# یادداشت‌های درس سیستم‌های عامل - بخش نهم

در این بخش از درس مطالعه‌ی روش‌های مقابله با بن‌بست را ادامه می‌دهیم.

- برای حالتی که از برخی از منابع چند نمونه موجود باشد نمی‌توان از گراف تخصیص منابع برای اجتناب از بن‌بست استفاده کرد.
- الگوریتم بانکدار (Banker's Algorithm) برای این حالت مناسب است.
- این الگوریتم از دو قسمت تشکیل شده است:

  الف) الگوریتم تشخیص امن بودن سیستم (Safety)

  ب) الگوریتم تخصیص منابع

## نمایش وضعیت

از چند بردار و ماتریس در الگوریتم بانکدار استفاده می‌شود (فرض کنید n تعداد پردازه‌ها و m تعداد منابع باشد):

**`Available(m)`**: تعداد نمونه‌های آزاد از هر منبع.

**`Allocation(n, m)`**: تعداد نمونه‌های تخصیص یافته به هر پردازه از هر منبع.

**`Maximum(n, m)`**: حداکثر نمونه‌هایی که هر پردازه می‌تواند از هر منبع درخواست دهد.

**`Need(n, m)`**: تعداد نمونه‌های جدیدی که می‌تواند هر پردازه از هر منبع درخواست دهد.

برای مثال:

- **`Available(2)`** نشان می‌دهد چند نمونه از منبع **R2** آزاد است.
- مقدار **`Allocation(1, 3)`** نشان می‌دهد چند نمونه از منبع **R3** به پردازه‌ی **P1** تخصیص یافته است.
- مقدار **`Maximum(1, 3)`** نشان می‌دهد حداکثر چند نمونه از منبع **R3** می‌تواند به پردازه‌ی **P1** تخصیص یابد.
- مقدار **`Need(1, 3)`** نشان می‌دهد چند نمونه‌ی دیگر از منبع **R3** می‌تواند به پردازه‌ی **P1** تخصیص یابد.

دقت کنید که همواره رابطه‌ی زیر برقرار است:

```
Maximum(i, j) = Allocation(i, j) + Need(i, j)
```

# عملیات برداری

برای ساده‌تر شدن بیان الگوریتم بانکدار از عملیات برداری استفاده می‌کنیم.

- دو بردار **A** و **B** را با اندازه‌ی پنج در نظر بگیرید.

| | | | | | |
|---|---|---|---|---|---|
| **A** | 4 | 2 | 3 | 4 | 1 |

| | | | | | |
|---|---|---|---|---|---|
| **B** | 8 | 3 | 5 | 4 | 3 |

- حاصل عملیاتی مثل جمع روی دو بردار، برداری است که هر عنصرش از انجام عمل مورد نظر روی عناصر متناظرش در آرایه‌ی ورودی حاصل می‌شود. برای مثال، اگر **C** برابر **A + B** باشد، خواهیم داشت.

| | | | | | |
|---|---|---|---|---|---|
| **C** | 12 | 5 | 8 | 8 | 4 |

- عملیات مقایسه هم به صورت برداری تعریف می‌کنیم. برای نمونه **A ≤ B** هست اگر و تنها اگر هر عنصر **A** کوچک‌تر یا مساوی عنصر متناظرش در **B** باشد.
- برای آرایه‌های بالا، **A ≤ B** برقرار است ولی **A < B** برقرار نیست.
- همچنین برای یک ماتریس مثل **Allocation**، مقدار **Allocation(i)** برابر برداری است که از سطر **i** این ماتریس حاصل می‌شود.

