دانشگاه علم و صنعت - دانشکده ریاضی و علوم کامپیوتر

سیستم‌های عامل

(فصل پنجم: بن بست)

استاد: حمید حاج سیدجوادی

ایمیل: h.s.javadi.hamid@gmail.com

یکشنبه و سه‌شنبه ساعت ۱۰:۳۰ تا ۱۲:۰۰

۱۴۰۲ - ۱۴۰۳

بخش اول **مقدمه**

بخش دوم **منبع**

بخش سوم **بن بست**

بخش چهارم **شرایط وقوع بن بست**

بخش پنجم **گراف تخصیص منابع**

بخش ششم **روش‌های اداره کردن بن بست**

بخش هفتم **جمع‌بندی و پایان**

مقدمه

سیستم‌های کامپیوتری دارای منابع متعدد و مختلفی هستند. برخی از این منابع قابلیت استفاده همزمان توسط چند کاربر را دارند ولی برخی دیگر در هر لحظه فقط در اختیار یک پردازه هستند. در سیستم‌های چند برنامگی، ممکن است چندین پردازه برای دسترسی به منابع محدود سیستم با هم رقابت داشته باشند. بنابراین، سیستم عامل باید منابع و دسترسی به آنها را طوری مدیریت کند که مشکلی در دسترسی‌های همزمان وجود نداشته باشد.

در ادامه مفهوم بن بست، شرایطی که به واسطه آن بن بست رخ می‌دهد و همچنین روش‌های اداره کردن و پیشگیری از بن بست را شرح می‌دهیم.

منبع (Resource)

منابع سیستم به دو دسته کلی تقسیم می‌شوند:

۱. **منابع اختصاصی:** این منابع را نمی‌توان در آن واحد و به طور مشترک در اختیار دو یا چند پردازه قرار داد (مانند چاپگر).

۲. **منابع اشتراکی:** این منابع این قابلیت را دارند که به طور همزمان مورد استفاده چندین پردازه قرار بگیرند (مانند دیسک).

به طور کلی زمانی که پردازه به منابعی نیاز دارد، مراحل زیر انجام می‌شود:

۱. **درخواست منبع (Request):** زمانی که پردازه‌ای به منابعی نیاز دارد، وقفه IO Request (درخواست ورودی/خروجی) رخ می‌دهد.

۲. **استفاده از منبع (Use):** اگر منبع آزاد باشد، در اختیار پردازه قرار می‌گیرد. در غیر این صورت، پردازه باید منتظر بماند تا منبع مورد نظر آزاد شود.

۳. **آزادسازی منبع (Release):** وقتی کار پردازه با منبع به پایان می‌رسد، وقفه IO Complete (کامل شدن عملیات ورودی/ خروجی) رخ می‌دهد و منبع آزاد می‌گردد.

همان‌طور که گفتیم سیستم دارای منابعی است و منابع انواع مختلفی نظیر R2 ،R1 ، ... و Rm دارند:

(CPU cycles , Memory spaces , I/O devices)

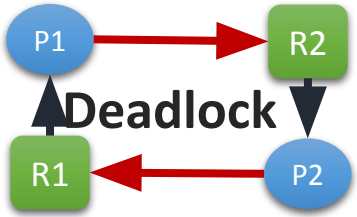
هر نوع منبع Ri دارای Wi نمونه از آن منبع است.

بن بست (Deadlock)

فرض کنید چندین پردازه در سیستم وجود دارند و برای دسترسی به منابع سیستم با هم به رقابت می‌پردازند.

اگر حالتی پیش بیاید که هر پردازه منبعی را در اختیار داشته باشد و برای تکمیل کار خود به منبع دیگری نیاز داشته باشد که در اختیار پردازه دیگر باشد، در این حالت کار هیچ یک از پردازه‌ها پیش نمی‌رود و همگی منتظر خواهند ماند. چنین وضعیتی را بن بست می‌نامند.

بن بست می‌تواند با درخواست عملیات ورودی/خروجی روی یک منبع اختصاصی (منبعی که قابل استفاده همزمان توسط چند پردازه نمی‌باشد) رخ دهد.



بن بست ممکن است هم روی منابع سخت‌افزاری و هم روی منابع نرم‌افزاری به وقوع بپیوندد.

ساده ترین شکل بن بست با دو پردازه P۱ و P۲ و دو منبع R۱ و R۲ می‌باشد:

"اگر پردازه P۱ منبع R۱ را در اختیار داشته باشد و منبع R۲ را درخواست دهد و همچنین پردازه P۲ منبع R۲ را در اختیار داشته باشد و منبع R۱ را درخواست کند، بن بست رخ می‌دهد."

شرایط وقوع بن بست

اگر تمام شرایط زیر در سیستم برقرار باشد، امکان وقوع بن بست وجود دارد. حتی اگر یکی از این شرایط نیز برقرار نباشد، هرگز بن بست رخ نمی‌دهد:

شرط اول: انحصار متقابل (Mutual Exclusion):

برای وقوع بن بست، باید منبعی داشته باشیم که به طور همزمان قابل استفاده چند پردازه نباشد (مانند چاپگر) . اگر بتوان منبع را بطور همزمان به پردازه‌ها تخصیص داد، هرگز بن بست رخ نمی‌دهد.

شرط دوم: نگهداری و انتظار (Hold & Wait):

هر پردازه بتواند منبعی را در بگیرد و تا زمانی که سایر منابع مورد نیازش را نگرفته است، منتظر بماند و منبع در اختیار خود را نیز رها نکند.

شرط سوم: انحصاری بودن (Non-preemptive):

وقتی پردازه منابعی را در اختیار میگیرد، تا زمانی که کارش با آن منبع به پایان نرسیده است، نتوان منبع را از پردازه گرفت.

