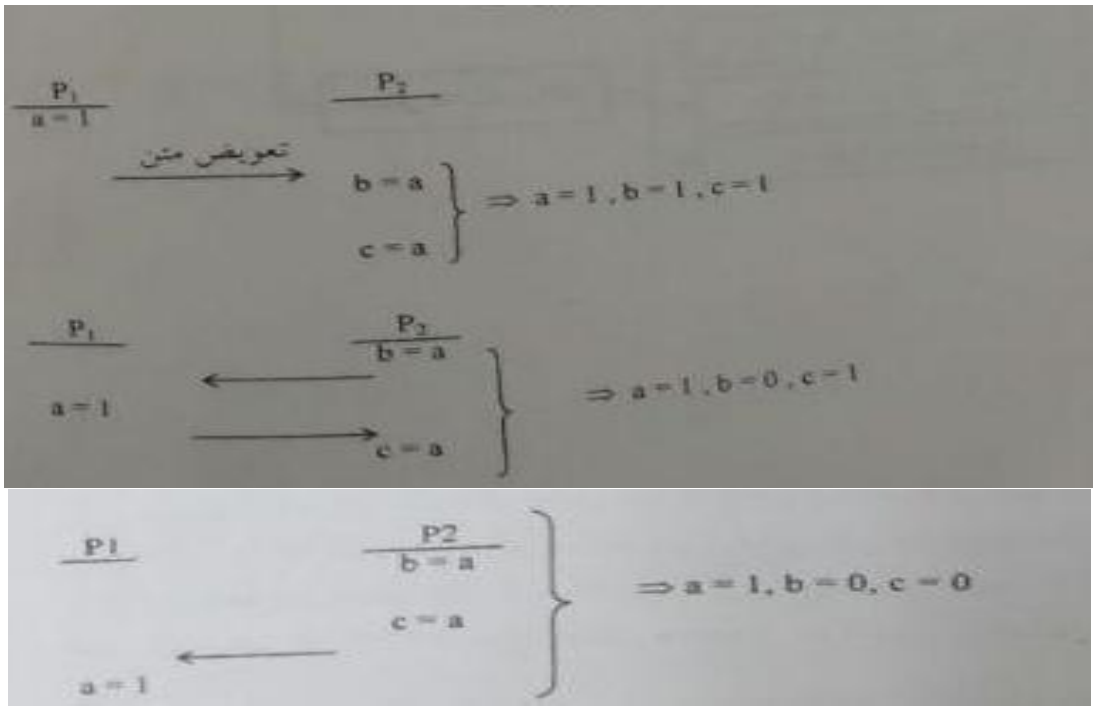
# مفهموم همزمانی پردازش ها

یک پردازش همکار(cooperating) پردازشی است که میتواند به سایر پردازش های در حال اجرا اثر کند یا از انها متاصر گردد. دو پردازه ا مستقل از هم هستند و یا همکار میباشند. پردازه های مستقل هیچ اثری بر هم ندارند و به هر ترتیبی که اجرا شوند خروجی نهایی هردو همواره یکسان است. ولی دو پردازه همکار بر هم اثر دارند و اینکه cpu چگونه و به چه ترتیبی و در چه زمانهایی بین ان دو سوییچ میکند در ایجاد پاسخ نهایی تاثیر دارد مثال ساده این موضوع را نشان میدهد.

مثال 1: دو فرایند p1 و p2 زیر به صورت همروند (concurrent) اجرا میشوند و امکان اجرای انها به صورت interleaved به این معناست که در بین اجرای یک پردازه در هر زمان امکان تعویض متن (context switch) به پروسس دیگر وجود دارد. در صورتی که مقدار اولیه متغییر سراسری a صفر باشد بد از اجرای کامل دو فرایند، بعد از اجرای کامل دو فرایند مقادیر c,b,a چه خواهد شد؟

