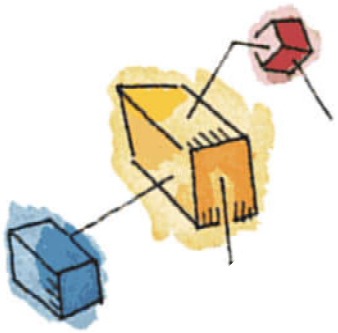
به نام خدا

فصل پنجم

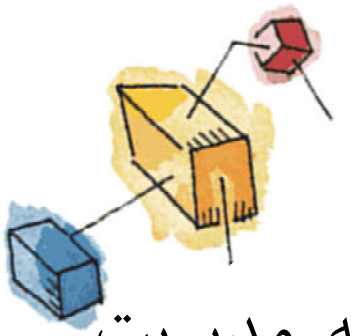
همروندی:
انحصار متقابل و همگام سازی
(بخش اول)

Concurrency: Mutual Exclusion and Synchronization



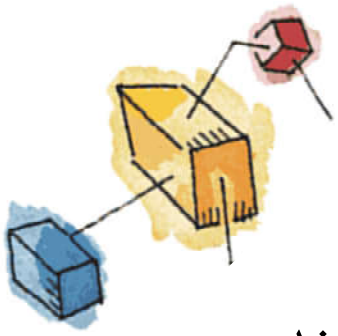
سرفصل مطالب

- اصول همروندی (همزمانی)
- انحصار متقابل: حمایت سخت افزار
- راهنماها (سمافورها)
- ناظرها (مانیتورها)
- تبادل پیام
- مساله خوانندگان و نویسندگان



چند فرآیندی

- همه موضوعات محوری در طراحی سیستم عامل، به مدیریت فرآیندها و نخ ها مربوط می شود:
  - **چند برنامه ای**: مدیریت فرآیندهای متعدد در داخل یک سیستم تک پردازنده ای
  - **چند پردازشی**: مدیریت فرآیندهای متعدد در داخل یک سیستم چندپردازنده ای
  - **پردازش توزیعی**: مدیریت فرآیندهای متعدد که روی سیستم های کامپیوتری متعدد و توزیع شده اجرا می شوند.
- برای هر سه موضوع فوق، همزمانی مساله اساسی است.
- همزمانی(همروندی) به معنی مدیریت ارتباط ها بین این فرآیندها است.



پیش زمینه

- بخش های مختلف سیستم از منابع مشترک استفاده می کنند.
- به ویژه با رشد سیستم های چندهسته ای و توسعه برنامه های چندنخی این مشکل شدید تر می شود.
- زیرا چندین نخ که احتمالا داده های مشتری دارند توسط چندین هسته به طور موازی اجرا می شوند.
- می خواهیم تغییرات حاصل از فعالیت ها با همدیگر تداخلی نداشته باشند.
- این فصل به این موضوع یعنی همگامی و هماهنگی فرآیندها می پردازد.

اجرای در بین هم (Interleaving) و همپوشانی در اجرای فرایندها

- در سیستم های تک پردازنده ای، فرآیندها در بین یکدیگر اجرا می شوند و به نظر می رسد که همزمان در حال اجرا هستند.
- در سیستم های چندپردازنده ای نه تنها ممکن است فرآیندها در بین یکدیگر اجرا شوند، بلکه می توانند واقعا به موازات و با همپوشانی اجرا شوند.

- با وجود این تفاوت، مشکلات یکسانی در سیستم های تک پردازنده ای و چندپردازنده ای رخ می دهد.