## مثال

- فرض کنید سه پردازه و دو منبع در سیستمی موجود باشند.
- جدول زیر به صورت خلاصه ماتریس‌های **Available**، **Maximum** و **Need** را نشان می‌دهد.

| Process | R1 (current/maximum) | R2 (current/maximum) |
|---|---|---|
| P1 | 2/3 | 1/1 |
| P2 | 1/3 | 1/2 |
| P3 | 1/1 | 0/0 |

- در هر خانه از این ماتریس دو عدد قرار دارند: عدد بالای علامت ممیز تعداد نمونه‌های تخصیص یافته به پردازه و عدد زیر علامت ممیز حداکثر تعداد نمونه‌هایی است که پردازه می‌تواند از یک منبع در اختیار بگیرد.
- فرض کنید مقدار همه‌ی عناصر آرایه‌ی **Available** برابر صفر باشد:

| Resource | Available |
|---|---|
| R1 | 0 |
| R2 | 0 |

- در این صورت تعداد کل نمونه‌های منبع **R1** برابر چهار و تعداد کل نمونه‌های منبع **R2** برابر دو خواهد بود.

- در الگوریتم تشخیص امن بودن، بررسی می‌شود که آیا یک دنباله‌ی امن وجود دارد یا خیر.
- در این الگوریتم در حلقه‌ای پردازه‌ای که می‌تواند همه‌ی منابع مورد نیازش را درخواست دهد انتخاب می‌شود و فرض می‌شود خاتمه یافته است.
- اگر همه‌ی پردازه‌ها به این شکل خاتمه یابند، وضعیت سیستم امن است.
- جزئیات این الگوریتم را در صفحه‌ی بعد می‌بینید.

## الگوریتم وضعیت امن

**الگوریتم تشخیص وضعیت امن از الگوریتم بانکدار**

ورودی: بردارهای وضعیت سیستم.

خروجی: آیا وضعیت سیستم امن هست یا خیر.

از آرایه‌ی **Left (m)** در الگوریتم برای نگهداری تعداد منابع آزاد از هر منبع استفاده می‌کنیم.

از آرایه‌ی **Finished (n)** در الگوریتم برای نگهداری پردازه‌های خاتمه یافته استفاده می‌کنیم.

۱ مقدار آرایه‌ی **Left** را برابر **Available** قرار بده.

۲ مقدار همه‌ی عناصر **Finished** را برابر **False** قرار بده.

۳ تکرار کن

۴ اندیسی از **i** را بیاب که دو ویژگی زیر را داشته باشد

الف) **Finished (i)** برابر **False** باشد.

ب) رابطه‌ی **Need(i) ≤ Left** برقرار باشد.

۵ اگر این اندیس موجود نبود از حلقه خارج شو.

۶ **Left** را برابر **Left + Allocation(i)** قرار بده.

۷ **Finished(i)** را برابر **True** قرار بده.

۸ اگر مقدار **Finished(i)** به ازای همه‌ی پردازه‌ها **True** بود

۹ وضعیت امن است.

۱۰ در غیر این صورت

۱۱ وضعیت ناامن است.

## الگوریتم وضعیت امن - مثال یک

وضعیت زیر را برای سه پردازه و دو منبع در نظر بگیرید و فرض کنید از هر منبع صفر نمونه آزاد باشند (**Available = <0, 0>**).

| Process | R1 (current/maximum) | R2 (current/maximum) |
|---|---|---|
| P1 | 2/3 | 1/1 |
| P2 | 1/3 | 1/2 |
| P3 | 1/1 | 0/0 |

الگوریتم بررسی وضعیت امن را اجرا می‌کنیم.

- خط اول و دوم: **Left: <0 0>** و **Finished: <False, False, False>**.
- حلقه‌ی خط سوم تا هفتم: برای **i = 3** دو شرط الف و ب برقرار است.
- تغییرات خط ششم و هفتم: **Left: <1 0>** و **Finished: <False, False, True>**.
- حلقه‌ی خط سوم تا هفتم: برای **i = 1** دو شرط الف و ب برقرار است.
- تغییرات خط ششم و هفتم: **Left: <3 1>** و **Finished: <True, False, True>**.
- حلقه‌ی خط سوم تا هفتم: برای **i = 2** دو شرط الف و ب برقرار است.
- تغییرات خط ششم و هفتم: **Left: <4 2>** و **Finished: <True, True, True>**.
- شرط خط هشتم برقرار است و **Finished** برای همه‌ی پردازه‌ها **True** است.
- وضعیت امن است (دنباله‌ی امن برابر **<P3, P1, P2>**).

