سمافورها می توانند سبب بن بست شوند دو پردازه p_0 , p_1 را در نظر بگرید.
فرض کنید p_0 دستور $p(a);$ را اجرا کرده باشد، در همین لحظه عمل تعویض متن رخ دهد و پردازه p_1 دستور $p(b);$ را اجرا کند، حال اگر پرداز p_1 دستور $p(a);$ را اجرا کند، یا پردازه p_0 دستور $p(b);$ را اجرا کند، هر دو بلاک می شوند(هر دو منتظر هم باقی می مانند)، در این حالت می گوئیم بین پردازه ها بن بست رخ داده است، پس سمافور ها می توانند سبب بن بست شوند.

```
Semaphor a=1,b=1;
P0 :                p1 :
p(a);               p(b);
p(b);               p(a);
⋮                   ⋮
v(a);               v(b);
v(b);               v(a);
```

7- مانیتور(monitor)**:**

سمافور ها با همه مزایائی که دارند،پیچیده می باشند به عبارتی، در استفاده از سمافورها می بایست خیلی دقت کرد، زیرا یک دستور v اضافی سبب مشکل می شود. مانیتور ها ابزارهای سطح بالائی هستند که این مشکل را ندارند، مانیتور یک ساختمان داده ای شامل توابع،متغیر های معمولی، متغیر های شرطی می باشند، که این متغیر های شرطی فقط از طریق دستورات **wait** و **signal** قابل دستیابی هستند، اگر پردازه ای در داخل مانیتور دستور **wait** را صادر کند، آن پردازه در لیست پردازه های آن متغیر شرطی قرار گرفته، و بلاک می شود و پردازه های دیگر می توانند وارد مانیتور شوند، یکی از توابع داخل مانیتور را اجرا کنند. در هر لحظه فقط یک پردازه می تواند در داخل مانیتور باشد. یعنی فقط یک پردازه می تواند در هر آن در حال اجرای توابع داخل مانیتور باشد.

- متغیر های شرطی، مقدار دهی نمی شوند و مقدار نمی گیرند.
- دستور **signal** سبب می شود اگر پردازه ای در صف بلاک آن متغیر شرطی باشد، بیدار شود و وارد مانیتور شده و اجرایش را از محلی که قبلا بلاک شده بود، ادامه دهد.
- کنترل ورود به ناحیه بحرانی را به هنگام استفاده از مانیتور، کامپایلر انجام می دهد، به طوری که کامپایلر اجازه نمی دهد بیش از یک پردازه در مانیتور اجرا شود، پس بهتر است نواحی بحرانی در مانیتور نوشته شود.

مسئله تولید کننده-مصرف کننده با استفاده از مانیتور

```
Monitor Tolid - Masraf
Condition full , empty
garardadan ( ){
if (count == max)
wait ( full );
Buffer [count ]= item ;
count ++;
if (count ==1)
signal (empty )
}

Bardash tan( ){
if (count == 0)
wait (empty );
count --;
item = Buffer [count ];
if (count == max -1)
signal ( full );
}

producer ( ){
while (1){
produce an item ;
tolid - masraf .garardadan ()
⋮
}}

consumer ( ){
while (1){
Tolid - masraf .bardash tan();
consum item ;
⋮
}}
```

☐ مانیتور را می بایست حتما کامپایلر پشتیبانی کند، ولی برای استفاده سمافور لزومی به پشتیبانی از طرف کامپایلر نیست، فقط کافی است سیستم عامل سمافور را پشتیبانی کند، که در این صورت می توان با استفاده از دستورات p,v پیاده سازی کرد.

بن بست پردازه ها(Process Deadlocks)

اگر مجموعه ای از پردازه ها در سیستم منتظر وقوع حادثه ای باشند که توسط دیگری انجام شود و هیچ کاری در سیستم پیش نرود، گوئیم سیستم دچار بن بست شده است.

شرایط وقوع بن بست: برای رخ دادن یک بن بست هر چهار شرط زیر باید برقرار باشند.

1- انحصار متقابل(Mutual Exclusion)

2- گرفتن و منتظر ماندن(Hold and wait)

3-عدم پس گرفتن(انحصاری بودن)(No preemption)

4- انتظار چرخشی(Circular Wait)

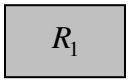
انحصار متقابل بدین معناست که منبع یا منابع درهر آن، فقط توسط یک پردازه قابل استفاده باشند.

گرفتن و منتظر ماندن یعنی پردازه یک سری منابع مورد نیازش را در اختیار گرفته، و منتظر منابعی است که در اختیار دیگر پردازه ها است

عدم پس گرفتن بدین معناست که به اجبار نمی توان منبع یا منابعی را از پردازه پس گرفت.

انتظار چرخشی یعنی بایستی مجموعه ای از پردازش ها $\{p_0, p_1, \cdots p_n\}$وجود داشته باشد به طوری که p_0 منتظر منبعی از p_1, p_1 منتظر منبع از p_2 و ... p_n منتظر منبعی از p_0 باشد.

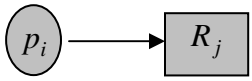
روش توصیف بن بست: استفاده از گراف تخصیص منابع(Resource Graph) $G(V,E) \leftarrow$

منابع = R= مستطیل 

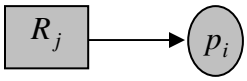
پردازه =P= دایره 

در گراف تخصیص منابع، گره ها از نوع منابع یا پردازه اند، که منابع با مستطیل نمایش داده می شوند و تعداد نمونه های آن با نقطه داخل آن مشخص می شوند و پردازه ها با دایره مشخص می شوند.