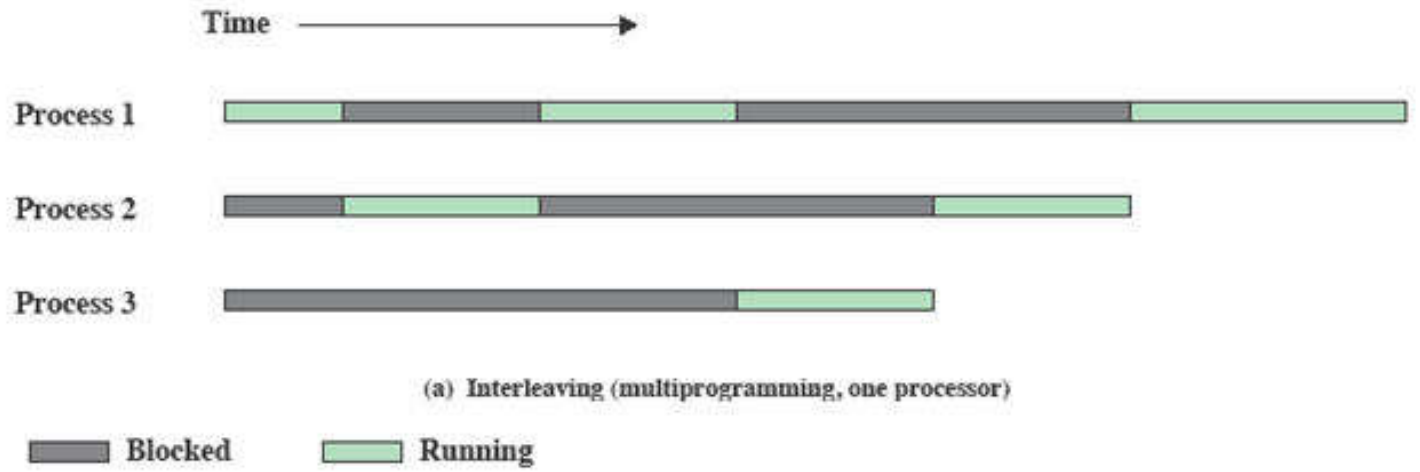
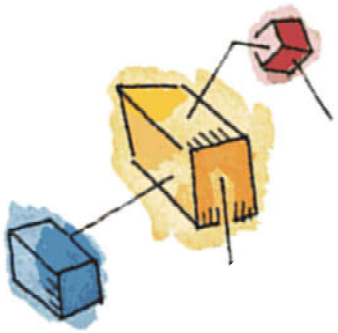
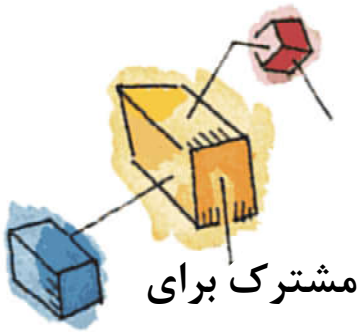


Figure 2.12 Multiprogramming and Multiprocessing



مشکلات همروندی

- به اشتراک گذاری منابع عمومی سیستم با مشکلات و مسائل مختلفی همراه است.
  - مثلا اگر دو فرآیند از یک متغیر سراسری استفاده می کنند، و خواندن و نوشتن را روی آن انجام می دهند، ترتیب اجرای خواندن و نوشتن ها بحرانی است.
- مدیریت تخصیص بهینه منابع به فرآیندها توسط سیستم عامل دشوار است.
  - ممکن است باعث بن بست شود.
- یافتن محل خطای برنامه نویسی مشکل است.
  - چون نتایج قطعی نیست و به راحتی نمی توان آنها را دوباره تولید کرد.



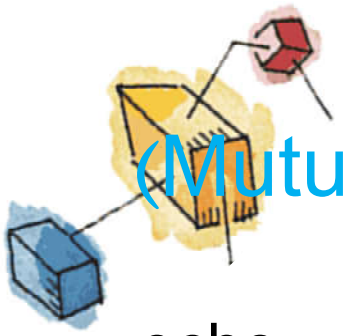
یک مثال ساده

متغیرهای chin و chout سراسری هستند و تابع echo بین فرآیندها به صورت مشترک برای گرفتن و نمایش کاراکترها استفاده می شود.

```
void echo()
{
   chin = getchar();
   chout = chin;
   putchar(chout);
}
```

```
Process P1
.
chin = getchar();
.
chout = chin;
putchar(chout);
.
.
```

```
Process P2
.
.
chin = getchar();
chout = chin;
.
putchar(chout);
.
```



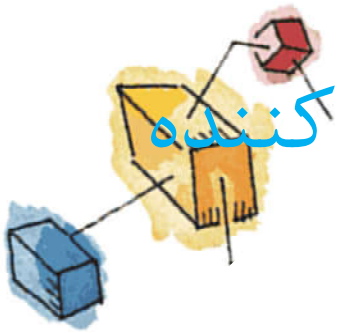
یک مثال ساده: انحصار متقابل(Mutual exclusion)

- اگر فرض کنیم تنها یک فرآیند ممکن است در حال اجرای تابع echo باشد، آنگاه دنباله زیر اتفاق می افتد:

1. فرآیندهای P1 و P2 روی پردازنده های جداگانه در حال اجرا هستند. P1 رویه echo را فراخوانی می کند.