## الگوریتم وضعیت امن - مثال دو

وضعیت زیر را برای سه پردازه و دو منبع در نظر بگیرید و فرض کنید از هر منبع صفر نمونه آزاد باشند (**Available = <1, 1>**).

| Process | R1 (current/maximum) | R2 (current/maximum) |
|---|---|---|
| P1 | 2/4 | 1/1 |
| P2 | 1/3 | 1/2 |
| P3 | 0/1 | 0/1 |

الگوریتم بررسی وضعیت امن را اجرا می‌کنیم.

- خط اول و دوم: **Left: <1 1>** و **Finished: <False, False, False>**.
- حلقه‌ی خط سوم تا هفتم: برای **i = 3** دو شرط الف و ب برقرار است.
- تغییرات خط ششم و هفتم: **Left: <1 1>** و **Finished: <False, False, True>**.
- حلقه‌ی خط سوم تا هفتم: دو شرط برای هیچ اندیس **i** برقرار نیست؛ خروج از حلقه.
- شرط خط هشتم برقرار نیست و **Finished** برای دو پردازه **False** است.
- وضعیت ناامن است.

**الگوریتم تخصیص منابع از الگوریتم بانکدار**

ورودی: بردارهای وضعیت سیستم.

ورودی: بردار **Request** که درخواست پردازه‌ی **i**-ام را نشان می‌دهد.

خروجی: تخصیص منابع درخواست شده یا متوقف کردن پردازه.

۱ اگر **Request ≤ Need(i)** نباشد

۲ خطا گزارش بده؛ خروج.

۳ اگر **Request ≤ Available** نباشد

۴ پردازه باید منتظر شود؛ خروج.

۵ مقدار **Available** را قرار بده **Available – Request**.

۶ مقدار **Allocation(i)** را قرار بده **Allocation(i) + Request**.

۷ مقدار **Need(i)** را قرار بده **Need(i) – Request**.

۸ اگر وضعیت سیستم امن باشد (با کمک الگوریتم تشخیص وضعیت امن)

۹ منابع درخواست شده را تخصیص بده؛ خروج.

۱۰ در غیر این صورت

۱۱ تغییرات متغیرها در خط‌های پنجم تا هفتم را برگردان.

۱۲ پردازه باید منتظر شود؛ خروج.

## الگوریتم تخصیص منابع - مثال

وضعیت زیر را برای سه پردازه و دو منبع در نظر بگیرید و فرض کنید: **Available = <1, 2>**

| Process | R1 (current/maximum) | R2 (current/maximum) |
|---|---|---|
| P1 | 2/3 | 1/1 |
| P2 | 1/3 | 1/2 |
| P3 | 1/1 | 0/1 |

فرض کنید پردازه‌ی **P2** درخواست یک نمونه از منبع **R1** و یک نمونه از منبع **R2** داشته باشد:

**Request = <1, 1>**

الگوریتم تخصیص منابع الگوریتم بانکدار را اجرا می‌کنیم.

- شرط خط اول و سوم برقرار نیست.
- خط سوم: **Available: <0, 1>**
- خط چهارم: **Allocation(2) = <2, 2>**
- خط پنجم: **Need(2) = <1, 0>**
- خط هشتم: اجرای الگوریتم بررسی وضعیت امن بانکدار: وضعیت امن است.
- خط نهم: تخصیص انجام می‌شود.

روش دیگر مقابله با بن‌بست، تشخیص بن‌بست و سپس حل آن است.