یال های گراف تخصیص منابع جهت دار می باشند که یا از پردازه به منابع می باشد(بدین معناست که پردازه p_i منتظر منبع R_j است) به این شکل.



یا از منبع به پردازه می باشد(بدین معنا است که یک نمونه از منبع R_j در اختیار پردازه p_i می باشد) به این شکل.



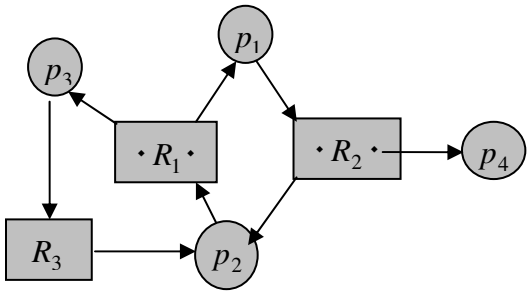
☐می توا اثبات کرد که اگر گراف دارای هیچ سیکلی(حلقه یا $loop$) نباشد، هیچ پردازشی در سیستم در بنبست نخواهد بود.

☐ اگر گراف دارای سیکلی باشد، احتمال دارد بن بست وجود داشته باشد. پس وجود حلقه در گراف شرط لازم برای بن بست است و نه شرط کافی

☐ اگر در گراف هر منبع دقیقا یک نمونه داشته باشد،آنگاه اگر گراف حلقه داشته باشد، بدین معناست که حتما بن بست رخ داده است، ولی اگر هر نوع منبع نمونه های متعددی داشته باشد، انگاه حلقه الزاما به معنای وقوع بن بست نیست

☐ اگر در گراف حلقه ای وجود نداشته باشد، آنگاه سیستم در حالت بن بست نیست

مثال: آیا گراف زیر در بن بست قرار دارد.

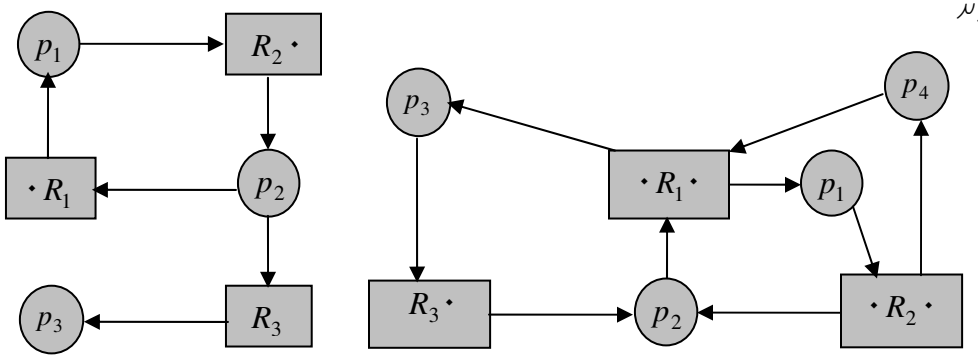


حل: خیر

توضیح:در شکل گفته شده در گراف حلقه وجود دارد ولی سیستم(پردازه ها) در بن بست نیستند، زیرا پردازه p_4 به منابع دیگری نیاز ندارد و بنابراین اجرایش را به اتمام رسانده و منبع R_2 را رها می سازد با رها ساختن منبع R_2 بن بست شکسته میشود. منبع R_2 به پردازه ی p_1 تخصیص داده میشود پردازه P_1 تمام منابع موردنیازش را در اختیار دارد، اجرایش را به اتمام رسانده و منابع را آزاد می سازد. در این حال منبع R_1 را به پردازه p_2 داده، پردازه p_2 نیز با در اختیار داشتن تمام منابع مورد نیازش، اجرایش را به پایان رسانده و تمام منابع را آزاد می سازد. که یکی از منابع R_3 است. منبع R_3 به پردازه p_3 تخصیص داده شده و این پردازه با در اختیار داشتن منابع اجرایش را به پایان می رساند.

مثال: آیا گراف های زیر در بن بست قرار دارند

حل:بله در هر دو بن بست وجود دارد.



فرایند استفاده از منبع:

1- در خواست منبع(Request): اگر پردازه ای به منبعی نیاز داشته باشد، دستور سیستمی درخواست منبع را می دهد، که اگر آن منبع آزاد باشد، به پردازه درخواست کننده تخصیص داده می شود، و گرنه پردازه می بایست منتظر منبع باشد

2- بکارگیری منبع(allocation): در صورت وجود منبع درخواست شده، منبع به پردازه درخواست کننده توسط سیستم تخصیص داده می شود.

3- آزاد کردن منبع(Release): پردازه پس از استفاده از منبع تخصیص یافته، منبع را به سیستم برمی رداند.

☐ در خواست منبع و آزاد کردن آن، توسط دستورات سیستم عامل(System Call) انجام می شود. مثلا برای فایل دستورات ***open*** و ***close*** و برای سمافور ***wait*** و ***signal*** انجام می شود. و همواره سیستم کلیه درخواست ها، تخصیص ها و آزاد شدن منابع را تحت نظر دارد

پایان جلسه ششم.