2. هنگامی که P1 داخل رویه echo است P2 رویه echo را احضار می کند. نظر به اینکه P1 هنوز در echo است (چه P1 معلق و چه در حال اجرا باشد)، P2 از ورود به رویه منع می شود. بنابراین P2 در انتظار دسترسی به رویه echo معلق می ماند.

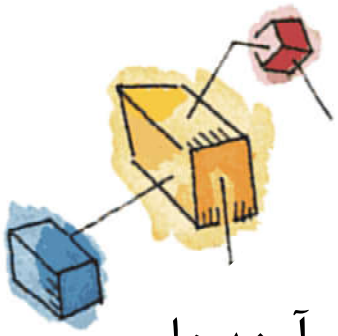
3. بعد از مدتی فرآیند P1 اجرای echo را کامل کرده و از آن خارج می شود و از اجرا باز می ایستد. به محض خروج P1 از echo، P2 از سرگرفته می شود و شروع به اجرای echo می کند.



یک مثال دیگر: مساله تولیدکننده - مصرف کننده
Producer-Consumer Problem

- نمونه ای از فرآیندهای همکار:
- فرآیند تولیدکننده اطلاعاتی را تولید می کند و در حافظه میانگیر (buffer) قرار می دهد که توسط فرآیند مصرف کننده مورد استفاده قرار گیرند.

- دو فرض متفاوت برای حل مساله وجود دارد:
  - حافظه میانگیر محدود (bounded-buffer)
  - حافظه میانگیر نامحدود (unbounded-buffer).

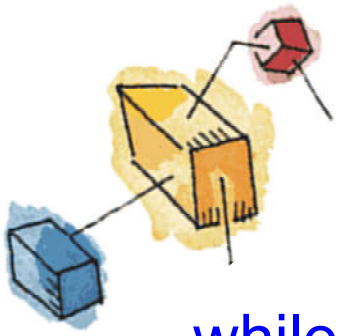


مسأله تولیدکننده -مصرف کننده

- بکارگیری یک بافر محدود: امکانی برای اشتراک حافظه توسط فرآیندها
- یک counter را با مقدار اولیه ۰ داریم.

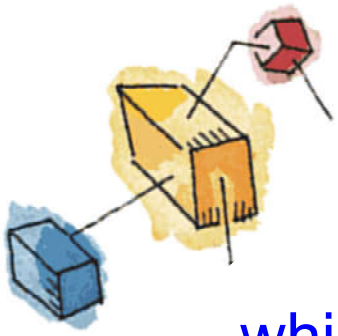
- هر بار که قلم داده جدیدی به بافر اضافه شود، counter یک واحد افزایش می یابد.

- هر بار که قلم داده جدیدی از بافر حذف شود، counter یک واحد کاهش می یابد.



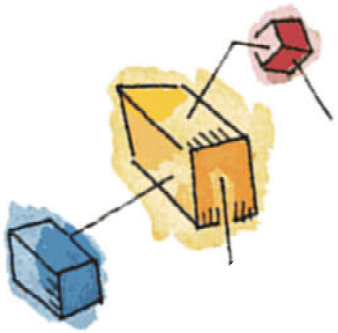
تولید کننده

```
while (true) {

        /*  produce an item and put in nextProduced  */
         while (count == BUFFER_SIZE)
                 ; // do nothing
          buffer [in] = nextProduced;
          in = (in + 1) % BUFFER_SIZE;
          count++;
}
```

