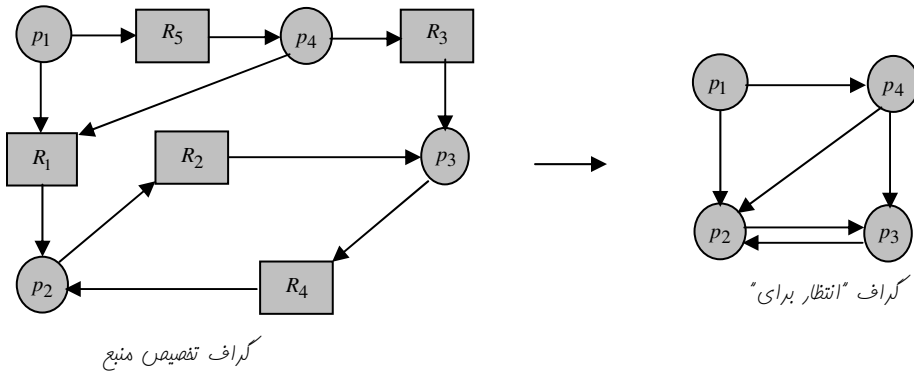


3- روش تشخیص بن بست و بازیافت سیستم :

تشخیص بن بست برای حالت یک نمونه از هر منبع

در این روش از روی کراف تفصیص منبع، کراف انتظار (wait-for graph) را بدست می آوریم. برای بدست آوردن کراف انتظار، کره های منبع را از کراف تفصیص حذف کرده و کمان های مناسبی را باهم ترکیب می کنیم. مثال. فرض کنید کراف تفصیص منابعی به شکل زیر باشد، اولاً کراف "انتظار برای" را رسم کنید ثانیاً مشخص نمائید که سیستم دچار بن بست شده است یا نه.



چون حلقه وجود دارد سیستم دچار بن بست شده است

کراف "انتظار برای"

یک پیکان از پردازش p_i به p_j در کراف انتظار، نمایانگر این است که پردازش p_i در انتظار پردازش p_j است تا منبعی را که p_i به آن نیاز دارد، آزاد کند. سیستم عامل، کراف انتظار را نگهداری کرده و مرتباً آن را بررسی می کند. اگر در کراف انتظار حلقه وجود داشته باشد، آنگاه سیستم به بن بست رسیده است. الگوریتم تشخیص بن بست در کراف از مرتبه $O(n^2)$ می باشد که n تعداد رئوس کراف است.

تشخیص بن بست برای حالت چند نمونه از هر منبع

روش کراف انتظار برای این حالت قابل استفاده نیست. در حالتی که هر منبع چند نمونه دارد می بایست از روشی شبیه بانگداران استفاده کنیم. در این روش به جای ماتریس Need از ماتریس $n \times m$ به نام Request استفاده می شود که نیازهای فعلی هر پردازش را نشان می دهد. اگر $Request[i][j] = k$ باشد یعنی پردازش p_i ، k نمونه بیشتر از منبع R_j را درخواست کرده است. الگوریتم این روش به شکل زیر خواهد بود.

- 1- ابتدا بردار $work$ را مساوی Available قرار می دهیم (work = Available) اگر به ازای $i = 0 \dots n$ ، $Allocation(i) \neq 0$ ، $finish[i] = fals$ و $finish[i] = True$ و کمره $finish[i] = True$
- 2- به ازای $i = 0 \dots n$ ، یک i چنان پیدا کنیم که $finish[i] = fals$ باشد و $Request[i] > work$ (سطر مربوط به پردازش p_i در ماتریس Request) باشد. اگر چنین i پیدا نشد برو به مرحله 4.

3- به ازای i پیدا شده در مرحله دو $finish[i]$ را برابر True قرار می دهیم و $work$ را با Allocation مربوطه جمع می کنیم ($work += Allocation$)، زیرا پردازشهای پیدا شده که با منابع موجود میتواند اجرایش را کامل کند و پس از فاصله اجراء منابع در اختیارش را آزاد سازد، دوباره برو به مرحله 2.