شرط چهارم: انتظار چرخشی (Circular Wait):

چرخه ای از دو یا چند پردازه وجود داشته باشد، بطوری که هر پردازه منتظر منبعی باشد که در اختیار پردازه دیگر است. به عبارت دیگر، پردازه اول منتظر منبعی است که در اختیار پردازه دوم قرار دارد، پردازه دوم منتظر منبعی است که پردازه سوم آن را در اختیار دارد، ... و پردازه آخر منتظر منبعی است که در اختیار پردازه اول است.

گراف تخصیص منابع (Resource Allocation Graph)

برای نشان دادن درخواست منبع، در اختیار داشتن منبع، بن بست و شرایط آن می‌توان از یک گراف جهت‌دار به نام گراف تخصیص منابع استفاده کرد.

در این گراف، دو نوع گره (node) وجود دارد:

۱. **گره پردازه:** پردازه‌ها را با دایره نشان می‌دهیم.

۲. **گره منبع:** منابع را با مربع نشان می‌دهیم.

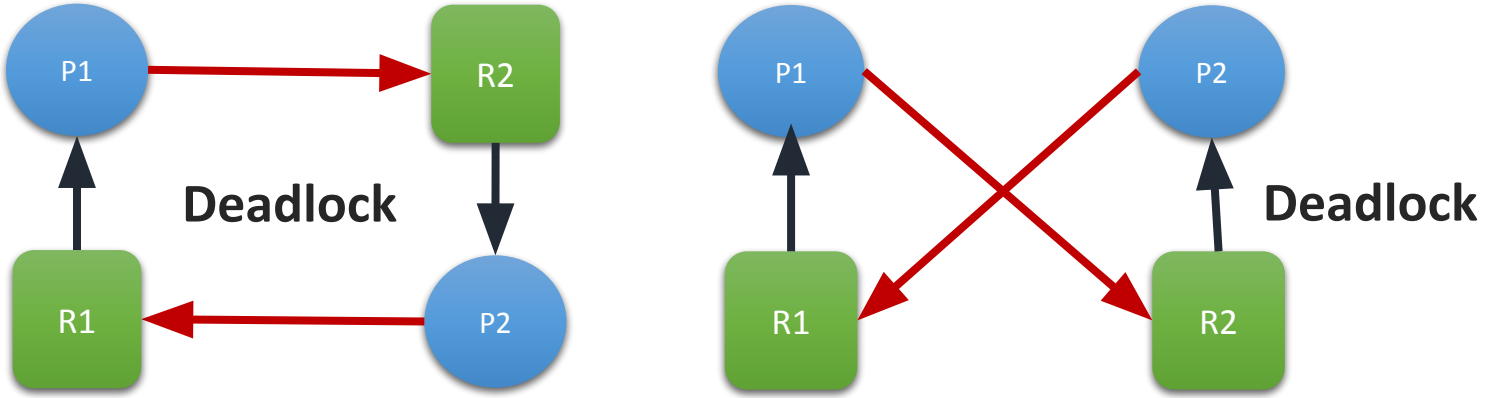
در گراف تخصیص منابع موارد زیر را در نظر داشته باشید:

۱. اگر پردازه P منبع R را درخواست کرده باشد ولی هنوز آن را در اختیار نگرفته باشد، فلشی از پردازه به سوی منبع رسم می‌شود.

۲. اگر پردازه P منبع R را در اختیار داشته باشد، فلشی از منبع به سمت پردازه رسم می‌شود.

۳. در ساده ترین شکل، بن بست از دو پردازه و دو منبع تشکیل می‌شود، به طوری که پردازه P۱ منبع R۱ را در اختیار دارد و منبع R۲ را درخواست کرده است و پردازه P۲ منبع R۲ را در اختیار دارد و منبع R۱ را درخواست کرده است.

گراف تخصیص منابع (Resource Allocation Graph)



برای درک بهتر گراف تخصیص منابع به ذکر مثال می‌پردازیم. در این مثال عمل request به معنی درخواست منبع و عمل release با معنی آزادسازی منبع است.

مثال گراف تخصیص منابع

مرحله اول: پردازه A منبع T را درخواست می‌کند. چون منبع T آزاد است در اختیار پردازه A قرار می‌گیرد.

مرحله دوم: پردازه B منبع S را درخواست می‌کند. چون منبع S آزاد است در اختیار پردازه B قرار می‌گیرد.

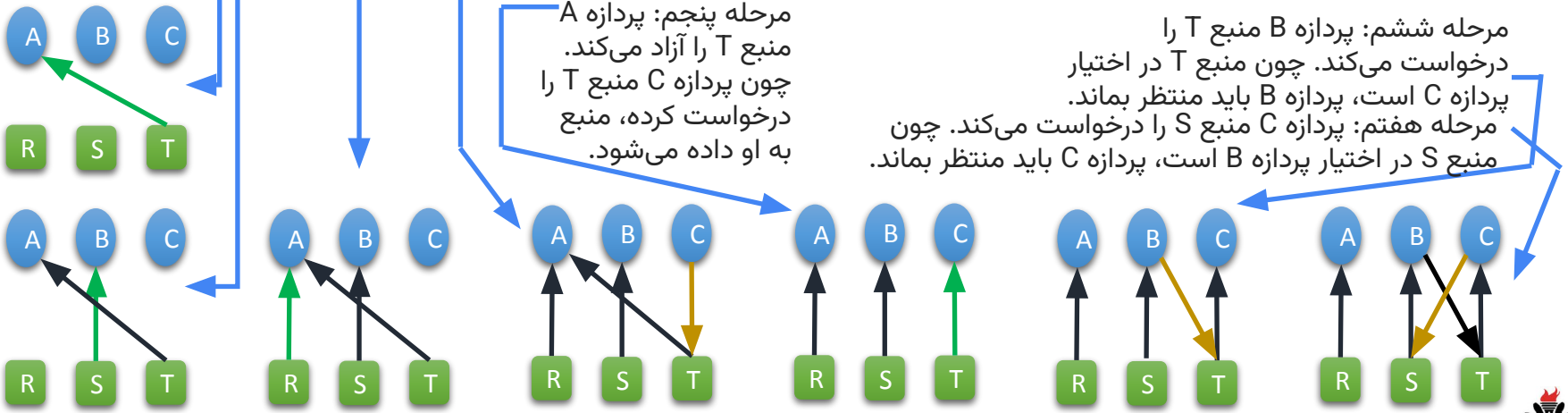
مرحله سوم: پردازه A منبع R را درخواست می‌کند. چون منبع R آزاد است در اختیار پردازه A قرار می‌گیرد.

