به نام خدا

سیستم عامل

بن‌بست

حمید فدیشه‌ای، دانشگاه بجنورد

ترم دوم 94-1393

مقدمه

مقابله با بن‌بست

- رویکردها
  - جلوگیری از بن‌بست
  - اجتناب از بن‌بست
  - کشف بن‌بست
  - بی‌خیال بن‌بست

مقابله با بن‌بست

- رویکردها
  - جلوگیری از بن‌بست — deadlock prevention
  - اجتناب از بن‌بست — deadlock avoidance
  - کشف بن‌بست — deadlock detection
  - بی‌خیال بن‌بست — never mind!

مقابله با بن‌بست

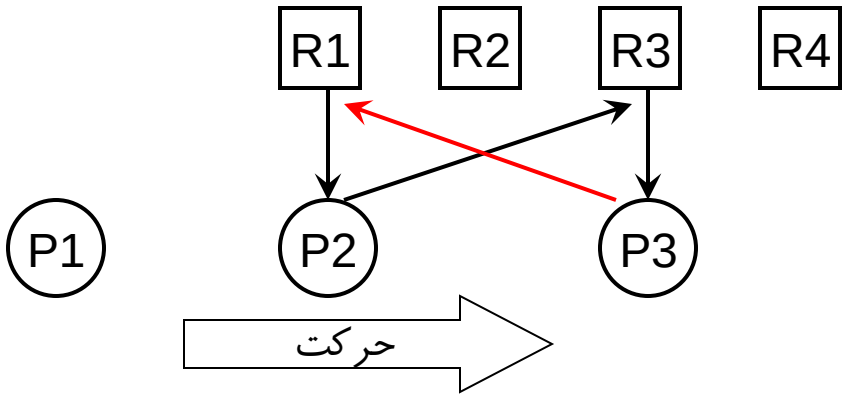
- رویکردها
  - جلوگیری از بن‌بست — deadlock prevention — احتمال بن‌بست را کاملا از بین ببریم
  - اجتناب از بن‌بست — deadlock avoidance — هنگام دادن منبع به فرایند اگر احتمال بن‌بست ایجاد شود نمی‌دهیم
  - کشف بن‌بست — deadlock detection — علاج واقعه بعد از وقوع
  - بی‌خیال بن‌بست — never mind! — رها کردن سیستم به امان خدا

مقابله با بن‌بست

- جلوگیری از بن‌بست
  - در طراحی سیستم کاری کنیم که یکی از شروط لازم بن‌بست امکان‌پذیر نباشد
    - شرط انحصار متقابل
      - گاهی می‌تواند حذف شود. مثلا منبع فایل با عمل خواندن
    - شرط نگه‌داشتن و انتظار
      - می‌توان تمام منابع فرایند را در ابتدا به او داد... اما مشکلاتی دارد
        - مسبب اتلاف منابع
        - مسبب انتظار طولانی شروع
        - عدم آگاهی از منابع مورد نیاز در ابتدای کار
      - یا می‌توان شرط گذاشت که هر فرایند قبل از تقاضای یک منبع جدید منبع قبلی را رها کند
        - کار برنامه نویس سخت می‌شود
    - شرط قبضه نکردن
      - معمولا مجبوریم این شرط را داشته باشیم و حذف آن اثرات مخرب دارد
    - انتظار مدور
      - صفحه بعد

مقابله با بن‌بست

- جلوگیری از بن‌بست
  - انتظار مدور
    - برای امکان‌ناپذیر کردن آن می‌توان مثلا منابع سیستم را به ترتیب خاصی مرتب کرد
    - فرایندها باید طبق همان ترتیب از منابع استفاده کنند و حق برگشت به عقب ندارند
    - مثل سلف سرویس
    - مشکلات این روش
      - منابع را تلف می‌کند
      - مثلا اگر فرایندی به منبعی آزاد نیاز دارد که هنوز به آن نرسیده است
      - برنامه‌نویسی را مشکل می‌کند



