سؤال اول)

۱) سیستمی با ۵ پردازه و ۴ منبع در حالت زیر قرار دارد:

Allocation

	A	B	C	D
P_0	0	0	1	2
P_1	1	0	0	0
P_2	1	3	5	4
P_3	0	6	3	2
P_4	0	0	1	4

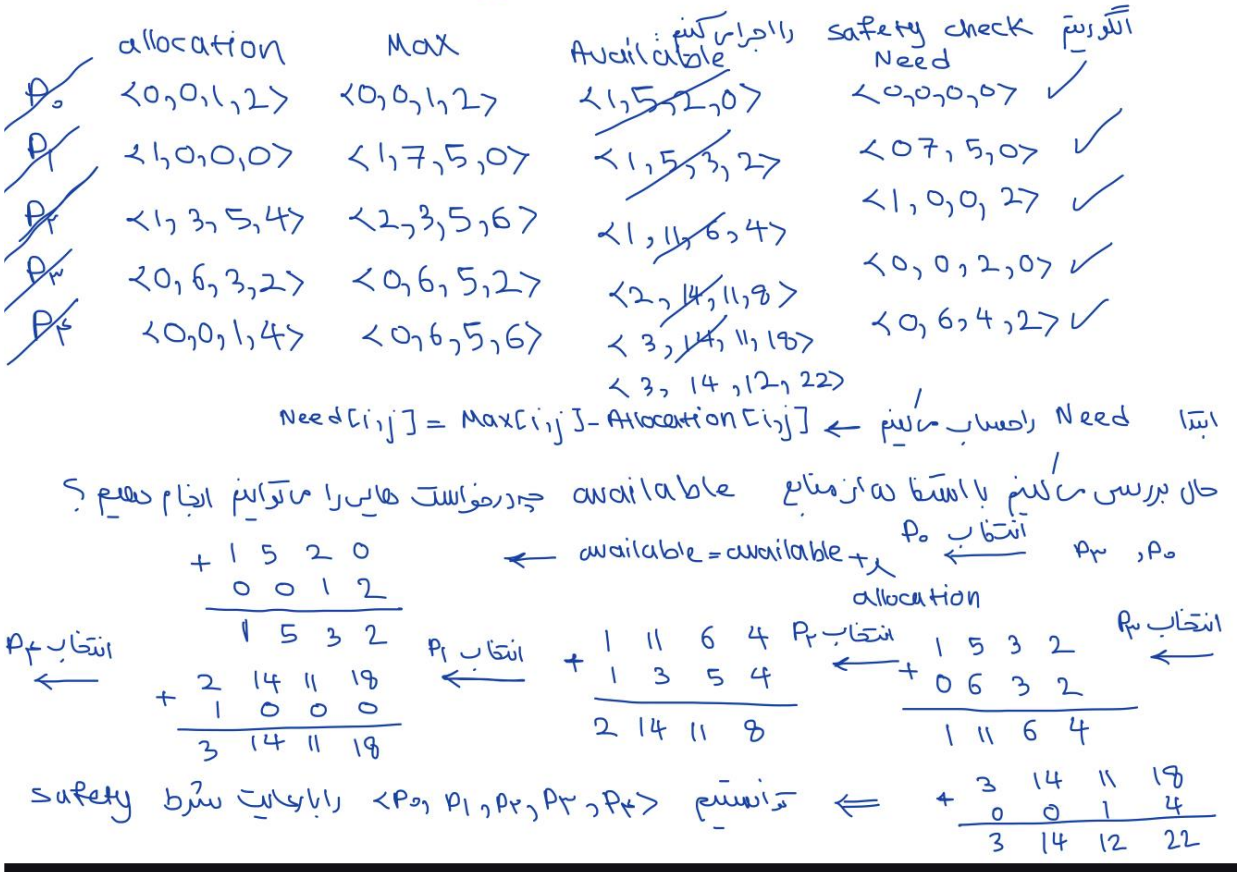
Max

	A	B	C	D
P_0	0	0	1	2
P_1	1	7	5	0
P_2	2	3	5	6
P_3	0	6	5	2
P_4	0	6	5	6

Available

A	B	C	D
1	5	2	0

الف) محاسبه کنید که آیا سیستم در حالت امن است؟ بله ، زیرا :



