

رابطه بین پردازش‌ها (Interprocess Communication)

پردازش‌ها از جهت وابستگی به یکدیگر در دو دسته قرار می‌گیرند و توجه به این مسئله در نحوه فاصله آنها حائز اهمیت است

**الف- پردازش‌های مستقل:** هر پردازش‌ای که داده‌ای را (چه موقت و چه دائمی) با دیگر پردازش‌ها به اشتراک نگذارد مستقل است و یا به عبارت دیگر یک پردازش در صورتی مستقل از دیگر پردازش‌ها است که شرایط زیر را برآورده کند.

1- حالت آن به اشتراک‌گذارده نشده باشد.

2- نتیجه اجرای آن کاملاً مشخص باشد.

3- نتیجه اجرا در دفعات مختلف به ازای ورودی‌های یکسان، مساوی باشد.

4- توقف و شروع آن بدون تأثیر بر سایر پردازش‌ها امکان‌پذیر باشد.

**ب- پردازش‌های همکاری‌کننده:** اگر دو پردازش نسبت به هم یکی از شرایط فوق را نداشته باشد گوئیم پردازش‌ها نسبت به هم همکاری‌کننده به عبارت دیگر پردازش‌هایی هستند که شرایط زیر در مورد آنها صادق باشد.

1- حالت آن به اشتراک‌گذارده شود

2- نتیجه اجرا به دلیل وابستگی به سایر پردازش‌ها کاملاً مشخص نباشد

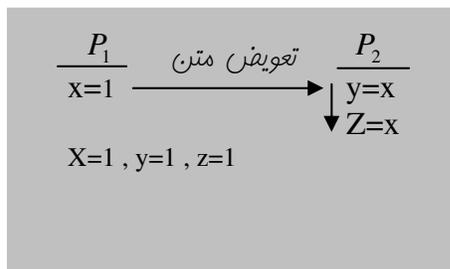
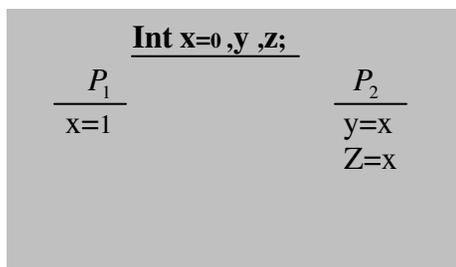
3- نتیجه اجرا برای ورودی‌های یکسان در دفعات مختلف، یکسان نباشد

منظور از اجرای Interleaved چیست؟

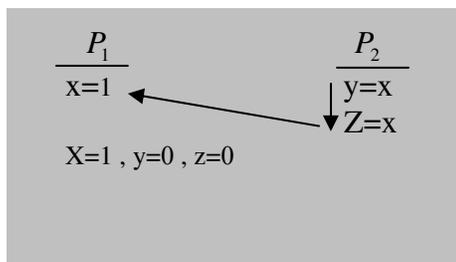
به این معناست که در بین اجرای یک پردازش در هر زمان امکان وقفه یا تعویض متن (Context Switching) به پردازش دیگر وجود دارد

**مثال:** اگر دو پردازش  $P_1$  و  $P_2$  به صورت هم‌روند اجرا شوند مقادیر نهایی  $x$ ،  $y$  و  $z$  را بیابید

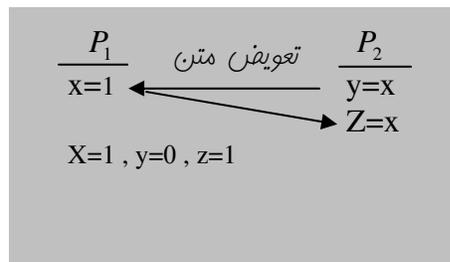
حل: در اجرای هم‌روند این دو پردازش سه ترتیب زیر امکان‌پذیر است



