Amirali Farazmand

99522329

HW4

---

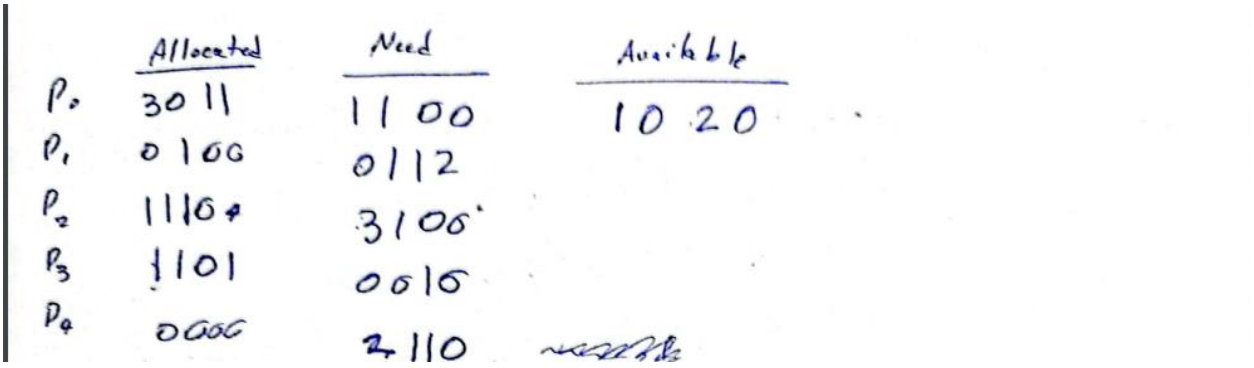
1. سیستمی دارای ۵ فرایند و چهارمنبع در حالت زیر به سر میبرد ، در چه صورتی وقوع بن بست حتمی است ؟

	منابع تخصیص یافته			
	R_0	R_1	R_2	R_3
P_0	۳	۰	۱	۱
P_1	۰	۱	۰	۰
P_2	۱	۱	۱	۰
P_3	۱	۱	۰	۱
P_4	۰	۰	۰	۰

	منابع مورد نیاز			
	R_0	R_1	R_2	R_3
P_0	۱	۱	۰	۰
P_1	۰	۱	۱	۲
P_2	۳	۱	۰	۰
P_3	۰	۰	۱	۰
P_4	۲	۱	۱	۰

کل منابع اولیه			
R_0	R_1	R_2	R_3
۶	۳	۴	۲

a) فرایند P1 یک واحد از منبع R2 درخواست کند .
b) فرایند R1 یک واحد از منبع R2 درخواست کند و فرایند P4 اولین واحد R2 را درخواست نماید .
c) فرایند P3 یک واحد از منبع R2 را درخواست کند و فرایند P4 کلیه منابع مورد نیازش را درخواست کند .
d) فرایند P1 یک واحد از منبع R2 را درخواست کند و فرایند P4 آخرین واحد R2 را درخواست نماید .



a) $Request_1 = (0010)$

$Request_1 \le Need_1$ ✓

$Request_1 \le Available$ ✓ => allocate صورت میگیرد =>

	Allocated
P_0	3011
P_1	0110
P_2	1110
P_3	1101
P_4	0000

Available: 1010

$Need_1 = (0,1,0,2)$

=> P_1 میتواند منابع را با فاصله بگیرد و کارش را انجام دهد

اگر الگوریتم detection را اجرا کنیم میفهمیم که با Available فعلی و Need ها میتوان ترتیب $<P_3, P_4, P_0, P_1, P_2>$ را یافت پس ددلاک نشود

b) $Request_1 = (0010)$ $Request_4 = (0010)$

$Request_1 < Need_1$ =>

$Request_4 <$ Available

$\{ Request_4 \le Need_4$, $Request_4 \le Available$ => P_1 و P_4 هم منبع را میگیرند و پروسس هایی که بعدا میرسند به چیزی نمیرسند

	Allocated	Need	Available
P_1	0110	0102	1000
P_4	0010	2101	

→ با Available و Need ها دنباله ای را نمیتوان پیدا کرد؛ سیستم در وضعیت ناامن قرار دارد و ممکن است ددلاک اتفاق بیفتد. فرض بر این شده که request های دیگری ممکن است بیاید