ب) اگر درخواستی از پردازه p_1 به صورت (0,4,2,0) به شکل تعداد منابع درخواستی برای (A, B, C, D) وارد شود آیا می توان فورا به آن تخصیص داد؟ بعد از تخصیص، سیستم در وضعیت امن قرار میگیرد؟

	allocation	Max	Need	Available
P_0	<0,0,1,2>	<0,0,1,2>	<0,0,0,0> ✓	<1,5,2,0>
P_1	<1,4,2,0>	<1,7,5,0>	<0,3,3,0> ✓	<1,1,0,0>
P_2	<1,3,5,4>	<2,3,5,6>	<1,0,0,2> ✓	<1,1,1,2>
P_3	<0,6,3,2>	<0,6,5,2>	<0,0,2,0> ✓	<2,4,6,6>
P_4	<0,0,1,4>	<0,6,5,6>	<0,6,4,2> ✓	<2,10,9,8>

request <0,4,2,0> from P_0

ابتدا پیش شرایط را چک میکنیم:

Request$_i$ ≤ Need$_i$ → <0,4,2,0> ≤ <0,7,5,0> ✓

Request$_i$ ≤ Available → <0,4,2,0> ≤ <1,5,2,0> ✓

```
 1 5 2 0        1 0 0 0        0 7 5 0
-0 4 2 0       +0 4 2 0       -0 4 2 0
---------      ---------      ---------
 1 1 0 0        1 4 2 0        0 3 3 0
```

حال safety check را اجرا میکنیم:

```
 0 0 1 2        1 1 1 2        2 4 6 6
+1 1 0 0       +1 3 5 4       +0 6 3 2
---------      ---------      ---------
 1 1 1 2        2 4 6 6        2 10 9 8
```

انتخاب P_0 ← انتخاب P_2 ← انتخاب P_3 ←

```
 2 10 9 8       2 10 9 8
+0 0 1 4       +1 4 2 0
---------      ---------
 2 10 10 12     3 14 11 8
```

انتخاب P_4 ← انتخاب P_1 ←

⇐ درحالت safe قرار میگیرد.

⇐ بله، می توان تخصیص داد.

با تشکر از خانم مینا بیکی

سوال ۲)
یک صف (queue) مجازی به اندازه تعداد process ها میسازیم در اینجا چون N عدد process داریم به اندازه N. هر پردازه ایی که بخواهد وارد ناحیه بحرانی (critical section) شود به این صف وارد میشود سپس با توجه به مکانیزم صف موقعی که زمان خارج شدنش از صف فرا رسید اجازه ورود به ناحیه بحرانی را پیدا میکند.
بدین منظور دو تابع به نام های lock, unlock تعریف میکنیم و با اجرای تابع lock برای هر process باعث میشویم تا زمانی که تا اخر صف پیش نرفته است وارد ناحیه بحرانی نشود و بعد از آن که همه process های جلوتر کارشان تمام شد و به اخر صف رسیدیم وارد ناحیه بحرانی میشود و در نهایت unlock را اجرا میکند.

```
LOCK( Process PID){
    for( int i = 0;i<N;++i){
        Turn[i]=PID;
        Flag[PID]=i;
        while((for all k != PID, Flag[k]<i) or (Turn[i] != PID))
        ;
    }
}
/* Critical Section */
UNLOCK(Process PID){
    Flag[PID]=-1
    else
    Flag[PID]=0
}
```

اکنون شروط ۳ گانه را بررسی میکنیم :

شرط انحصار متقابل (mutual exclusion) : با توجه به اینکه در هر لحظه فقط یک process میتواند در انتهای صف باشد بنابراین پس فقط یک process میتواند در ناحیه بحرانی باشد پس این شرط برقرار است.