مصرف کننده

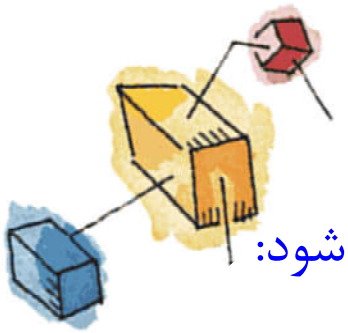
```
while (true)  {
        while (count == 0)
              ; // do nothing
         nextConsumed =  buffer[out];
         out = (out + 1) % BUFFER_SIZE;
         count--;

          /*  consume the item in nextConsumed
}
```

مساله تولیدکننده - مصرف کننده

- هر دو روال تولید کننده و مصرف کننده به طور جداگانه درست عمل می کنند.

- اما هنگامی که به طور همزمان اجرا شوند ممکن است به درستی عمل نکنند.



مثالی از شرایط مسابقه

- دستور count++ می تواند به زبان ماشین به صورت زیر پیاده سازی شود:

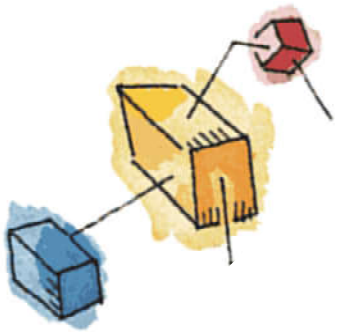
```
register1 = count
register1 = register1 + 1
count = register1
```

- دستور count-- می تواند به زبان ماشین به صورت زیر پیاده سازی شود:

```
register2 = count
register2 = register2 - 1
count = register2
```

- حالتی را در نظر بگیرید که در ابتدا "count = 5" و دستورات به صورت زیر در بین هم اجرا می شوند.

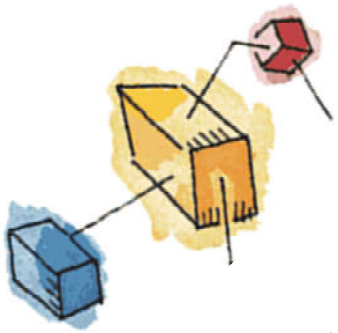
S0: producer execute register1 = count {register1 = 5}
S1: producer execute register1 = register1 + 1 {register1 = 6}
S2: consumer execute register2 = count {register2 = 5}
S3: consumer execute register2 = register2 - 1 {register2 = 4}
S4: producer execute count = register1 {count = 6 }
S5: consumer execute count = register2 {count = 4}



شرایط مسابقه (Race Condition)

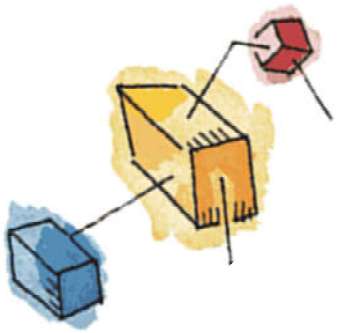
- شرایط مسابقه وقتی رخ می دهد که:
  - چند نخ یا فرآیند بتوانند داده هایی را بخوانند یا بنویسند
  - و این کار را به نحوی انجام دهند که نتیجه نهایی وابسته به ترتیب اجرای فرآیندها یا نخ ها در بین هم باشد.

- خروجی وابسته به این است که کدام یک، مسابقه را به پایان می رساند.



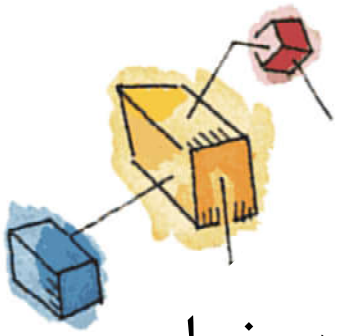
وظایف سیستم عامل در ارتباط با همروندی

- سیستم عامل باید بتواند فرآیند های فعال را دنبال کند.
- سیستم عامل باید بتواند منابع را به فرآیندها تخصیص دهد و بگیرد.
- سیستم عامل باید داده ها و منابع هر فرآیند را از دسترسی سایر فرآیندها محافظت کند.
- نتایج یک فرآیند باید مستقل از سرعت پیشرفت فرآیندهای دیگر باشد.



