

Semaphor $a = 1, b = 1$;

P_0 :	P_1 :
$p(a)$;	$p(b)$;
$p(b)$;	$p(a)$;
\vdots	\vdots
$v(a)$;	$v(b)$;
$v(b)$;	$v(a)$;

سمافورها می توانند سبب بن بست شوند دو پردازش P_0, P_1 را در نظر بگیرید.
فرض کنید P_0 دستور $p(a)$ را اجرا کرده باشد، در همین لحظه عمل تعویض متن رخ دهد و پردازش P_1 دستور $p(b)$ را اجرا کند، حال اگر پردازش P_1 دستور $p(a)$ را اجرا کند، یا پردازش 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 پردازش ها بن بست رخ داده است، پس سمافور ها می توانند سبب بن بست شوند.

7- مانیتور (monitor):

سمافور ها با همه مزایائی که دارند، پیچیده می باشند به عبارتی، در استفاده از سمافور ها 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 سافتمان 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 پردازش ای در داخل مانیتور دستور $wait$ را صادر کند، آن پردازش در لیست پردازش های آن متغیر شرطی قرار گرفته، و بلاک می شود و پردازش های دیگر می توانند وارد مانیتور شوند، یکی از توابع داخل مانیتور را اجرا کنند. در هر لحظه فقط یک پردازش می تواند در داخل مانیتور باشد. یعنی فقط یک پردازش 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 پردازش ای در صف بلاک آن متغیر شرطی باشد، بیدار شود و وارد مانیتور شده و اجرائش را از مملى که قبلا بلاک شده بود، ادامه دهد.

☐ کنترل ورود به نامیه بهرانی را به هنگام استفاده از مانیتور، کامپایلر انجام می دهد، به طوری که کامپایلر اجازه نمی دهد بیش از یک پردازش در مانیتور اجرا شود، پس بهتر است نوامی بهرانی در مانیتور نوشته شود.

مسئله تولید کننده-مصرف کننده با استفاده از مانیتور

Monitor Tolid - Masraf

Condition full, empty

garardadan () {	Bardash tan () {	producer () {	consumer () {
if (count == max)	if (count == 0)	while (1) {	while (1) {
wait (full);	wait (empty);	produce an item;	Tolid - masraf .bardash tan();
Buffer [count] = item;	count --;	tolid - masraf .garardadan ()	consum item;
count ++;	item = Buffer [count];	:	:
if (count == 1)	if (count == max - 1)	}	}
signal (empty)	signal (full);		
}	}		

□ 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 پردازش ها (Process Deadlocks)

اگر مجموعه ای از پردازش ها در سیستم منتظر وقوع حادثه ای باشند که توسط دیگری انجام شود و هیچ کاری در سیستم پیش نرود، کوئیم 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 در هر آن، فقط توسط یک پردازش قابل استفاده باشند.

گرفتن و منتظر ماندن یعنی پردازش یک سری منابع مورد نیازش را در اختیار گرفته، و منتظر منابعی است که در اختیار دیگر پردازش ها است عدم پس گرفتن بدین معناست که به اجبار نمی توان منابع یا منابعی را از پردازش پس گرفت.

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

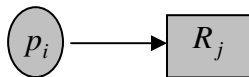
روش توصیف بن بست: استفاده از گراف تخصیص منابع (Resource Graph) $G(V, E) \leftarrow$

مستطیل $R =$ منابع 

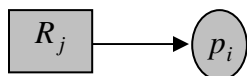
دایره $P =$ پردازش 

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

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



یا از منبع به پردازش می باشد (بدین معناست که یک نمونه از منبع R_j در اختیار پردازش p_i می باشد)



به این شکل.

□ می توان اثبات کرد که اگر گراف دارای هیچ سیکلی (حلقه یا loop) نباشد، هیچ پردازشی در سیستم در بن بست نخواهد بود.

