

سیستم‌های عامل

Dr. Taghinezhad



Operating System Concepts

TENTH EDITION

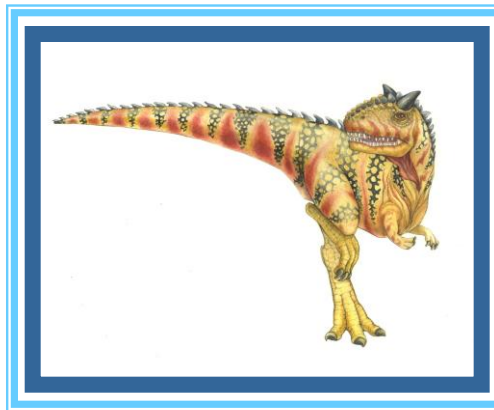
ABRAHAM SILBERSCHATZ • PETER BAER GALVIN • GREG GAGNE



WILEY

Website: ataghinezhad.github.io, Email: a0taghinezhad@gmail.com

فصل ۸: بن بست





Outline

- مشخصه بن بست (Deadlock Characterization)
- روش‌های مدیریت بن بست (Methods for Handling Deadlocks)
- جلوگیری از بن بست (Deadlock Prevention)
- جلوگیری از قفل شدن (Deadlock Avoidance)
- تشخیص بن بست (Deadlock Detection)
- بازیابی از بن بست (Recovery from Deadlock)



Chapter Objectives

- نمونه‌ای از ایجاد بن‌بست با استفاده از قفل‌های متقابل (Illustrate how deadlock can occur when mutex locks are used)
- تعریف چهار شرط لازم برای مشخصه بن‌بست (Define the four necessary conditions that characterize deadlock)
- شناسایی وضعیت بن‌بست در نمودار تخصیص منابع (Identify a deadlock situation in a resource allocation graph)
- ارزیابی چهار رویکرد مختلف برای جلوگیری از بن‌بست (Evaluate the four different approaches for preventing deadlocks)
- اعمال الگوریتم بانکدار برای جلوگیری از بن‌بست (Apply the banker's algorithm for deadlock avoidance)
- اعمال الگوریتم تشخیص بن‌بست (Apply the deadlock detection algorithm)
- ارزیابی رویکردهای بازیابی از بن‌بست (Evaluate approaches for recovering from deadlock)



مدل سیستم

- سیستم شامل منابع است
- انواع منابع R_1, R_2, \dots, R_m
- چرخه‌های پردازنده، فضای حافظه، دستگاه‌های ورودی/خروجی (CPU cycles, memory space, I/O devices)
- هر نوع منبع R_i دارای W_i نمونه است
- هر فرایند به روش زیر از یک منبع استفاده می‌کند :
 - درخواست (request)
 - استفاده (use)
 - آزادسازی (release)



بن بست با سمافور

■ داده‌ها:

- یک سمافور S1 با مقدار اولیه ۱
- یک سمافور S2 با مقدار اولیه ۱
- دو رشته (ترد) T1 و T2

■ T1:

- wait(s1)
- wait(s2)

■ T2:

- wait(s2)
- wait(s1)



ویژگی‌های بن‌بست

- **موقعیت بن‌بست زمانی اتفاق می‌افتد که همه شرایط زیر محیا باشد.**
- **منع دسترسی همزمان (Mutual exclusion):** در هر لحظه تنها یک رشته می‌تواند از یک منبع استفاده کند.
- **نگهداشتن و انتظار (Hold and wait):** یک رشته که حداقل یک منبع را در اختیار دارد، منتظر است تا منابع دیگری را که توسط رشته‌های دیگر نگهداری می‌شوند، به دست آورد.
- **عدم تصاحب اجباری (No preemption):** یک منبع تنها می‌تواند به صورت داوطلبانه توسط رشته‌ای که آن را در اختیار دارد، پس از اتمام کار آن رشته، آزاد شود.
- **انتظار دوره‌ای (Circular wait):** مجموعه‌ای از رشته‌های در حال انتظار $\{T_0, T_1, \dots, T_n\}$ وجود دارد که به گونه‌ای است که T_0 منتظر منبعی است که توسط T_1 نگهداری می‌شود، T_1 منتظر منبعی است که توسط T_2 نگهداری می‌شود، \dots T_{n-1} منتظر منبعی است که توسط T_n نگهداری می‌شود و T_n منتظر منبعی است که توسط T_0 نگهداری می‌شود.

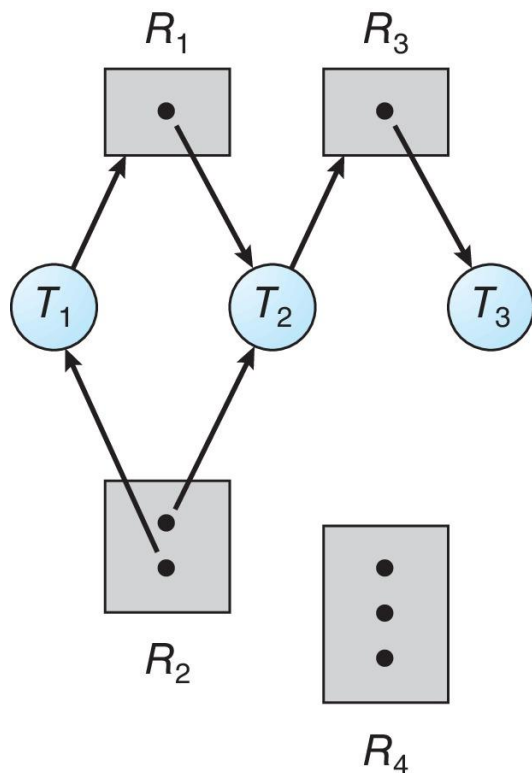


گراف تخصیص منابع

- مجموعه‌ای از رئوس V و لبه (کمان‌های) E
- V به دو دسته تقسیم می‌شود :
- شامل همه تردهای سیستم, $T = \{T_1, T_2, \dots, T_n\}$
- شامل همه منابع سیستم, $R = \{R_1, R_2, \dots, R_m\}$
- انواع لبه در نمودار تخصیص منابع:
 - لبه درخواست (request edge) – $T_i \rightarrow R_j$ جهت‌دار
 - لبه انتساب (assignment edge) – $R_j \rightarrow T_i$ جهت‌دار