شرط پیشرفت (progress) : با توجه به اینکه هر process پس از خروجش از ناحیه بحرانی با اجرای تابع unlock باید flag خود را 1- کند پس این اجازه را به بقیه process ها میدهد که وارد ناحیه بحرانی بشوند بنابراین این شرط نیز برقرار است.

شرط انتظار محدود (bounded waiting) : با توجه به اینکه سایز صف را میدانیم امکان ندارد بیش از N – 1 پردازه طول بکشد که نوبت process جدید شود چون هر process ایی که خارج میشود با false کردن نوبت خود باعث میشود بقیه process ها بتوانند وارد شوند و اگر بخواهد دوباره وارد شود دوباره

باید به اول صف برود پس نمیتواند بی نهایت بار وارد و خارج شود و موجب انتظار باقی process ها شود بنابراین زمان محدودی طول میکشد تا هر process وارد ناحیه بحرانی بشود.

سوال ۳)
الگوریتم ارائه شده انحصار متقابل ندارد ، پیش روی ندارد ، انتظار محدود دارد.
انحصار متقابل نداریم ، برای مثال:
فرایند ۰ وارد enter_region می شود ، به دلیل برقرار نبود شرط:
Interested[other1] || interested[other2]
وارد بخش بحرانی می شود. سپس فرایند ۲ وارد این تابع شده و تا خط قبل از while اجرا می شود. سپس فرایند ۱ وارد شده و تا خط قبل از while اجرا می شود. حال مقدار turn برابر id فرایند ۲ است و به همین خاطر شرط process != turn برای فرایند ۲ برقرار نیست و وارد ناحیه بحرانی می شود)که فرایند ۰ هم هنوز ممکن است داخل آن باشد)
پیش روی نداریم ، برای مثال:
فرض کنید دو فرایند ۰ و ۱ درخواست ورود به بخش بحرانی را داشته باشند اما فرایند ۲ نه. درین صورت هرگز فرایند های ۰ و ۱ از خط while عبور نخواهند کرد. چرا که turn برابر فرایند ۲ است و بخش other2[interested] || other1[interested] هم صحیح است.
انتظار محدود داریم چرا که تعداد دفعاتی که یک فرایند می تواند پشت سر هم وارد ناحیه بحرانی شود **محدود** است و اگر فرایند بعد درخواست ورود داشته باشد ، همان فرایند وارد خواهد شد (پس دچار قحطی نمی شویم) چرا که با هر دفعه ورود فرایند قبلی turn به فرایند بعدی داده می شود ، هرچند برای ورود فرایند بعدی **نیاز به ورود فرایند قبلی هم داریم** که مشکل عدم پیش روی را ایجاد می کرد.

سوال ۴)

۴) سیستم ما داری 4 منبع A، B، C، D و به ترتیب 6، 4، 4، 2 تا از هر کدام از این منابع را دارا است. با استفاده از الگوریتم بانکدار به سوالات زیر پاسخ دهید؟

Available = <6, 4, 4, 2>

	اختصاص داده شده (alloc)				حداکثر نیاز (max)				مورد نیاز (Need)				موجود			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	1	1	3	2	1	1	1	2	0	0	6	4	4	2
P1	1	1	0	0	1	2	0	2	0	1	0	2				
P2	1	0	1	0	3	2	1	0	2	2	0	0				
P3	0	1	0	1	2	1	0	1	2	0	0	0				

الف) در ابتدا حساب کنید کدام یک از پردازه‌ها می‌تواند درخواست منابع کند (الگوریتم بانکدار را انجام دهید)؟

ابتدا Need را حساب می‌کنیم که از طریق Need = max − alloc به دست می‌آید.