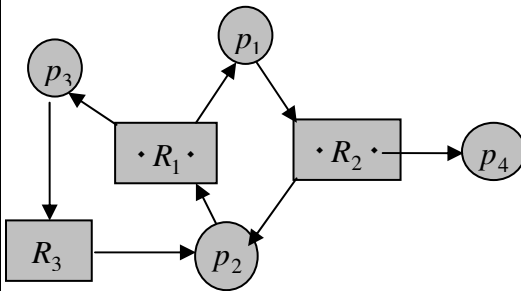
□ اگر گراف دارای سیکلی باشد، احتمال دارد بن بست وجود داشته باشد. پس وجود حلقه در گراف شرط لازم برای بن بست است و نه شرط کافی

□ اگر در گراف هر منبع دقیقاً یک نمونه داشته باشد، آنگاه اگر گراف حلقه داشته باشد، بدین معناست که حتماً بن بست رخ داده است.

ولی اگر هر نوع منبع نمونه‌های متعددی داشته باشد، آنگاه حلقه الزاماً به معنای وقوع بن بست نیست

□ اگر در گراف حلقه‌ای وجود نداشته باشد، آنگاه سیستم در حالت بن بست نیست

مثال: آیا گراف زیر در بن بست قرار دارد.



حل: خیر

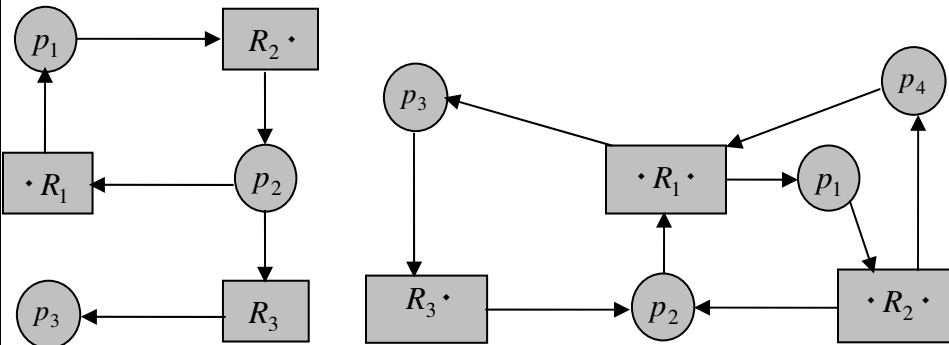
توضیح: در شکل گفته شده در گراف حلقه وجود دارد ولی سیستم (پردازه‌ها)

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 می‌سازد با رها ساختن منبع R_2 بن بست شکسته می‌شود. منبع R_2 به پردازه P_1 تفصیص داده می‌شود پردازه P_1 تمام منابع مورد نیازش را در اختیار دارد، اجرایش را به اتمام رسانده و منابع را آزاد می‌سازد. 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 می‌سازد. که یکی از منابع R_3 است. منبع R_3 به پردازه P_3 تفصیص داده شده و این پردازه با در اختیار داشتن منابع اجرایش را به پایان می‌رساند.

مثال: آیا گراف‌های زیر در بن بست قرار دارند

حل: بله در هر دو بن بست وجود دارد.



فهرایند استفاده از منبع:

1- در خواست منبع (Request): اگر پردازه‌ای به منبعی نیاز داشته باشد، دستور سیستمی درخواست منبع را می‌دهد، که اگر آن منبع آزاد باشد، به پردازه درخواست‌کننده تفصیص داده می‌شود، و گرنه پردازه می‌بایست منتظر منبع باشد

2- بکارگیری منبع (allocation): در صورت وجود منبع درخواست شده، منبع به پردازه درخواست‌کننده توسط سیستم تفصیص داده می‌شود.

3- آزاد کردن منبع (Release): پردازه پس از استفاده از منبع تفصیص یافته، منبع را به سیستم برمی‌رداند.

□ در خواست منبع و آزاد کردن آن، توسط دستورات سیستم عامل (System Call) انجام می‌شود. مثلاً برای فایل دستورات open

و close و برای سمافور wait و signal انجام می‌شود. و همواره سیستم کلیه درخواست‌ها، تفصیص‌ها و آزاد شدن منابع را تحت نظر دارد

پایان جلسه ششم.