* اگر این دو درخواست همزمان نیایند میتوان به آنها تخصیص داد

c) $Request_1 = (0010)$, $Request_4 = (2,1,1,0)$

* میتوانیم اول برای P_1 شرایط را بررسی کنیم و تخصیص بدهیم بخش a و سپس برای P_4 متوجه شویم $Request_4 < Available$ نیست پس P_4 باید صبر کند تا بقیه پراسس ها ریسورس آزاد کنند. در صورتی که P_4 هم اول بررسی کنیم هم به همین نتیجه میرسیم که باید صبر کند و P_1 هم میتواند منبعش را بگیرد و مشکلی ندارد. در هر صورت P_4 باید wait بخورد. Available ما که (1,0,1,0) شده است فرض بر این است که P_4 از این منابع چیزی برنمیدارد و سپس منتظر دیگر منابع باشد. در این صورت به حالت مطرح شده در (a) میرسیم با این تفاوت که P_4 در اولین فرصت ریکوئستش را میگیرد و کارش را پیش میبرد.

* چون فرض نشده که در ریسورس ها اولویتی وجود ندارد هر کدام در طول کل کلاس هم به چنین چیزی برنخوردیم. پس حالت d مثل b است

چون ۲: در سوالات بررسی شده وجود ددلاک با Request/Need همزمان بررسی نشده بود و در کتاب هم 8.3 ما اینجا هم میبینیم که آنها هم بررسیده است که در هنگام تخصیص میتوان برطرف کرد. در کل سوال خوبی نبود

8.6.3.2 Resource-Request Algorithm

Next, we describe the algorithm for determining whether requests can be safely granted. Let $\boldsymbol{Request}_i$ be the request vector for thread T_i. If $\boldsymbol{Request}_i[j] == k$, then thread T_i wants k instances of resource type R_j. When a request for resources is made by thread T_i, the following actions are taken:

1. If $\boldsymbol{Request}_i \leq \boldsymbol{Need}_i$, go to step 2. Otherwise, raise an error condition, since the thread has exceeded its maximum claim.
2. If $\boldsymbol{Request}_i \leq \boldsymbol{Available}$, go to step 3. Otherwise, T_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to thread T_i by modifying the state as follows:

$$\boldsymbol{Available} = \boldsymbol{Available} - \boldsymbol{Request}_i$$
$$\boldsymbol{Allocation}_i = \boldsymbol{Allocation}_i + \boldsymbol{Request}_i$$
$$\boldsymbol{Need}_i = \boldsymbol{Need}_i - \boldsymbol{Request}_i$$

   If the resulting resource-allocation state is safe, the transaction is completed, and thread T_i is allocated its resources. However, if the new state is unsafe, then T_i must wait for $\boldsymbol{Request}_i$, and the old resource-allocation state is restored.

2. شرایطی که باعث به وجود آمدن بن بست میشود را شرح دهید .

ددلاک هنگامی رخ میدهد که 4شرط مربوطه اش برقرار نباشند.

4شرط بدین صورت هستند:

1. Mutual-exclusion: ریسورس non-sharable تنها توسط 1 پراسس مورد استفاده باشد و دیگران به آن در آنموقع دسترسی نداشته باشند.
2. Hold & wait: پراسس ریسورس هایی که میخواهد و در دسترس اند را بگیرد و آنهایی که دست دیگر پراسس ها اند را صبر کند تا آزاد شوند و بعد آنهارا بگیرد.
3. No-preemption: ریسورس ها وسط کار با پراسسی از آن گرفت مگر آنکه پراسس خودش آنرا آزاد کند.
4. Circular-wait: هنگامی که پراسس ها برای بدست آوردن ریسورس دایره وار منتظر پراسس بعدی خود باشند و wait کنند تا ریسورس آزاد شود رخ میدهد. (پراسس 1منتظر2 ،2منتظر3، ....،n منتظر1).

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

1. **Mutual exclusion**. At least one resource must be held in a nonsharable mode; that is, only one thread at a time can use the resource. If another thread requests that resource, the requesting thread must be delayed until the resource has been released.
2. **Hold and wait**. A thread must be holding at least one resource and waiting to acquire additional resources that are currently being held by other threads.
3. **No preemption**. Resources cannot be preempted; that is, a resource can be released only voluntarily by the thread holding it, after that thread has completed its task.
4. **Circular wait**. A set $\{T_0, T_1, ..., T_n\}$ of waiting threads must exist such that T_0 is waiting for a resource held by T_1, T_1 is waiting for a resource held by T_2, ..., T_{n-1} is waiting for a resource held by T_n, and T_n is waiting for a resource held by T_0.