- پس جلوگیری از بن‌بست جز در موارد خاص غیر قابل استفاده است

مقابله با بن‌بست

- اجتناب از بن‌بست
  - یک الگوریتم به نام بانکداران (Bankers)
  - وقتی فرایندی تقاضای منبع کند با این الگوریتم چک می‌شود آیا با تخصیص منبع احتمال بن‌بست پیش نمی‌آید؟ اگر محتمل باشد تخصیص انجام نمی‌دهد
- کشف بن‌بست
  - الگوریتم دارد
  - سیستم عامل هر از چندگاهی با اجرای آن چک می‌کند آیا بن‌بستی رخ داده؟
  - اگر رخ داده باشد یکی از فرایندهای گرفتار بن‌بست نابود شود تا بقیه ادامه یابند
- اکثر سیستم عامل‌ها روش چهارم را در پیش می‌گیرند
  - سه روش اول پرهزینه یا غیر ممکن هستند
  - به هر حال فرایندی که زیاد بماند باید اخراج شود
    - چه در اثر حلقه بی‌نهایت باشد چه در اثر خطای برنامه نویسی باشد چه در اثر بن‌بست...

الگوریتم بانکداران

- الگوریتم بانکداران
  - برای اجتناب از بن‌بست
  - وقتی تقاضایی هست و منبعی وجود دارد، تخصیص زمانی انجام می‌شود که مطمئن باشیم احتمال بن‌بست صفر است
  - وگرنه با این که منبع موجود است، تخصیص انجام نمی‌شود
  - حالت‌هایی که احتمال بن‌بست صفر است را حالت امن (Safe State) می‌نامیم

الگوریتم بانکداران

- مثال
  - حالت سیستم در یک لحظه:

claim

	R1	R2
P1	6	4
P2	2	3

alloc.

	R1	R2
P1	4	1
P2	1	0

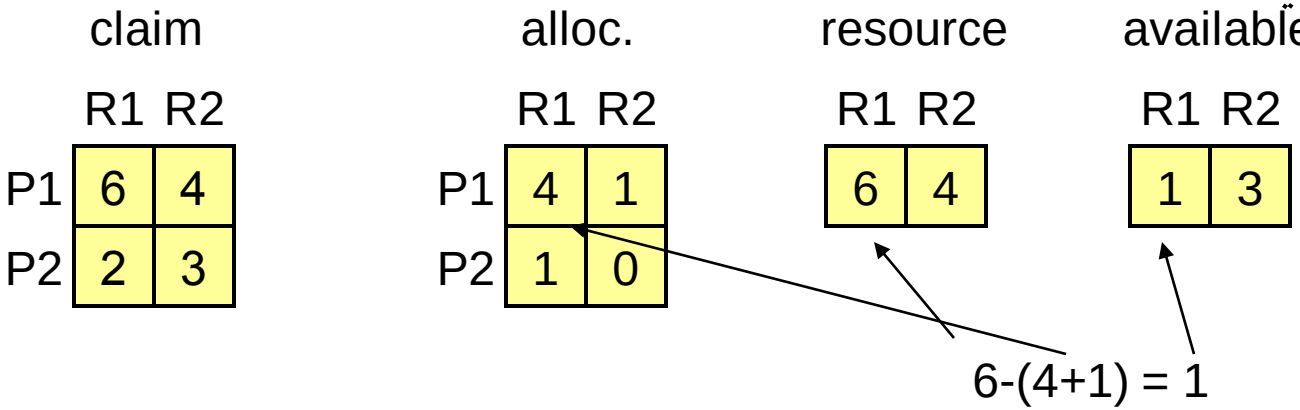
resource

R1	R2
6	4

  - resource[j]=k یعنی k مورد از منبع Rj در سیستم وجود دارد
  - cliam[i][j] حاوی حداکثر نیاز فرایند Pi به منبع Rj است
  - alloc[i][j]=k یعنی تعداد k مورد از منبع Rj به فرایند Pi تخصیص داده شده است