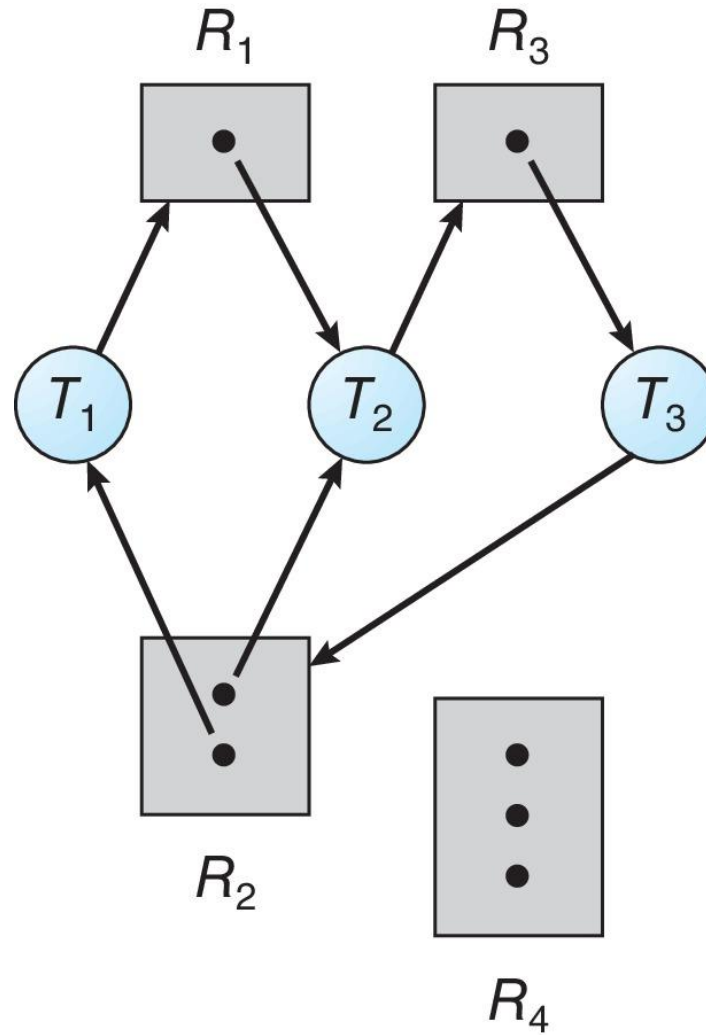
نمونه‌ای از گراف تخصیص منابع



- یک نمونه از R_1
- دو نمونه از R_2
- یک نمونه از R_3
- سه نمونه از R_4
- T_1 یک نمونه از R_2 را در اختیار دارد و منتظر یک نمونه از R_1 است.
- T_2 یک نمونه از R_1 و یک نمونه از R_2 را در اختیار دارد و منتظر یک نمونه از R_3 است.
- T_3 یک نمونه از R_3 را در اختیار دارد.

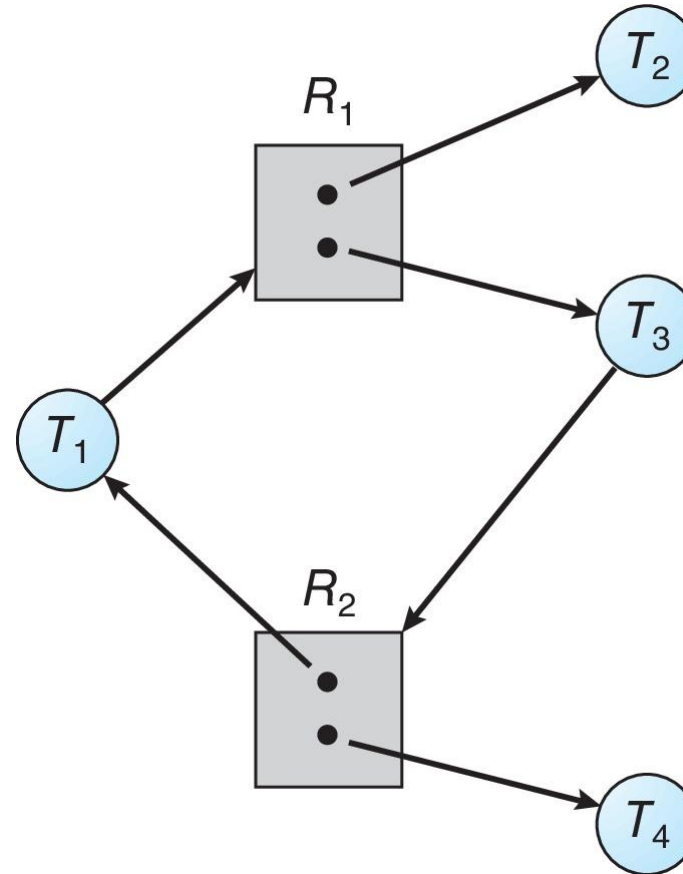


گراف تخصیص منابع با بن بست





گراف با چرخه بدون بن بست





اطلاعات پایه‌ای

- اگر نمودار هیچ سیکلی نداشته باشد، بن‌بستی وجود ندارد.
- اگر نمودار یک سیکل داشته باشد :
- اگر برای هر نوع منبع تنها یک نمونه وجود داشته باشد، بن‌بست رخ می‌دهد.
- اگر برای هر نوع منبع چندین نمونه وجود داشته باشد، احتمال بن‌بست وجود دارد.



