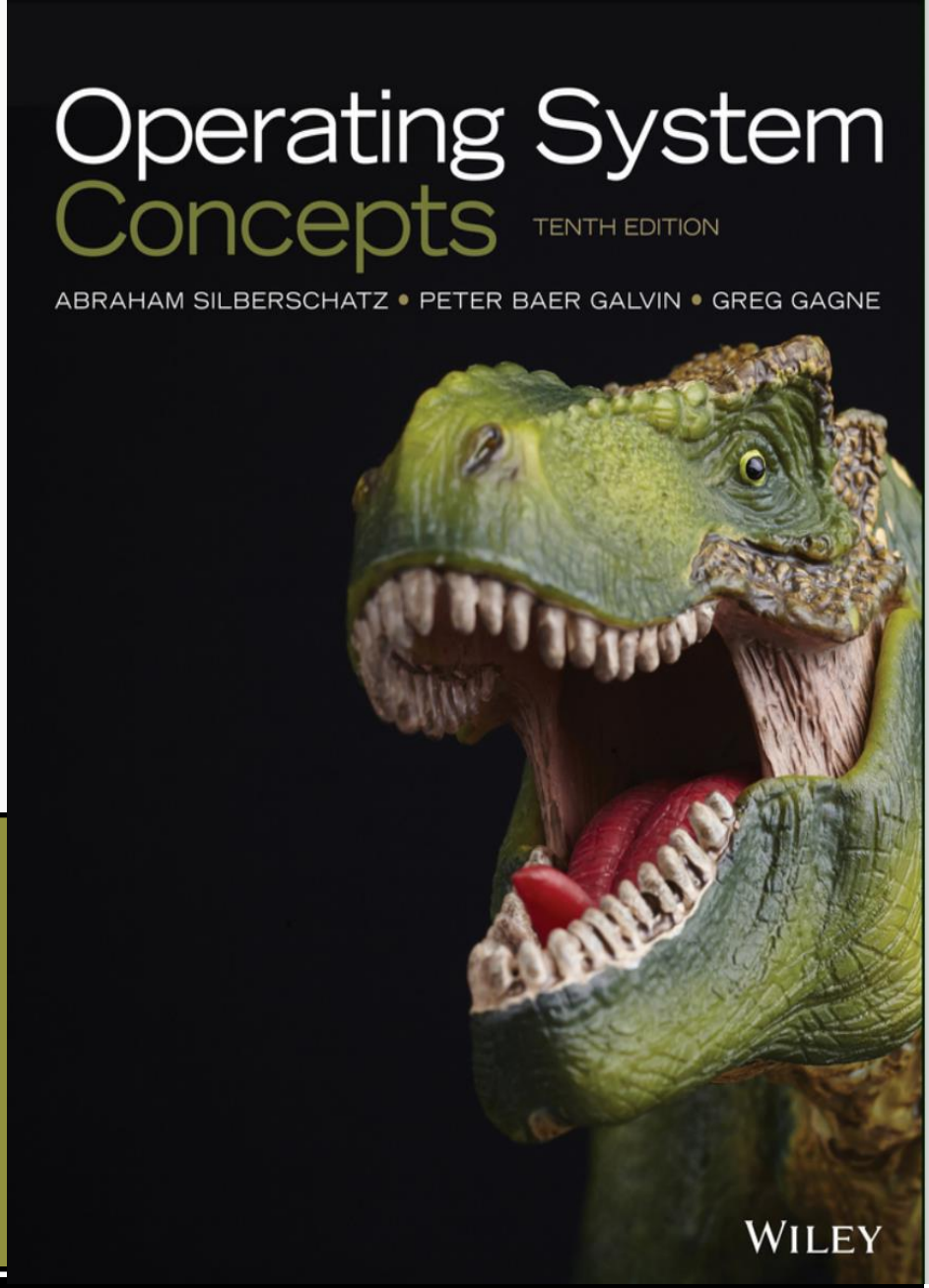
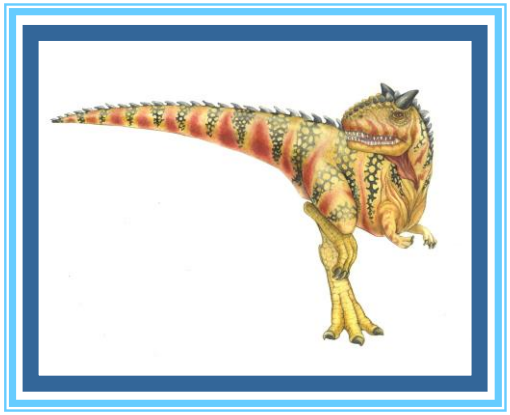
سیستم‌های عامل