الگوریتم بانکداران

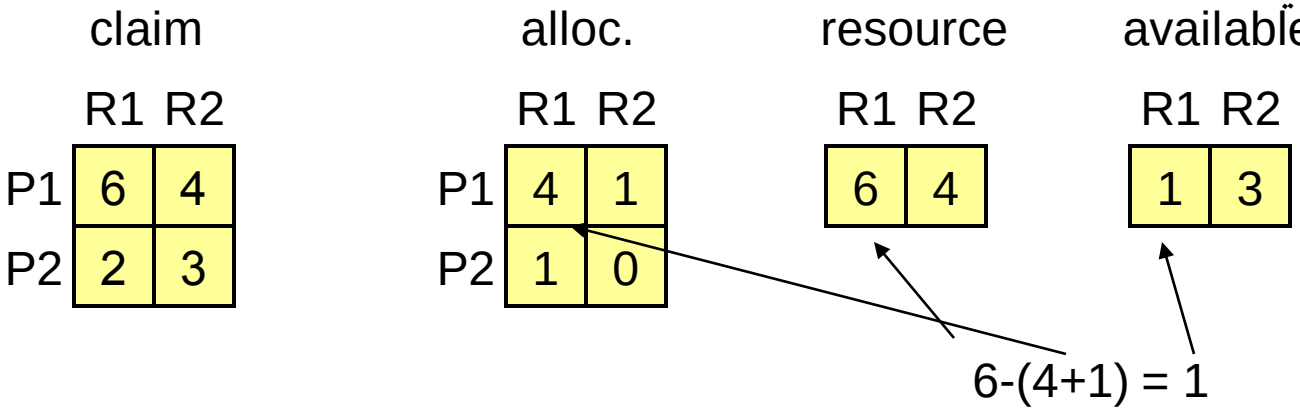
- مثال
  - حالت سیستم در یک لحظه:



  - resource[j]=k یعنی k مورد از منبع Rj در سیستم وجود دارد
  - cliam[i][j] حاوی حداکثر نیاز فرایند Pi به منبع Rj است
  - alloc[i][j]=k یعنی تعداد k مورد از منبع Rj به فرایند Pi تخصیص داده شده است
  - available[j]=k یعنی k مورد از منبع Rj آزاد است

الگوریتم بانکداران

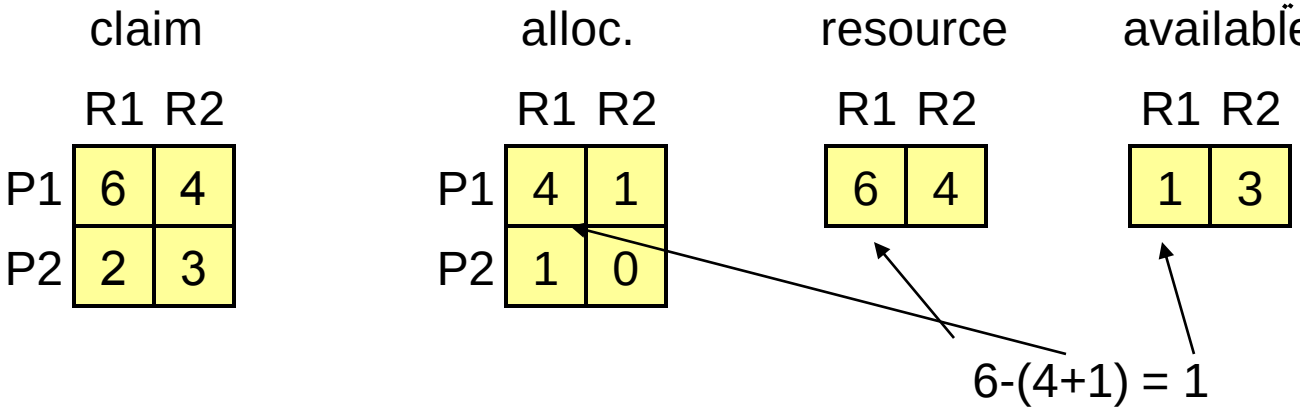
- مثال
  - حالت سیستم در یک لحظه:



  - آیا این حالت امن است؟
    - آیا ترتیبی از فرایندها می‌توان یافت که طبق آن نیاز همه با موفقیت پاسخ داده شود؟