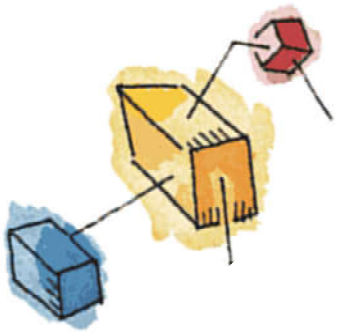
محاوره فرآیندها

- بی اطلاعی فرآیندها از یکدیگر
  - این فرآیندها مستقل از یکدیگرند و خواستار همکاری با یکدیگر نیستند.
- اطلاع غیرمستقیم فرآیندها از یکدیگر
  - این فرآیندها لزوما از شناسه یکدیگر اطلاع ندارند، ولی در دسترسی به بعضی اشیاء مثل بافر ورودی–خروجی با یکدیگر مشترک اند.
- اطلاع مستقیم از یکدیگر
  - این ها فرآیندهایی هستند که قادرند با استفاده از شناسه با یکدیگر ارتباط برقرار کنند.
  - برای کار مشترک بر روی بعضی فعالیت ها ساخته شده اند.



رقابت میان فرآیندها برای منابع

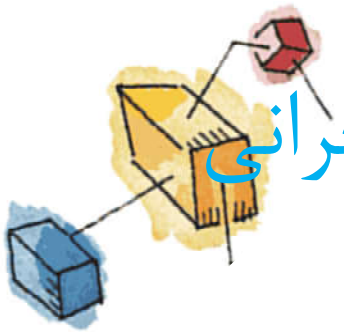
- در مورد فرآیندهای رقیب با سه مساله کنترلی برخورد خواهیم داشت:
- **۱. انحصار متقابل (Mutual Exclusion)**
  - بخش بحرانی (Critical Section)
    - هر فرآیند دارای قطعه کدی است به نام بخش بحرانی که در این ناحیه ممکن است متغیرهای مشترک تغییر نمایند، جدولی به روز شود، و یا فایلی به اشتراک گذاشته شود.
  - در هر لحظه فقط یک برنامه اجازه دارد به بخش بحرانی خود وارد شود. به این موضوع، انحصار متقابل گفته می شود.
  - به عنوان مثال در هر لحظه تنها یک فرآیند اجازه دارد پیامی را به چاپگر بفرستد.



رقابت میان فرآیند ها برای منابع

- **۲. بن بست (Deadlock)**
  - هنگام اعمال انحصار متقابل، در صورتیکه یک فرآیند کنترل منبعی را در اختیار بگیرد و در انتظار منبع دیگری برای اجرا باشد، ممکن است بن بست رخ دهد.

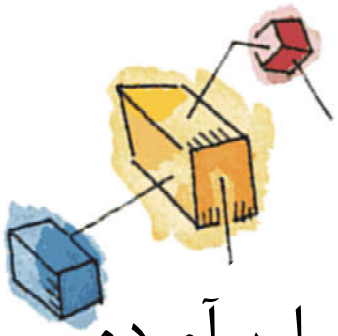
- **۳. گرسنگی یا قحطی زدگی (Starvation)**
  - ممکن است یکی از فرآیندهای مجموعه برای مدتی نامحدود از دسترسی به منابع مورد نیازش محروم بماند، چرا که سایر فرآیندها منابع را به طور انحصاری بین یکدیگر مبادله می کنند. به این حالت گرسنگی یا قحطی زدگی می گویند.



رقابت میان فرآیند ها برای منابع و ناحیه بحرانی

- در ادامه فصل خواهیم دید که:
  - هر فرآیند برای ورود به ناحیه بحرانی خود باید اجازه بگیرد.
  - بخشی از کد که این درخواست را انجام می دهد، ناحیه ورودی نام دارد.
  - ناحیه بحرانی ممکن است ناحیه خروج هم داشته باشد. بقیه کد، ناحیه باقیمانده است.