الف-



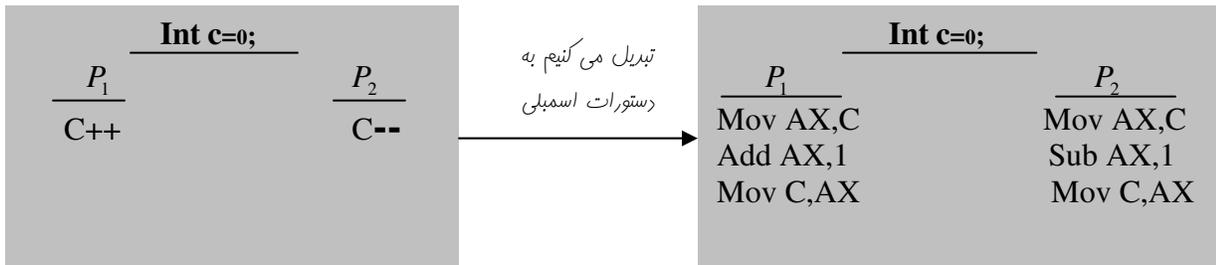
ج-



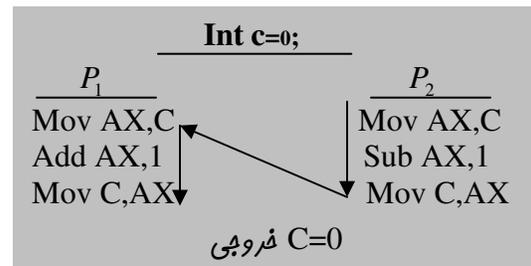
ب-

## منظور از همگام سازی پردازش ها (Synchronization) چیست؟

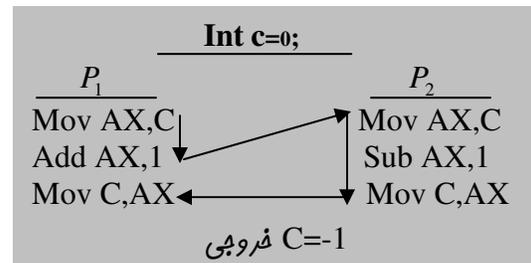
زمانی که پردازش ها به هم وابسته اند نتیجه اجرا به ترتیب اجرای دستورات برنامه بستگی دارد، بنا براین می بایست با توجه به نتایج مورد انتظار ترتیب اجرای دستورات پردازش ها را مشخص کرد که به این مسئله همگام سازی پردازش ها گویند  
 مثال: اگر دو پردازش  $P_1$  و  $P_2$  به صورت همروند اجرا شوند مقدار نهائی  $C$  بیاید.



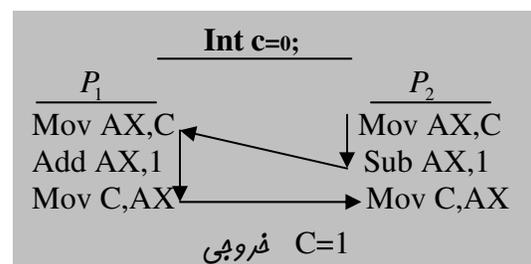
الف-



ب-



ج-



به علت اینکه هر دو پدازه از متغیر مشترک C استفاده کرده اند، مقدار نهائی C وابسته به ترتیب اجرای پدازه ها می باشد که در این مثال بسته به ترتیب اجرای دستورات C می تواند مقادیر 0، 1، -1، را اتخاذ نماید، ولی نتیجه مورد نظر صفر می باشد.

### منظور از وضعیت مسابقه یا Race Condition چیست؟

وضعیت هائی که در آن فرایندهای متعددی، داده یکسانی را به طور همروند دستیابی و دستکاری می کنند و حاصل اجرا بستگی به ترتیب خاص دسترسی ها دارد، وضعیت مسابقه یا Race Condition گفته می شود.

### نوامی بحرانی (Critical Section):

برای جلوگیری از شرایط رقابتی باید راهی را پیدا کنیم که از خواندن و نوشتن داده های مشترک، به طور همزمان، توسط بیش از یک پروسس جلوگیری به عمل آید. به عبارت دیگر ما به «انحصار متقابل» یا Mutual Exclusion نیاز داریم، به عبارتی دیگر اگر یکی از پدازه ها در حال استفاده از داده مشترک است باید مطمئن باشیم که دیگر پدازه ها، در آن زمان از انجام همان کار معروم می باشند.

بفشی از برنامه که به حافظه اشتراکی دسترسی دارد را قسمت یا ناحیه بحرانی (Critical Section) می نامیم.

هر پدازه برای ورود به بخش بحرانی اش باید اجازه بگیرد. بفشی از کد پدازه که این اجازه گرفتن را پیاده سازی می کند بخش ورودی یا Entry section نام دارد.

بخش بحرانی می تواند با بخش خروجی یا exit section دنبال شود. این بخش خروجی کاری می کند که پدازه های دیگر بتوانند وارد ناحیه بحرانی شان بشوند. بقیه کد پدازش را بخش باقی مانده یا remainder section گوئیم. بنا براین سافتکار کلی پدازش ها به صورت زیر می باشد.

```
While(True){
```

```
    Entry section
```

```
    Critical_section();
```

```
    exit section
```

```
    Remainder_section();
```

```
}
```

باید جهت رفع مشکل وضعیت مسابقه چهار شرط زیر رعایت گردد تا یک راه حل خوب بدست آید

### 1- شرط انحصار متقابل (مانعه الجمعی Mutual Exclusion):

هنگامی که پدازشی در ناحیه بحرانی اش اجرا می گردد، هیچ پدازش دیگری نباید در ناحیه بحرانی حضور داشته باشد.