الگوریتم بانکداران

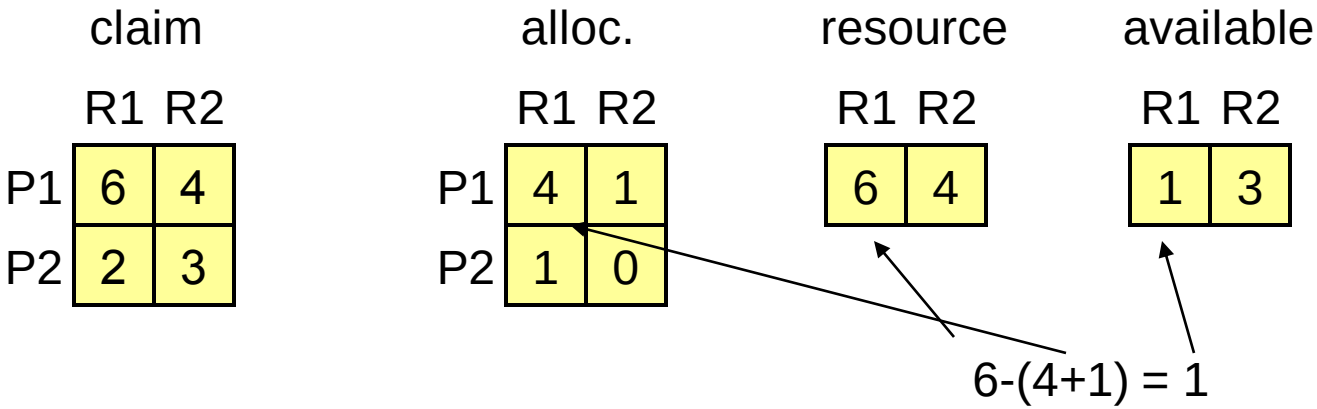
- مثال
  - حالت سیستم در یک لحظه:



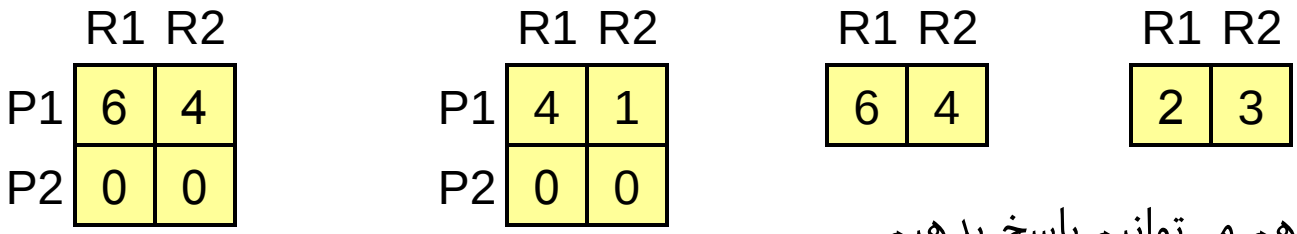
  - نیاز P1 را نمی‌توانیم پاسخ بدهیم
    - چون به 2 واحد دیگر از R1 نیاز دارد (6-4=2)
    - اما فقط 1 واحد از R1 آزاد است

الگوریتم بانکداران

- مثال
  - حالت سیستم در یک لحظه:



  - نیاز P2 را می‌توانیم پاسخ بدهیم و بعد از این کار حالت سیستم جدید می‌شود:



  - سپس نیاز P1 را هم می‌توانیم پاسخ بدهیم
  - پس <P2,P1> یک ترتیب امن است و حالت سیستم در شکل اول یک حالت امن بوده

الگوریتم بانکداران

- الگوریتم بانکداران دو بخش اصلی دارد
  - الگوریتم تشخیص امن بودن حالت سیستم
  - الگوریتم تصمیم برای تخصیص منابع درخواستی فرایند در صورت تشخیص امن بودن
- ایرادهای الگوریتم بانکداران
  - باید در ابتدا بدانیم هر فرایند در کل قرار است چه تعداد از منابع مختلف طلب کند (ماتریس claim)
  - تحمیل ترتیب اجرای خاص به فرایندها که قدرت سیستم عامل در ایجاد زمان‌بندی مناسب را می‌کاهد
  -

الگوریتم بانکداران

- ساختارهای داده مورد استفاده

claim

	R1	R2
P1	6	4
P2	2	3

alloc.

	R1	R2
P1	4	1
P2	1	0

resource

R1	R2
6	4

available

R1	R2
1	3

- resource[j]=k یعنی k مورد از منبع Rj در سیستم وجود دارد
- cliam[i][j] حاوی حداکثر نیاز فرایند Pi به منبع Rj است
- alloc[i][j]=k یعنی تعداد k مورد از منبع Rj به فرایند Pi تخصیص داده شده است
- available[j]=k یعنی k مورد از منبع Rj آزاد است
- need[i][j]=k یعنی فرایند Pi برای اتمام نیاز به k مورد دیگر از منبع Rj دارد

need = claim - alloc

الگوریتم بانکداران

- هر فرایند Pi درخواستش را به صورت ماتریس یک بعدی request می‌دهد.
- request[j]=k یعنی از منبع Rj تعداد k عدد درخواست دارد