Dr. Taghinezhad

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

فصل ۸: بن‌بست



Outline

- مشخصه بن‌بست (Deadlock Characterization)
- روش‌های مدیریت بن‌بست (Methods for Handling Deadlocks)
- جلوگیری از بن‌بست(Deadlock Prevention)
- جلوگیری از قفل شدن(Deadlock Avoidance)
- تشخیص بن‌بست(Deadlock Detection)
- بازیابی از بن‌بست(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، .... 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)

بن‌بست با سمافور

- **داده‌ها:**
  - یک سیمافور S1 با مقدار اولیه ۱
  - یک سیمافور S2 با مقدار اولیه ۱
  - دو رشته (ترد) T1 وT2
- **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 {T0, T1, ... Tn} وجود دارد که به گونه‌ای است که T0 منتظر منبعی است که توسط T1 نگهداری می‌شود، T1منتظر منبعی است که توسط T2 نگهداری می‌شود، ... Tn–1 منتظر منبعی است که توسط Tn نگهداری می‌شود و Tn منتظر منبعی است که توسط T0نگهداری می‌شود.

گراف تخصیص منابع

- مجموعه‌ای از رئوس V و لبه (کمان‌های) E
- Vبه دو دسته تقسیم می‌شود :

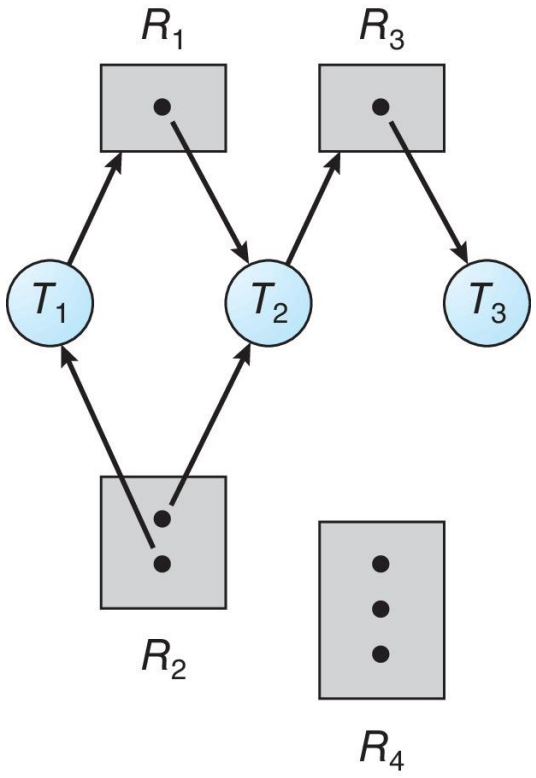
- T = {T1, T2, …, Tn}, شامل‌همه تردهای سیستم