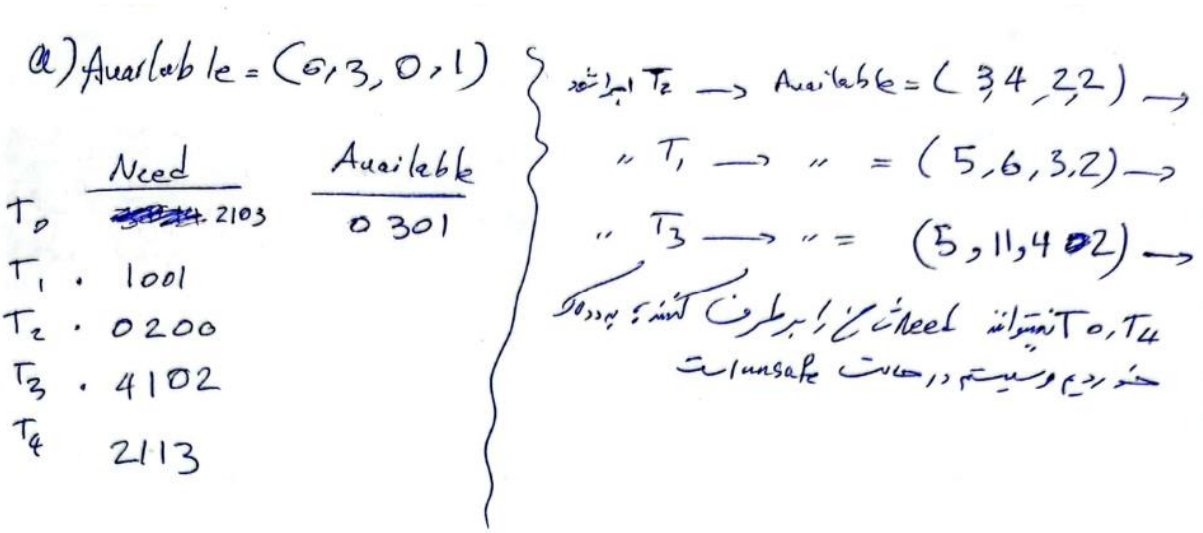
We emphasize that all four conditions must hold for a deadlock to occur. The circular-wait condition implies the hold-and-wait condition, so the four

3. اسنپ شات زیر را برای سیستم در نظر بگیرید :

	Allocation	Max
	A B C D	A B C D
T_0	3 0 1 4	5 1 1 7
T_1	2 2 1 0	3 2 1 1
T_2	3 1 2 1	3 3 2 1
T_3	0 5 1 0	4 6 1 2
T_4	4 2 1 2	6 3 2 5

با استفاده از الگوریتم بانکدار ، تعیین کنید که آیا هر کدام از حالات زیر در حالت نا امن هستند یا خیر ،اگر یک استیت درحالت امن بود ترتیب اجرا را بنویسید و اگر در حالت نا امن بود توضیح دهید که چرا نا امن است ؟

a) Available = (0,3,0,1)
b) Available = (1,0,0,2)



b) Available = (1, 0, 0, 2)