روش‌های برای مدیریت بن‌بست

- برای اطمینان از اینکه سیستم هرگز وارد حالت بن‌بست نمی‌شود، از روش‌های زیر استفاده می‌کنیم :

- جلوگیری از بن‌بست (Deadlock prevention)

- اجتناب از بن‌بست (Deadlock avoidance)

■ روش‌های دیگر برای مدیریت بن‌بست:

- اجازه دادن به سیستم برای ورود به حالت بن‌بست و سپس بازیابی
- نادیده گرفتن مشکل و وانمود کردن اینکه بن‌بست هرگز در سیستم رخ نمی‌دهد



Deadlock Prevention پیشگیری از بن بست

یکی از چهار شروط را برای بن بست باطل کنید:

- منع دسترسی همزمان - برای منابع اشتراکی (مانند فایل های فقط خواندنی) الزامی نیست، اما برای منابع غیرقابل اشتراک الزامی است.
- نگهداری و انتظار - باید تضمین کند که یک رشته زمانی درخواستی برای یک منبع ارسال می کند که هیچ منبع دیگری را در اختیار نداشته باشد.
- الزام رشته ها برای درخواست و تخصیص تمام منابع خود قبل از شروع اجرا یا اجازه دادن به رشته برای درخواست منابع فقط در صورتی که هیچ منبعی به آن اختصاص داده نشده باشد .
- استفاده از این رویکرد می تواند منجر به پایین آمدن بهره وری منابع و همچنین گرسنگی (عدم تخصیص منابع به یک رشته برای مدت طولانی) شود.

(Cont.) پیشگیری از بن بست



■ عدم تصاحب اجباری: (No Preemption)

- اگر فرآیندی که برخی منابع را در اختیار دارد، درخواست منبع دیگری را بدهد که بلافاصله قابل تخصیص به آن نباشد، در این صورت تمام منابعی که در حال حاضر در اختیار فرآیند هستند، آزاد می‌شوند.
- منابع پس گرفته شده به لیست منابعی که رشته برای آن‌ها در حال انتظار است، اضافه می‌شوند.
- رشته تنها زمانی مجدداً راه‌اندازی می‌شود که بتواند منابع قدیمی خود و همچنین منابع جدیدی را که درخواست می‌کند، دوباره به دست آورد.

■ انتظار دوره‌ای: (Circular Wait)

- برای باطل کردن شرط انتظار دوره‌ای، معمولاً ترتیب کلی بر روی تمام انواع منابع اعمال می‌شود و از هر رشته خواسته می‌شود که منابع را به ترتیب افزایشی شماره‌گذاری درخواست کند.



انتظار چرخشی Circular Wait

■ ساده‌ترین روش این است که به هر منبع (مانند قفل‌های متقابل) یک

شماره منحصر به فرد اختصاص دهیم .

■ منابع باید به ترتیب به دست آورده شوند .

■ برای مثال:

اگر:

first_mutex = 1

second_mutex = 5

■ کد برای رشته‌ی دو نمی‌تواند به صورت زیر نوشته شود:

■ باید ترتیب اول و دوم رعایت شود.

```
/* thread_one runs in this function */  
void *do_work_one(void *param)
```

```
{  
    pthread_mutex_lock(&first_mutex);  
    pthread_mutex_lock(&second_mutex);  
    /**  
     * Do some work  
     */  
    pthread_mutex_unlock(&second_mutex);  
    pthread_mutex_unlock(&first_mutex);  
  
    pthread_exit(0);  
}
```

```
/* thread_two runs in this function */  
void *do_work_two(void *param)
```

```
{  
    pthread_mutex_lock(&second_mutex);  
    pthread_mutex_lock(&first_mutex);  
    /**  
     * Do some work  
     */  
    pthread_mutex_unlock(&first_mutex);  
    pthread_mutex_unlock(&second_mutex);  
  
    pthread_exit(0);  
}
```




Deadlock Avoidance اجتناب از بن بست

مستلزم آن است که سیستم اطلاعات قبلی اضافی در دسترس داشته باشد

■ ساده‌ترین و مفیدترین مدل، نیازمند این است که هر رشته حداکثر تعداد منابعی را که از هر نوع ممکن است نیاز داشته باشد، اعلام کند.

■ الگوریتم اجتناب از بن بست به صورت پویا وضعیت تخصیص منابع را بررسی می‌کند تا اطمینان حاصل کند که هرگز شرایط انتظار دوره‌ای وجود نداشته باشد .

■ وضعیت تخصیص منابع با تعداد منابع در دسترس و تخصیص داده شده و حداکثر تقاضاهای فرآیندها تعریف می‌شود .



حالت امن Safe State

- هنگامی که یک رشته یک منبع در دسترس را درخواست می کند، سیستم باید تصمیم بگیرد که آیا تخصیص فوری سیستم را در حالت امن نگه می دارد یا خیر.
- سیستم در حالت امن قرار دارد، اگر دنباله ای $\langle T_1, T_2, \dots, T_N \rangle$ از تمام رشته های موجود در سیستم وجود داشته باشد، به گونه ای که برای هر T_i ، منابعی که T_i هنوز می تواند درخواست کند، توسط منابع موجود فعلی + منابع نگهداری شده توسط تمام T_j ، با $I < j$ قابل برآورده شدن باشد.
- به عبارت دیگر:
- اگر نیازهای منابع T_i بلافاصله در دسترس نیستند، پس T_i می تواند تا زمانی که همه T_j (تراکنش های قبلی) به پایان برسند، صبر کند.
- هنگامی که T_j تمام شد، T_i می تواند منابع مورد نیاز را به دست آورد، اجرا کند، منابع اختصاص یافته را برگرداند و خاتمه دهد.
- هنگامی که T_i خاتمه می یابد، $1 + T_i$ می تواند منابع مورد نیاز خود را به دست آورد و به همین ترتیب ادامه یابد.