- R = {R1, R2, …, Rm}, شامل همه منابع سیستم

- **انواع لبه در نمودار تخصیص منابع:**

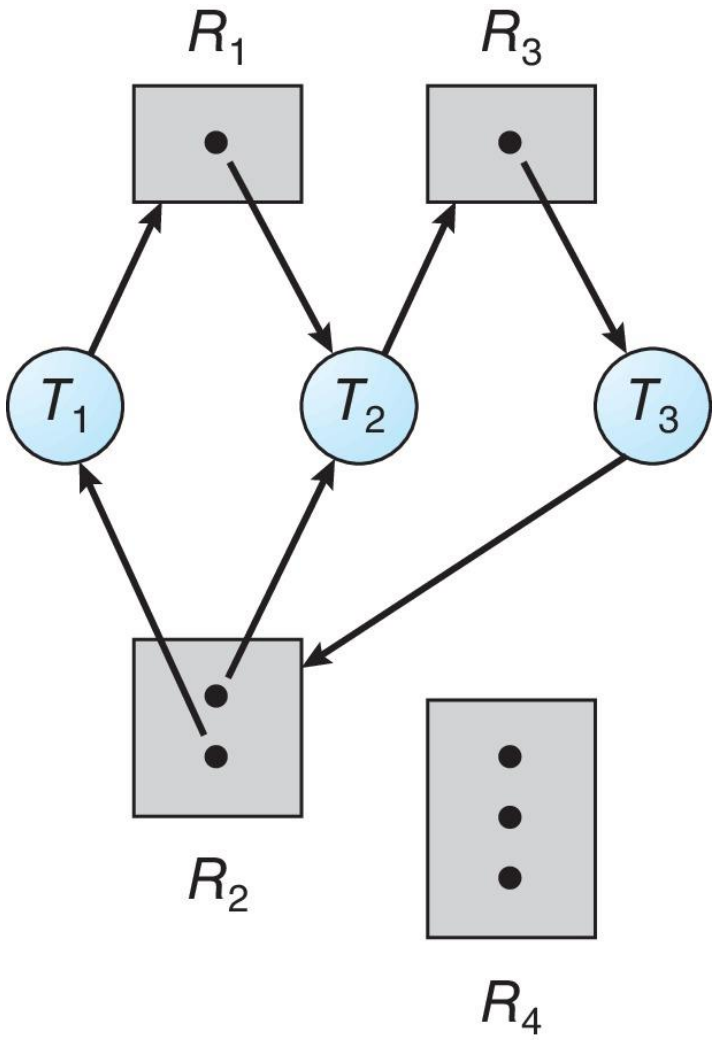
- لبه درخواست – (request edge) جهت‌دار Ti → Rj
- لبه انتساب – (assignment edge) جهت‌دار Rj → Ti

نمونه‌ای از گراف تخصیص منابع

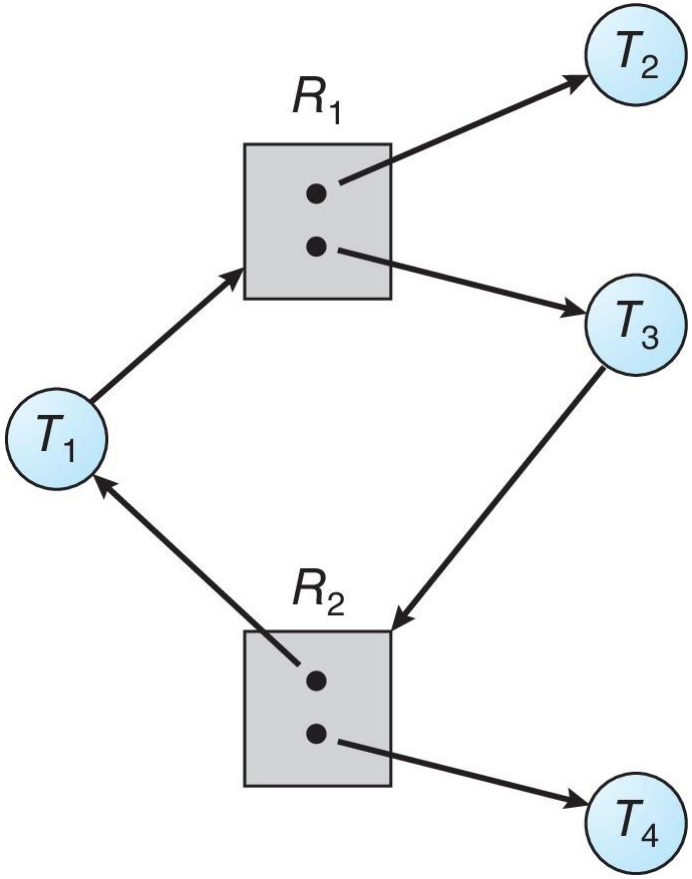


- یک نمونه از R1
- دو نمونه از R2
- یک نمونه از R3
- سه نمونه از R4
- T1 یک نمونه از R2 را در اختیار دارد و منتظر یک نمونه از R1 است.
- T2 یک نمونه از R1 و یک نمونه از R2 را در اختیار دارد و منتظر یک نمونه از R3 است.
- T3 یک نمونه از R3 را در اختیار دارد.