حال با توجه به موجودیت R ها بررسی می‌کنیم کدام Pi را می‌توانیم انجام دهیم تا در نهایت بتوانیم Need ها را با Available برابر کنیم. است دهگی می‌توانند درخواست منابع کنند. برای اینکار می‌توانیم Safety را هم حساب کنیم که در قسمت ج حساب کردم.

ب) ایا سیستم ما به بن بست می‌خورد؟ چرا؟

خیر، زیرا اگر از یک پردازه شروع کنیم با توجه به need منابع را اختصاص دهیم برای همه‌ی پردازه های بعدی نیز منابع به اندازه‌ی کافی داریم و سیستم در حالت safe است. با اجرای الگوریتم Safety check نیز می‌توانیم به این نتیجه برسیم.

ج) ایا سیستم ما امن هست؟ چرا؟

بله، همانطور که در قسمت ب اشاره کردم با اجرای الگوریتم safety check مطمئن می‌شویم:

انتخاب P_0 ← ۶ ۴ ۴ ۲ + ۲ ۰ ۱ ۱ = ۸ ۴ ۵ ۳

انتخاب P_1 ← ۸ ۴ ۵ ۳ + ۱ ۱ ۰ ۰ = ۹ ۵ ۵ ۳

انتخاب P_2 ← ۹ ۵ ۵ ۳ + ۱ ۰ ۱ ۰ = ۱۰ ۵ ۶ ۳

انتخاب P_3 ← ۱۰ ۵ ۶ ۳ + ۰ ۱ ۰ ۱ = ۱۰ ۶ ۶ ۴

د) اگر p_3 در این لحظه (2,1,0,0) منبع درخواست کند، آیا می‌توانیم درخواستش را اجابت کنیم؟ خیر، زیرا:

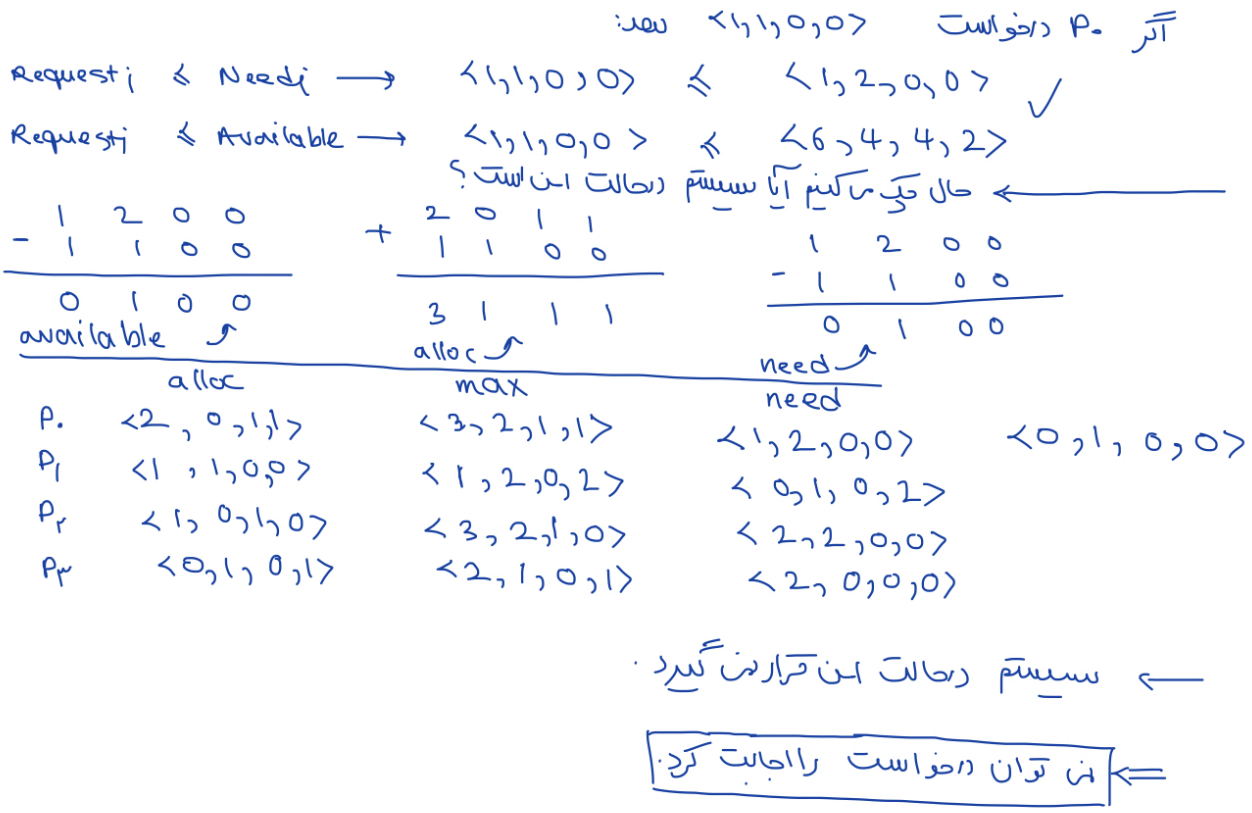
ابتدا بیش شرایط را چک می‌کنیم:

Request$_i$ ≤ Need$_i$ ⟶ <2,1,0,0> ≤ <2,0,0,0> ✗

Request$_i$ ≤ Available ⟶ <2,1,0,0> ≤ <6,4,2,2> ✓

شرط اول برقرار نیست، یعنی مقدار منابعی که در خواست کرده از مقداری که نیاز دارد (دقیقاً مشخص کرده بودن max) بیشتر است، پس نمی‌توانیم درخواستش را اجابت کنیم.

ه) یک مثال دیگر مانند قسمت د بزنید و الگوریتم بانکدار را جلو ببرید.



با تشکر از خانم مینا بیکی

سوال ۵)

الف)

انحصار متقابل:

وجود دارد، فرآیندها برای ورود به ناحیه ی بحرانی نیاز دارند که نوبت به آن ها تعارف شود.
فرض می کنیم فرآیند i زودتر به حلقه while رسیده است. پس از آنکه فرآیند j نوبت را به فرآیند i تعارف کرد، فرآیند i وارد ناحیه ی بحرانی می شود. اما فرآیند i دیگر نمی تواند نوبت را به فرآیند j تعارف کند. پس فرآیند j نمی تواند وارد شود و انحصار متقابل حفظ می شود.

پیشرفت:

به دلیل شرط flag[j]!، اگر فقط یکی از فرآیندها اجرا شود، پیشرفت نخواهیم داشت اما اگرهر دو اجرا شوند و در قسمت remainder دچار قفل نشوند، پیشرفت وجود خواهد داشت.

انتظار محدود:

وجود دارد، زیرا فرآیندی که از ناحیه ی بحرانی خارج شود، ناچار است دو مرتبه نوبت را به فرآیند دیگر تعارف کند. البته این مورد نیز به شرط آن است که قسمت remainder در زمان متناهی به اتمام برسد.

ب)

انحصار متقابل:

وجود دارد، فرآیندی که نوبت را به فرآیند دیگر بدهد خود نخواهد توانست وارد بخش بحرانی شود تا هنگامی که فرآیند دیگر از بخش بحرانی خارج شود.

پیشرفت:

وجود دارد، زیرا حالتی وجود ندارد که هر دو فرآیند قادر به عبور از حلقه while نباشند.

انتظار محدود:

وجود دارد، چون به دلیل تعارف نوبت امکان ندارد که یک فرآیند برای همیشه پشت حلقه while بافی بماند.

راه حل ساده تر:

این الگوریتم همان الگوریتم پترسون است زیرا تنها اندیس های i و j در همه کاربردهای متغیر flag باهم جا به جا شده اند. پس همه شروط برقرار هستند.