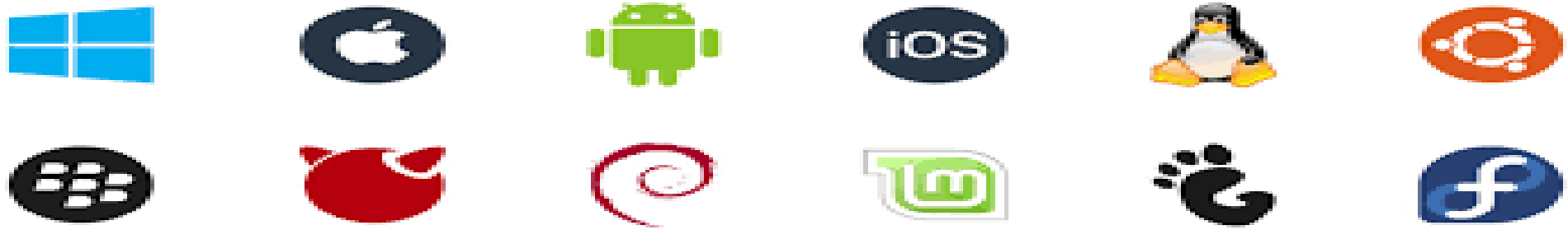


سیستم های عامل

Operating Systems

میلاد سلطانی

فصل هفتم

بن بست

- مفهوم بن بست
- شرایط لازم برای ایجاد بن بست
  - ✓ انحصار متقابل
  - ✓ گرفتن و منتظر ماندن
  - ✓ انحصاری بودن
  - ✓ انتظار چرخشی
- گراف تخصیص منابع
- نحوه اداره کردن بن بست
  - ✓ پیشگیری یا جلوگیری
  - ✓ اجتناب (الگوریتم بانکدار)
  - ✓ آشکار سازی و بازیافت
  - ✓ نادیده گرفتن
- شرط رخ ندادن بن بست

مفهوم بن بست

- اگر هر فرآیندی متعلق به یک مجموعه، منتظر یک حادثه است که تنها توسط فرآیند دیگری در آن مجموعه می تواند رخ دهد، بن بست پدید می آید.

مثال: سیستمی یک چاپگر و یک درایو سی دی دارد. اگر در حال حاضر، فرآیند P1 از چاپگر و فرآیند P2 از درایو سی دی استفاده کنند و فرآیند P1 درخواست چاپگر و فرآیند P2 درخواست درایو سی دی را داشته باشند، آنگاه بن بست اتفاق می اُفتد.

تذکر! در سیستم عامل های امروزی احتمال بن بست خیلی کم بوده و در اکثر آنها روش های رفع بن بست طراحی نشده است. مباحث این فصل احتمالاً در آینده به سیستم های عامل اضافه می شود.

مفهوم بن بست

- منظور از حادثه مورد انتظار در این فصل بحث دریافت منابع و رها سازی آنها می باشد.
  - ✓ منبع می تواند فیزیکی (مثل چاپگر) یا منطقی (مثل فایل ها) باشد.
  - ✓ ممکن است از هر منبع در سیستم چند نوع موجود باشد مثلاً ۲ چاپگر و ۳ درایو سی دی
  - ✓ تعداد منابع درخواستی از سوی فرآیند ها نباید از مجموع کل منابع موجود بیشتر باشد.

- سیستم عامل برای مدیریت منابع به چهار عمل زیر نیاز دارد:
  - ★ تعیین وضعیت لحظه ای منابع (Resource Status)
  - ★ زمانبندی درست استفاده از منابع (Scheduling)
  - ★ تخصیص صحیح منابع به فرآیندها (Allocation)
  - ★ باز پس گیری منابع از فرآیندها (Release)

شرایط لازم برای ایجاد بن بست

- شرایط زیر که معروف به شرایط کافمن هستند، برای رخ دادن بن بست باید برقرار باشند:

✓ انحصار متقابل (Mutual Exclusion)

- یعنی حداقل یک منبع غیر قابل اشتراک باشد. یعنی در هر لحظه، یک فرآیند می تواند از این منبع استفاده کند.

✓ گرفتن و منتظر ماندن (Hold and Wait)

- یعنی باید فرآیندی باشد که یک حداقل منبع گرفته و منتظر منابع دیگر می باشد.