مرحله چهارم: پردازه C منبع T را درخواست می‌کند. چون منبع T در اختیار پردازه A است، پردازه C باید منتظر بماند.

مرحله پنجم: پردازه A منبع T را آزاد می‌کند. چون پردازه C منبع T را درخواست کرده، منبع به او داده می‌شود.

مرحله ششم: پردازه B منبع T را درخواست می‌کند. چون منبع T در اختیار پردازه C است، پردازه B باید منتظر بماند.

مرحله هفتم: پردازه C منبع S را درخواست می‌کند. چون منبع S در اختیار پردازه B است، پردازه C باید منتظر بماند.

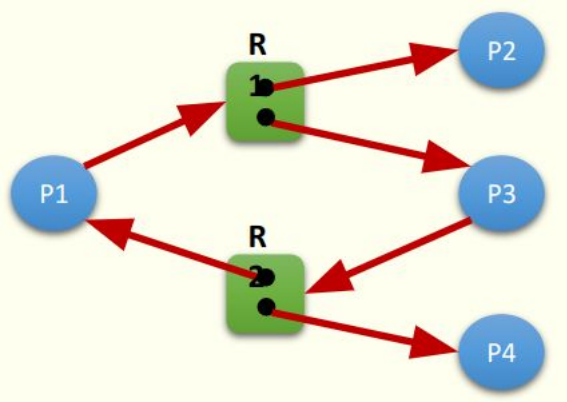


نکات مهم گراف تخصیص منابع

نکته اول: وجود حلقه در گراف تخصیص منابع برای وقوع بن بست شرطی لازم است ولی کافی نیست. اگر از هر نوع منبع چند نمونه داشته باشیم، ممکن است حتی با وجود حلقه در گراف، بن بست نداشته باشیم.

همان‌طور که در شکل مقابل ملاحظه می‌کنید، دو نمونه از منبع R1 داریم که یکی در اختیار P2 و دیگری در اختیار P3 می‌باشد. از منبع R2 نیز دو نمونه داریم که یکی در اختیار P1 و دیگری در اختیار P4 است. پردازه P1 یک نمونه از منبع R2 را در اختیار دارد و یک نمونه از R1 را درخواست کرده است. پردازه P2 یک نمونه از R1 را در اختیار دارد. پردازه P3 یک نمونه از R1 را در اختیار دارد و یک نمونه از R2 را درخواست کرده است. پردازه P4 یک نمونه از R2 را در اختیار دارد.

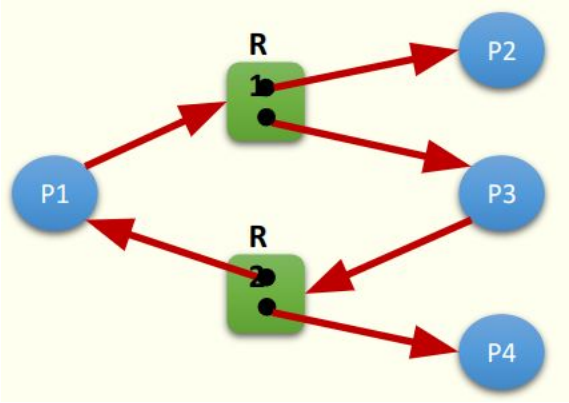
بابراین یسن P3 ، R1 ، P1 و R2 حلقه وجود دارد ولی بن بست رخ نمی‌دهد. زیرا پردازه P2 و P4 در حلقه نیستند و منتظر هیچ منبعی نیستند و بعد از مدتی کارشان به پایان می‌رسد و یک نمونه از R1 و یک نمونه از R2 آزاد می‌شود. که به ترتیب به P1 و P3 داده می‌شود و کار تمام پردازه‌ها بدون مشکل پیش می‌رود.