- این الگوریتم دو بخش دارد:

  الف) تشخیص بن‌بست

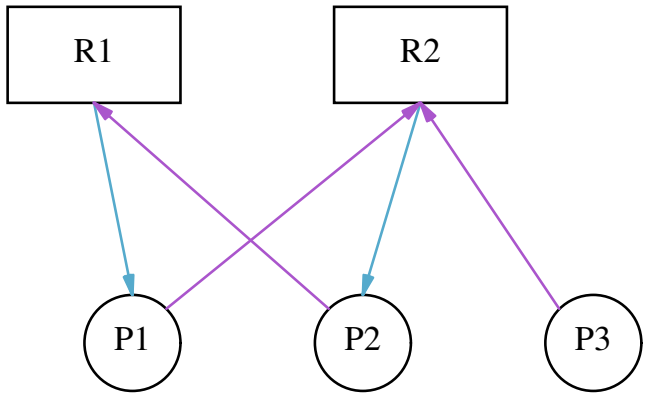
  ب) رهایی از بن‌بست

- هر یک از دو بخش را بررسی می‌کنیم.

- تشخیص بن‌بست را در دو حالت بررسی می‌کنیم:
- اگر از هر منبع فقط یک نمونه موجود باشد از گراف تخصیص منابع یا از گراف انتظار پردازه‌ها برای تشخیص بن‌بست استفاده می‌کنیم.
- اگر از برخی از منابع بیش از یک نمونه موجود باشد، از الگوریتمی مشابه الگوریتم بانکدار استفاده می‌نماییم.

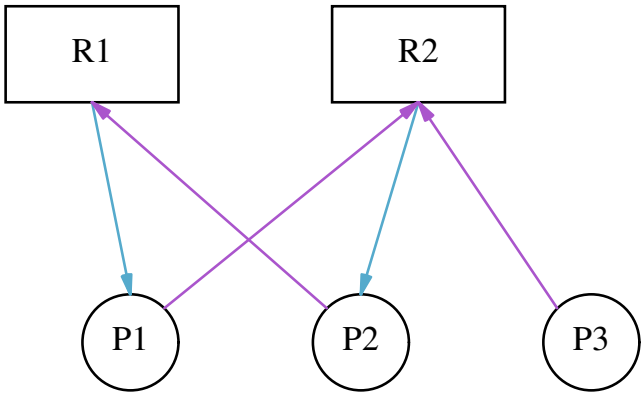
## تشخیص بن‌بست - گراف تخصیص منابع

- اگر گراف تخصیص منابع دور داشته باشد یعنی بن‌بست رخ داده است.
- این روش تنها وقتی درست کار می‌کند که از هر منبع فقط یک نمونه موجود باشد.
- گراف تخصیص منابع زیر برای دو منبع و سه پردازه است.

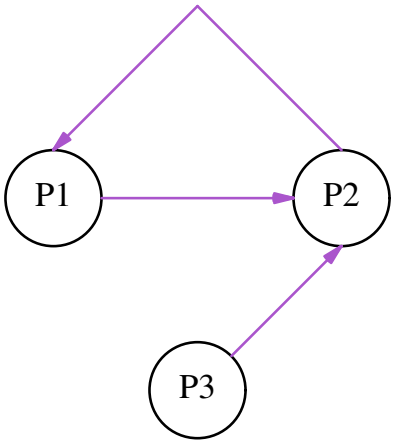



- چون یک دور در این گراف وجود دارد (دور <P1 R2 P2 R1>)، بن‌بست رخ داده است.

- از گراف انتظار پردازه‌ها (Wait-for Graph) هم می‌توان برای تشخیص بن‌بست استفاده کرد؛ اگر این گراف دور داشته باشد، بن‌بست رخ داده است.
- این روش هم تنها وقتی درست کار می‌کند که از هر منبع فقط یک نمونه موجود باشد.
- در گراف انتظار پردازه‌ها، به هر پردازه یک رأس تخصیص داده می‌شود.
- یک یال از پردازه‌ی A به پردازه‌ی B نشان می‌دهد که منبعی در اختیار پردازه‌ی B قرار دارد که پردازه‌ی A درخواست استفاده از آن را دارد.
- گراف تخصیص منابع صفحه‌ی قبل را در نظر بگیرید.




- گراف انتظار پردازه‌های متناظر با آن را در ادامه می‌بینید.




- چون یک دور در این گراف وجود دارد، بن‌بست رخ داده است.

این الگوریتم به الگوریتم بانکدار شباهت دارد اما در این الگوریتم حداکثر تعداد نمونه‌هایی که هر پردازه از منابع درخواست می‌کند مشخص نیست.

