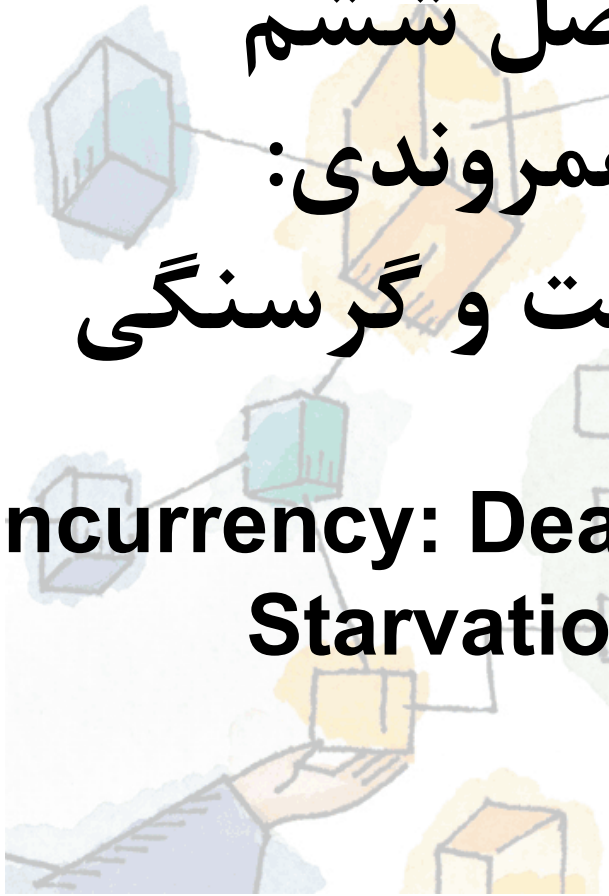
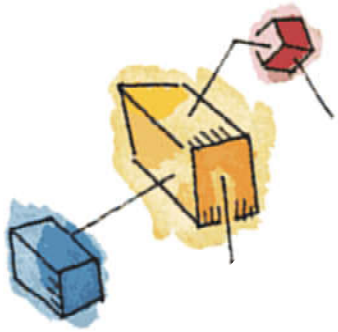
به نام خدا

فصل ششم
همروندی:
بن بست و گرسنگی

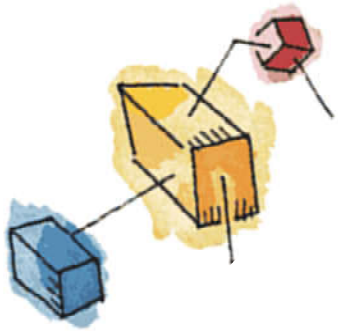
Concurrency: Deadlock and Starvation





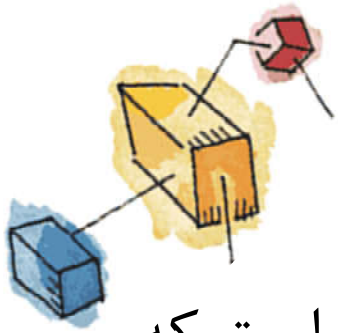
سرفصل مطالب

- **اصول بن بست**
- **راه های برخورد با مساله بن بست**
  - **پیش گیری از بن بست** (Deadlock Prevention)
  - **اجتناب از بن بست** (Deadlock Avoidance)
  - **کشف بن بست** (Deadlock Detection)



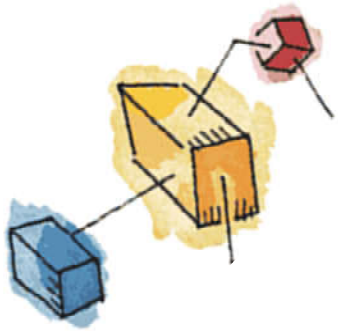
سرفصل مطالب

- **اصول بن بست**
- **راه های برخورد با مساله بن بست**
  - **پیش گیری از بن بست** (Deadlock Prevention)
  - **اجتناب از بن بست** (Deadlock Avoidance)
  - **کشف بن بست** (Deadlock Detection)



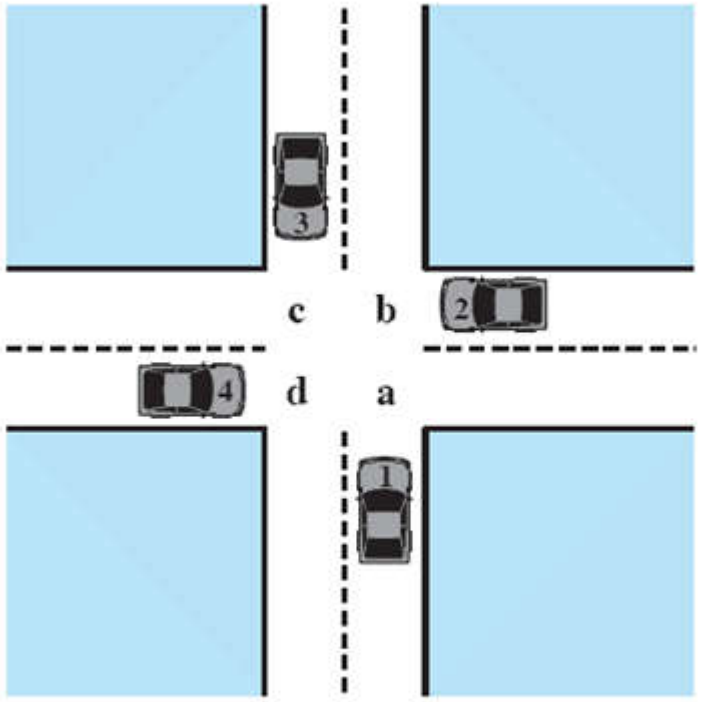
بن بست

- بن بست به معنی مسدود بودن دائمی مجموعه ای از فرآیندها است که یا برای به دست آوردن منابع سیستم رقابت می کنند و یا با یکدیگر در ارتباط هستند.
- بن بست همواره شامل تداخل نیاز دو یا چند فرآیند برای منابع سیستم است.
- هرگاه هر فرآیندی در مجموعه ای از فرآیندها در انتظار رویدادی باشد که تنها به وسیله فرآیندی دیگر از آن مجموعه قابل وقوع است، این مجموعه از فرآیندها در وضعیت بن بست قرار دارند.

