جلسه چهارم

رابطه بین پردازه ها(Interprocess Communication)

پردازه ها از جهت وابستگی به یکدیگر در دو دسته قرار می گیرند و توجه به این مسئله در نحوه خاتمه آنها حائز اهمیت است

الف- پردازه های مستقل: هر پردازه ای که داده ای را (چه موقت و چه دائمی) با دیگر پردازه ها به اشتراک نگذارد مستقل است و یا به عبارت دیگر یک پردازه در صورتی مستقل از دیگر پردازه ها است که شرایط زیر را برآورده کند.

1- حالت آن به اشتراک گذارده نشده باشد.

2- نتیجه اجرای آن کاملا مشخص باشد.

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

4- توقف و شروع آن بدون تاثیر بر سایر پردازه ها امکان پذیر باشد.

ب- پردازه های همکاری کننده: اگر دو پردازه نسبت به هم یکی از شرایط فوق را نداشته باشد گوئیم پردازه ها نسبت به هم همکارند به عبارت دیگر پردازه هائی هستند که شرایط زیر در مورد آنها صادق باشد.

1- حالت آن به اشتراک گذارده شود

2- نتیجه اجرا به دلیل وابستگی به سایر پردازه ها کاملا مشخص نباشد

3- نتیجه اجرا برای ورود ی های یکسان در دفعات مختلف، یکسا ن نباشد

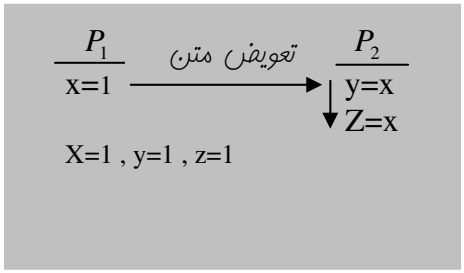
منظور از اجرای Interleaved چیست؟

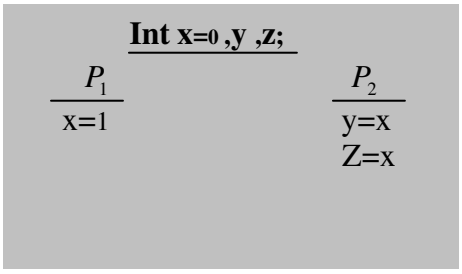
به این معناست که در بین اجرا ی یک پردازه در هر زمان امکان وقفه یا تعویض متن (Context Switching) به پردازه دیگر وجود دارد

مثال: اگر دو پردازه P_1 و P_2 به صورت همروند اجرا شوندمقادیر نهائی x ، y و z را بیابید

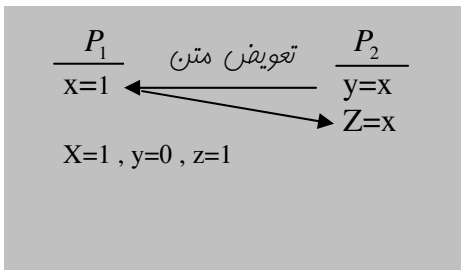
حل:در اجرای همروند این دو پردازه سه ترتیب زیر امکان پذیر است