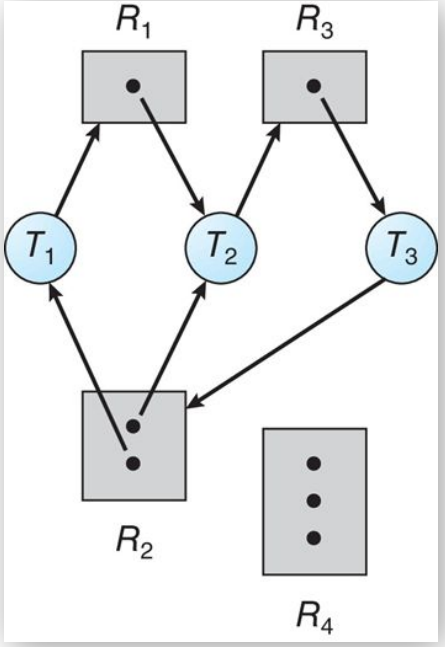
نکات مهم گراف تخصیص منابع

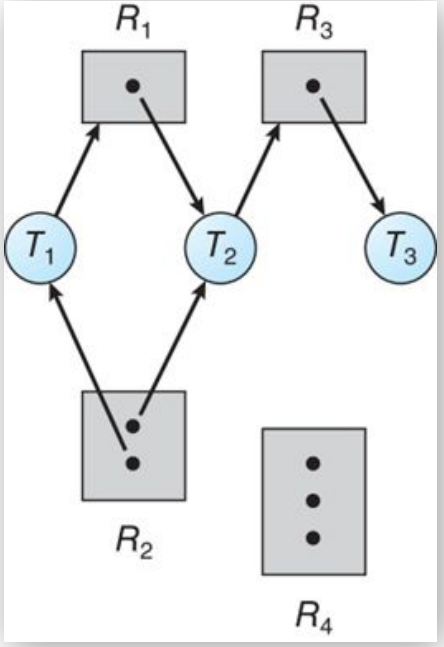
نکته دوم: اگر از هر نوع منبع فقط یک نمونه داشته باشیم، وجود حلقه در گراف، نشان دهنده بن بست خواهد بود.

نکته سوم: اگر گراف فاقد حلقه باشد، قطعا بن بست رخ نخواهد داد.



مثال گراف تخصیص منابع





روش‌های اداره کردن بن بست

- الگوریتم شترمرغی Ostrich Algorithm
- الگوریتم تشخیص و اصلاح Detection and Recovery
- الگوریتم پیشگیری از بن بست Deadlock Prevention
- اجتناب از بن‌بست Deadlock Avoidance

روش اول: الگوریتم شترمرغی (Ostrich Algorithm)

این روش، ممکن است در برخورد با بن بست در برنامه نویسی همزمان مورد استفاده قرار گیرد.

اگر اعتقاد بر این باشد که در سیستمی بن بست به ندرت رخ می‌دهد (مثلاً هر 10 سال یکبار) و هزینه تشخیص یا پیشگیری از بن بست بالا باشد، یک راه‌اندازی دستی مجدد (restart) می‌تواند مشکل را حل کند.

الگوریتم شترمرغی وانمود می‌کند مشکلی وجود ندارد و در صورتی که بن بست ها به ندرت اتفاق بیفتد و هزینه پیشگیری بالا باشد، قابل استفاده است.

دلیل این نامگذاری نیز به عملکرد شترمرغ برمی‌گردد. شترمرغ در هنگام بروز مشکل سر خود را در شن فرو می‌برد و وانمود می‌کند مشکلی پیش نیامده است.

روش دوم: الگوریتم تشخیص و اصلاح (Detection & Recovery)

در این روش نیز اجازه می‌دهیم بن بست رخ دهد، سپس دو مرحله داریم: 1– الگوریتم تشخیص بن بست و 2– الگوریتم اصلاح

الگوریتم تشخیص:

اگر از هر نوع منبع فقط یک نمونه موجود باشد، از گراف تخصیص منابع استفاده می‌کنیم.

اگر از هر نوع منبع چند نمونه داشته باشیم، از روی گراف نمی‌توان به سادگی بن بست را تشخیص داد و از الگوریتم بانکداران استفاده می‌کنیم.

الگوریتم اصلاح یا ترمیم:

پس از تشخیص بن بست نوبت به ترمیم و اصلاح آن می‌رسد. برای این منظور دو راه‌کار وجود دارد:

1 – تمام پردازه‌های مرتبط با بن بست را از سیستم خارج می‌کنیم.

2 – یکی از پردازه‌های مرتبط با بن بست را از سیستم خارج می‌کنیم. اگر با این کار مشکل بن بست حل نشد، پردازه دیگر را خارج می‌کنیم. این کار را آنقدر ادامه می‌دهیم که تا حلقه انتظار شکسته شود و بن بست برطرف گردد.

روش دوم: الگوریتم تشخیص و اصلاح (Detection & Recovery)

نکته مهم:

در راه‌کار دوم، برای انتخاب پردازه‌ای که باید از سیستم خارج شود، می‌توان یکی از شرایط زیر را در نظر گرفت :

1 – پردازه‌ای باید از سیستم خارج شود که مدت زمان کمتری از پردازنده استفاده کرده است.

2 – پردازه‌ای باید از سیستم خارج شود که تعداد منابع بیشتری را در اختیار دارد، تا احتمال برطرف شدن بن بست بیشتر شود.

3 – پردازه‌ای باید از سیستم خارج شود که تا کنون کمترین خروجی را داشته است.

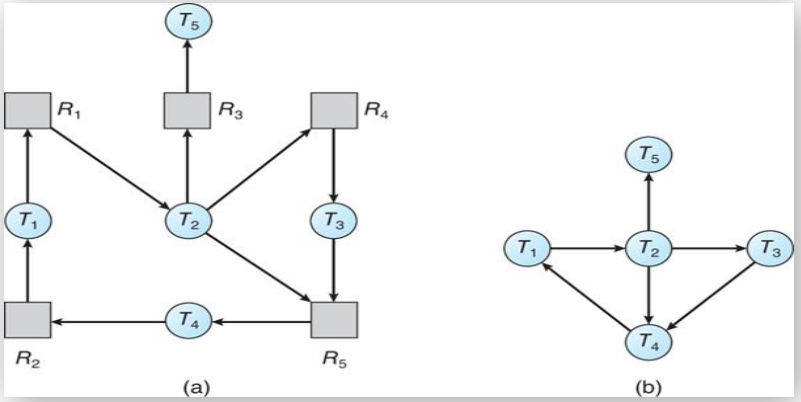
4 – پردازه‌ای باید از سیستم خارج شود که طبق تخمین اولیه، بیشترین زمان باقیمانده کار را داشته باشد.

5 – پردازه‌ای باید از سیستم خارج شود که اولویت کمتری نسبت به سایر پردازه‌ها داشته باشد.

6 – و غیره .