### 2- شرط پیشرفت یا پیشروی (Progress):

هنگامی که هیچ پدازشی در قسمت بحرانی در حال اجرا نباشد و تقاضاهائی برای ورود به بخش بحرانی وجود دارد، فقط پدازش هائی در تصمیم گیری برای ورود دالت می کنند که هنوز به ناحیه بحرانی شان نرسیده باشند. به عبارت دیگر اگر پدازشی در قسمت باقی مانده (remainder) خود باشد، در تصمیم گیری اینکه که کدام پدازه وارد بخش بحرانی شود، شرکت داده نمی شود. به عبارت دیگر هیچ پدازشی نباید از بیرون ناحیه بحرانی خود امکان بلوکه کردن پدازش های دیگر را داشته باشد.

3- شرط انتظار مقید یا محدود (Bounded Waiting): یک برنامه منتظر ورود به ناهیه بهرانی، نباید به طور نامحدود در حالت انتظار باقی بماند

4- هر پردازشی با سرعت غیر صفر اجرا می شود ولی هیچ فرضی در مورد سرعت نسبی n پردازش و نیز تعداد CPU ها نمی کنیم

روش های جلوگیری از مسابقه (همزمانی پردازش ها)

1- غیر فعال ساختن وقفه ها:

هر پردازش بلا فاصله پس از ورود به ناهیه بهرانی اش کلیه وقفه ها را از کار بیندازد و درست قبل از خروج از ناهیه بهرانی دوباره همه آنها را فعال کند. با خاموش ساختن وقفه ها CPU به هیچ عنوان نمی تواند از پردازشی به پردازش دیگر سوئیچ کند. پس پردازش می تواند بدون ترس از مداخله دیگر پردازش ها به دستکاری قسمت مشترک بپردازد.

این روش دو مشکل دارد

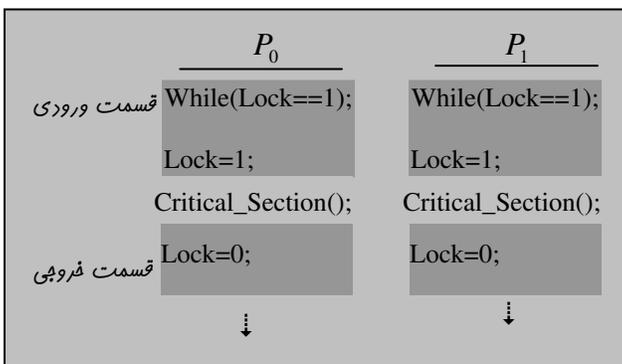
1- ممکن است کاربر وقفه ها را خاموش کند ولی دوباره آنها را فعال نسازد (یادش برود) بدین ترتیب سیستم از کار خواهد افتاد. پس اعطای قدرت غیر فعال ساختن وقفه ها به پردازش کاربران عاقلانه نیست.

2- در سیستم های چند پردازنده ای غیر فعال ساختن وقفه ها، فقط در CPU ای اثر دارد که دستور از کار انداختن وقفه را اجرا می کند، بقیه CPU ها می توانند کار خودشان را ادامه داده و به حافظه مشترک دسترسی پیدا کنند.

2- استفاده از متغیر های قفل (Lock Variables):

فرض کنید یک متغیر قفل یکتا و مشترک با مقدار اولیه صفر وجود دارد (متغیر مثلا با نام Lock). هنگامی که پردازشی می خواهد وارد ناهیه بهرانی خود شود، ابتدا Lock را آزمایش می کند اگر Lock=0 بود آن را برابر "1" کرده و وارد ناهیه بهرانی می شود ولی اگر Lock=1 بود، باید در یک حلقه منتظر بماند تا Lock برابر صفر شود. بنا براین "0" به این معناست که هیچ پردازشی در ناهیه بهرانی نیست و "1" به این معناست که پردازشی در ناهیه بهرانی اش قرار دارد. کد زیر این روش را نشان می دهد، مقدار اولیه Lock برابر صفر است.

نکته: این روش شرط اصلی انحصار متقابل را ندارد



توضیح:

لظنه ای را تصور کنید که متغیر Lock برابر صفر است، پردازش  $P_0$  در حلقه While متغیر Lock را چک می کند و چون برابر صفر است به سراغ خط بعدی می رود تا Lock را برابر "1" کند. ولی قبل از آنکه عدد "1" را در Lock بریزد، CPU به پردازش  $P_1$  سوئیچ می کند. در این حال  $P_1$  نیز متغیر Lock را برابر صفر می بیند و از حلقه While خارج می شود، آنگاه در

ادامه Lock را برابر "1" کرده و وارد ناهیه بهرانی خود می شود. حال دوباره پردازنده به  $P_0$  سوئیچ می کند، عدد "1" را در Lock ریخته و وارد ناهیه بهرانی  $P_0$  می شود. یعنی هر دو پردازش  $P_0$  و  $P_1$  همزمان در ناهیه بهرانی می باشند!