از چند بردار و ماتریس در این الگوریتم استفاده می‌شود (فرض کنید n تعداد پردازه‌ها و m تعداد منابع باشد):

**`Available(m)`**: تعداد نمونه‌های آزاد از هر منبع.

**`Allocation(n, m)`**: تعداد نمونه‌های تخصیص یافته به هر پردازه از هر منبع.

**`Request(n, m)`**: تعداد نمونه‌هایی درخواست شده توسط هر پردازه از هر منبع (که تخصیص داده نشده‌اند).

برای مثال:

- **`Available(2)`** نشان می‌دهد چند نمونه از منبع **R2** آزاد است.
- مقدار **`Allocation(1, 3)`** نشان می‌دهد چند نمونه از منبع **R3** به پردازه‌ی **P1** تخصیص یافته است.
- مقدار **`Request(1, 3)`** نشان می‌دهد پردازه‌ی **P1** چند نمونه از منبع **R3** را درخواست کرده است که هنوز در اختیارش قرار نگرفته است.

جزئیات الگوریتم تشخیص بن‌بست در ادامه نمایش داده می‌شود.

**الگوریتم تشخیص بن‌بست**

ورودی: بردارهای وضعیت سیستم.

خروجی: آیا بن‌بست رخ داده است یا خیر.

از آرایه‌ی **Left (m)** در الگوریتم برای نگهداری تعداد منابع آزاد از هر منبع استفاده می‌کنیم.

از آرایه‌ی **Finished (n)** در الگوریتم برای نگهداری پردازه‌های خاتمه یافته استفاده می‌کنیم.

۱ مقدار آرایه‌ی **Left** را برابر **Available** قرار بده.
۲ مقدار همه‌ی عناصر **Finished** را برابر **False** قرار بده.
۳ تکرار کن
۴ اندیسی از **i** را بیاب که دو ویژگی زیر را داشته باشد
الف) **Finished (i)** برابر **False** باشد.
ب) رابطه‌ی **Request (i) ≤ Left** برقرار باشد.
۵ اگر این اندیس موجود نبود از حلقه خارج شو.
۶ **Left** را برابر **Left + Allocation (i)** قرار بده.
۷ **Finished (i)** را برابر **True** قرار بده.
۸ اگر مقدار **Finished** به ازای همه‌ی پردازه‌ها **True** بود
۹ بن‌بست رخ نداده است.
۱۰ در غیر این صورت
۱۱ بن‌بست رخ داده است.

- ایده‌ی کلی الگوریتم این است که آیا ترتیبی از پردازه‌ها وجود دارد که در آن ترتیب پردازه‌ها همه‌ی منابع درخواست شده‌شان را دریافت کنند و خاتمه یابند یا خیر.
- در حلقه‌ی خط سوم تا هفتم، یک پردازه که منابع درخواست شده‌اش از منابع آزاد بیشتر نیست انتخاب می‌شود.

## الگوریتم مشابه بانکدار - مثال

وضعیت زیر را برای سه پردازه و دو منبع در نظر بگیرید و فرض کنید از از منبع **R1** یک نمونه و از منبع **R2** صفر نمونه آزاد باشد.

(**Available = <1, 0>**).

| Process | R1 (current/request) | R2 (current/request) |
|---|---|---|
| P1 | 1/2 | 1/0 |
| P2 | 1/3 | 1/1 |
| P3 | 1/1 | 0/0 |

الگوریتم تشخیص بن‌بست را اجرا می‌کنیم.

- خط اول و دوم: **Left: <1 0>** و **Finished: <False, False, False>**.
- حلقه‌ی خط سوم تا هفتم: برای **i = 3** دو شرط الف و ب برقرار است.
- تغییرات خط ششم و هفتم: **Left: <2 0>** و **Finished: <False, False, True>**.
- حلقه‌ی خط سوم تا هفتم: برای **i = 1** دو شرط الف و ب برقرار است.
- تغییرات خط ششم و هفتم: **Left: <3 1>** و **Finished: <True, False, True>**.
- حلقه‌ی خط سوم تا هفتم: برای **i = 2** دو شرط الف و ب برقرار است.
- تغییرات خط ششم و هفتم: **Left: <4 2>** و **Finished: <True, True, True>**.
- شرط خط هشتم برقرار است و **Finished** برای همه‌ی پردازه‌ها **True** است.
- همه‌ی پردازه‌ها خاتمه یافته‌اند و در نتیجه بن‌بست وجود ندارد.