تشخیص بن بست با گراف (از هر نوع منبع فقط یک نمونه)

اگر از هر نوع منبع فقط یک نمونه داشته باشیم، می‌توانیم با استفاده از گراف تخصیص منابع وجود بن بست را تشخیص دهیم. برای این منظور از روی گراف تخصیص منبع (شکل a) گراف دیگری به نام wait-for (شکل b) ایجاد می‌کنیم. گره‌های این گراف پردازه‌ها هستند. اگر پردازه Pi برای انجام کارش به منبعی نیاز داشته باشد که در اختیار پردازه Pj باشید، می‌گوییم پردازه Pi منتظر پردازه Pj است و یالی از Pi به Pj خواهیم داشت.



اگر در گراف wait-for حلقه وجود داشته باشد، قطعا بن بست رخ داده است.

روش سوم: پیشگیری از بن بست (Deadlock Prevention)

در این روش، سیستم عامل سعی می‌کند با نقض یکی از شرایط بن بست از وقوع آن پیشگیری کند. حال به بررسی چگونگی نقض هر یک از شرایط می‌پردازیم :

انحصار متقابل: نقض این شرط در مورد تمامی منابع امکان پذیر نیست. به عبارت دیگر، نمی‌توان برخی منابع را به طور همزمان در اختیار چند پردازه قرار داد. بنابراین در مورد منابعی اختصاصی نظیر چاپگر نقض انحصار متقابل امکان پذیر نیست.

نگهداری و انتظار: نقض این شرط به دو صورت امکان پذیر است :

1 – هر پردازه قبل از شروع به کار، تمام منابع مورد نیازش را درخواست کند و تا زمانی که همه را در اختیار نگرفته باشد، کارش آغاز نمی‌شود. معایب این راه‌کار عبارتند از :

الف – ممکن است پردازه‌ای برای مدت طولانی منتظر بماند تا تمام منابع مورد نیازش را در اختیار بگیرد.

ب – ممکن است پردازه در ابتدای کار تمام منابع مورد نیازش را نداند.

ج – پردازه ممکن است در ابتدای کار منابعی را در اختیار بگیرد که در آن لحظه به آنها نیازی نداشته باشد.

2 – در هر لحظه فقط به هر پردازه یک منبع را اختصاص می‌دهیم. اگر پردازه‌ای منبع دوم را نیاز داشته باشد، باید منبعی که در اختیار دارد را به طور موقت رها کند.

انحصاری بودن: در این حالت، می‌توان بر حسب نیاز منبع در اختیار پردازه را از او گرفت و به پردازه دیگر تخصیص داد. پس باید بتوان وضعیت جاری منابع را ذخیره کرد تا در صورت نیاز پردازه از همان وضعیت از سر گرفته شود.

انتظار چرخشی: برای نقض این شرط این قانون را وضع می‌کنیم که پردازه‌ها مجاز نیستند به هر ترتیبی منابع را درخواست کنند. به هر منبع یک شماره اختصاص می‌دهیم و به پردازه‌ها اجازه می‌دهیم که فقط به ترتیب شماره منبع، آن را درخواست کنند. در واقع، هر پردازه فقط می‌تواند منبعی را درخواست کند که شماره آن بیشتر از سایر منابعی باشد که در اختیار دارد. مشکل این است که ترتیب درخواست منابع مورد نیاز یک پردازه ممکن است متفاوت باشد.

روش چهارم : اجتناب از بن بست (Deadlock Avoidance)

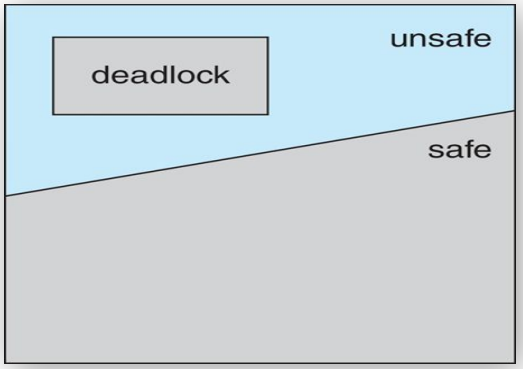
در این روش، با در اختیار داشتن یکسری اطلاعات در مورد پردازه‌ها و منابع مورد نیازشان وضعیت‌های مختلف ناشی از تخصیص منابع را مورد بررسی قرار می‌دهیم تا ببینیم سیستم در حالت امن (safe) است یا ناامن (unsafe) . یکی از اطلاعاتی که در این روش مورد نیاز است، حداکثر تعداد منابعی است که هر پردازه ممکن است نیاز داشته باشد.

حالت امن: می‌گوییم سیستم در حالت امن است اگر بتواند به یک ترتیب خاص همه منابع مورد نیاز پردازه‌ها را تخصیص دهد و از بن بست جلوگیری کند. به عبارت دیگر، سیستم در حالت امن است اگر یک ترتیب امن وجود داشته باشد که بر اساس آن بتوان نیاز تمام پردازه‌ها را برآورده کرد. ترتیب (P1 , P2 , ... , Pn) را ترتیب امن می‌نامیم اگر برای هر پردازه Pi بتوان با منابع موجود نیاز آن را برآورده کرد.

حالت ناامن: می‌گوییم سیستم در حالت ناامن است اگر هیچ ترتیب امنی برای آن یافت نشود. به عبارت دیگر، تمام ترتیب‌های تخصیص ممکن ناامن باشند. ترتیب نامن ترتیبی است که بر اساس آن نتوان نیاز تمام پردازه‌ها را با منابع موجود برآورده کرد.