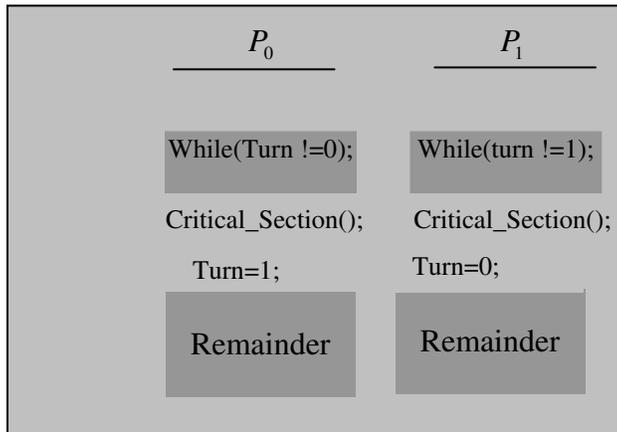
3- روش تناوب قطعی (Strict Alternation):

در این روش از یک متغیر نوبت استفاده می شود طوری که اگر دو پردازش  $P_1$  و  $P_2$  داشته باشیم بعد از این که یکی از پردازش ها وارد ناهیه بهرانی شد، تا این که پردازش دیگر وارد ناهیه بهرانی نشود، این پردازش نمی تواند مجددا وارد ناهیه بهرانی بشود.

```

P_i
Other = 1-i
While(Turn==Other);
Critical_Section();
Turn=Other;
Remainder Code
    
```

**نکته:** با این که این الگوریتم شرط انحصار متقابل را دارد ولی شرط پیشرفت در آن برقرار نیست



**توضیح:** حالتی را در نظر بگیرید که پردازنده  $P_0$  وارد ناحیه بحرانی می شود که هنگام خروج از ناحیه بحرانی مقدار Turn را برابر 1 قرار می دهد و در این موقع پردازنده  $P_1$  می تواند وارد ناحیه بحرانی شود، پردازنده  $P_1$  پس از خروج از ناحیه بحرانی Turn را برابر صفر قرار می دهد، فرض کنید پردازنده  $P_1$  در Remainder Code باشد اگر در همین اثنا پردازنده  $P_0$  بخواهد وارد ناحیه بحرانی بشود چون Turn برابر صفر می باشد، وارد ناحیه بحرانی می شود این پردازنده پس از خروج از ناحیه بحرانی Turn را برابر 1 می کند. حال اگر قسمت Remainder Code مربوط به  $P_1$  طولانی باشد، اگر  $P_0$  بخواهد مجددا وارد

ناحیه بحرانی بشود، نمی تواند زیرا Turn برابر با 1 است، پس پردازنده  $P_1$  که در قسمت Remainder Code خود قرار دارد در تصمیم گیری ورود به ناحیه بحرانی دقالت دارد و این یعنی نقض شرط پیشرفت

#### 4- استفاده از دستور سفت افزاری (TSL):

**دستورات اتمیک:** دستوراتی هستند که تیزی نا پذیرند (وقفه ناپذیر)

Mov Ax, Count اتمیک است

Count++ اتمیک نیست

در این روش از دستور اتمیک TSL استفاده می شود که طرز کار این دستور به صورت زیر است

پردازنده ها در هنگام ورود به ناحیه بحرانی دستور TSL را فراخوانی می کنند، که اگر مقدار Lock مخالف صفر باشد در حلقه گیر کرده و منتظر می مانند و اگر مقدار Lock صفر باشد، پردازنده ای که این دستور را اجرا کرده مقدار Lock را صفر می بیند و وارد ناحیه بحرانی می شود (در این لحظه مقدار Lock برابر 1 " می شود)، اگر در همین عین پردازنده دیگری TSL را اجرا کند، چون مقدار Lock برابر 1 " است نمی تواند وارد ناحیه بحرانی بشود، پردازنده ها پس از خروج از ناحیه بحرانی مقدار Lock را صفر می کنند تا دیگر پردازنده ها بتوانند وارد ناحیه بحرانی بشوند. (دقت شود که در اولین اجرا مقدار Lock برابر صفر است)

تابع Tsl را می توانیم به این شکل بنویسیم

```

     $P_i$ 
    While(1) {
    While(TSL(& Lock))
    Critical Section
    Lock=0 ;
    Rimainder Section
    }

```

```

Tsl(x){
Int temp;
Temp=x;
x=1;
return (temp);
}

```

پایان جلسه چهارم