✓ انحصاری بودن (No Preemption)

- نمی توان منبع را به اجبار از فرآیندی گرفت. فقط فرآیند داوطلبانه منبع را آزاد می کند.

✓ انتظار چرخشی (Circular Wait)

- مجموعه ای از فرآیند ها موجود باشند که بصورت زنجیره ای متقاضی منابع هم باشند.

گراف تخصیص منبع

- یک گراف جهت دار به نام گراف تخصیص منبع (Resource Allocation Graph) برای نمایش بن بست استفاده می شود.

- اجزای گراف تخصیص منبع:

۱) فرآیندها (R_i) با دایره نشان داده می شوند.

۲) منابع (P_i) با مستطیل نمایش داده می شوند.

۳) تعداد هر منبع را با نقطه درون مستطیل نشان می دهند.

۴) درخواست منبعِ یک فرآیند را با کمان جهت داری از فرآیند به سمت منبع نشان می دهند.

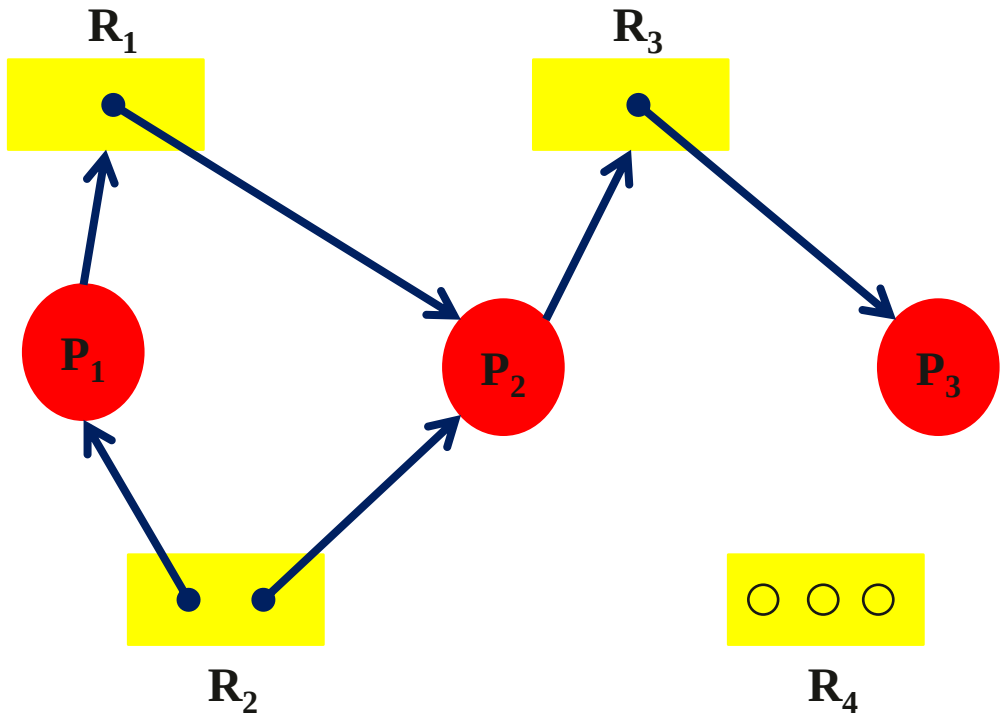
۵) اگر منبعی در اختیار فرآیندی باشد، با یک کمان جهت دار از منبع به سمت فرآیند نمایش می دهند.

مثال: گراف تخصیص منابع مربوط به سیستم زیر را ترسیم کنید.

فرآیندها $P = \{ P_1 ، P_2 ، P_3\}$

منابع $R = \{R_1$ (یک نمونه) $، R_2$ (دو نمونه) $، R_3$ (یک نمونه) $، R_4$ (سه نمونه)$\}$

وضعیت سیستم $E = \{P_1 \text{ -> } R_1 ، P_2 \text{ -> } R_3 ، R_1 \text{ -> } P_2 ، R_2 \text{ -> } P_2 ، R_2 \text{ -> } P_1 ، R_3 \text{ -> } P_3\}$