روش چهارم : اجتناب از بن بست (Deadlock Avoidance)

نکته : اگر سیستمی امن باشد ، هرگز بن بست رخ نخواهد داد ولی اگر سیستمی ناامن باشد، احتمال وقوع بن بست وجود دارد. عدم قطعیت بن بست را می‌توان به‌طور قطع اعلام کرد ولی وقوع بن بست را نمی‌توان با قاطعیت بیان کرد، چون در بررسی حالت امن حداکثر نیاز پردازه‌ها در نظر گرفته می‌شود. در حالی که ممکن است در عمل نیاز پردازه کمتر از آنچه به عنوان حداکثر نیاز برآورده شده است باشد و بن بست رخ ندهد.



الگوریتم‌های اجتناب از بن بست

الگوریتم‌های اجتناب از بن بست :

- اگر از هر نوع منبع یک نمونه داشته باشیم، از گراف تخصیص منبع استفاده می‌کنیم.
  - یال Ti -> Ri یک clime edge است و نشان می‌دهد که پردازه Ti ممکن است منبع Ri را درخواست کند.
  - اگر پردازه منبع را درخواست کند، clime edge به request edge تبدیل می‌شود.
  - اگر منبع در اختیار پردازه قرار گیرد، request edge به assignment edge تبدیل می‌شود.
  - اگر پردازه منبع را آزاد کند، assignment edge به claim edge تبدیل خواهد شد.

- اگر از هر نوع منبع چند نمونه داشته باشیم، از الگوریتم بانکداران (Bankers) استفاده خواهیم کرد.

نکته خیلی مهم : اگر در سیستمی n پردازه و m نمونه از یک نوع منبع داشته باشیم و شرط زیر برقرار باشد، هرگز بن بست رخ نخواهد داد :

$\sum_{i=1}^{n} request[i] < m + n$

الگوریتم بانکداران (Bankers Algorithm)

در این روش، برای بررسی امن و ناامن بودن سیستم باید چندین ساختار داده را در سیستم نگهداری کنیم:

n: تعداد پردازه‌ها را مشخص می‌کند.

m: تعداد انواع منابع را نشان می‌دهد.

آرایه یک بعدی available (به طول m): تعداد نمونه‌های موجود (آزاد) از هر نوع منبع را نشان می‌دهد.

آرایه یک بعدی total (به طول m): تعداد کل نمونه‌های موجود از هر منبع را بیان می‌کند.

ماتریس max (با ابعاد n x m): حداکثر نیاز هر پردازه را از هر نوع منبع مشخص می‌کند.

ماتریس allocate (با ابعاد n x m): تعداد منابع تخصیص یافته از هر نوع منبع را به هر پردازه نشان می‌دهد.

ماتریس need (با ابعاد n x m): باقیمانده نیاز هر پردازه را نگهداری می‌کند. این ماتریس را می‌توان به صورت زیر محاسبه کرد :

```
need[i][j] = max[i][j] – allocate[i][j]
```

الگوریتم بانکداران (Bankers Algorithm)

نکته 1 : در مسائلی که مطرح می‌شود تعداد پردازه‌ها، تعداد منابع و ماتریس allocate باید به ما داده شود. از دو گزینه available و total یکی را به ما می‌دهند. ما برای تشخیص امن/ناامن بودن سیستم به available نیاز داریم. حال اگر total را بدهند باید از روی آن available را به دست آوریم. همچنین از بین دو گزینه need و max یکی را به ما می‌دهند. ما برای حل مسئله به ماتریس need نیاز داریم، پس اگر max را داده باشند باید از روی آن need را به دست آوریم.

نکته 2 : برای حل مسئله با الگوریتم بانکداران باید available ، allocate و need را داشته باشیم.

الگوریتم بانکداران از دو بخش تشکیل شده است:

- **بخش اول، الگوریتم امنیت (Safety):** از این الگوریتم برای تشخیص امن/ناامن بودن سیستم استفاده می‌کنیم.
- **بخش دوم، الگوریتم درخواست منبع (Resource-Request):** از این الگوریتم برای رسیدگی به درخواست منبع استفاده می‌کنیم.

الگوریتم امنیت (Safety Algorithm)

مراحل الگوریتم امنیت به صورت زیر می‌باشد :

مرحله اول : دو آرایه یک بعدی work و finish را به ترتیب با طول m و n به صورت زیر تعریف می‌کنیم :

work = available , finish[i] = false for i=1,...,n

در واقع آرایه work همان موجودی سیستم است و آرایه finish نیز برای هر پردازه مشخص می‌کند که کارش تمام شده است یا خیر.

مرحله دوم : در ماتریس need به دنبال سطری مانند سطر i می‌گردیم که دو ویژگی زیر را داشته باشد :

finish[i] = false , need[i] ≤ work

به عبارت دیگر در ماتریس need به دنبال پردازه‌ای می‌گردیم که هنوز به پایان نرسیده باشد و نیازش کمتر یا مساوی موجودی باشد و با منابع موجود بتوان نیاز آن پردازه را برآورده کرد.

مرحله سوم : اگر هیچ سطری با مشخصات فوق یافت نشود، به مرحله پنجم می‌رویم.

مرحله چهارم : اگر سطری مانند سطر i با شرایط فوق یافت شود، منابع مورد نیازش را به او می‌دهیم. با تخصیص منابع به Pi تغییرات زیر اعمال می‌شود :

work_after= work_before – need[i] (I) → (منابع داده شده از موجودی کم می‌شود)

allocate_after[i] = allocate_before[i] + need[i] (II) → (منابع داده شده، به منابع تخصیص یافته به پردازه اضافه می‌شود)

work = work_after + allocate_after[i] (III) → (وقتی پردازه تمام منابعش را در اختیار می‌گیرد، کارش به پایان می‌رسد و منابع در اختیارش را آزاد می‌کند)

finish[i] = true → (پردازه به پایان می‌رسد)

حال با موجودی جدید عملیات را از مرحله دوم تکرار می‌کنیم.