تصمیم گیری تخصیص منابع در الگوریتم بانکداران

1- اگر به ازای هر j ، request[j] <= need[i][j] باشد برو به مرحله بعد وگرنه خطا

2- اگر به ازای هر j ، request[j]<=available[j] باشد برو به مرحله بعد وگرنه انتظار

3- حالت بعد از اختصاص منابع را به شکل زیر محاسبه کن:

- availeble = available – request
- alloc[i] = alloc[i] + request
- need[i] = need[i] – request

4- حالت جدید اگر امن باشد تخصیص را انجام بده وگرنه انتظار

الگوریتم بانکداران

- آزمایش امن بودن در الگوریتم بانکداران
  1- work = available
  2- تمام اعضای آرایه finish را به false مقدار اولیه بده
  3- i را به گونه ای پیدا کن که finish[i]=false باشد و need[i][j]<=work[j] به ازای هر j برقرار باشد (فرایندی که می‌تواند ادامه دهد)
     اگر پیدا نشد برو به 5
  4-
     - work = work + alloc[i]
     - finish[i] = true
     - goto 3
  5- اگر برای تمام i ها مقدار finish[i]=true باشد حالت امن داریم

الگوریتم بانکداران

- مثال

alloc

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	2
P4	2	1	1
P5	0	0	2

claim

	R1	R2	R3
P1	7	3	5
P2	3	2	2
P3	9	0	2
P4	2	2	2
P5	4	3	3

available

R1	R2	R3
3	3	2

الگوریتم بانکداران

- مثال
  - محاسبه need

alloc

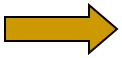
	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	2
P4	2	1	1
P5	0	0	2

claim

	R1	R2	R3
P1	7	3	5
P2	3	2	2
P3	9	0	2
P4	2	2	2
P5	4	3	3

available

R1	R2	R3
3	3	2



need

	R1	R2	R3
P1	7	2	5
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

الگوریتم بانکداران

- مثال
  - درخواست جدید

alloc

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	2
P4	2	1	1
P5	0	0	2

claim

	R1	R2	R3
P1	7	3	5
P2	3	2	2
P3	9	0	2
P4	2	2	2
P5	4	3	3

available

R1	R2	R3
3	3	2

need

	R1	R2	R3
P1	7	2	5
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

- اگر فرایند P2 تقاضای request = [1 0 2] بدهد:

available = available - request

R1	R2	R3
2	3	0

alloc[i] = alloc[i] + request

	R1	R2	R3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

need[i] = need[i]-request

	R1	R2	R3
P1	7	2	5
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

- حال باید امن بودن حالت جدید بررسی شود

الگوریتم بانکداران

- مثال
  - بررسی امن بودن حالت جدید

available

R1	R2	R3
2	3	0

alloc

	R1	R2	R3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

need

	R1	R2	R3
P1	7	2	5
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

work

R1	R2	R3
2	3	0

finish

P1	P2	P3	P4	P5
f	f	f	f	f

الگوریتم بانکداران

- مثال
  - بررسی امن بودن حالت جدید

available

R1	R2	R3
2	3	0

alloc

	R1	R2	R3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

need

	R1	R2	R3
P1	7	2	5
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

i	work			finish					
	R1	R2	R3	P1	P2	P3	P4	P5	
	2	3	0	f	f	f	f	f	
2	5	3	2	f	t	f	f	f	

الگوریتم بانکداران

- مثال
  - بررسی امن بودن حالت جدید

available

R1	R2	R3
2	3	0

alloc

	R1	R2	R3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

need

	R1	R2	R3
P1	7	2	5
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

i	work			finish					
	R1	R2	R3	P1	P2	P3	P4	P5
	2	3	0	f	f	f	f	f
2	5	3	2	f	t	f	f	f
4	7	4	3	f	t	f	t	f

الگوریتم بانکداران

- مثال
  - بررسی امن بودن حالت جدید

available

R1	R2	R3
2	3	0

alloc

	R1	R2	R3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

need

	R1	R2	R3
P1	7	2	5
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

i	work R1	work R2	work R3	finish P1	finish P2	finish P3	finish P4	finish P5
	2	3	0	f	f	f	f	f
2	5	3	2	f	t	f	f	f
4	7	4	3	f	t	f	t	f
3	10	4	5	f	t	t	t	f