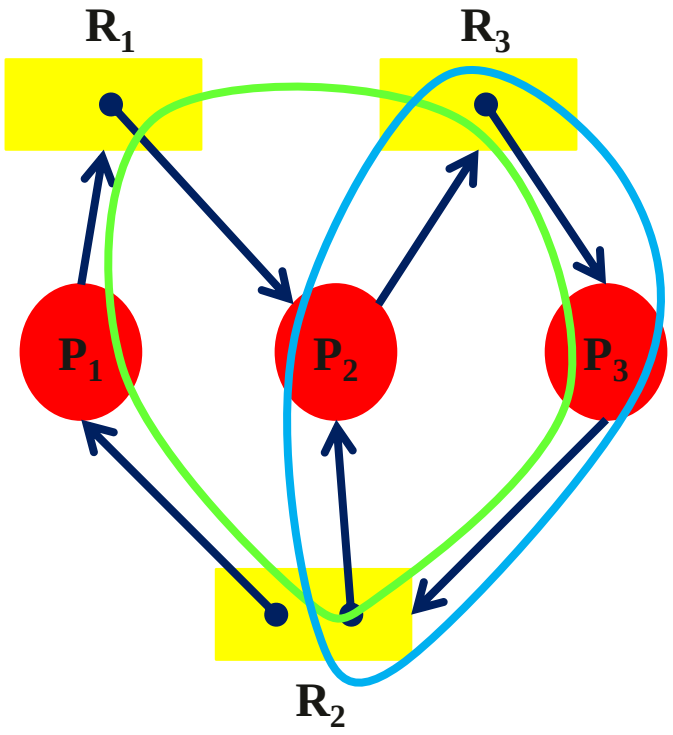
گراف تخصیص منبع

- اگر گراف هیچ حلقه ای نباشد، هیچ فرآیندی در سیستم، در بن بست قرار نخواهد گرفت.

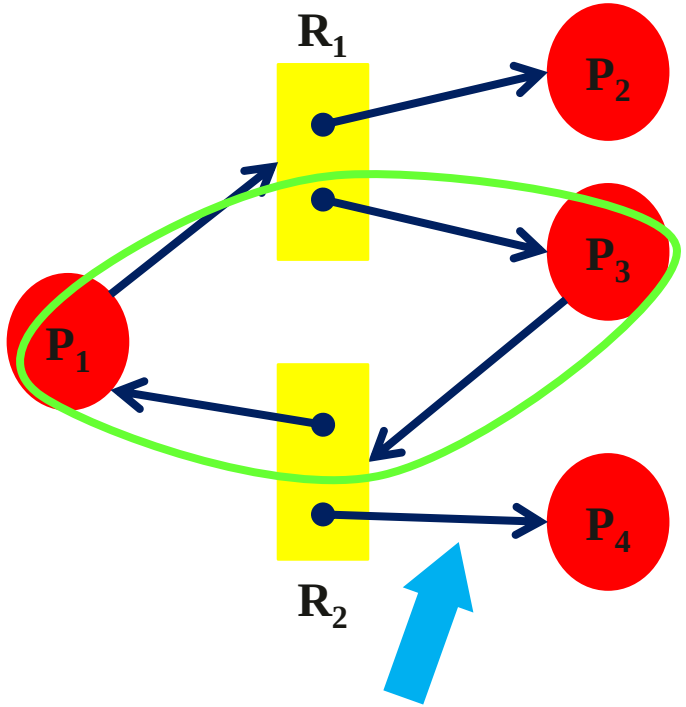
- اگر گراف دارای حلقه باشد:
  - ✓ اگر از هر منبع دقیقاً یک عدد در سیستم باشد، وقوع بن بست حتمی است.
  - ✓ اگر از هر منبع بیش یک عدد در سیستم باشد، حلقه الزاماً به معنای وقوع بن بست نیست.

نکته! وجود حلقه در گراف شرط لازم برای بن بست است ولی شرط کافی نیست.

مثال: آیا گراف های زیر در بن بست قرار دارند؟



بن بست وجود دارد.



بن بست وجود ندارد.

نحوه اداره کردن بن بست

- برای برخورد با بن بست ۴ روش وجود دارد
  - پیشگیری یا جلوگیری از بن بست (Deadlock Prevention)
  - اجتناب از بن بست (Deadlock Avoidance)
  - آشکار سازی و بازیافت (کشف) (Detection and Recovery)
  - صرف نظر کردن از بن بست

پیشگیری یا جلوگیری از بن بست

- منظور اینست که هیچگاه نگذاریم یکی از چهار شرط لازم برای بن بست رخ دهد.