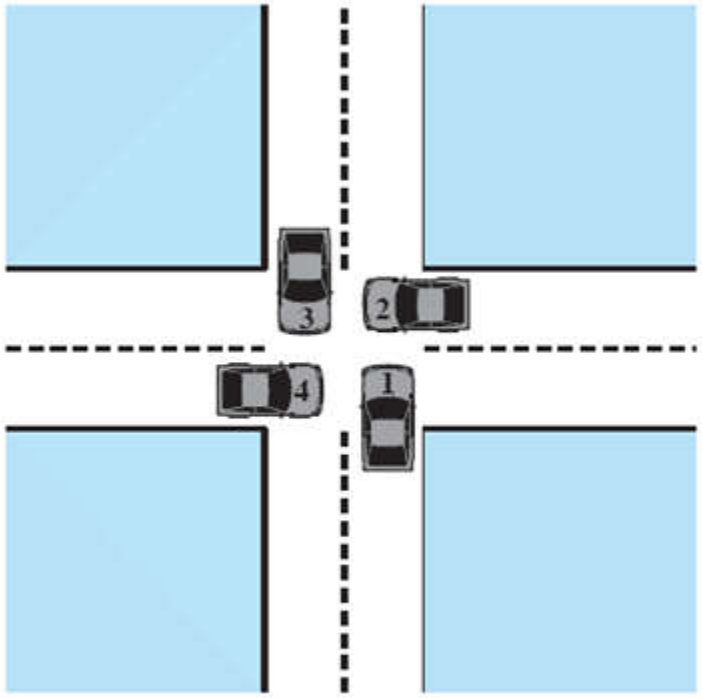


بن بست

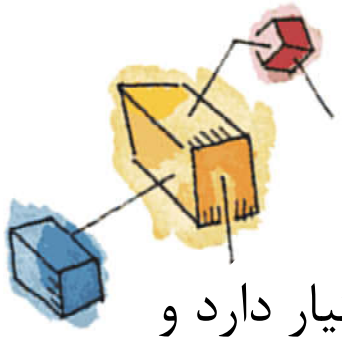
- یک مثال برای بن بست، ترافیک است.



(a) Deadlock possible



(b) Deadlock



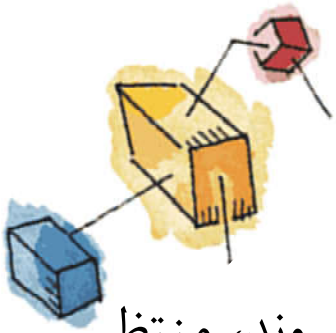
مثال هایی از بن بست

- مجموعه ای از فرآیندهای مسدود داریم که هر یک از آنها منبعی را در اختیار دارد و منتظر منبعی است که در اختیار فرآیند دیگری از آن مجموعه است.

- مثال ۱:
  - سیستم دارای دو درایو دیسک است.
  - هر یک از فرآیندهای P_1 و P_2 ، یکی از درایوهای دیسک را در اختیار دارد و درایو دیسک دیگر را نیاز دارد.

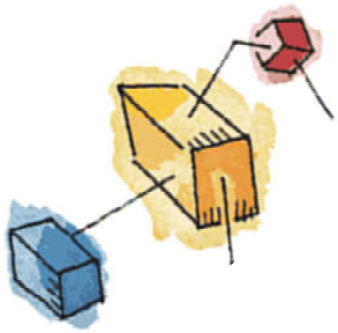
- مثال ۲:
  - سمافورهای A و B با مقدار ۱ مقدار دهی اولیه شده اند:

P_0	P_1
wait (A);	wait(B)
wait (B);	wait(A)



بن بست در فرآیند ها

- ساده ترین نوع بن بست زمانی اتفاق می افتد که دو فرآیند مجزا و همروند، منتظر بدست آوردن منبعی هستند که در اختیار دیگری است و خود نیز منابعشان را تا اتمام کار آزاد نمی کنند.



انواع منابع

- منابع نقش تعیین کننده ای در بن بست دارند.
- منابع به دو دسته تقسیم می شوند:
  - منابع قابل استفاده مجدد
  - منابع مصرف شدنی یا غیرقابل استفاده مجدد

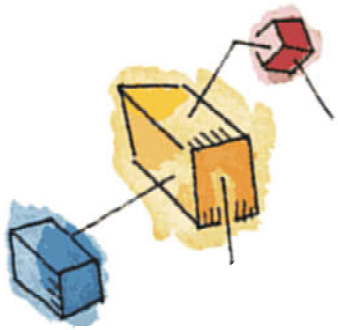
منابع قابل استفاده مجدد

- منابع قابل استفاده مجدد، منابعی هستند که در هر لحظه تنها توسط یک فرآیند قابل استفاده اند و استفاده از آنها موجب به پایان رسیدنشان نمی شود (بدون آسیب دیدن آزاد می شوند).

- فرآیندها منابع را بدست می آورند و سپس آنها را برای استفاده مجدد توسط سایر فرآیندها آزاد می کنند.

- پردازنده، حافظه اصلی و جانبی، دستگاه های I/O، و ساختمان داده هایی مثل فایل ها، پایگاه های داده و راهنماها از این دسته اند.

- چنانچه هر یک از فرآیندها یک منبع را در اختیار گرفته و منبع دیگری را درخواست کند که در اختیار دیگری است (مشابه مثال هایی که گفته شد)، بن بست رخ می دهد.



منابع قابل استفاده مجدد

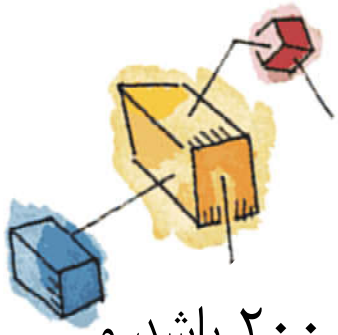
Process P

Step	Action
p_0	Request (D)
p_1	Lock (D)
p_2	Request (T)
p_3	Lock (T)
p_4	Perform function
p_5	Unlock (D)
p_6	Unlock (T)

Process Q

Step	Action
q_0	Request (T)
q_1	Lock (T)
q_2	Request (D)
q_3	Lock (D)
q_4	Perform function
q_5	Unlock (T)
q_6	Unlock (D)

Figure 6.4 Example of Two Processes Competing for Reusable Resources

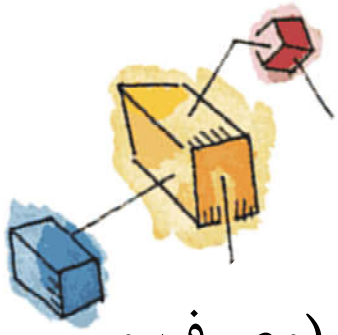


مثالی از بن بست منابع قابل استفاده مجدد

- فرض کنید فضای حافظه ای که می تواند به فرآیندها تخصیص یابد ۲۰۰ KB باشد و دنباله ای از رویدادها به صورت زیر رخ دهد.
- در این صورت چنانچه هر دو فرآیند تا درخواست دوم خود پیش روند، بن بست رخ می دهد.

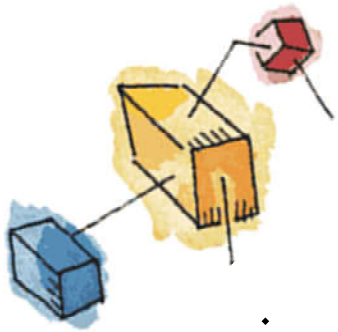
P1
. . .
Request 80 Kbytes;
. . .
Request 60 Kbytes;

P2
. . .
Request 70 Kbytes;
. . .
Request 80 Kbytes;



منابع مصرف شدنی

- منابع مصرف شدنی، ایجاد (تولید) می شوند و از بین می روند (مصرف می شوند).
- وقفه ها، علامت ها، پیام ها و اطلاعات بافر I/O از این نمونه اند.
- به عنوان مثال چنانچه دریافت پیام به صورت مسدود شونده باشد، ممکن است بن بست رخ دهد.
- کشف بن بست های حاصل از این منابع بسیار مشکل است و ممکن است ترکیب نادری از حوادث آنها را ایجاد کند.



مثالی از بن بست

- اگر receive ها به صورت مسدود شونده انجام شوند، بن بست رخ می دهد.