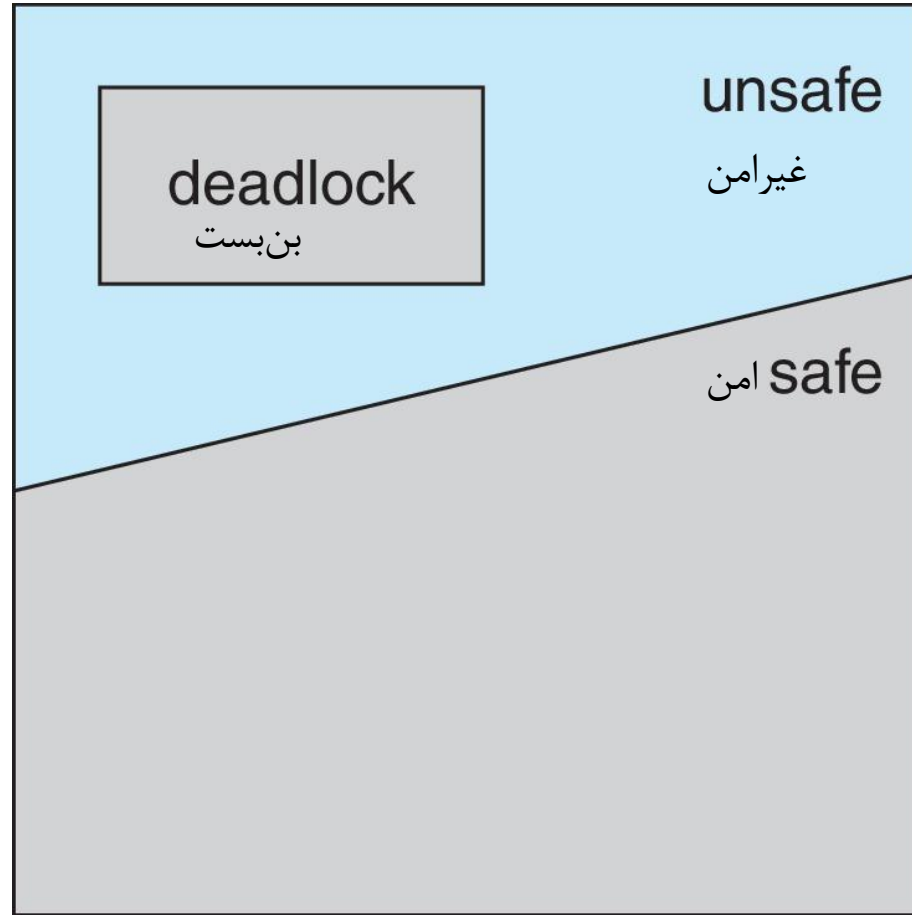


حقایق اساسی Basic Facts

- اگر سیستم در حالت امن باشد، بن بست وجود ندارد.
- اگر سیستم در حالت ناامن باشد، احتمال بن بست وجود دارد.
- اجتناب: اطمینان حاصل می کند که سیستم هرگز وارد حالت ناامن نمی شود.



Safe, Unsafe, Deadlock State





الگوریتمهای اجتناب Avoidance Algorithms

- یک نمونه از یک نوع منبع:

- برای نشان دادن تخصیص منابع از یک نمودار تخصیص منابع استفاده کنید.

- چندین نمونه از یک نوع منبع:

- از الگوریتم بانکدار (Banker's Algorithm) استفاده کنید.