| کد p1 | کد p2 |
| --- | --- |
| a =1 | b = a |
| | c = a |

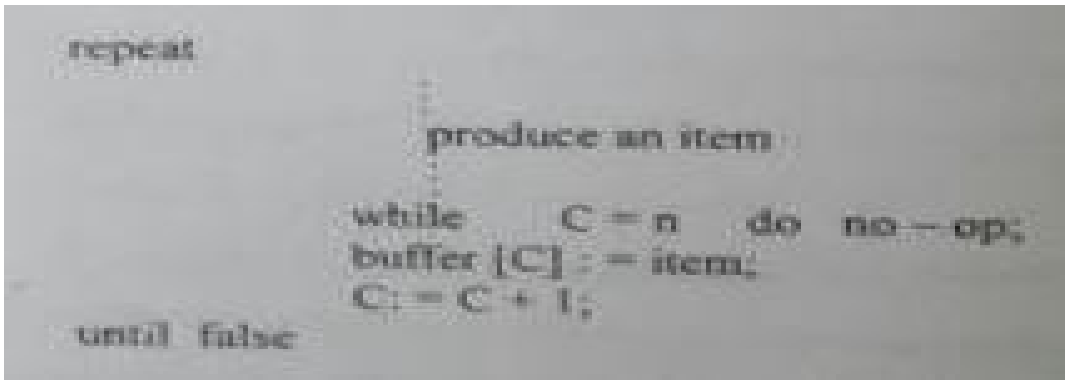
حل : در اجرای همروند این دو پردازه سه ترتیب زیر امکان پذیر است



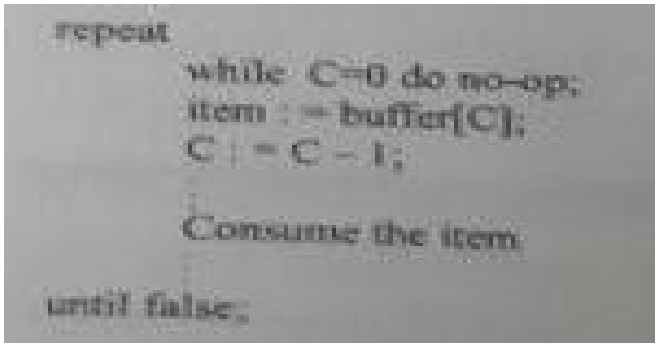

با توجه به مثال فوق مسخص می گردد که بحث ارتباط بین پروسس ها یا ( interprocessor communication) IPC یکی از مباحث مهم در درس سیستم عامل میباشد. هنگامی که بین پردازه ها وابستی وجود دارد ترتیب صحیح انجام کارها (یعنی مساله همگام سازی یا synchronization پروسس ها) اهمیت زیادی پیدا میکند. مثلا اگر فرایند A داده ای را تولید میکند که فرایند B باید ان را چاپ کند پردازه B باید تا پردازه A مقداری داده برایش فراهم کرده و بعد شروع به چاپ انها کند. در زیر نمونه دیگری را شرح می دهیم.

مثال 2: بافری که به کمک یک ارایه n خانه ای پیاده سازی شده است را در نظر بگیرد. دو پردازه بر روی این بافر نوسته شده است که هر دو موازی با هم اجرا میگردند. یکی پردازش تولید کننده که داده هایی را تولید کرده و در بافر می ریزد و یکی پردازش مصرف کننده که داده ها را از بافر برداشته و دستکاری میکند. هر دو این پردازنده ها درحقه دائم می باشند. نمونه ای از کد این پردازش ها را در زیر اورده ایم.

کد پردازش تولید کننده:




کد پردازش مصرف کننده:




فرض کنید مقدار شمارنده C در حال حاضر برابر 5 باشد و پردازش های تولید کننده و مصرف کننده جملات C=C-L و C=C+L راهمزمان اجرا کنند پس ا اجرای این دو عبارت متغیر C ممکن است 4 یا 5 یا 6 گردد ولی نتیجه درست فقط مقدار 5 می باشد. روند زیر نمونه ای از محاسبه اشتباه C می باشد فرض کنید دستور C=C+1 در سطح اسمبلی به صورت سه دستور زیر اجرا گردد:

MOV AX.C

ADD AX.1

MOV C.AX

به همین ترتیب دستور :C=C-1 در سطح اسمبلی مثلا به صورت سه دستور زیر اجرا می شود:

MOV BX.C

SUB BX.1

MOV C.BX

حال هنگامی که CPU بین پردازش های تولید کننده و مصرف کننده سوییچ میکند ممکن است ترتیب و روند زیر اجرا می گردد