P1

```
. . .
Receive(P2);
. . .
Send(P2, M1);
```

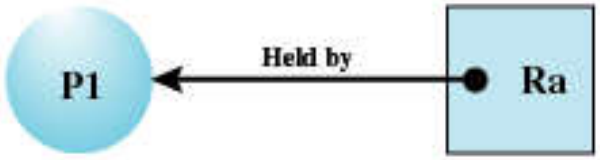
P2

```
. . .
Receive(P1);
. . .
Send(P1, M2);
```

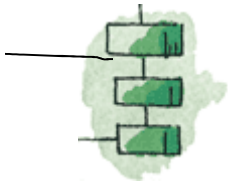
گراف تخصیص منابع

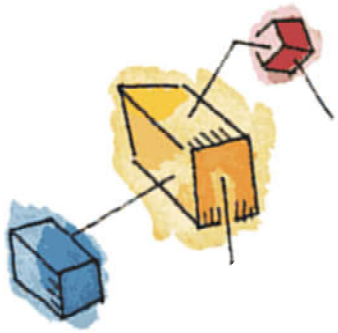
- گراف تخصیص منابع یک گراف جهت دار است که نحوه تخصیص منابع به فرآیندها را در هر لحظه از زمان نشان می دهد.
  - دایره ها: نشان دهنده فرآیندها
  - مربع ها: نشان دهنده منابع
  - تعداد نقاط داخل مربع، تعداد منابع از آن نوع را نشان می دهد.
- برای تشخیص بن بست باید گراف تخصیص منابع را بعد از هر درخواست، هر تخصیص، یا هر آزاد سازی (ترخیص) به روز کرد.

(a) Resouce is requested

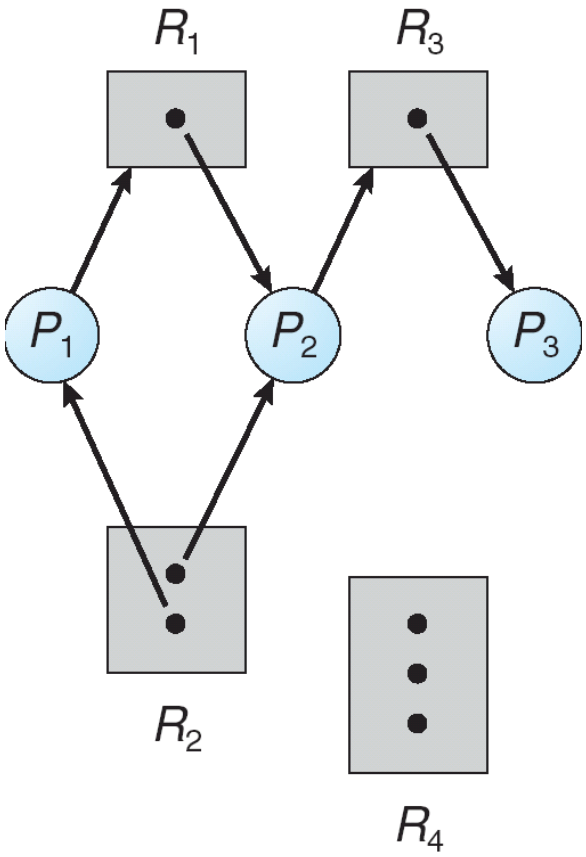


(b) Resource is held

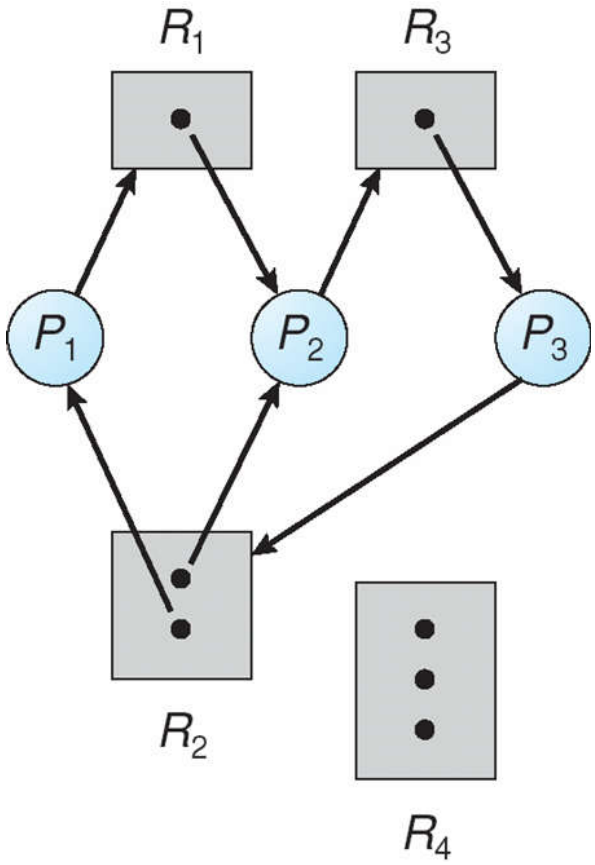




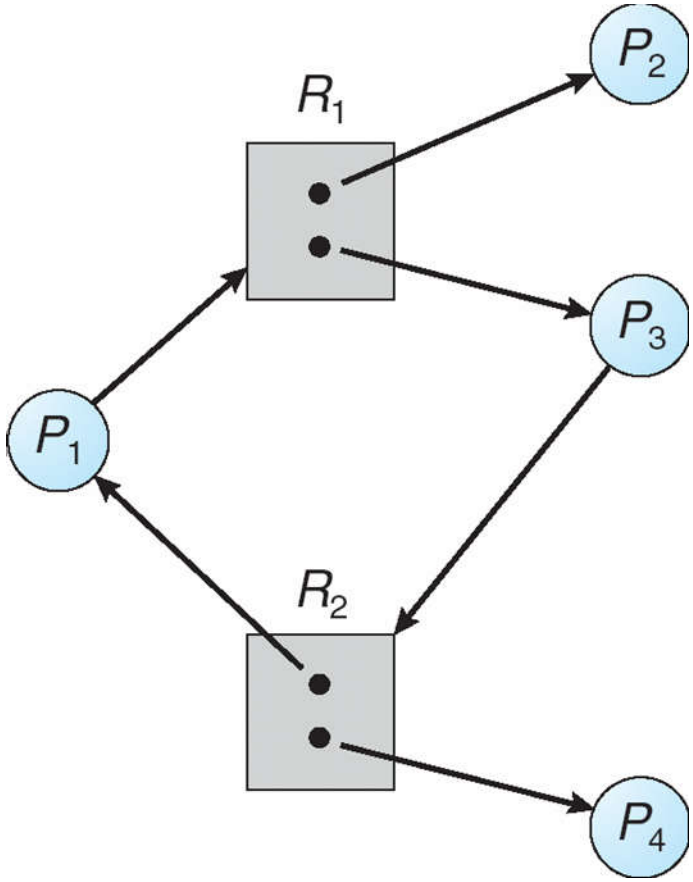
مثالی از گراف تخصیص منابع

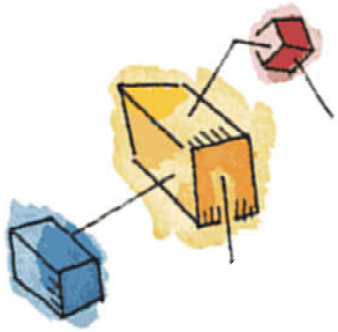


نمونه ای از گراف تخصیص منابع که دارای بن بست است



گرافی که دارای دور است ولی دچار بن بست نشده است

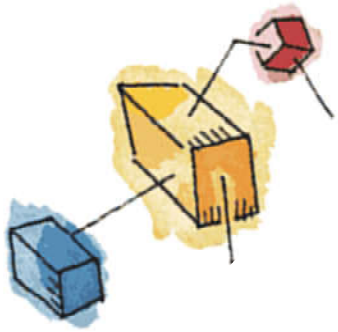




گراف تخصیص منبع

- اگر گراف حاوی هیچ دوری نباشد آنگاه بن بست وجود ندارد.