مرحله پنجم : اگر برای تمام پردازه‌ها finish[i] = true باشد سیستم در حالت امن است، در غیر این صورت سیستم در حالت ناامن خواهد بود.

نکته ۱ : دستور (II) و (III) را می‌توان به صورت زیر در یک دستور خلاصه کرد :

work = work_before + allocate_before[i]

نکته ۲ : الگوریتم امنیت از مرتبه $m \times n^2$ می‌باشد.

الگوریتم درخواست منبع (Resource-Request Algorithm)

مراحل الگوریتم در خواست منبع با فرض اینکه پردازه Pi درخواست request[i] را داشته باشد، به صورت زیر می‌باشد :

مرحله اول : ابتدا نیاز پردازه با درخواست مقایسه می‌شود. (request[i] ≤ need[i])
اگر درخواست پردازه از نیاز او بیشتر باشد نمی‌توان به درخواست رسیدگی کرد و الگوریتم به پایان می‌رسد. در غیر اینصورت، به مرحله بعد می‌رویم.
مرحله دوم : درخواست پردازه با موجودی مقایسه می‌شود. (request[i] ≤ available[i])
اگر درخواست پردازه از موجودی بیشتر باشد نمی‌توان به درخواست رسیدگی کرد و الگوریتم به پایان می‌رسد. در غیر این صورت به مرحله بعد می‌رویم.
مرحله سوم : حال می‌توانیم به درخواست رسیدگی کنیم و منابع مورد نیاز پردازه را به او بدهیم. با این کار تغییرات زیر اعمال می‌شود :
available = available – request[i] → (منابع درخواست شده از موجود کم می‌شود)
allocate[i] = allocate[i] + request[i] → (منابه درخواست شده به منابع تخصیص یافته اضافه می‌شود)
need[i] = need[i] – request[i] → (منابع درخواست شده از نیاز پردازه کم می‌شود)

نکته ۱ : اگر الگوریتم درخواست منبع با موفقیت به پایان برسد و یا حتی در مراحل یک و دو متوقف شود، حتماً باید پس از آن برای تشخیص امن / ناامن بودن سیستم، الگوریتم امنیت را اجرا می‌کنیم.
نکته ۲ : مسائل بانکداران به دو صورت مطرح می‌شوند:
حالت اول : با خواستی برای یک یا چند پردادن اطلاعات در مورد پردازه‌ها و منابع ، وضعیت امن / ناامن بودن سیستم خواسته می‌شود. در این صورت الگوریتم امنیت را بکار می‌بریم.
حالت دوم : دردازه مطرح می‌شود و وضعیت سیستم را بعد از رسیدگی به درخواست‌ها می‌خواهند. در این حالت ابتدا باید الگوریتم درخواست منبع و سپس الگوریتم امنیت را اجرا کنیم.

چند نمونه مثال برای بن بست

پردازه	منابع تخصیص یافته			حداکثر منابع مورد نیاز		
	A	B	C	A	B	C
P0	0	1	0	7	5	3
P1	2	0	0	3	2	2
P2	3	0	2	9	0	2
P3	2	1	1	2	2	2
P4	7	2	5	کل منابع تخصیص یافته		

منابع مورد نیاز		
A	B	C
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

available = total – allocated = (10,5,7) – (7,2,5) = (3,3,2)

مثال اول : فرض کنید پنج پردازه P0 تا P4 و سه نوع منبع A ، B و C وجود دارد. 10 نمونه از منبع A ، 5 نمونه از منبع B و 7 نمونه از منبع C کل منابع سیستم را تشکیل می‌دهند. اطلاعات مربوط به حداکثر نیاز و منابع تخصیص یافته به پردازه‌ها در جدول زیر مشخص شده است.

الف– مشخص کنید با شرایط فعلی سیستم در حالت امن است یا خیر.

ب– اگر پردازه P1 یک نمونه از منبع A و دو نمونه از منبع C درخواست کرده باشد. پس از اعمال درخواست وضعیت سیستم چگونه خواهد بود؟

حال با داشتن این اطلاعات به حل قسمت‌هال الف و ب می‌پردازیم.

مثال دوم : فرض کنید در یک سیستم 8 نمونه از یک منبع وجود دارد که n پردازه برای دسترسی به آنها با هم رقابت می‌کنند. با فرض اینکه هر پردازه برای اجرا حداکثر به سه نمونه منبع نیاز داشته باشد، این سیستم به ازای چه مقادیری از n فاقد بن بست می‌باشد؟

مثال دوم : سیستمی شامل چهار پردازه و سه منبع را در نظر بگیرید. اگر هر پردازه حداکثر به سه منبع نیاز داشته باشد، تعداد وضعیت‌های بن بست در این سیستم حداکثر چقدر است ؟

حل مثال اول – قسمت الف

موجودی (2و3و3) را با سرهای ماتریس need مقایسه می‌کنیم. همان‌طور که می‌بینید نیاز P1 یعنی (2و2و1) کمتر یا مساوی موجودی است. بنابراین داریم :

```
work = work + allocate[P1] = (3,3,2) + (2,0,0) = (5,3,2)
```

حال موجودی جدید را با سایر سطرهای need مقایسه می‌کنیم. نیاز پردازه P3 یعنی (1و1و0) کمتر یا مساوی موجودی (2و3و5) است. بنابراین داریم :

```
work = work + allocate[P3] = (5,3,2) + (2,1,1) = (7,4,3)
```

موجودی جدید را با سایر سطرهای need مقایسه می‌کنیم. نیاز پردازه P0 یعنی (3و4و7) مساوی موجودی (3و4و7) است. بنابراین داریم :

```
work = work + allocate[P0] = (7,4,3) + (0,1,0) = (7,5,3)
```