۱. **انحصار متقابل:** اگر از منابع اشتراکی استفاده کنیم، بن بست هیچ گاه رخ نخواهد داد.

۲. **گرفتن و منتظر ماندن:**
   - یا هر فرآیند قبل از اجرا تمام منابع خود را بگیرد.
   - یا هر فرآیند قبل از درخواست منابع بیشتر، منابع قبلی در اختیار خود را رها کند.

۳. **انحصاری بودن:** اگر فرآیندی منبعی را تقاضا کند که نمی توان به آن داد، کلیه منابع دیگر از آن گرفته شود. برای آغاز مجدد، فرآیند مذکور باید کلیه منابع قدیم و جدید را کسب کرده باشد.

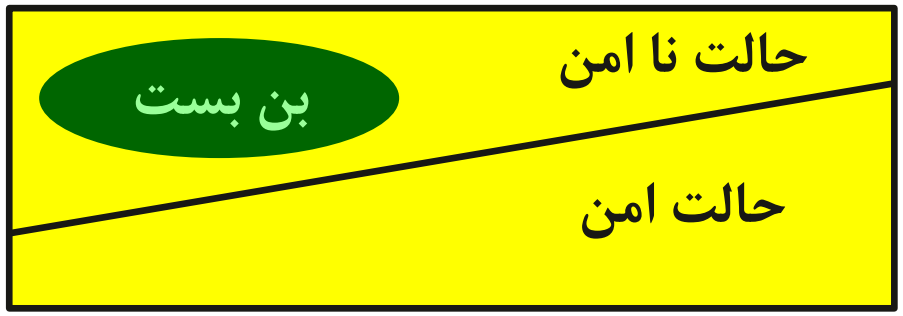
۴. **انتظار چرخشی:** به هر منبع شماره ای یکتا اختصاص داده شود و هر فرآیند فقط منابع را در جهت صعودی شماره منابع دریافت کند.

مشکلات روش های پیشگیری از بن بست

- تمامی منابع را نمی توان این گونه مدیریت کرد. (روش اول، سوم و چهارم)
- بهره وری پایین منابع سیستم به دلیل بیکاری برخی از منابع به مدت طولانی (روش دوم)
- قحطی زدگی به این معنی که یک فرآیند وقتی نیاز به منابع زیادی دارد بطور نا معین باید در انتظار باشد. (روش دوم)

اجتناب از بن بست

- در این روش سعی می شود، سیستم همیشه در حالت امن (Safe State) قرار بگیرد.
  - ✓ اگر سیستم منابع درخواستی را طوری در اختیار فرآیندها قرار دهد که از بروز بن بست اجتناب شود، سیستم در حالت امن قرار خواهد گرفت.
- در این روش هرگاه فرآیندی درخواست منبع می کند، سیستم تصمیم می گیرد که آیا فوراً تقاضا پاسخ داده شود یا فرآیند باید صبر کند. منبع به شرطی به فرآیند داده می شود که سیستم در حالت امن باقی بماند.



الگوریتم بانکدار

- این الگوریتم اولین بار توسط دیجسترا (Dijkstra) در سال ۱۹۶۵ ارائه گردید و به نام الگوریتم بانکدار (Banker's Algorithm) معروف شد.
  - ✓ یک بانکدار همیشه تمام سرمایه اش را به مشتریان نمی دهد و طوری عمل می کند که بتواند نیازهای کلیه مشتریان را برآورده کند.
- مواد لازم برای الگوریتم بانکدار:

۱. آرایه یک بعدی available (در دسترس) به طول m که تعداد منابع آزاد از هر نوع را نشان می دهد. مثلاً اگر available [j]=k باشد، یعنی از منبع R_j در حال حاضر k عدد آزاد داریم.

۲. آرایه دوبعدی (ماتریس) max (ماکزیمم) به ابعاد m×n که ماکزیمم نیاز هر فرآیند را مشخص می کند. مثلاً اگر max[i][j]=h باشد، یعنی فرآیند P_i حداکثر h عدد از منبع R_j را درخواست خواهد کرد

الگوریتم بانکدار

- ادامه مواد لازم برای الگوریتم بانکدار:

۳. آرایه دوبعدی (ماتریس) allocation (تخصیص یافته) به ابعاد m×n که تعداد منابعی که در حال حاضر از هر منبع به فرآیند داده شده را مشخص می کند. مثلاً اگر allocation[i][j]=d باشد، یعنی فرآیند P_i در حال حاضر d عدد از منبع R_j را در اختیار خود دارد.