- اگر گراف حاوی دور باشد آنگاه:
  - اگر از هر نوع منبع تنها یک نمونه داشته باشیم، آنگاه در سیستم بن بست وجود دارد.
  - اگر از بعضی منابع، چند نمونه داشته باشیم، آنگاه ممکن است بن بست وجود داشته باشد.

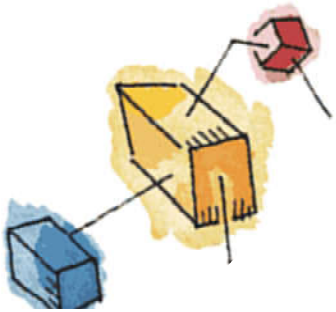


شرایط وقوع بن بست
(۴ شرط کافمن)

1. **انحصار متقابل (Mutual Exclusion)**
   - در هر لحظه تنها یک فرآیند می تواند از یک منبع استفاده کند.

2. **نگهداری و انتظار (Hold and Wait)**
   - هنگام درخواست و انتظار برای منبع جدید، فرآیند می تواند منابع قبلی تخصیص یافته را نیز نگه دارد.

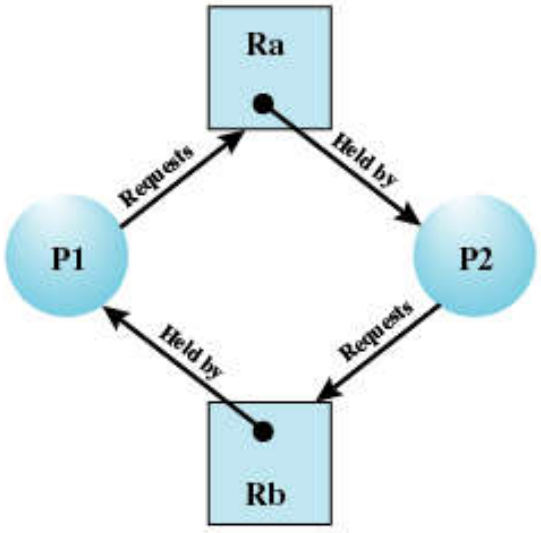
3. **انحصار یا عدم باز پس گیری (Non-Preemption)**
   - منابع به زور قابل پس گیری نیستند.



شرایط وقوع بن بست

4. انتظار حلقوی (Circular wait)

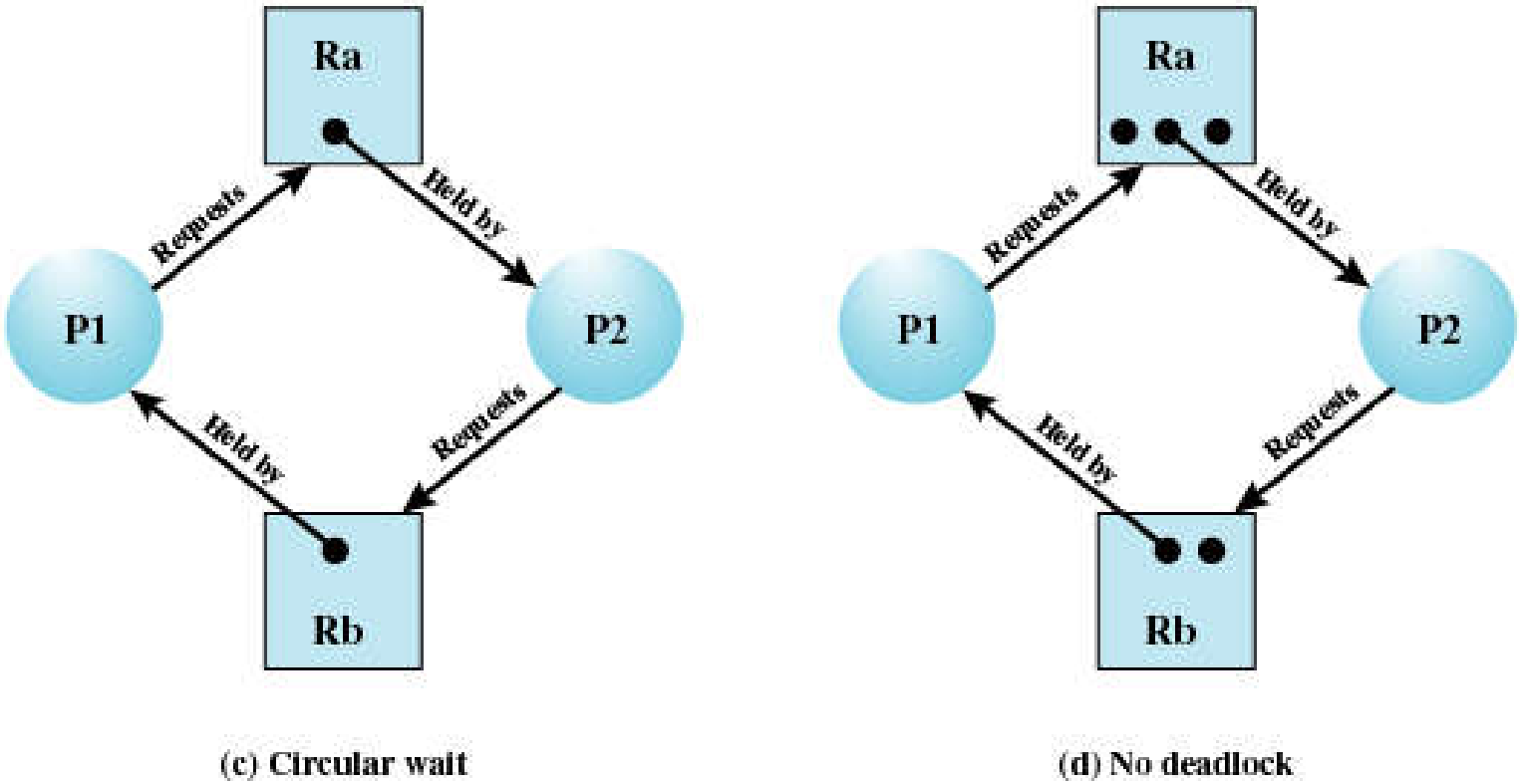
- زنجیر بسته ای از فرآیندها وجود دارد که در آن هر فرآیند حداقل یک منبع مورد نیاز توسط فرآیند بعدی در حلقه را در اختیار دارد.



(c) Circular wait

- برقرار بودن سه شرط اول، نشانه محتمل بودن وقوع بن بست بوده و برقراری تمامی ۴ شرط، بیانگر وقوع بن بست می باشد.