موجودی جدید را با سایر سطرهای need مقایسه می‌کنیم. نیاز پردازه P2 یعنی (1و1و0) کمتر یا مساوی موجودی (3و5و7) است. بنابراین داریم :

```
work = work + allocate[P2] = (7,5,3) + (3,0,2) = (10,5,5)
```

در نهایت موجودی جدید را با سایر سطرهای need مقایسه می‌کنیم. نیاز آخرین پردازه یعنی P4 که (2و0و0) است کمتر یا مساوی موجودی (5و5و10) است. بنابراین داریم :

```
work = work + allocate[P2] = (10,5,5) + (0,0,2) = (10,5,7) = total
```

در نتیجه طبق ترتیب امن (P1 , P3 , P0 , P2 , P4) سیستم در حالت امن است. البته پس از تخصیص منابع به P3 موجودی برابر (3و4و7) می‌شود و چون از تمام سطرهای باقیمانده need بزرگتر یا مساوی است، دیگر نیازی به ادامه دادن نیست و می‌گوییم سیستم در حالت امن است.

حل مثال اول – قسمت ب

ابتدا باید الگوریتم درخواست منبع را اعمال کنیم. پردازه P1 یک واحد از منبع A و دو واحد از منبع C درخواست کرده است ← request[P1] = (1,0,2)

مرحله اول :

درخواست پردازه با نیازش مقایسه می‌شود :

چون (۲و۰و۱) از (۲و۲و۱) کوچکتر یا مساوی است به مرحله بعد می‌رود. request[P1] ≤ need[P1] → (1,0,2) ≤ (1,2,2) √

مرحله دوم :

درخواست پردازه با موجودی مقایسه می‌شود :

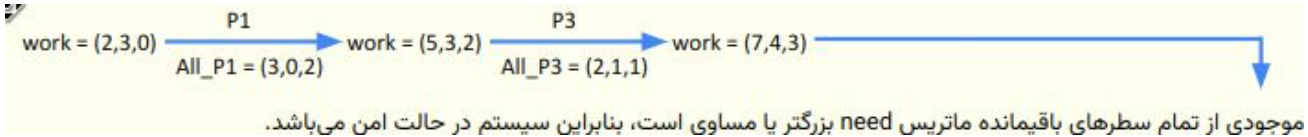
چون (۲و۰و۱) از (۲و۳و۳) کوچکتر یا مساوی است به مرحله بعد می‌رود. request[P1] ≤ available → (1,0,2) ≤ (3,3,2) √

مرحله سوم :

تخصیص منابع درخواست شده به P1 :

available = available – request[P1] = (3,3,2) – (1,0,2) = (2,3,0)
allocate[P1] = allocate[P1] + request[P1] = (2,0,0) + (1,0,2) = (3,0,2)
need[P1] = need[P1] – request[P1] = (1,2,2) – (1,0,2) = (0,2,0)

حال با موجودی جدید الگوریتم امنیت را اعمال می‌کنیم:



موجودی از تمام سطرهای باقیمانده ماتریس need بزرگتر یا مساوی است، بنابراین سیستم در حالت امن می‌باشد.

حل مثال دوم

اگر nپردازه و m نمونه از یک منبع داشته باشیم، برای آنکه هرگز بن بست رخ ندهد باید شرط زیر برقرار باشد :

$\sum_{i=1}^{n} request[i] < m + n$

در این جا، n پردازه داریم که هر کدام سه منبع نیاز دارند، پس :

$\sum_{i=1}^{n} request[i] = 3 * n$

همچنین ۸ منبع داریم، بنابراین :

$3 * n < 8 + n \rightarrow 2 * n < 8 \rightarrow n < 4$

در نتیجه برای n های ۱ ، ۲ و ۳ سیستم فاقد بن بست است.

حل مثال سوم

هر پردازه حداکثر سه منبع نیاز دارد، اگر سه منبع را به یک پردازه بدهیم مشکلی پیش نمی‌آید و بن بست رخ نمی‌دهد.

در دو مورد زیر به مشکل بر می‌خوریم:

مورد اول: هر منبع را به یک پردازه بدهیم.

مورد دوم: دو منبع را به یک پردازه و منبع سوم را به پردازه دیگر بدهیم.

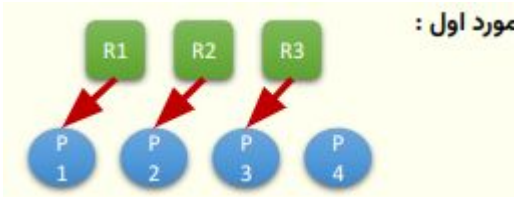
حال باید برای هر مورد تعداد حالت‌های ممکن را بدست آورده و با هم جمع کنیم.

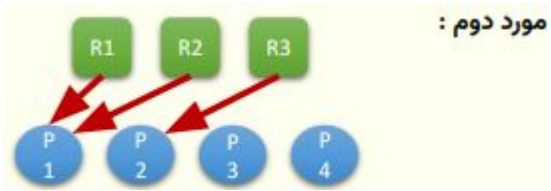
حالت‌های مختلف این مورد عبارتند از:

(P1,P2,P3), (P1,P2,P4), (P1,P3,P4), (P2,P3,P4)

بنابراین در مورد اول 4 حالت داریم که به مشکل برمی‌خوریم.

در مجموع تعداد وضعیت‌های بن بست در این سیستم حداکثر 16 حالت می‌باشد.





حالت‌های مختلف این مورد عبارتند از:

(P1,P1,P2), (P1,P1,P3), (P1,P1,P4)
(P2,P2,P1), (P2,P2,P3), (P2,P2,P4)
(P3,P3,P1), (P3,P3,P2), (P3,P3,P4)
(P4,P4,P1), (P4,P4,P2), (P4,P4,P3)

بنابراین در مورد دوم 12 حالت داریم که به مشکل برمی‌خوریم.

پایان فصل پنجم