اجرا شود T_1 ⟶ Available = (3, 2, 1, 2) ⟶

" T_2 ⟶ " = (6, 3, 3, 3) ⟶

" T_0 ⟶ " = (9, 3, 4, 7) ⟶

" T_3 ⟶ " = (9, 8, 5, 7) ⟶

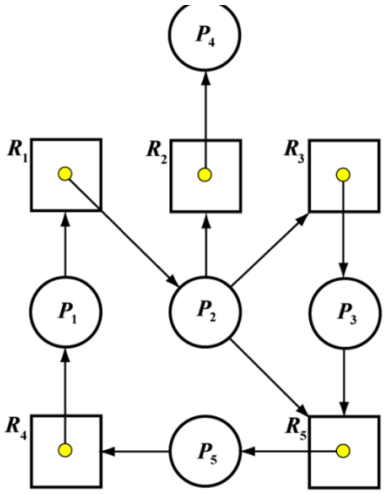
" T_4 ⟶ " = (13, 10, 6, 9) ⟶

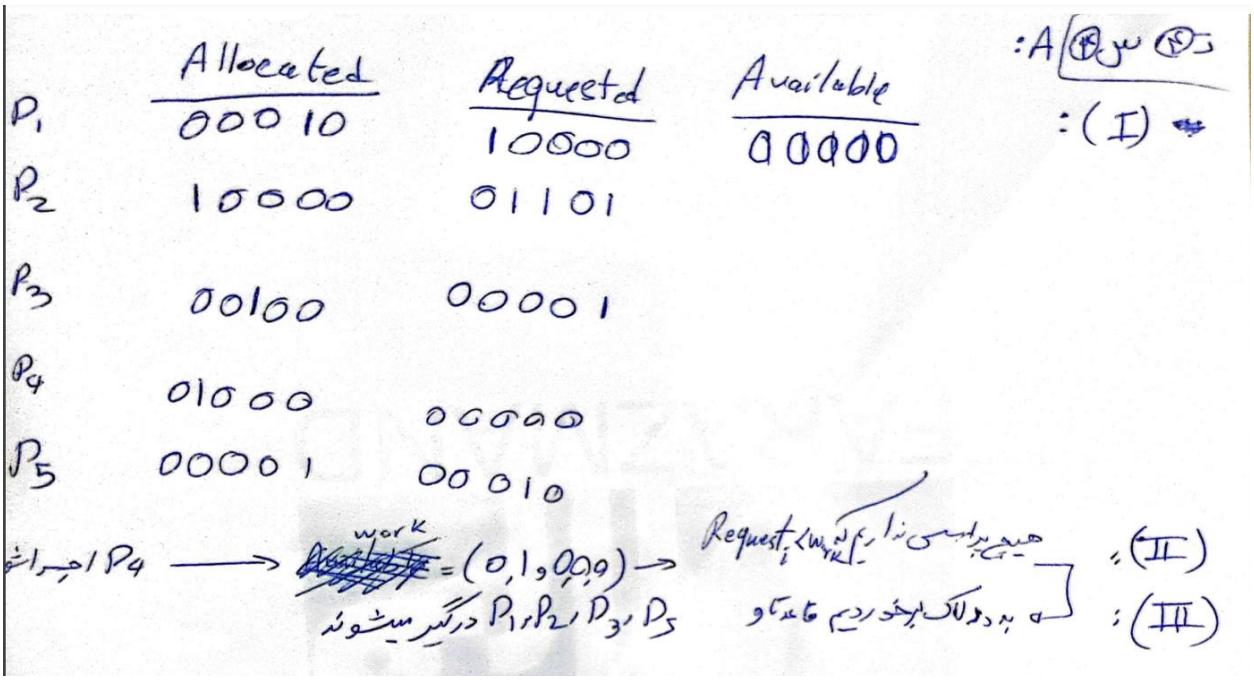
دنباله T_1, T_2, T_0, T_3, T_4 را داریم که نشانگر این است که به ددلاک نمیخوریم و سیستم هم در حالت safe قرار دارد

4. گراف های اختصاص زیر را در نظر بگیرید و با توجه به آنها برای هرکدام به سوالات زیر پاسخ دهید .

I. ماتریس های requested , allocation , available را بکشید .
II. الگوریتم مرحله به مرحله تشخیص بن بست را بنویسید .
III. آیا بن بست وجود دارد ؟ در صورت وجود کدام فرایند ها درگیر هستند ؟

A)





با کشیدن wait-for-graph هم میشد به ددلاک داشتن رسید.