## رهایی از بن‌بست

دو روش برای رهایی از بن‌بست وجود دارد:

الف) خاتمه‌ی پردازه‌ها

ب) به زور گرفتن منابع

- این دو روش را بررسی می‌کنیم.

## رهایی - خاتمه‌ی پردازه‌ها

دو راه کلی برای خاتمه‌ی پردازه‌ها وجود دارد:

الف) خاتمه‌ی همه‌ی پردازه‌های درگیر در بن‌بست

ب) خاتمه‌ی پردازه‌ها یکی پس از دیگری تا بن‌بست رفع شود

در حالت ب، برای انتخاب یک پردازه، متغیرهایی مثل موارد زیر در نظر گرفته می‌شوند:

اولویت پردازه

زمان پردازش پردازه

تعداد و انواع منابع در اختیار پردازه

نوع پردازه (محاوره‌ای یا دسته‌ای)

در این روش گام‌های زیر انجام می‌شود:

الف) یک پردازه‌ی قربانی انتخاب می‌شود

ب) منابع این پردازه به زور گرفته می‌شوند و به سایر پردازه‌ها داده می‌شوند

ج) پردازه‌ی قربانی ادامه می‌یابد (گام Rollback)

- در گام ج، لازم است پردازه‌ی قربانی ادامه پیدا کند. اگر وضعیت پردازه و منابعی که در اختیارش بودند در زمانی از اجرایش ذخیره شده باشد، پردازه به آن وضعیت انتقال می‌یابد و اجرای آن ادامه می‌یابد. در غیر این صورت، پردازه دوباره شروع می‌شود.
- یکی از مشکلاتی که در این روش ممکن است رخ دهد گرسنگی است: ممکن است پردازه‌ای هر بار پس از بروز بن‌بست به عنوان قربانی انتخاب شود.

در این روش مشکل بن‌بست را نادیده می‌گیریم؛ بن‌بست رخ می‌دهد ولی برای آن کاری نمی‌کنیم.

- بسیاری از سیستم‌های عامل امروزی از جمله لینوکس و ویندوز این روش را انجام می‌دهند.
- به این الگوریتم گاهی الگوریتم شترمرغ (Ostrich) هم می‌گویند: گفته می‌شود وقتی شترمرغ احساس خطر کند سرش را در ماسه فرو می‌کند.
- توجیه این روش:

  بن‌بست به ندرت رخ می‌دهد و هزینه و پیچیدگی روش‌های مقابله با بن‌بست زیاد است.

  اگر بن‌بست رخ داد، می‌توان سیستم عامل را Restart کرد که بسیار کم‌هزینه است.
- اما دقت کنید که در این سیستم‌های عامل کتابخانه‌هایی موجود هستند که در فضای کاربری با بن‌بست مقابله می‌کنند.

خلاصه‌ای از مطالب ارائه شده در این بخش از درس:

چهار شرط لازم برای بروز بن‌بست

۱ انحصار متقابل

۲ داشتن و انتظار

۳ نبودن Preemption

۴ انتظار چرخشی

روش‌های مقابله با بن‌بست

۱ پیشگیری از بن‌بست

نقض یکی از چهار شرط لازم برای بن‌بست

۲ اجتناب از بن‌بست

گراف تخصیص منابع + یال Claim

الگوریتم بانکدار

۳ تشخیص و رهایی از بن‌بست

تشخیص

گراف تخصیص منابع یا انتظار پردازه‌ها

الگوریتم مشابه بانکدار

رهایی

خاتمه‌ی پردازه‌ها

به زور گرفتن منابع

۴ نادیده گرفتن بن‌بست