گراف تخصیص منابع با بن‌بست



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 Preventionپیشگیری از بن بست

یکی از چهار شروط را برای بن‌بست باطل کنید:

- **منع دسترسی همزمان** – برای منابع اشتراکی (مانند فایل‌های 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 می‌کند که هیچ منبع دیگری را در اختیار نداشته باشد.
- الزام رشته‌ها 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 می‌تواند منجر به پایین آمدن بهره‌وری منابع و همچنین گرسنگی (عدم تخصیص منابع به یک رشته برای مدت طولانی) شود.

پیشگیری از بن بست(Cont.)

- **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 پس‌گرفته 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.

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 Avoidanceاجتناب 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 بنبست 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 .

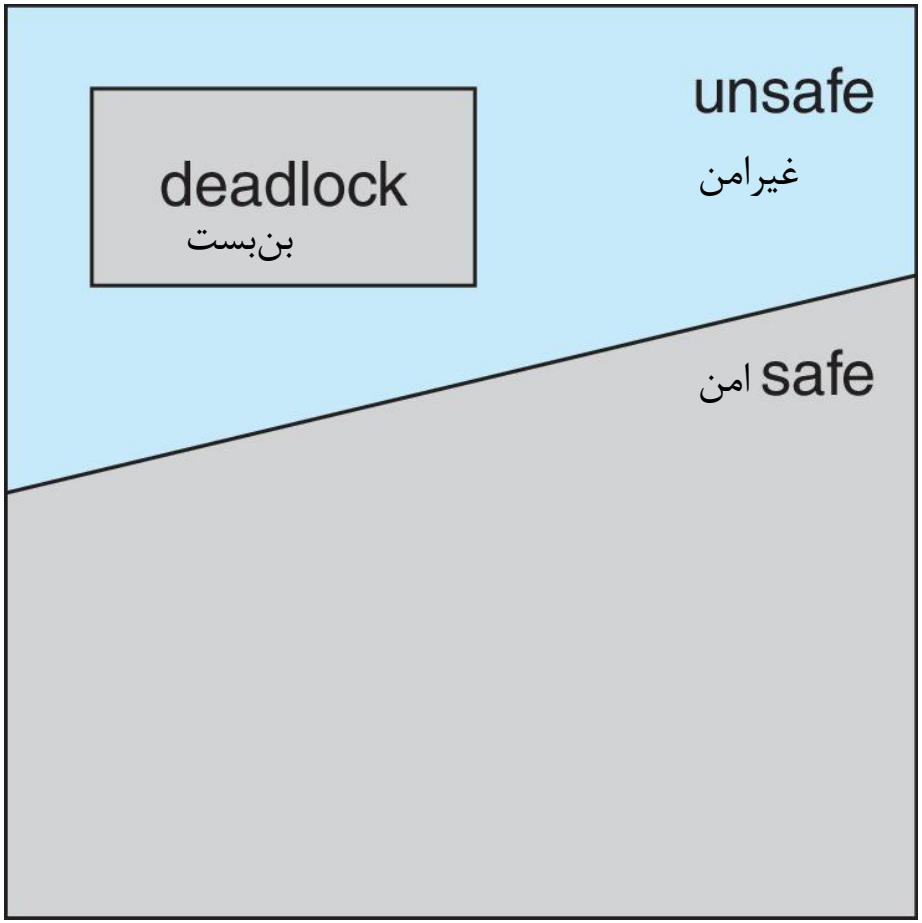
حالت امن Safe State

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

حقایق اساسی Basic Facts

- اگر سیستم در حالت امن باشد، بن‌بستی وجود ندارد.
- اگر سیستم در حالت ناامن باشد، احتمال بن‌بست وجود دارد.
- اجتناب :اطمینان حاصل می‌کند که سیستم هرگز وارد حالت ناامن نمی‌شود.