الگوریتم بانکداران

- مثال
  - بررسی امن بودن حالت جدید

available

R1	R2	R3
2	3	0

alloc

	R1	R2	R3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

need

	R1	R2	R3
P1	7	2	5
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

i	work R1	work R2	work R3	finish P1	finish P2	finish P3	finish P4	finish P5
	2	3	0	f	f	f	f	f
2	5	3	2	f	t	f	f	f
4	7	4	3	f	t	f	t	f
3	10	4	5	f	t	t	t	f
1	10	5	5	t	t	t	t	f

الگوریتم بانکداران

- مثال
  - بررسی امن بودن حالت جدید

available

R1	R2	R3
2	3	0

alloc

	R1	R2	R3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

need

	R1	R2	R3
P1	7	2	5
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

i	work R1	work R2	work R3	finish P1	finish P2	finish P3	finish P4	finish P5
	2	3	0	f	f	f	f	f
2	5	3	2	f	t	f	f	f
4	7	4	3	f	t	f	t	f
3	10	4	5	f	t	t	t	f
1	10	5	5	t	t	t	t	f
5	10	5	7	t	t	t	t	t

الگوریتم بانکداران

- مثال
  - بررسی امن بودن حالت جدید

available

R1	R2	R3
2	3	0

alloc

	R1	R2	R3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

need

	R1	R2	R3
P1	7	2	5
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

i	work R1	work R2	work R3	finish P1	finish P2	finish P3	finish P4	finish P5
	2	3	0	f	f	f	f	f
2	5	3	2	f	t	f	f	f
4	7	4	3	f	t	f	t	f
3	10	4	5	f	t	t	t	f
1	10	5	5	t	t	t	t	f
5	10	5	7	t	t	t	t	t

→ حالت امن

ترتیب امن

الگوریتم بانکداران

- در مثال قبل
  - اگر P4 تقاضای [3 3 0] می داد به خطا می‌انجامید
  - اگر P1 تقاضای [0 2 0] می‌داد به حالت ناامن می‌انجامید

الگوریتم کشف بن‌بست

- الگوریتم کشف بن‌بست
  - یک مشکل الگوریتم بانکداران، نیاز به از پیش دانستن حداکثر نیاز فرایند (claim) بود
  - الگوریتم کشف بن‌بست نیاز به این ماتریس ندارد
    - ولی بن‌بست را بعد از وقوع کشف می‌کند
  - بعد از بن‌بست چه باید کرد؟
    - در اصل باید فرایند واقع در بن‌بست به عقب بازگردد
    - معمولا غیر ممکن
    - به همین خاطر معمولا فرایند واقع در بن‌بست محکوم به نابود شدن می‌شود
  - ساختارهای داده الگوریتم کشف بن‌بست
    - شبیه آن چه در بانکداران بود
    - ماتریس request این جا درخواست همه فرایندها را در خود دارد (دوبعدی)

الگوریتم کشف بن‌بست

- الگوریتم کشف بن‌بست
  - 1- مقداردهی اولیه
    - work = available
    - for each process Pi
      - if alloc[i] <> 0 then finished[i] = false
      - else finished[i] = true
  - 2- مقدار i را به گونه‌ای پیدا کن که finished[i] = false و request[i]<=work باشد. اگر پیدا نشد پایان (فرایندی که می‌تواند ادامه دهد)
  - 3- اعمال زیر را انجام بده و به 2 برگرد
    - work = work + alloc[i]
    - finished[i] = true
    - goto 2
- در انتهای این الگوریتم فرایندهایی که finished[i] برای آن‌ها false است در بن‌بست قرار دارند (الگوریتم هم وقوع بن‌بست و هم فرایندهای واقع در آن را نشان می‌دهد)

الگوریتم کشف بن‌بست

- مثال
  - آیا در وضعیت مقابل فرایندها در بن‌بست هستند؟

alloc

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

request

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

resource

R1	R2	R3
7	2	6

الگوریتم کشف بن‌بست

- مثال

	alloc			request			resource			available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	0	0	0	7	2	6	0	0	0
P2	2	0	0	2	0	2						
P3	3	0	3	0	0	0						
P4	2	1	1	1	0	0						
P5	0	0	2	0	0	2						