۴. آرایه دوبعدی (ماتریس) need (نیاز) به ابعاد m×n که نیاز های فرآیند را در حال حاضر مشخص می کند. مثلاً اگر need[i][j]=g باشد، یعنی فرآیند P_i در حال حاضر g عدد از منبع R_j را برای تکمیل کارش نیاز دارد.

نکته! با توجه به موارد بالا کاملاً واضح است که:

تخصیص یافته – ماکزیمم = نیاز

need[i][j] = max [i][j] – allocation [i][j]

الگوریتم بانکدار

- الگوریتم بانکدار با مواد قبلی به شرح زیر عمل می کند:

۱) در ماتریس نیاز (need) دنبال سطری باشید که کوچکتر از منابع در دسترسی (available) باشد. اگر چنین سطری نیست، سیستم در حالت نا امن است و الگوریتم تمام می شود.

نکته! دو آرایه x و y را می گوییم $x \le y$ است اگر و فقط اگر $x[i] \le y[i]$ باشد به اِزای تمام مقادیر i خواهیم داشت $i = 1,2,\ldots,n$ مثلاً

$$(0,4,2,3) \le (2,8,3,5)$$

۲) اگر در ماتریس نیاز (need) سطری یافتید که کوچکتر از منابع در دسترسی (available) بود، یعنی می توانید منابع موجود را به فرآیند بدهید تا کارش به اتمام برسد و تمام منابع در دسترس خود را (allocation) را آزاد کند.

- اعداد مربوط به فرآیند مورد نظر در ماتریس allocation را به آرایه available اضافه کنید و این فرآیند را به عنوان خاتمه یافته علامت بزنید.

الگوریتم بانکدار

- ادامه شرح الگوریتم بانکدار:

۳) مراحل ۱ و ۲ را مرتب تکرار کنید تا اینکه

- یا تمام فرآیندها به صورت خاتمه یافته علامت بخورند که سیستم امن می باشد.
- یا در مرحله اول از الگوریتم خارج شوید که سیستم نا امن می باشد.

نکته! اگر در مرحله ۲ چند فرآیند با شرایط یکسان وجود داشت، ترتیب اجرای آنها اهمیت ندارد.

نکته! برای اجرای الگوریتم بانکدار شرایط زیر بایستی وجود داشته باشند:

a. حداکثر نیاز برنامه ها از نظر منابع از پیش تعیین شده باشد. (ایراد الگوریتم است)
b. برنامه ها مستقل از هم بوده و ترتیب اجرای آنها مهم نباشد.

نکته ! نام دیگر این روش، الگوریتم هابِرمَن (Habermann Algorithm) است.

مثال ۱: سیستمی دارای پنج فرآیند P_0 تا P_4 و سه منبع A و B و C است. سیستم دارای 10 عدد منبع A، دارای 5 عدد منبع B و دارای 7 عدد منبع C می باشد. در زمان t این سیستم در وضعیت زیر است. آیا سیستم در حالت امن است ؟

	Allocation			max		
	A	B	C	A	B	C
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3

مثال ۲: در مثال ۱ فرض کنید فرآیند P_1 در لحظه t ، از منبع A یک عدد دیگر و از منبع C دو عدد دیگر درخواست کند. در چنین وضعیتی آیا سیستم در حالت امن است ؟

وضعیت مثال ۱

	Allocation			max		
	A	B	C	A	B	C
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3

مثال ۳: در مثال ۱ فرض کنید فرآیند P_3 در لحظه t ، از منبع B دو عدد دیگر درخواست کند. در چنین وضعیتی آیا سیستم به حالت امن می رود ؟

وضعیت مثال ۱

	Allocation			max		
	A	B	C	A	B	C
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3

مثال ۴: در سیستم زیر فرض کنید فرآیند P_0 در لحظه t ، درخواست (A,B,C) = (0,2,0) کند. در چنین وضعیتی آیا باید به این درخواست پاسخ داده شود؟

Allocation

	A	B	C
P_0	0	1	0
P_1	3	0	2
P_2	3	0	2
P_3	2	1	1

need

A	B	C
7	4	3
0	2	0
6	0	0
0	1	1

available

A	B	C
2	3	0

آشکارسازی یا تشخیص بن بست

- اگر یک سیستم از الگوریتم های پیشگیری و اجتناب از بن بست استفاده نکند، ممکن است بن بست رخ دهد. در این وضعیت سیستم باید دو عمل زیر را انجام دهد:
  - ✓ وضعیت سیستم بررسی شود و تعیین کند بن بست رخ داده است یا نه ؟
  - ✓ اگر بن بست اتفاق افتاده، سیستم باید از بن بست خارج شود و در اصطلاح سیستم را بازیافت (Recover) کند.