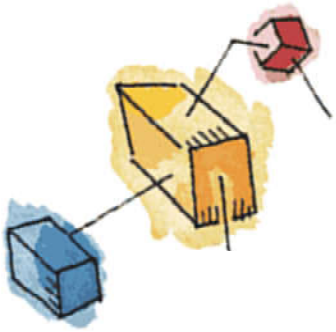
مثال هایی از گراف تخصیص منابع



(c) Circular wait

(d) No deadlock

Figure 6.5 Examples of Resource Allocation Graphs



مثال هایی از گراف تخصیص منابع

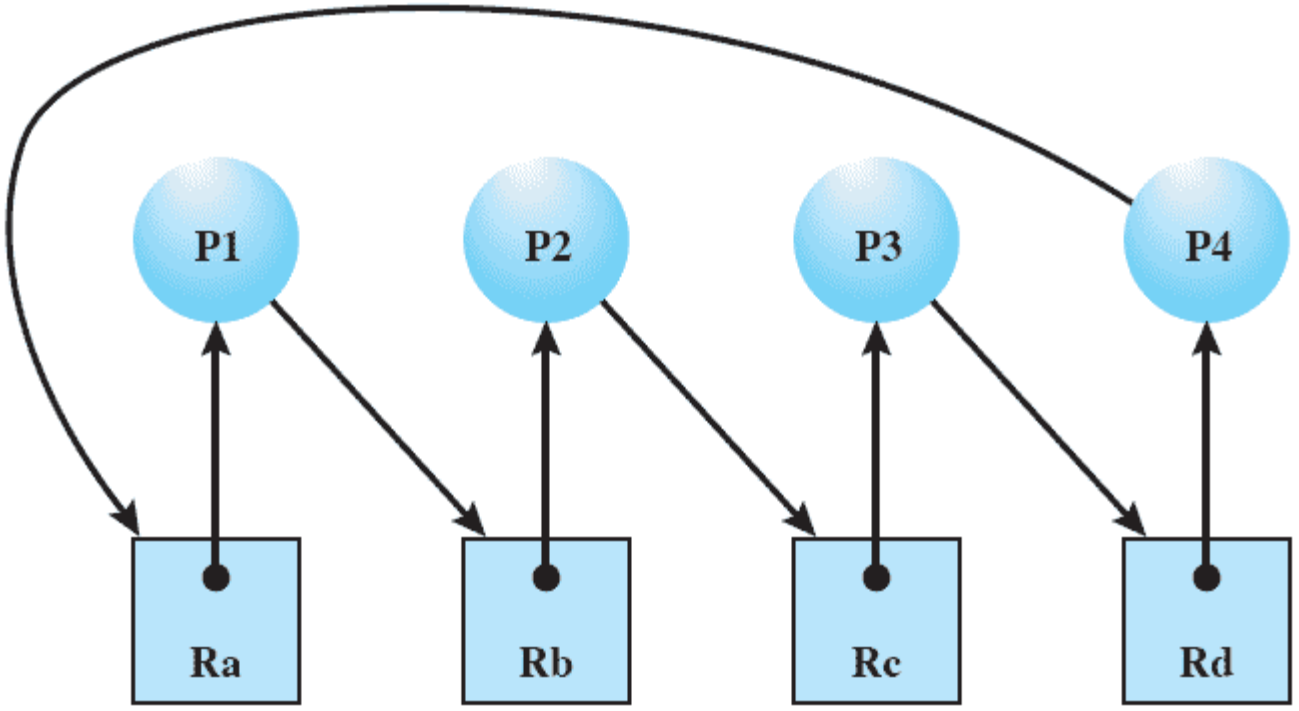
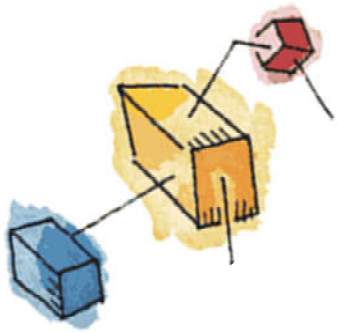


Figure 6.6 Resource Allocation Graph for Figure 6.1b



سرفصل مطالب

- اصول بن بست
- راه های برخورد با مساله بن بست
  - پیش گیری از بن بست (Deadlock Prevention)
  - اجتناب از بن بست (Deadlock Avoidance)
  - کشف بن بست (Deadlock Detection)

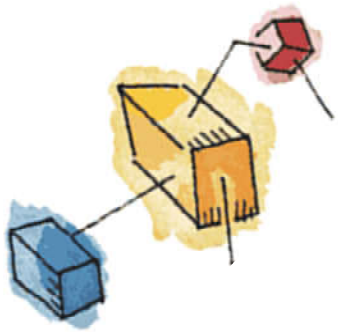
روش های اداره کردن بن بست

۱. می توان از روشی برای پیشگیری یا اجتناب از بن بست استفاده کرد تا تضمین شود که سیستم هرگز به حالت بن بست نرود.

- **پیشگیری از بن بست**
- **اجتناب از بن بست**

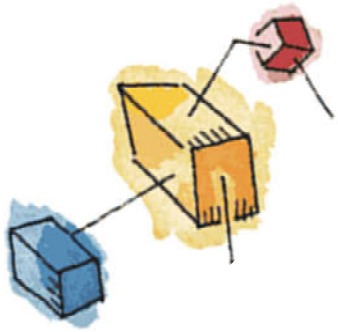
۲. می توان اجازه داد که سیستم وارد بن بست شود. سپس بن بست را تشخیص داد و سیستم را از حالت بن بست خارج کرد (**کشف بن بست**)

۳. می توان از مساله بن بست صرف نظر کرد و وانمود کرد که بن بست در سیستم رخ نمی دهد.



سرفصل مطالب

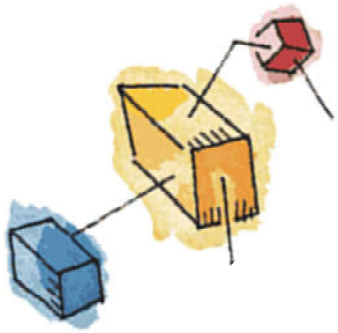
- اصول بن بست
- راه های برخورد با مساله بن بست
  - پیش گیری از بن بست (Deadlock Prevention)
  - اجتناب از بن بست (Deadlock Avoidance)
  - کشف بن بست (Deadlock Detection)



پیشگیری از بن بست
Deadlock Prevention

- پیش گیری از بن بست با نقض کردن یکی از شرایط چهارگانه لازم برای بن بست انجام می شود.

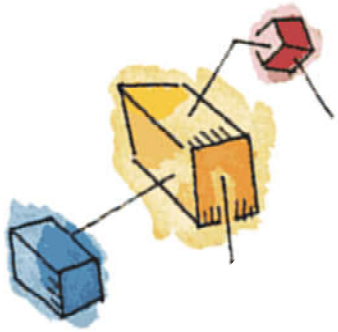
- **نقض انحصار متقابل**
  - نقض کردن این شرط نیاز به حمایت سیستم عامل دارد.
  - ولی این شرط را همیشه نمی توان رد کرد، چرا که بعضی از منابع ذاتاً انحصاری هستند.
  - مثلاً یک چاپگر تنها می تواند به یک فرآیند پاسخ دهد یا نوشتن بر روی یک بانک اطلاعاتی تنها توسط یک فرآیند انجام می شود.