الف-

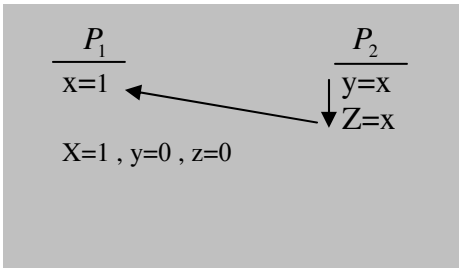




ب-



ج-

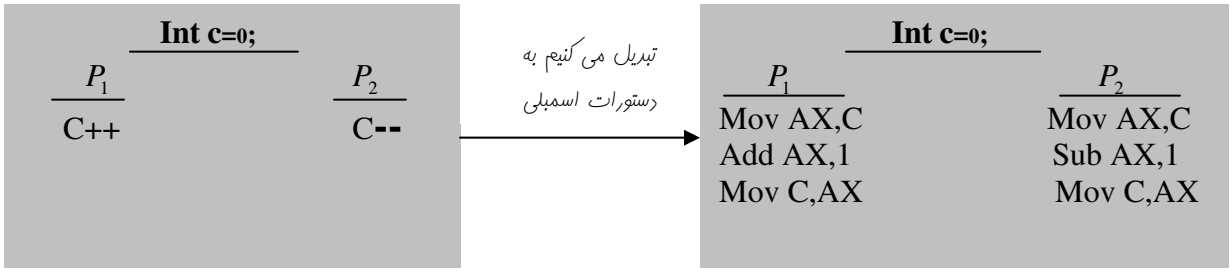


Pnu-Soal.ir

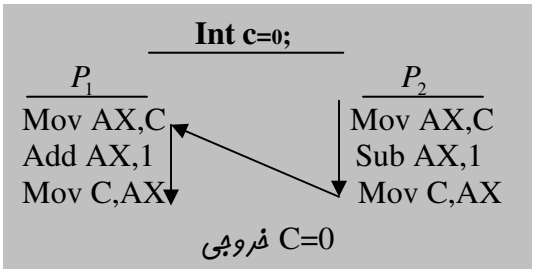
منظور از همگام سازی پردازه ها(Synchronization) چیست؟

زمانی که پردازه 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 پردازه ها را مشخص کرد که به این مسئله همگام سازی پردازه ها گویند

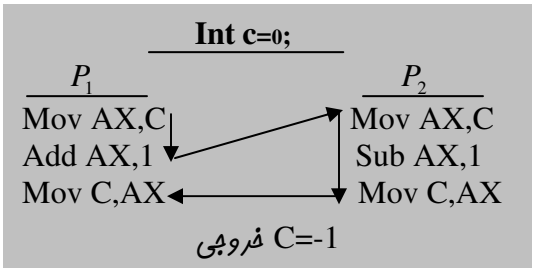
مثال: اگر دو پردازه P_1 و P_2 به صورت همروند اجرا شوندمقادیر نهائی c بیابید.



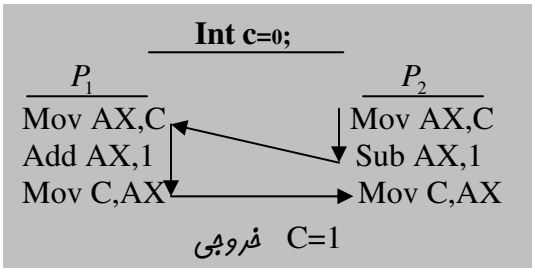
الف-



ب-



ج-



S.Hemmati 2006

به علت اینکه هر دو پردازه از متغیر مشترک C استفاده کرده اند، مقدار نهائی C وابسته به ترتیب اجرای پردازه 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 وضیعت مسابقه یا Race Condition چیست؟

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

نواحی 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 Eclusion نیاز داریم، به عبارتی دیگر اگر یکی از پردازه ها در حال استفاده از داده مشترک است باید مطمئن باشیم که دیگر پردازه ها، در آن زمان از انجام همان کار محروم می باشند.

- بخشی از برنامه که به حافظه اشتراکی دسترسی دارد را قسمت یا ناحیه بحرانی (Critical Section) می نامیم.
- هر پردازه برای ورود به بخش بحرانی اش باید اجازه بگیرد. بخشی از کد پردازه 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 پردازه های دیگر بتوانندوارد ناحیه بحرانی شان بشوند. بقیه کد پردازش را بخش باقی مانده یا remainder section گوئیم. بنا براین ساختار کلی پردازش ها به صورت زیر می باشد.

```
While(True){
        Entry section
        Critical_section();
        exit section
        Remainder_section();
}
```

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

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

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

2- شرط پیشرفت یا پیشروی(Progress):

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

S.Hemmati 2006

3- شرط انتظار مقید یا محدود(Bounded Waiting):یک برنامه منتظر ورود به ناحیه بحرانی،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 پردازه بلا فاصله پس ا ز ورود به ناحیه بحرانی اش کلیه وقفه ها را از کار بیندازد و درست قبل از خروج از ناحیه بحرانی 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 پردازه می تواند بدون ترس از مداخله دیگر پردازه ها به دستکاری قسمت مشترک بپردازد .

این روش دو مشکل دارد

1-ممکن است کاربر وقفه ها را خاموش کند ولی دوباره آنهارا 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 دستیابی پیدا کنند.