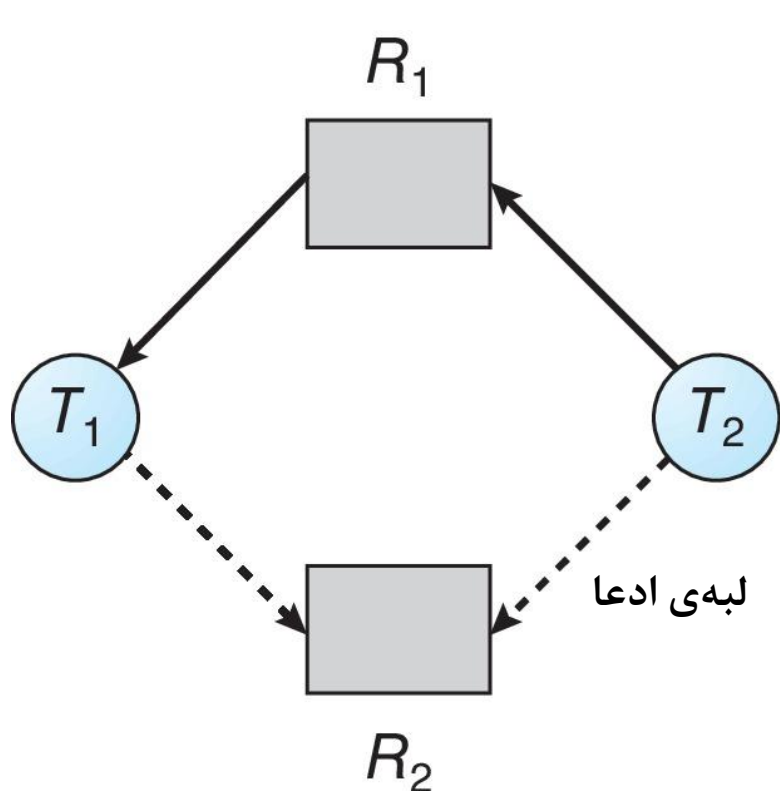


الگوی (اسکیما) الگوریتم گراف تخصیص منابع

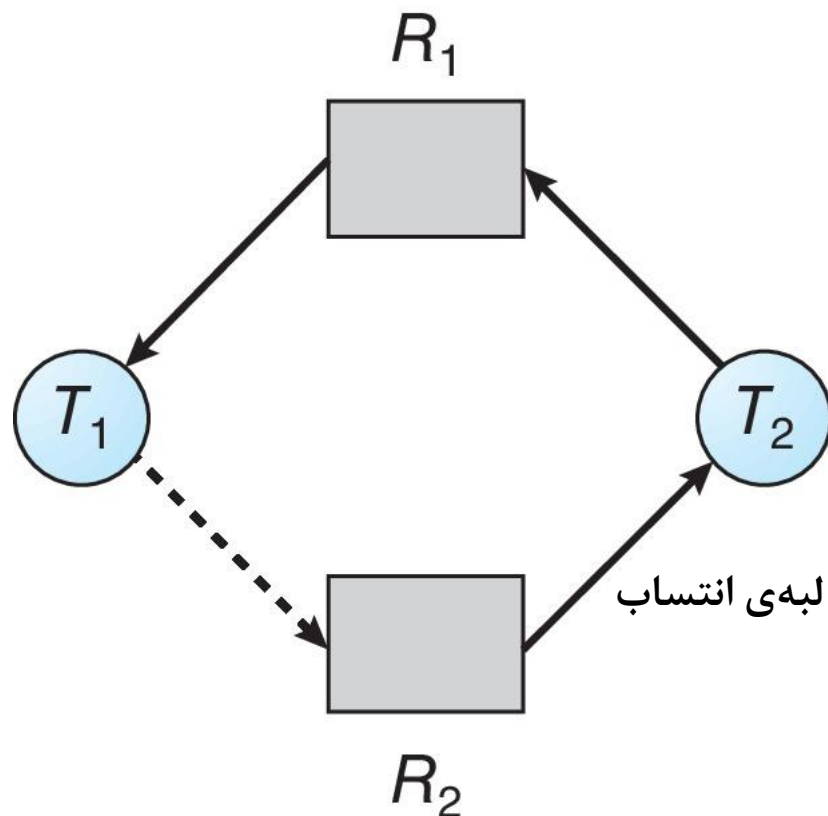
- **لبه‌ی ادعا: (claim edge):** جهت‌دار $R_j \rightarrow T_i$ نشان می‌دهد که فرآیند T_j ممکن است منبع R_j را درآینده درخواست کند. این لبه با خط چین نمایش داده می‌شود. $\cdots \rightarrow$
- **تبدیل لبه‌ی ادعا به لبه‌ی درخواست:** هنگامی که یک رشته منبعی را درخواست می‌کند، لبه‌ی ادعا به لبه‌ی درخواست (request edge) تبدیل می‌شود.
- **تبدیل لبه‌ی درخواست به لبه‌ی انتساب:** زمانی که منبع به رشته اختصاص یابد، لبه‌ی درخواست به لبه‌ی انتساب (assignment edge) تبدیل می‌شود. خط کامل نشان داده می‌شود. \longrightarrow
- **بازگشت لبه‌ی انتساب به لبه‌ی ادعا:** هنگامی که منبعی توسط یک رشته آزاد می‌شود، لبه‌ی انتساب دوباره به لبه‌ی ادعا تبدیل می‌شود.
- **منابع باید قبلاً در سیستم ادعا شوند. (claimed a priori)**



الگوریتم گراف تخصیص منابع



فرض کنید که T_2 درخواست R_2 را می‌کند .
اگرچه R_2 در حال حاضر آزاد است، نمی‌توانیم
آن را به T_2 اختصاص دهیم، زیرا این عمل یک
چرخه در نمودار ایجاد می‌کند .



وضعیت ناامن در نمودار تخصیص منابع



الگوریتم گراف تخصیص منابع

- فرض کنید که رشته‌ی T_1 درخواست یک منبع R_j را می‌کند.
- این درخواست تنها در صورتی قابل اعطا است که تبدیل لبه‌ی درخواست به لبه‌ی انتساب منجر به ایجاد **چرخه** در نمودار تخصیص منابع نشود.



الگوریتم بانکدار Banker's Algorithm

- چند نمونه‌ای از منابع وجود دارد.
- هر رشته باید از قبل حداکثر استفاده از منابع را ادعا کند
- هنگامی که یک رشته درخواستی برای یک منبع ارسال می‌کند، ممکن است مجبور به انتظار شود (wait).
- زمانی که یک رشته تمام منابع مورد نیاز خود را به دست آورد، باید آن‌ها را در مدت زمان محدودی برگرداند.



ساختارهای داده برای الگوریتم بانکدار

□ فرض کنید n تعداد فرآیندها، و m تعداد انواع منابع.

■ بردار در دسترس یا **Available** به طول m اگر $available[j] = k$ ، به این معنی است که k نمونه از نوع منبع R_j در دسترس است.

■ ماتریس بیشینه یا **Max**: ماتریس $n \times m$ اگر $Max[i,j] = k$ ، پس فرآیند T_i می تواند حداکثر k نمونه از نوع منبع R_j را درخواست کند.

■ ماتریس تخصیص یا **Allocation**: ماتریس $n \times m$ اگر $Allocation[i,j] = k$ ، در این صورت فرآیند T_i در حال حاضر k نمونه از نوع منبع R_j را در اختیار دارد.

■ ماتریس نیازمندی **Need**: ماتریس $n \times m$ اگر $Need[i,j] = k$ ، پس T_i ممکن است برای تکمیل کار خود به k نمونه دیگر از نوع منبع R_j نیاز داشته باشد.

$$Need[i,j] = Max[i,j] - Allocation[i,j] \blacktriangleright$$



الگوریتم ایمنی

حالا می‌شود الگوریتم برای اینکه سیستم امن است یا خیر داد.