$T_1$ = از پردازش تولید کنند {AX = 5} MOV AX.C

$T_2$ = از پردازش تولید کننده {AX = 6} ADD AX.1

$T_3$ = از پردازش مصرف کننده {BX = 5} MOV BX.C

$T_4$ = از پردازش مصرف کننده {BX = 4} SUB AX.1

$T_5$ = از پردازش تولید کننده {C = 6} MOV C.AX

$T_6$ = از پردازش مصرف کننده {C = 4} MOV C.BX

نهایتا متغیر C برابر مقدار نادرست 4 می گردد به همین ترتیب اگر جای مراحل $T_5$ و $T_6$ را عوض کنیم به جهات نادرست C=6 می رسیم دلیل رسیدن به این حالات نادرست ان است که اجازه دادیم هر دو فرایند به متغیر C به طور همزمان دسترسی داشته باشند.

به وضعیت های مشابه مثال فوق که در آن فرایند های متعددی داده یکسانی را به طور همروند دستیابی و همکاری میکنند میکند و حاصل اجرا بستگی به ترتیب خاص دسترسی ها دارد، وضعیت مسابقه یا RACE CONDITIONE 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 اید. به عبارت دیگر ما به ((انحصار متقابل)) یا Mutually exclusive نیاز داریم به عبارت دیگر اگر یکی از پردازه ها در حال استفاده از داده مشترک است باید مطمئن باشیم که دیگر پردازه ها در ان 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. اگر ترتیبی را فراهم کنیم که هیچ دو پردازه ای در یک زمان در ناحیه بحرانی خود وارد نشوند از شرایط رقابتی اجتناب کرده ایم. به عبارت دیگر وقتی یک پردازش در حال اجرای بخش بحرانی اش است، هیچ پردازه دیگری مجاز نیست در بخش بحرانی خود اجرا گردد.

هر پردازه برای ورود به بخش بحرانی اش باید اجازه بگیرد، بخشی از کد پردازه که این اجازه گرفتن را پیاده سازی میکند بخش ورودی یا entry section نام دارد. بخش بحرانی میتواند با بخش خروجی یا exit section دنبال شود. این بخش خروجی کاری میکند که پردازه های دیگر بتوانند وارد ناحیه بحرانی شن بشوند. بقیه کد پردازش را بخش باقی مانده یا reminder section گوییم. بنابراین ساختار کلی پردازش ها به صورت زیر میباشد:

While (TRUE) (

Entry section

Critical_section ( ):

Exit section

Reminder_section ( ):

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

1. شرط انحصاری متقابل (ماتعه الجمعی mutual excusion): هنگامی که پردازشی درناحیه بحرانی اش اجرا میگردد، 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 تقاضا هایی برای ورود به بخش بحرانی وجود دارد، فقط پردازشهایی در تصیم گیری برای ورود دخالت میکنند که هنوز به ناحیه بحرانی شان نرسیده باشند. به عبارت دیگر اگر پردازشی در قسمت باقی مانده (reminder) خود باشد در تصمیم گیری اینکه کدام پردازش وراد بخش بحرانی ود، شرکت داده نمی شود. به عبارت دیگر هیچ پردازشی نباید از بیرون ناحیه بحرانی خود امکان بلوکه کردن پردازشهای دیگر را داشته باشد.

3. شرط انتظار مقید یا محدوده (bounded waiting) یک برنامه منتظر ورود به ناحیه بحرانی، نباید به طور نا محدود 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 تعدادCPU ها نمی کنیم.

**نکته** : اگر پردازشی در حال اجرا و پردازش دیگری بلوکه یا اماده اجرا باشد و هر دوی انها در ناحیه بحرانی خود باشند، شرط انحصار متقابل رعایت نشده است بنابراین هنگام بحث درباره شرط ماتعه الجمعی وضعیت پردازش ها مهم نمی باشد.