P_1	P_2	P_n
do {	do {	do {
ناحیه ورودی	ناحیه ورودی	ناحیه ورودی
ناحیه بحرانی	**ناحیه بحرانی**	**ناحیه بحرانی**
ناحیه خروجی	ناحیه خروجی	ناحیه خروجی
ناحیه باقیمانده	ناحیه باقیمانده	ناحیه باقیمانده
} while (true)	} while (true)	} while (true)



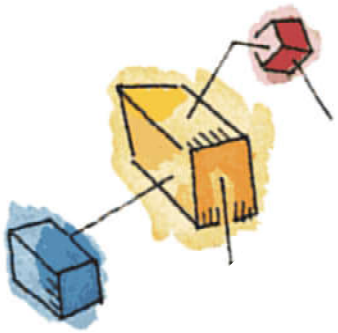
شرایط لازم برای انحصار متقابل

(ویژگی های راه حل ها برای ناحیه بحرانی)

- راه حل مسأله ناحیه بحرانی بایستی نیازمندی های زیر را برآورده نماید:

1. از میان فرآیندهایی که برای منبع یکسان یا شیء مشترکی دارای بخش بحرانی هستند، در هر لحظه تنها یک فرآیند مجاز است در بخش بحرانی خود باشد (شرط انحصار متقابل یا Mutual Exclusion).
2. فرآیندی که در بخش غیربحرانی خود متوقف می شود، باید طوری عمل کند که هیچ دخالتی در عملکرد فرآیندهای دیگر نداشته باشد.
3. برای فرآیندی که درخواست دسترسی به یک بخش بحرانی دارد، نباید امکان به تاخیر انداختن نامحدود آن درخواست وجود داشته باشد (شرط انتظار محدود یا Bounded Waiting).

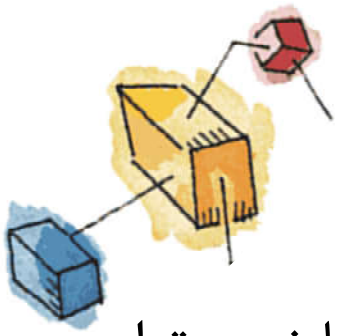
– بدین معنی که بن بست یا گرسنگی مجاز نیست.



شرایط لازم برای انحصار متقابل
(ویژگی های راه حل ها برای ناحیه بحرانی)

4. هنگامی که هیچ فرآیندی در بخش بحرانی خود نیست، هر فرآیند متقاضی ورود به ناحیه بحرانی بایستی بدون تأخیر اجازه دسترسی پیدا کند.

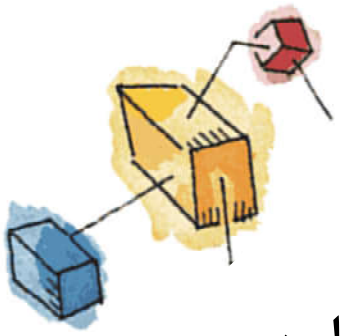
- می توان گفت: اگر هیچ فرآیندی در حال اجرای بخش بحرانی نباشد و برخی از فرآیندها بخواهند به بخش بحرانی خودشان وارد شوند، در اینصورت بر سر اینکه کدام یک از فرآیندها می تواند در ادامه وارد بخش بحرانی خود شود تنها فرآیندهایی در تصمیم گیری سهیم هستند که در حال اجرای بخش باقیمانده خود نباشند و این انتخاب نمی تواند به طور نامعینی به تعویق بیفتد (شرط پیشرفت یا Progress)



شرایط لازم برای انحصار متقابل
(ویژگی های راه حل ها برای ناحیه بحرانی)

5. هیچ فرضی در مورد سرعت نسبی فرآیندها یا تعداد آنها نمی توان در نظر گرفت.

6. هر فرآیند فقط برای مدت زمانی محدود در داخل ناحیهٔ بحرانی خود می ماند.



راه حل ها

- راه های مختلفی برای حل مساله ناحیه بحرانی وجود دارد:

  - در برخی راه حل ها، مسئولیت به فرآیندهایی که می خواهند به طور همزمان اجرا شوند واگذار می شود (راه حل های **نرم افزاری**).
  - رویکرد دوم، استفاده از حمایت **سخت افزار** است.
  - رویکرد سوم، استفاده از حمایت **سیستم عامل یا زبان برنامه سازی** است (سمافور و ناظر).

- رویکرد دوم و سوم در ادامه بررسی می شوند.