پیشگیری از بن بست
Deadlock Prevention

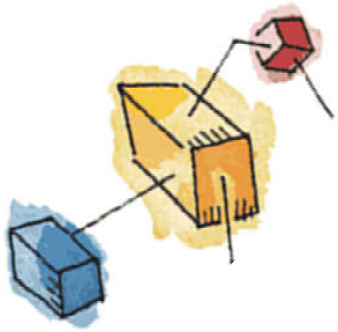
- **نقض نگهداشتن و انتظار**
  - می توان فرآیندها را ملزم ساخت که اختصاص منابع تنها زمانی انجام شود که تمام منابع مورد نیاز فرآیند آزاد باشد و حتی اگر یک منبع آماده نبود هیچ اختصاصی انجام نشود.

- معایب این روش:
  - انتظار طولانی فرآیند برای تکمیل منابعش
  - بیکار ماندن یک منبع به مدت طولانی
  - عدم اطلاع از منابع مورد نیاز در آینده



پیشگیری از بن بست
Deadlock Prevention

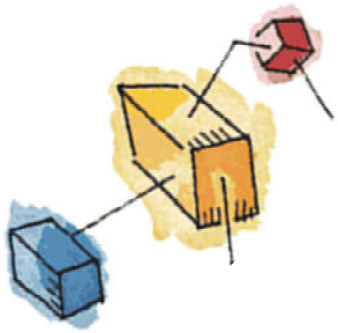
- **باز پس گیری یا نقض انحصار**
  - قبضه کردن فرآیند با اولویت پایین تر
  - زمانی که فرآیند ۱ نیاز به منبعی دارد که فرآیند ۲ آن را نگه داشته است، سیستم عامل آن منبع را قبضه کرده و آزاد می کند (باز پس می گیرد) و در اختیار فرآیند ۱ قرار می گیرد
  - یعنی فرآیند ۲ که منبع را در اختیار داشته بایستی منبع را آزاد کرده و بعدا مجددا آن منبع درخواست کند.



پیشگیری از بن بست
Deadlock Prevention

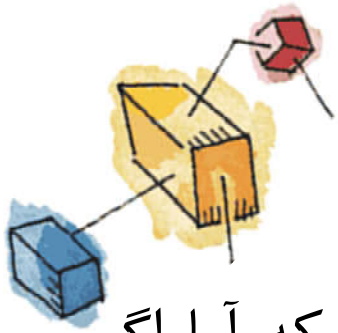
- **نقض شرط انتظار حلقوی**
  - مرتب کردن منابع به صورت خطی
  - به هر منبع یک شاخص نسبت داده می شود. در این صورت فرآیند می تواند ابتدا منبع R_i و سپس منبع R_j را درخواست کند اگر **i<j** باشد.

- معایب این روش:
  - کند شدن فرآیندها
  - رد کردن غیر ضروری دسترسی به منابع
  - هدر رفتن منابع و کاهش بهره وری
  - نیاز به پیش بینی آینده



سرفصل مطالب

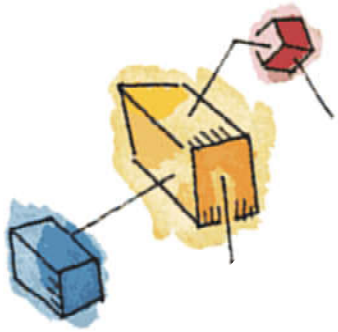
- اصول بن بست
- راه های برخورد با مساله بن بست
  - پیش گیری از بن بست (Deadlock Prevention)
  - اجتناب از بن بست (Deadlock Avoidance)
  - کشف بن بست (Deadlock Detection)



اجتناب از بن بست (Deadlock Avoidance)

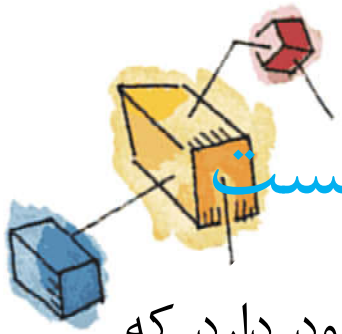
- در این رهیافت، دائما به صورت پویا تصمیم گرفته می شود که آیا اگر درخواست فعلی برای تخصیص منبع، برآورده شود، ممکن است منجر به بن بست شود یا خیر.

- در این استراتژی، نیاز است درخواست های آینده فرآیندها برای منابع را بدانیم.



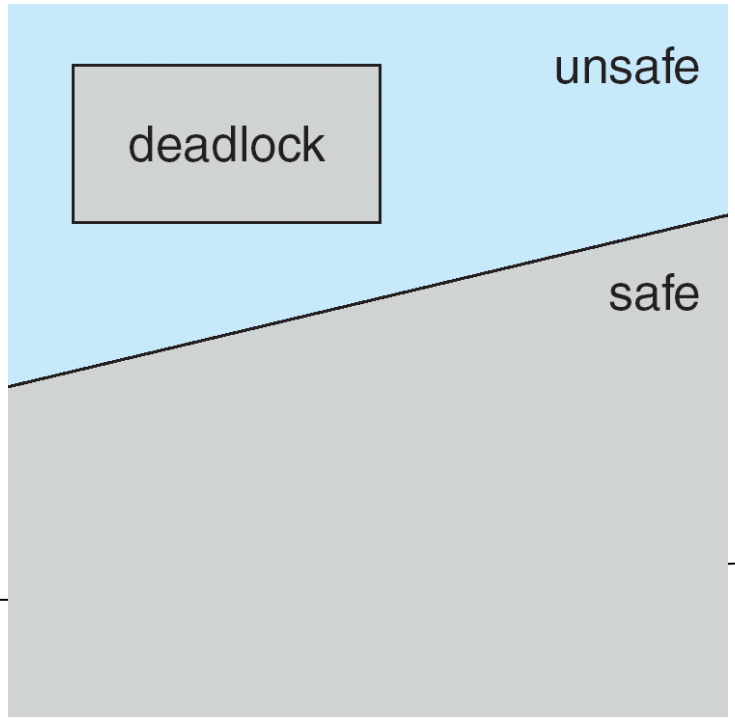
دو رهیافت برای جلوگیری از بن بست

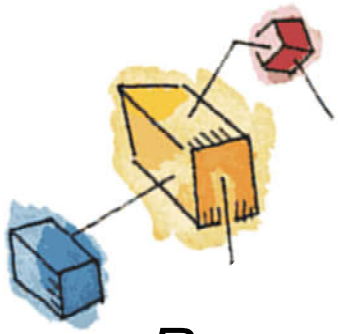
- **عدم شروع فرآیند:**
  - عدم شروع فرآیندی که ممکن است درخواست هایش موجب وقوع بن بست شود.
  - یک فرایند تنها در صورتی آغاز می شود که حداکثر درخواست های فرآیندهای موجود و این فرآیند جدید بتواند تامین گردد. این رهیافت بهینه نیست چون بدترین شرایط را در نظر می گیرد (این شرایط که همه فرآیندها حداکثر منابع موردنیاز خود را با هم اعلام کنند).
- **عدم تخصیص منبع:**
  - عدم پاسخ به درخواست فرآیندی برای دریافت منبع و افزایش منابع خود، در صورتی که این تخصیص ممکن است منجر به بن بست شود. دو الگوریتم برای این کار:
    1. الگوریتم گراف تخصیص منابع (این الگوریتم فقط وقتی قابل استفاده است که از هر منبع تنها یک نمونه داشته باشیم)
    2. الگوریتم بانکدار
  - هر دوی این الگوریتم ها سعی می کنند سیستم را در حالت امن نگه دارند.