## 1. از کار انداختن وقفه ها

ساده ترین راه آن است که هر پردازه بلافاصله پس از ورود به ناحیه بحرانی اش کلیه وقفه ها را از کار بیاندازد و درست قبل از خروج از ناحیه بحرانی دوباره همه انها را فعال کند. با خاموش ساختن وقفه ها CPU به هی عنوان نمی تواند از پردازشی به پردازش دیگر سوئیچ کند. بنابراین هنگامی که

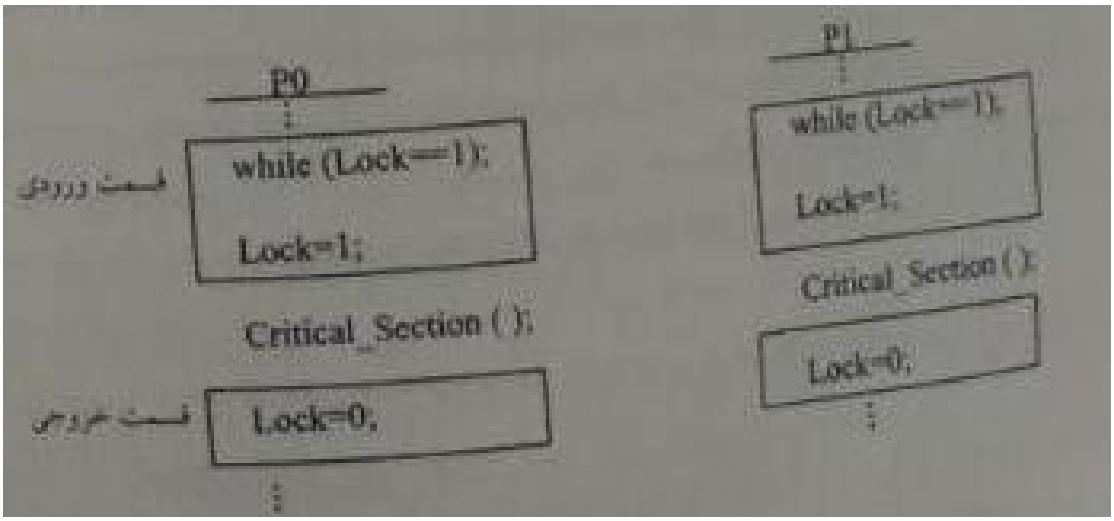
یک پردازه وقفه ها را غیر فعال می کندمی تواند بدون هیچ مشکلی و بدون ترس از مداخله دیگر پردازه ها به دستکاری قسمت مشترک بپردازد ولی این روش دو مشکل دارد یکی انکه ممکن است کاربر وقفه ها را خاموش کند ولی دوباره انها را فعال سازد. بدین ترتیب سیستم از کار خواهد افتاد پس اعطای قدرت غیر فعال ساختن وقفه ها به پردازش کاربران عاقلانه نیست از طرف دیگر در سیستم های چند پردازنده ای غیر فعال ساختن وقفه ها، فقط در CPU ی اثر دارد که دستور از کار انداختن وقفه ها را اجرا میکند بقیه CPU ها میتوانند به کار خودشان ادامه داده و به حالت مشترک دستیابی پیدا کنند ولی از کار انداختن وقفه ها برای تعداد کمی از دستورات برای خود هسته سیستم عامل مناسب می باشد.

پس، از کار انداختن وقفه ها غالبا در خود سیستم عامل استفاده می شود و به کار گیری ان برای پردازشهای کاربران مناسب نمی باشد.

## 2. استفاده از متغیر های قفل (Lock varisbles)

دومین راهی که ممکن است به ذهن خطور کند روش کاملا نرم افزاری با استفاده از متغیر هـای قفـل مـی باشد. فرض کنید یک متغیر قفل یکتا و مشترک با مقدار اولیه صفر وجود دارد (متغیری مثلا با نـام Lock) هنگامی که پردازش میخواهد وارد ناحیه بحرانی خود شود ابتدا Lock را ازمایش میکند.

اگر Lock=d بود انرا برابر "1" کرده و وارد ناحیه بحرانی میشود ولی اگر Lock = 1 بود، باید در یک حلقه منتظر بماند تا Lock برابر صفر شود. بنابراین "0" به این معناسـت کـه هـیچ پردازشـی در ناحیـه بحرانـی نیست و "1" به این معناست که پردازشی در ناحیه بحرانی اش قرار دارد کد زیر این روش را نشان میدهـد، مقدار اولیه Lock برابر صفر است.