Safe, Unsafe, Deadlock State



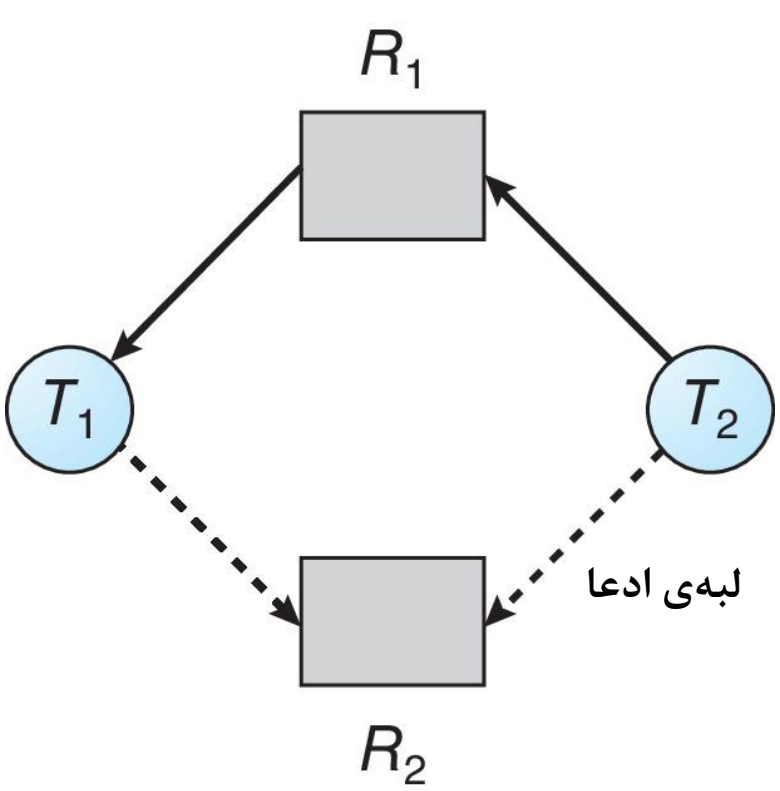
Avoidance Algorithms الگوریتمهای اجتناب

- **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.