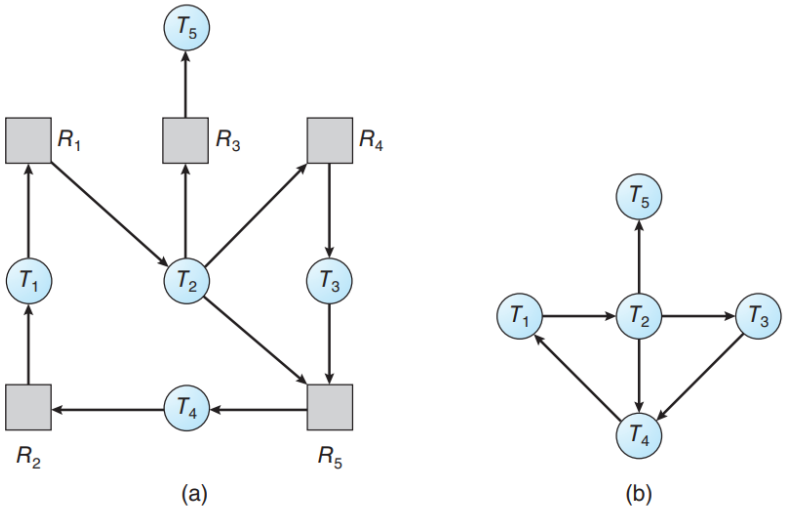
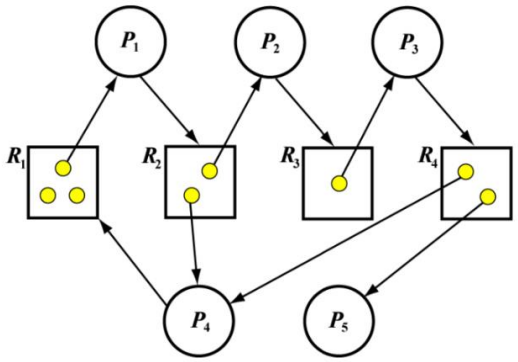
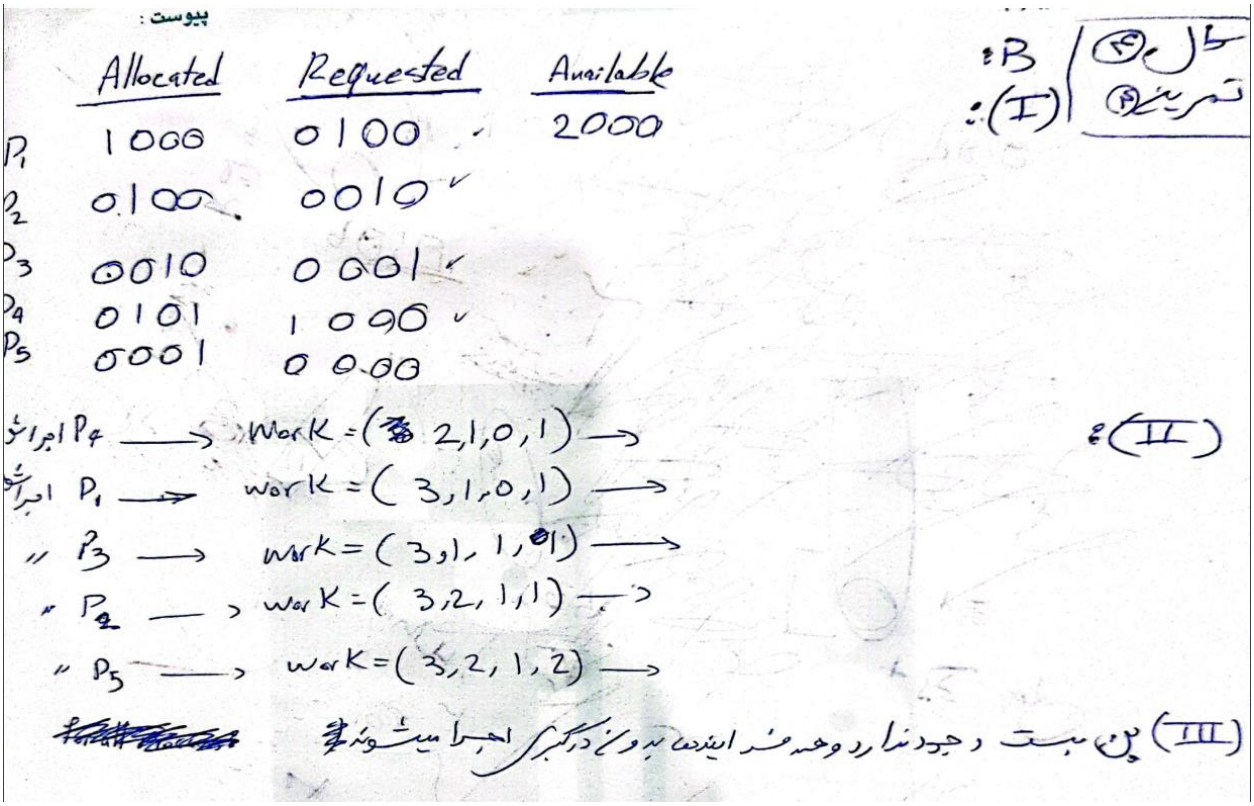


Figure 8.11 (a) Resource-allocation graph. (b) Corresponding wait-for graph.

B)





1. Let ***Work*** and ***Finish*** be vectors of length m and n, respectively. Initialize ***Work*** = ***Available.*** For $i = 0, 1, ..., n-1$, if $\textbf{\textit{Allocation}}_i \neq 0$, then ***Finish***[$i$] = ***false.*** Otherwise, ***Finish***[i] = ***true.***
2. Find an index i such that both
   a. ***Finish***[i] == ***false***
   b. $\textbf{\textit{Request}}_i \leq$ ***Work***

   If no such i exists, go to step 4.
3. ***Work*** = ***Work*** + $\textbf{\textit{Allocation}}_i$
   Finish[i] = ***true***
   Go to step 2.
4. If ***Finish***[i] == ***false*** for some i, $0 \leq i < n$, then the system is in a deadlocked state. Moreover, if ***Finish***[i] == ***false,*** then thread T_i is deadlocked.