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

فرض کنید یک متغیر قفل یکتا ومشترک با مقدار اولیه صفر وجود دارد(متغیر مثلا با نام Lock). هنگامی که پردازشی می خواهد وارد ناحیه بحرانی خود شود، ابتدا Lock را آزمایش می کند اگر Lock=0 بود آن را برابر "1" کرده و وارد ناحیه بحرانی 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 ناحیه بحرانی نیست و "1" به این معناست که پردازشی در ناحیه بحرانی اش قرار دارد. کد زیر این روش را نشان می دهد، مقدار اولیه Lock برابرصفر است.

نکته:این روش شرط اصلی انحصار متقابل را ندارد

	P_0	P_1
قسمت ورودی	While(Lock==1);	While(Lock==1);
	Lock=1;	Lock=1;
	Critical_Section();	Critical_Section();
قسمت خروجی	Lock=0;	Lock=0;
	⇓	⇓

توضیح:

لحظه 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 ناحیه بحرانی خود می شود . حال دوباره پردازنده به P_0 سوئیچ می کند، عدد "1" را در Lock ریخته و وارد ناحیه بحرانی P_0 می شود. یعنی هر دو پردازش P_0 و P_1 همزمان در ناحیه بحرانی می باشند!

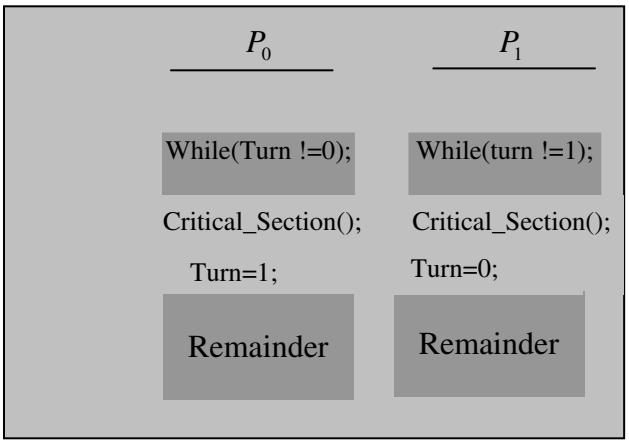
3-روش تناوب قطعی (Strict Alternation):

در این روش از یک متغیر نوبت استفاده می شود طوریکه اگر دو پردازه P_1 و P_2 داشته باشیم بعد از این که یکی از پردازه ها وارد ناحیه بحرانی شد، تا این که پردازه دیگر وارد ناحیه بحرانی نشود، این پردازه نمی تواند مجددا وارد ناحیه بحرانی بشود .

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


S.Hemmati 2006

نکته: با این که این الگوریتم شرط انحصار متقابل را داردولی شرط پیشرفت در آن برقرار نیست



توضیح: حالتی را در نظر بگیرید که پردازه 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 پردازه P_1 می تواند وارد ناحیه بحرانی شود، پردازه P_1 پس از خروج از ناحیه بحرانی Turn را برابر صفر قرار می دهد،فرض کنید پردازه P_1 در Remainder Code باشد اگر در همین اثنا پردازه P_0 بخواهد وارد ناحیه بحرانی بشود چون Turn برابر صفر می باشد، وارد ناحیه بحرانی می شود این پردازه 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 پردازه P_1 که در قسمت Remainder Code خود قرار دارد در تصمیم گیری ورود به ناحیه بحرانی دخالت دارد و این یعنی نقض شرط پیشرفت

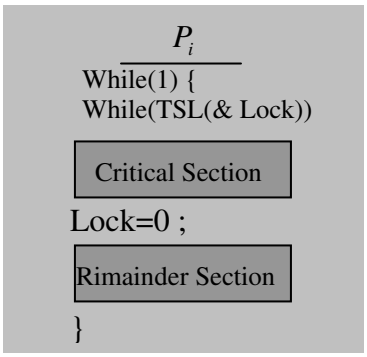
4- استفاده از دستور سخت افزاری(TSL):

دستورات اتمیک: دستوراتی هستند که تجزیه 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



پردازه 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، پردازه 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 حین پردازه ی دیگری TSL را اجرا کند ، چون مقدار Lock برابر "1 " است نمی تواند وارد ناحیه بحرانی بشود، پردازه ها پس از خروج از ناحیه بحرانی مقدار Lock را صفر می کنند تا دیگر پردازه 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

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

پایان جلسه چهارم

Pnu-Soal.ir