وضعیت امن / وضعیت ناامن / وضعیت بن بست

- **وضعیت امن**، حالتی است که در آن حداقل یک ترتیب از فرآیندها وجود دارد که می توانند اجرا و کامل شوند، بدون اینکه حالت بن بست ایجاد شود.
- چنانچه سیستم در چنین حالتی قرار نداشته باشد، اصطلاحا گفته می شود سیستم در **وضعیت نا امن** قرار دارد.
- تمام وضعیت های ناامن لزوما به بن بست منتهی نمی شوند.

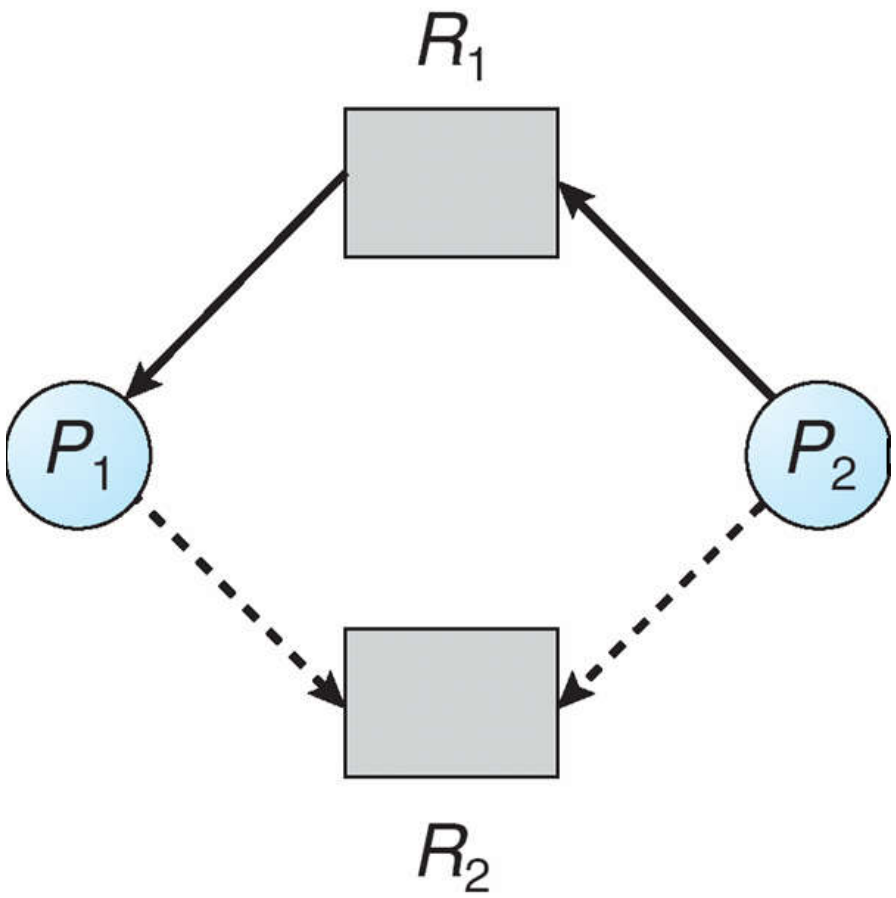




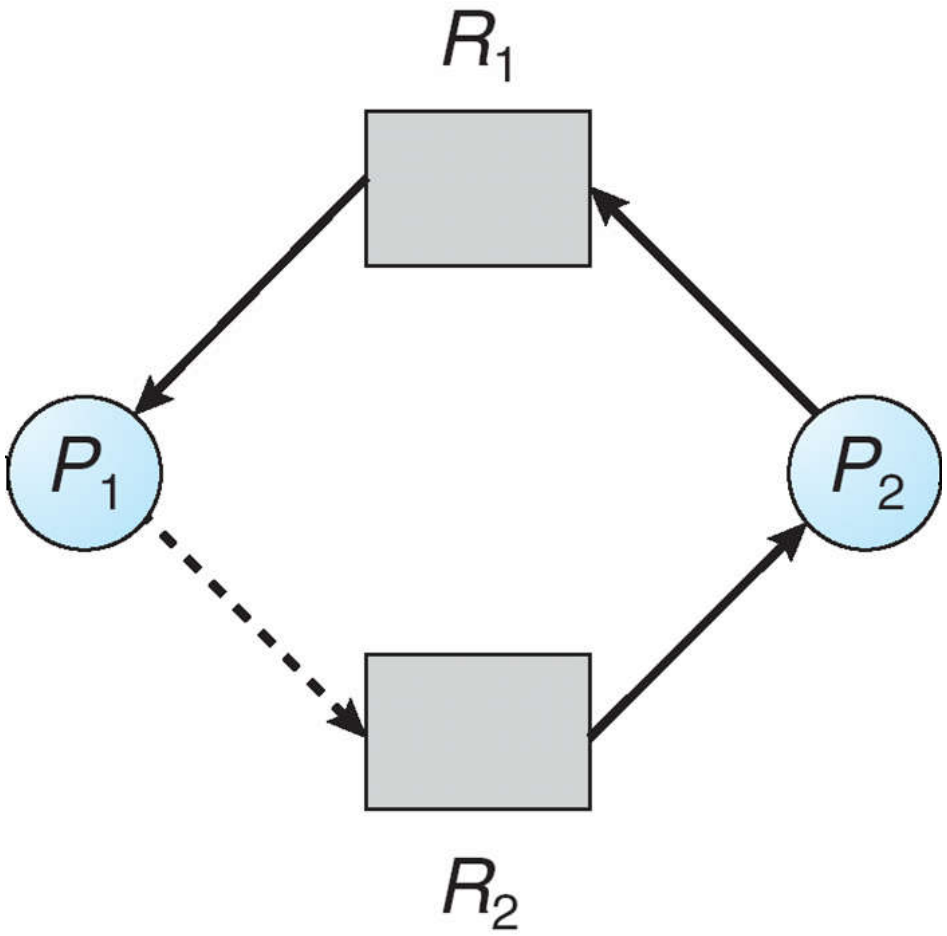
۱. الگوریتم گراف تخصیص منابع

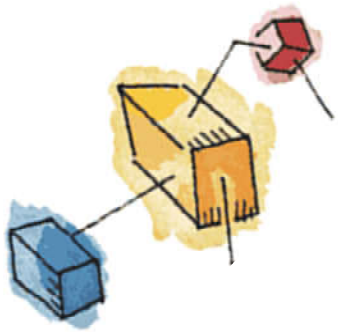
- یال ادعای $P_i \rightarrow R_j$ نشان می دهد که فرآیند P_j ممکن است در آینده منبع R_j را درخواست کند.
- یال های ادعا به صورت خط چین نشان داده می شوند.
- اگر فرآیندی در خواست یک منبع را بدهد، یال ادعا به یال درخواست تبدیل می شود.
- هرگاه فرآیندی منبعی را آزاد کند، یال تخصیص به یال ادعا تبدیل می شود.
- منابع باید پیشاپیش در سیستم ادعا شوند و به عنوان یال ادعا در گراف تخصیص منابع قرار بگیرند.