- ابتدا محاسبه آرایه available
  - با کم کردن جمع ستون‌های alloc از resource

الگوریتم کشف بن‌بست

- مثال

	alloc			request			resource			available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	0	0	0	7	2	6	0	0	0
P2	2	0	0	2	0	2						
P3	3	0	3	0	0	0						
P4	2	1	1	1	0	0						
P5	0	0	2	0	0	2						

i

work

R1	R2	R3
0	0	0

finish

P1	P2	P3	P4	P5
f	f	f	f	f

الگوریتم کشف بن‌بست

- مثال

	alloc			request			resource			available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	0	0	0	7	2	6	0	0	0
P2	2	0	0	2	0	2						
P3	3	0	3	0	0	0						
P4	2	1	1	1	0	0						
P5	0	0	2	0	0	2						

i	work			finish					
	R1	R2	R3	P1	P2	P3	P4	P5	
	0	0	0	f	f	f	f	f	
1	0	1	0	t	f	f	f	f	

الگوریتم کشف بن‌بست

- مثال

	alloc			request			resource			available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	0	0	0	7	2	6	0	0	0
P2	2	0	0	2	0	2						
P3	3	0	3	0	0	0						
P4	2	1	1	1	0	0						
P5	0	0	2	0	0	2						

i	work			finish					
	R1	R2	R3	P1	P2	P3	P4	P5	
	0	0	0	f	f	f	f	f	
1	0	1	0	t	f	f	f	f	
3	3	1	3	t	f	t	f	f	

الگوریتم کشف بن‌بست

- مثال

	alloc			request			resource			available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	0	0	0	7	2	6	0	0	0
P2	2	0	0	2	0	2						
P3	3	0	3	0	0	0						
P4	2	1	1	1	0	0						
P5	0	0	2	0	0	2						

i	work			finish					
	R1	R2	R3	P1	P2	P3	P4	P5	
	0	0	0	f	f	f	f	f	
1	0	1	0	t	f	f	f	f	
3	3	1	3	t	f	t	f	f	
2	5	1	3	t	t	t	f	f	

الگوریتم کشف بن‌بست

- مثال

alloc

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

request

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

resource

R1	R2	R3
7	2	6

available

R1	R2	R3
0	0	0

i	work R1	work R2	work R3	finish P1	finish P2	finish P3	finish P4	finish P5
	0	0	0	f	f	f	f	f
1	0	1	0	t	f	f	f	f
3	3	1	3	t	f	t	f	f
2	5	1	3	t	t	t	f	f
4	7	2	4	t	t	t	t	f

الگوریتم کشف بن‌بست

- مثال

	alloc			request			resource			available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	0	0	0	7	2	6	0	0	0
P2	2	0	0	2	0	2						
P3	3	0	3	0	0	0						
P4	2	1	1	1	0	0						
P5	0	0	2	0	0	2						

i	work			finish					
	R1	R2	R3	P1	P2	P3	P4	P5	
	0	0	0	f	f	f	f	f	
1	0	1	0	t	f	f	f	f	
3	3	1	3	t	f	t	f	f	
2	5	1	3	t	t	t	f	f	
4	7	2	4	t	t	t	t	f	
5	7	2	6	t	t	t	t	t	بن‌بست نیست

الگوریتم کشف بن‌بست

alloc

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

request

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	2
P4	1	0	0
P5	0	0	2

resource

R1	R2	R3
7	2	6

available

R1	R2	R3
0	0	0

- اگر request[P3][R3] به جای 0 مقدار 2 می‌داشت بن‌بست وجود داشت

i	work			finish				
	R1	R2	R3	P1	P2	P3	P4	P5
	0	0	0	f	f	f	f	f
1	0	1	0	t	f	f	f	f

الگوریتم کشف بن‌بست

- سؤال
  - الگوریتم کشف بن‌بست چه مواقعی اجرا شود؟
    - اگر با هر تقاضا انجام شود به محض وقوع بن‌بست کشف می‌شود
      - اما این کار سربار زیادی دارد
    - می‌توان به طور تناوبی این کار را انجام داد
      - بسته به بار کاری سیستم می‌توان دوره تناوب را کم یا زیاد کرد
- چرا در بانکداران حداکثر نیاز فرایند را لازم بود بدانیم اما در کشف لازم نیست؟
  - بانکداران در مورد احتمال وقوع بن‌بست در آینده صحبت می‌کرد. پس نیاز داشت بداند فرایند در آینده چه درخواست‌هایی خواهد داشت
  - الگوریتم کشف در مورد این که همین حالا آیا سیستم در بن‌بست هست یا خیر صحبت می‌کند