- روش های تشخیص بن بست به دو دسته تقسیم می شوند:
  - ✓ حالت اول : وقتی از هر منبع یک عدد در سیستم باشد (به کمک گراف تخصیص منابع)
  - ✓ حالت دوم : وقتی از هر منبع چند عدد در سیستم باشد (به کمک الگوریتم بانکدار)

آشکار سازی یا تشخیص بن بست

- سوال مهم این است که الگوریتم تشخیص بن بست چه زمان هایی باید اجرا شود ؟
  - ✓ هر بار درخواست یک فرآیند اجرا نمی شود.
  - ✓ در بازه های زمانی ثابت، مثلاً هر یک ساعت یک بار
  - ✓ هر بار که بهره وری CPU کمتر از یک حد معین می رسد، مثلاً زیر ۴۰٪

- وقتی بن بست تشخیص داده شد، باید برطرف شود. روش های بازیافت:
  - ✓ سیستم عامل به کاربر اعلام کند بن بست رخ داده تا خود کاربر آن را اداره کند.
  - ✓ سیستم عامل آن را اداره کند:
    - خاتمه دادن به فرآیندها
    - پس گرفتن منابع

بازیافت یا برطرف کردن بن بست

- خاتمه دادن به فرآیند:
  - ✓ خاتمه دادن به فرآیند برای شکستن حلقه انتظار چرخشی استفاده می شود.
  - ✓ یا همه فرآیندهای بن بست یا یکی از آنها را خاتمه می دهد. معمولاً یکی را پایان می دهد.
  - ✓ انتخاب یک فرآیند از بین بقیه بر اساس اولویت، مدت زمان اجرا، مدت باقی مانده از اجرا، تعداد و نوع منابع در اختیار فرآیند و ... صورت می گیرد.
- پس گرفتن منابع:
  - ✓ منابع از یک فرآیند گرفته شده و به فرآیند دیگری داده می شود.
  - ✓ برای پس گرفتن سه مورد را باید در نظر گرفت:
    - انتخاب منبع و فرآیند های مورد نظر
    - فرآیندی که منبع از آن گرفته شده به حالت امنی باز گردد. (Rollback)
    - تضمین شود منابع همواره از یک فرآیند گرفته نشود. (عدم قحطی زدگی)

نادیده گرفتن بن بست

- ساده ترین روش برخورد با بن بست، الگوریتم شتر مرغ (Ostrich) است.
  - ✓ شتر مرغ وقتی می ترسد، سر خود را در شن فرو می کند و وانمود می کند ابداً هیچ مشکلی وجود ندارد.

- این روش نیز بن بست را نادیده گرفته و اجازه می دهد که اتفاق بیفتد. پس از آن مجبور به Reset کردن سیستم هستید.

- بهانه طراحان این روش اینست که ممکن است خیلی کم بن بست رخ دهد و ارزش ندارد قوانین محدود کننده کشف و رفع بن بست را رعایت کرد.

- سیستم عامل یونیکس (Unix) از این روش استفاده می کند.

شرط رخ ندادن بن بست

- اگر در یک سیستم n فرآیند و m منبع از یک نوع وجود داشته باشد و شرط زیر برقرار باشد، هیچگاه بن بست رخ نمی دهد.

$$\sum_{i=1}^{n} \mathrm{Re}\,quest[i] < m + n$$

- مثال: اگر در یک سیستم ۶ فرآیند وجود داشته باشد و هر فرآیند حداکثر ۳ تقاضا برای منبع داشته باشد، چه تعداد منبع مشابه احتیاج است تا بن بست رخ ندهد؟

$$\sum_{i=1}^{6} \mathrm{Re}\,quest = 6 \times 3 < m + 6 \quad \Rightarrow \quad 18 < m + 6 \quad \Rightarrow \quad m > 12$$

حداقل ۱۳ منبع نیاز است.

سؤال ؟