(۱) فرض کنیم $Work$ و $Finish$ به ترتیب بردارهایی به طول m و n هستند.

• مقداردهی اولیه:
 $Work = Available$
 $Finish[i] = false \text{ for } i = 0, 1, \dots, n-1$

(۲) یک i پیدا کنید به گونه‌ای که هر دو شرط زیر برقرار باشند :

◦ (الف) $Finish[i] = false$ (فرآیند i هنوز تمام نشده است)

◦ (ب) $Need[i] \leq Work$ (فرآیند i برای اتمام به منابعی کمتر یا مساوی با منابع موجود نیاز دارد)

اگر چنین i وجود ندارد، به مرحله ۴ بروید (هیچ فرآیندی شرایط لازم برای تخصیص منابع را ندارد)

(۳) $Work = Work + Allocation_i$
 $Finish[i] = true$

برو به مرحله ۲

(۴) یک $Finish[i] == true$ برای همه i باشد. پس سیستم در وضعیت امن است.



الگوریتم درخواست منبع برای فرآیند P_i

- حال باید الگوریتمی ارائه شود که نشان دهد وقتی درخواستی برآورده می‌شود سیستم در وضعیت امنی است.
- فرض می‌کنیم.
- $Request_i$ برابر بردار درخواست برای فرآیند T_i است
- اگر $Request_i[j] = k$ پس فرآیند T_i از منبع R_j به تعداد k نمونه می‌خواهد.



الگوریتم درخواست منبع برای فرآیند P_i

۱. اگر درخواست از نیازمندی کمتر است: $Request_i \leq Need_i$ به مرحله ۲ بروید. در غیر این صورت، شرایط خطا را ایجاد کنید، زیرا فرآیند از حداکثر ادعای خود فراتر رفته است (درخواست فراتر از نیاز مجاز نیست)

۲. اگر درخواست از در دسترس کمتر است $Request_i \leq Available$ ، به مرحله ۳ بروید. در غیر این صورت، T_i باید منتظر بماند، زیرا منابع در دسترس نیستند (درخواست نباید بیشتر از منابع موجود باشد)

۳. با تغییر حالت به شرح زیر، فرض کنید منابع درخواستی را به T_i اختصاص می‌دهیم
ادامه اسلاید



الگوریتم درخواست منبع برای فرآیند P_i

۳. با تغییر حالت به شرح زیر، فرض کنید منابع درخواستی را به T_i اختصاص می‌دهیم

$$Available = Available - Request_i$$

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Need_i - Request_i$$

- اگر ایمن است (safe) منابع به T_i اختصاص داده می‌شوند (در صورتی که این تخصیص باعث ایجاد بن‌بست نشود، منابع به فرآیند اختصاص داده می‌شوند)
- اگر ناامن است T_i (unsafe) باید منتظر بماند و حالت تخصیص منابع قدیمی بازیابی شود (اگر تخصیص باعث بن‌بست شود، درخواست رد می‌شود و وضعیت به حالت قبل برمی‌گردد)



Example of Banker's Algorithm

- T_0 through T_4 ; ۵ تا فرایند داریم

3 resource types: انواع منابع (سه تا)

A (نمونه ۱۰), B (نمونه ۵), and C (نمونه ۷)

- وضعیت لحظه‌ای در زمان T_0

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C	A B C	A B C
T_0	0 1 0	7 5 3	3 3 2
T_1	2 0 0	3 2 2	
T_2	3 0 2	9 0 2	
T_3	2 1 1	2 2 2	
T_4	0 0 2	4 3 3	



Example of Banker's Algorithm

- سیستم در حالت ایمن قرار دارد زیرا دنباله بترتیب T_0, T_2, T_4, T_3, T_1 معیارهای ایمنی را برآورده می کند.

$$Available = Available - allocation$$

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
T_0	0 1 0	7 5 3	3 3 2	7 4 3
T_1	2 0 0	3 2 2		1 2 2
T_2	3 0 2	9 0 2		6 0 0
T_3	2 1 1	2 2 2		0 1 1
T_4	0 0 2	4 3 3		4 3 1

$$Need = Max - Allocation$$

$$7-0, 5-1, 3-0 = 7 \ 4 \ 3$$

$$3-2, 2-0, 2-0 = 1 \ 2 \ 2$$



Example of Banker's Algorithm

سیستم در حالت ایمن قرار دارد زیرا دنباله بترتیب T0 ، T2 ، T4 ، T3 ، T1 معیارهای ایمنی را برآورده می کند.

$$Available = Available - allocation$$

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>		<u>Need</u>
	A B C	A B C	A B C		A B C
T ₀	0 1 0	7 5 3	3 3 2	7-0,	7 4 3
T ₁	2 0 0	3 2 2			1 2 2
T ₂	3 0 2	9 0 2			6 0 0
T ₃	2 1 1	2 2 2			0 1 1
T ₄	0 0 2	4 3 3			4 3 1

sum: 7 2 5 - 10 5 7 = 3 3 2

کل منابع



Example of Banker's Algorithm

- حال فرض کنید که رشته T_1 (thread) درخواست یک نمونه اضافی از نوع منبع A و دو نمونه از نوع منبع C را داشته باشد، بنابراین $Request_1 = (1,0,2)$ برای تصمیم گیری در مورد اینکه آیا این درخواست را می توان بلافاصله اعطا کرد، ابتدا بررسی می کنیم که $Request_1 \leq Available$ یعنی اینکه $(1,0,2) \leq (3,3,2)$ باشد، که درست است. سپس وانمود می کنیم که این درخواست برآورده شده است، و به وضعیت جدید زیر می رسم:

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C	A B C	A B C
T_0	0 1 0	7 5 3	3 3 2
T_1	2 0 0	3 2 2	+(1,0,2)
T_2	3 0 2	9 0 2	
T_3	2 1 1	2 2 2	
T_4	0 0 2	4 3 3	

وضعیت قدیم

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
T_0	0 1 0	7 4 3	2 3 0
T_1	3 0 2	0 2 0	
T_2	3 0 2	6 0 0	
T_3	2 1 1	0 1 1	
T_4	0 0 2	4 3 1	

وضعیت جدید



Example of Banker's Algorithm

- اجرای الگوریتم ایمنی نشان می دهد که توالی $\langle T_2, T_0, T_4, T_3, T_1 \rangle$ نیازمندی ایمنی را برآورده می کند
- آیا درخواست $(3, 3, 0)$ توسط T_4 قابل اعطا است؟
- آیا درخواست $(0, 2, 0)$ توسط T_0 قابل اعطا است؟

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
T_0	0 1 0	7 4 3	2 3 0
T_1	3 0 2	0 2 0	
T_2	3 0 2	6 0 0	
T_3	2 1 1	0 1 1	
T_4	0 0 2	4 3 1	



Deadlock Detection

- اجازه دادن به سیستم برای ورود به حالت بن‌بست
- الگوریتم تشخیص
- طرح بازیابی

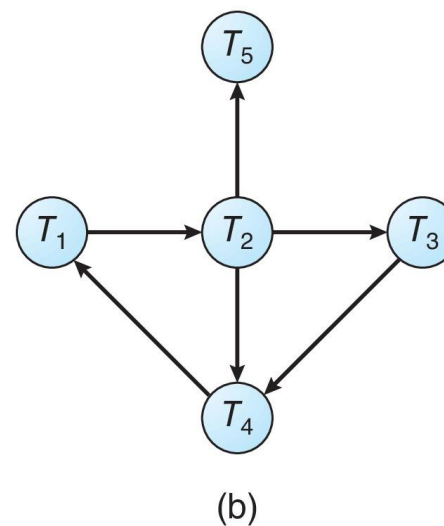
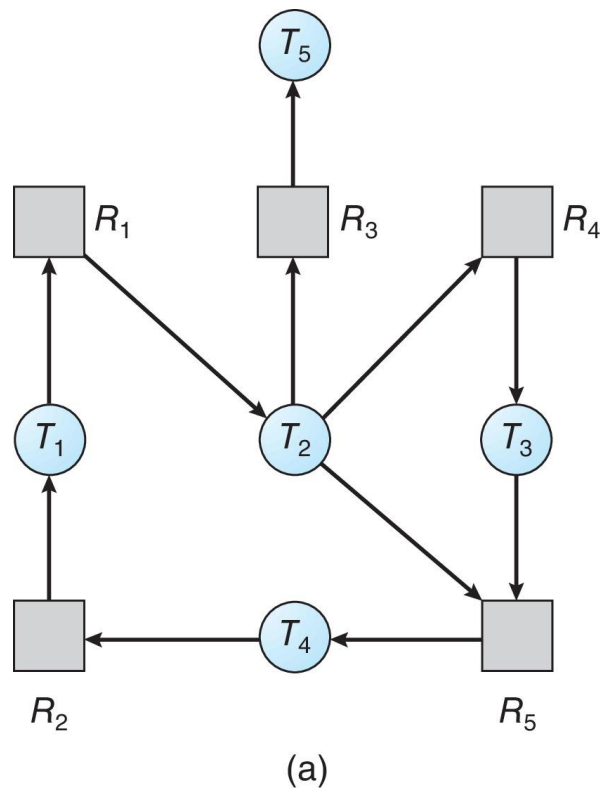


Single Instance of Each Resource Type

- این گراف وابستگی **wait-for** بین رشته‌ها را نشان می‌دهد.
 - گره‌ها نشان‌دهنده‌ی رشته‌ها Threads هستند.
 - $T_i \rightarrow T_j$ نشان می‌دهد که رشته‌ی T_i در حال انتظار برای رشته‌ی T_j است.
- به صورت دوره‌ای الگوریتمی اجرا می‌شود که به دنبال دور (Cycle) در گراف می‌گردد . وجود یک دور نشان‌دهنده‌ی بن‌بست است.
- پیچیدگی زمانی الگوریتم تشخیص دور در یک گراف از مرتبه‌ی n^2 است، که n تعداد گره‌های گراف است.



Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph



Several Instances of a Resource Type

- موجود (Available): برداری به طول m که تعداد منابع در دسترس از هر نوع را نشان می‌دهد .
- تخصیص یافته (Allocation): ماتریسی n در m که تعداد منابع از هر نوع که در حال حاضر به هر رشته تخصیص داده شده است را مشخص می‌کند .
- درخواست (Request): ماتریسی n در m که درخواست فعلی هر رشته را نشان می‌دهد.
- اگر $Request[i][j] = k$ ، به این معنی است که رشته‌ی T_i درخواست k واحد دیگر از نوع منبع R_j را دارد.



الگوریتم تشخیص بن بست

فرض کنید:

- **Work** یک بردار با طول m (تعداد نوع منابع) است.
- **Finish** یک بردار با طول n (تعداد فرآیندها) است.
- در این مرحله، مقدار اولیه **Work** برابر با بردار **Available** (مقدار منابع در دسترس) قرار داده می شود.
- سپس برای هر فرآیند $i = 1$ تا n ، بررسی می شود که آیا منابعی به آن اختصاص داده شده است ($\text{Allocation}[i] \neq 0$) یا نه.
- اگر اختصاصی وجود داشته باشد، یعنی ممکن است درگیر بن بست باشد، پس **Finish** برابر **false** می شود.
- در غیر این صورت، **Finish** برابر **true** در نظر گرفته می شود، چون فرآیند نیازی به منابع ندارد و "تمام شده" تلقی می شود.