این روش با انکه به نظر خیلی ساده است ولی شرط اصلی انحصار متقابل را ندارد در واقع در اینجا برای رفع مشکل شرایط رقابتی و حفاظت از ناحیه بحرانی یـک ناحیـه بحرانـی دیگـر یعنـی متغیـر مشـترک Lock استفاده شده است، امکان زیر را در نظر بگیرید:

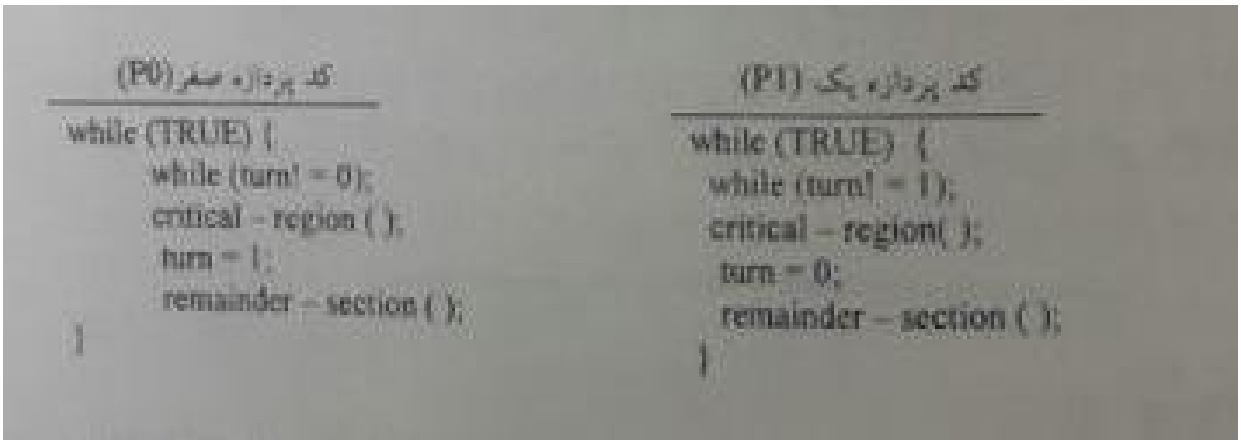
پردازش p0 در حلقه while متغیر Lock را چک میکند و چون برابر صفر است به سراغ خط بعـدی مـی رود Lock 0 را برابر 1 کند. ولی قبل از انکه عدد 1 را درLock بریزد، CPU بـــه پـــردازش P1 سوییچ میکند. در ایـن حالـت P1 نیـز متغیـر Lock را برابـر صـفر مـی بینـد و از حلقـه while خـارج میشود،انگاه در ادامه Lock را برابر "1" کرده و وارد ناحیه بحرانی خود میشود حال دوباره پردازنده به p0 سوییچ میکند، عدد"1" را در Lock ریخته و وارد ناحیه بحرانی p0 میشود. یعنـی هـردو پـردازش p0 و p1 همزمان در ناحیه بحرانی می باشند!

نتیجه انکه این روش اصلا مناسب نیست و در ادامه سعی می کنیم این مشکل را برطرف سازیم.

## 3. تناوب قطعی (strict alternation)

دو پردازش با شماره های 0 و 1 را در نظر بگیرید که هر دو از یک متغیر مشترک به نام turn با مقدار اولیه صفر استفاده می کند. این متغیر نوبت پردازش ها را برای ورود به ناحیه بحرانی معین می سازد در ابتدا پردازه صفر این متغیر را بررسی کرده و ون برابر صفر است وارد ناحیـه بحرانـی اش مـی شـود. در اینجا چون پروسس 1ان را مخالف 1 می یابد، در یک حلقه کوچک منتظر باقی می ماند تا وقتی کـه turn برابر 1 گردد.

عمل مداوم تست کردن یک متغیر تا زمانی که حاوی یک مقدار مشـخص گـردد Busy wating یـا انتظار مشغول نامیده میشود که حتی الامکان باید از ان اجتناب کرد چرا که وقت CPU را هدر میدهد بنابر این از این روش هنگامی استفاده می شود که انتظار کوتاه باشند:




در این روش بر خلاف روش متغیر های قفل (که مقدار اولیه متغیر همواره بود). مقدار اولیه turn می تواند صفر یا یک باشد. در واقع مقدار اولیه turn اولیت ورود پردازشها به ناحیه بحرانی را مشخص می سازد مثلا اگر مقداز اولیه Turn برابر یک باشد انگاه ابتدا پردازش p1 وارد ناحیه بحرانی می شود یا اینکـه الگـوریتم فوق متقابل را دارد ولی شرط پیشرفت در ان بر قرار نیست