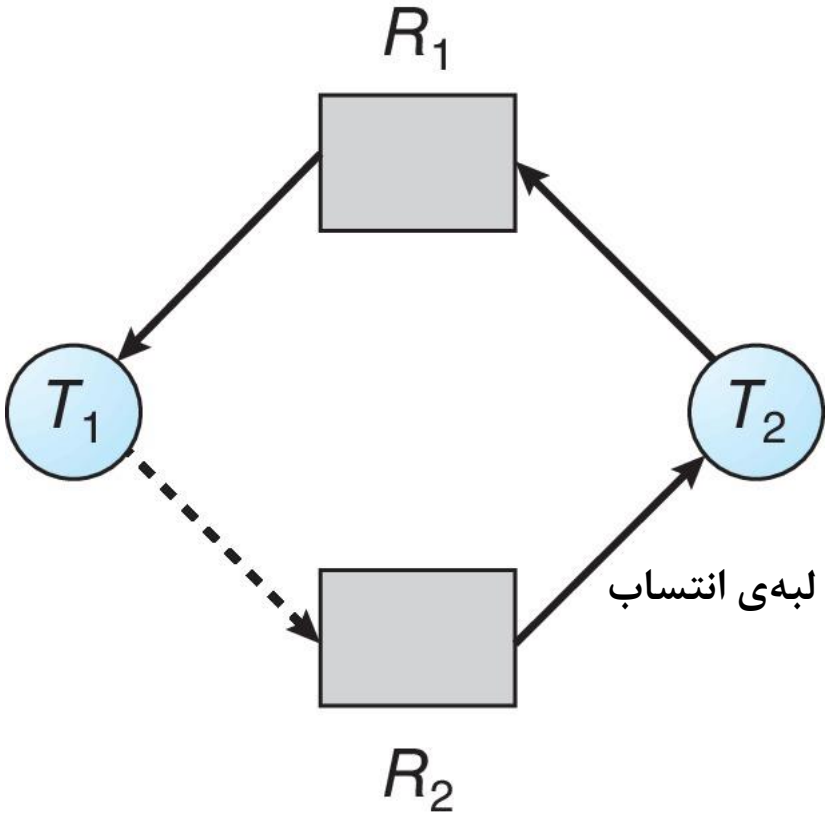
Resource-Allocation Graph Scheme

- **لبه‌ی ادعا (claim edge):** جهت‌دار `Ti` → `Rj` نشان می‌دهد که فرآیند `Tj` ممکن است منبع `Rj` را درآینده درخواست کند .این لبه با خط چین نمایش داده می‌شود.
- **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 نشان‌داده می‌شود.
- **بازگشت لبه‌ی انتساب به لبه‌ی ادعا :** هنگامی که منبعی توسط یک رشته آزاد می‌شود، لبه‌ی انتساب دوباره به لبه‌ی ادعا تبدیل می‌شود.
- منابع **باید قبلًا در سیستم ادعا شوند.(claimed a priori)**

Resource-Allocation Graph



فرض کنید که T2 درخواست R2را می کند .
اگرچه R2در حال حاضر آزاد است، نمی‌توانیم آن را به T2اختصاص دهیم، زیرا این عمل یک چرخه در نمودار ایجاد می‌کند.



وضعیت ناامن در نمودار تخصیص منابع

Resource-Allocation Graph Algorithm

- فرض کنید که رشته‌ی Tiدرخواست یک منبع Rjرا 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.

Banker's Algorithm الگوریتم بانکدار

- 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.

Data Structures for the Banker’ s Algorithm

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

- **بردار دردسترس** یا **Available** به طول m اگر available[j] = k، به این معنی است که kنمونه 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 می‌تواند 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 اینصورت فرآیند T_i در حال حاضر k نمونه از نوع منبع R_jرا در اختیار دارد .
- **ماتریس نیازمندی Need**: ماتریس $n \times m$ اگر Need[i,j] = k، پس T_iممکن 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]

Safety Algorithm

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 for i = 0, 1, …, n- 1
```

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

- لف) **Finish[i] = false** (فرآیند i هنوز تمام نشده است)
- ب) **Need[i] ≤ 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.

Resource-Request Algorithm for Process 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.

Resource-Request Algorithm for Process P_i

۱. اگر $\boldsymbol{Request_i \leq Need_i}$ **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 $\boldsymbol{Request_i \leq Available}$ ، **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

$$Available = Available - Request_i$$

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Need_i - Request_i$$

- 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

- 5 **threads T_0 through T_4;** ۵ تا فرایند

  3 **resource types:** انواع منابع

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

- وضعیت لحظه‌ای در زمان T_0

	Allocation	*Max*	*Available*
	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 T1 ، T3، T4، T2، T0 معیارهای ایمنی را برآورده می کند.

	Allocation	*Max*	*Available*
	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	

	Need
	A B C
T_0	7 4 3
T_1	1 2 2
T_2	6 0 0
T_3	0 1 1
T_4	4 3 1

Example of Banker's Algorithm

- حال فرض کنید که رشته (thread) T1 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 (3,3,2) ≤(1,0,2)باشد، که درست است .سپس وانمود می کنیم که این درخواست برآورده شده است، و به وضعیت جدید زیر می رسیم:

	Allocation	*Max*	*Available*
	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

	Allocation	*Need*	*Available*
	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 < T1, T3, T4, T0, T2> نیازمندی ایمنی را برآورده می کند
- آیا درخواست (۳,۳,۰) توسط T4 قابل اعطا است؟
- آیا درخواست (۰,۲,۰) توسط T0 قابل اعطا است؟

	Allocation	*Need*	*Available*
	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	