Step 1: Initialization

1. **Work** := **Available**
2. for $i := 1$ to n do
3. if $\text{Allocation}[i] \neq 0$ then
4. **Finish**[i] := **false**
5. else
6. **Finish**[i] := **true**



الگوریتم تشخیص بن بست (ادامه)

Step 2: Find an executable process

1. while there exists an index i such that:
2. $\text{Finish}[i] == \text{false}$ and
3. $\text{Request}[i] \leq \text{Work}$
4. do
5. $\text{Work} := \text{Work} + \text{Allocation}[i]$
6. $\text{Finish}[i] := \text{true}$

در این مرحله، دنبال فرآیندی می گردیم که:

- هنوز تمام نشده ($\text{Finish}[i] == \text{false}$)
- و تقاضای آن از منابع در دسترس (Work) کمتر یا مساوی است ($\text{Request}[i] \leq \text{Work}$)

اگر چنین فرآیندی یافت شود، فرض می کنیم که می تواند اجرا شده و تمام شود. پس منابع اختصاص داده شده به آن ($\text{Allocation}[i]$) آزاد شده و به Work اضافه می شود.

در نتیجه $\text{Finish}[i]$ برابر true قرار داده می شود و دوباره این حلقه تکرار می شود تا زمانی که هیچ فرآیند قابل اجرای دیگری باقی نماند.



الگوریتم تشخیص بن بست (ادامه)

Step 3: Check for deadlock

1. if there exists an index i such that $Finish[i] == false$ then
2. The system is in a deadlock state
3. The set of such i indicates the deadlocked processes
4. else
5. No deadlock is detected

- در پایان بررسی می شود که آیا هنوز فرآیندی وجود دارد که $Finish[i] == false$ باشد.
- اگر چنین باشد، به این معناست که آن فرآیند نتوانسته اجرا شود و منتظر منابع مانده است: یعنی سیستم در حالت بن بست قرار دارد. مجموعه ی چنین اها نشان دهنده فرآیندهایی است که در بن بست قرار دارند.
- در غیر این صورت، اگر همه ی $Finish[i] == true$ باشند، یعنی هیچ بن بستى در سیستم وجود ندارد.

پیچیدگی زمانی الگوریتم تشخیص بن بست از مرتبه ی $O(m \cdot n^2)$ است.



مثالی از الگوریتم تشخیص بن بست

پنج نخ (Thread) به نام‌های T0 تا T4 داریم؛ سه نوع منبع شامل:

A با ۷ نمونه **B** (با ۲ نمونه) و **C** با ۶ نمونه

وضعیت لحظه‌ای در زمان: T0

ترتیب اجرای نخ‌ها به صورت از چپ به راست $\langle T0, T2, T3, T1, T4 \rangle$

	تخصیص (Allocation)	درخواست (Request)	در دسترس (Available)
	A B C	A B C	A B C
T0	0 1 0	0 0 0	0 0 0
T1	2 0 0	2 0 2	
T2	3 0 3	0 0 0	
T3	2 1 1	1 0 0	
T4	0 0 2	0 0 2	

منجر به $Finish[i] = true$ برای همه‌ی آنها خواهد شد.



مثالی از الگوریتم تشخیص بن‌بست (ادامه)

■ نخ **T2** یک نمونه‌ی اضافی از نوع **C** درخواست می‌کند.

Request

A B C

T_0 0 0 0

T_1 2 0 2

T_2 0 0 1

T_3 1 0 0

T_4 0 0 2

وضعیت سیستم:

می‌توان منابع نگه‌داری شده توسط نخ **T0** را آزاد کرد،
اما منابع آزادشده برای برآورده کردن درخواست‌های سایر نخ‌ها کافی نیست.

■ بنابراین، بن‌بست **Deadlock** در سیستم وجود دارد،
که شامل فرآیندهای **T1**، **T2**، **T3**، و **T4** است.



کاربرد الگوریتم تشخیص

- زمان و دفعات فراخوانی به موارد زیر بستگی دارد:
 - احتمال وقوع بن بست چقدر است؟
 - چند فرآیند باید عقب گرد (Rollback) انجام دهند؟
 - برای هر چرخه مجزا، یک فرآیند
- اگر الگوریتم تشخیص به طور خودسرانه فراخوانده شود، ممکن است چرخه‌های زیادی در نمودار منابع وجود داشته باشد و بنابراین نمی‌توانیم تشخیص دهیم که کدام یک از رشته‌های بن بست شده «مسبب» بن بست هستند.



Recovery from Deadlock: Process Termination

- لغو (Abort) کردن تمام رشته‌های بن‌بست‌شده
- تک‌به‌تک فرآیندها را لغو کنید تا چرخه بن‌بست از بین برود
- چه ترتیبی برای لغو انتخاب کنیم؟
 - اولویت رشته
 - مدت زمان محاسبه رشته و زمان باقی‌مانده تا تکمیل
 - منابعی که رشته استفاده کرده است
 - منابعی که رشته برای تکمیل نیاز دارد
 - چند رشته باید خاتمه یابند
 - آیا رشته تعاملی است یا دسته‌ای؟



Recovery from Deadlock: Resource Preemption

- انتخاب قربانی - به حداقل رساندن هزینه
- عقب‌گرد - (Rollback) بازگشت به یک حالت امن، راه‌اندازی مجدد رشته برای آن حالت
- گرسنگی - (Starvation) ممکن است همیشه یک رشته خاص به عنوان قربانی انتخاب شود، تعداد عقب‌گردها را در عامل هزینه لحاظ کنید

پایان فصل ۸