گراف تخصیص منابع



حالت ناامن در گراف تخصیص منابع

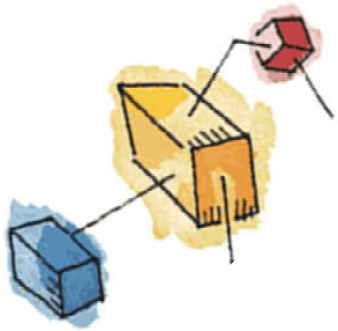




الگوریتم گراف تخصیص منابع

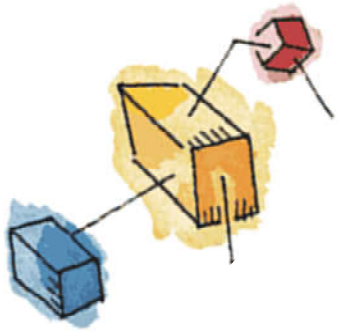
- فرض کنید فرآیند P_i یک منبع R_j را نیاز دارد.

- این درخواست تنها در صورتی پاسخ داده می شود که تبدیل کردن یال درخواست منبع به یال تخصیص، باعث ایجاد دور در گراف تخصیص منابع نشود.



۲. الگوریتم بانکدار

- این روش به نام **الگوریتم بانکدار** معروف است.
- در این الگوریتم نیز، در صورتی تخصیص منبع صورت می گیرد که سیستم در حالت امن باقی بماند.
- وضعیت سیستم، گویای چگونگی تخصیص فعلی منابع به فرآیندها است.
- وضعیت سیستم شامل دو بردار Available و Resource و همچنین دو ماتریس Max (یا ادعا) و Allocation می باشد.



الگوریتم بانکدار

1. نیاز (Need) هر فرآیند را بدست می آوریم.
2. فرآیندهای موجود در لیست (سیستم) را از بالا به پایین مورد بررسی قرار می دهیم.
3. برای هر فرآیند، میزان نیازهای آن را با منابع موجود مقایسه می کنیم.

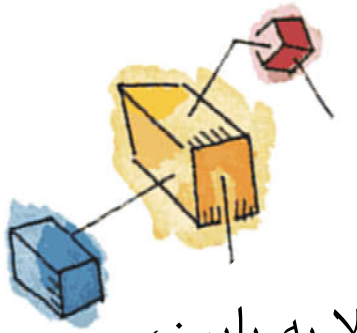
a) اگر تعداد نیاز فرآیند به هر یک از منابع، کوچک تر یا مساوی تعداد موجود (باقیمانده) از آن منابع باشد:

- منابعی که در اختیار داشته است را به منابع موجود(آزاد) سیستم اضافه می کنیم.
- آن فرآیند را از لیست(سیستم) حذف نموده و مجددا گام های ۲ و ۳ را تکرار می کنیم.

b) اگر حتی یکی از نیازهای فرآیند بیشتر از تعداد موجود از آن منبع باشد:

- در صورتیکه تمامی فرآیندها بررسی شده و این آخرین فرآیند در لیست باشد، نتیجه می گیریم که سیستم در حالت نا امن است.
- در غیر اینصورت این فرآیند را در لیست نگه داشته و گام ۳ را برای فرآیند بعدی اجرا می کنیم.

4. در صورتیکه نهایتا تمامی فرآیندها پردازش و حذف گردند، سیستم در حالت امن قرار دارد.



الگوریتم بانکدار

- **نکته:** برای حل سریع تر و راحت تر، بهتر است که به جای بررسی از بالا به پایین، فرآیندهایی که کمترین نیاز را دارند زودتر مورد بررسی قرار گیرند.

مثالی از بررسی امن بودن با استفاده از الگوریتم بانکدار

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**: Max

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C – A =Need

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

Available vector **V**

(a) Initial state

مثالی از بررسی امن بودن با استفاده از الگوریتم بانکدار

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
6	2	3

Available vector **V**

(b) P2 runs to completion

مثالی از بررسی امن بودن با استفاده از الگوریتم بانکدار

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
7	2	3

Available vector **V**

(c) P1 runs to completion

مثالی از بررسی امن بودن با استفاده از الگوریتم بانکدار

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
9	3	4

Available vector **V**

(d) P3 runs to completion

و نهایتا هم P4 اجرا می شود و همه منابع آزاد می شوند.

بنابراین یک ترتیب امن برای اجرای این چهار فرایند یافته شد (از چپ به راست):

P2, P1, P3, P4

پس سیستم در حالت امن است و بن بست رخ نمی دهد.

مثالی از بررسی امن بودن با استفاده از الگوریتم بانکدار

- اکنون فرض کنید سیستم در حالت زیر است و فرآیند P2، یک نمونه از R1 و یک نمونه از R3 را درخواست می دهد.
- می خواهیم بررسی کنیم که آیا پاسخ دادن به این درخواست باعث ناامن شدن سیستم می شود یا خیر.
- فرض می کنیم منابع درخواست داده شده را به P2 تخصیص داده ایم. با الگوریتم بانکدار بررسی می کنیم که آیا این حالت امن است یا خیر (همان مثال قبل)
- چون امن است، به طور واقعی هم به درخواست پاسخ می دهیم.

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
1	1	2

Available vector **V**

(a) Initial state

مثالی از بررسی امن بودن با استفاده از الگوریتم بانکدار

- اکنون فرض کنید فرآیند P1، یک نمونه از R1 و یک نمونه از R3 را درخواست می دهد.
- فرض می کنیم منابع درخواست داده شده را به P1 تخصیص داده ایم. با الگوریتم بانکدار بررسی می کنیم که آیا این حالت امن است یا خیر.
- با دنبال کردن الگوریتم بانکدار، نشان داده می شود که در این حالت، سیستم امن نیست. پس درخواست را پاسخ نمی دهیم.

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix C

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix A

	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

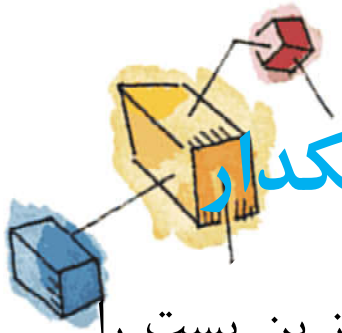
R1	R2	R3
9	3	6

Resource vector R

R1	R2	R3
0	1	1

Available vector V

(b) P1 requests one unit each of R1 and R3



معایب روش اجتناب از بن بست و الگوریتم بانکدار

- الگوریتم بانکدار و روش اجتناب از بن بست نمی تواند به طور حتم بروز بن بست را پیش بینی کند. وقوع آن را حدس می زند و عدم وقوع آن را اطمینان می دهد.
- معایب روش اجتناب از بن بست:
  - حداکثر منابع باید از قبل مشخص باشد.
  - فرآیندهای مورد نظر باید مستقل باشند (همگام سازی نشده باشند)
  - فرآیندی که منبعی را در اختیار داشته باشد نمی تواند خارج گردد.
  - تعداد منابع تخصیصی باید ثابت باشد.