4- اگر پردازش p_i یافت شود که $finish[i] = false$ باشد، بدین معناست که p_i در بن بست قرار دارد. (سیستم دچار بن بست شده است)

□ پیچیدگی این الگوریتم $(m \times n)$ می باشد که m تعداد منابع است و n تعداد پردازشها می باشد.

مثال. سیستمی با 5 پردازش $p_1 \dots p_5$ و سه نوع منبع A, B, C را در نظر می گیریم. A دارای 7 نمونه، منبع نوع B دارای 2 نمونه و C دارای 6 نمونه است. فرض کنید در زمان t_0 وضعیت سیستم به صورت زیر باشد آیا سیستم در بن بست قرار دارد یا خیر؟
ب. فرض کنید p_3 نمونه دیگری از منبع C را درخواست کند آیا سیستم دچار بن بست می شود یا خیر؟

وضعیت اولیه سیستم

Process	A	B	C
p_1	0	1	0
p_2	2	0	0
p_3	3	0	3
p_4	2	1	1
p_5	0	0	2

Allocation

Process	A	B	C
p_1	0	0	0
p_2	2	0	2
p_3	0	0	0
p_4	1	0	0
p_5	0	0	2

Request

حل: بردار Available برابر خواهد بود با: (0,0,0)، پس داریم

Work	A	B	C
	0	0	0

finish	p_1	p_2	p_3	p_4	p_5
	f	f	f	f	f

Work	A	B	C
	0	1	0

finish	p_1	p_2	p_3	p_4	p_5
	T	f	f	f	f

Work	A	B	C
	3	1	3

finish	p_1	p_2	p_3	p_4	p_5
	T	f	T	f	f

Work	A	B	C
	5	1	3

finish	p_1	p_2	p_3	p_4	p_5
	T	T	T	f	f

Work	A	B	C
	7	2	4

finish	p_1	p_2	p_3	p_4	p_5
	T	T	T	T	f

Work	A	B	C
	7	2	6

finish	p_1	p_2	p_3	p_4	p_5
	T	T	T	T	T

پس توالی $\langle p_1, p_3, p_2, p_4, p_5 \rangle$ امکان پذیر است و سیستم در بن بست قرار ندارد.

ب. در این صورت ماتریس Request به شکل زیر می باشد. هر که می تواند منابع p_1 را باز پس گرفت، ولی این تعداد منابع موجود برای انجام تقاضاهای هیچ پردازشی کافی نخواهد بود. لذا بن بستی شامل پردازش های p_2, p_3, p_4, p_5 پدید می آید.

Process	A	B	C
p_1	0	0	0
p_2	2	0	2
p_3	0	0	①
p_4	1	0	0
p_5	0	0	2

بن بست رخ می دهد $(0,0,0) \xrightarrow{p_1} (0,1,0)$ منابع

زمان فراخوانی الگوریتم تشفیص بن بست:

سؤال مهم این است که الگوریتم تشفیص بن بست را چه زمان هائی باید اجرا کنیم. در یک حالت حد، میتوان در هر بار درخواستی که سریعاً قابل اعطاء نمی باشد، این الگوریتم را اجرا کنیم. بدین ترتیب علاوه بر اینکه می توانیم مجموعه پردازش های موجود در بن بست را تشفیص دهیم، پردازش فاضی که باعث بن بست شده است نیز مشفیص می شود. ولی از طرف دیگر با توجه به زمانگیر بودن الگوریتم های تشفیص بن بست، اجرای مکرر آنها باعث کاهش کارائی سیستم می گردد. یک روش کم هزینه تر آن است که الگوریتم مذکور را با پریود کمتری اجرا کنیم. مثلاً در هر ساعت یکبار، یا هر بار که بهره وری CPU به زیر 40% برسد، یا هر بار که سیستم کم است. اگر این الگوریتم در زمانهای دلفواهی اجرا شود، ممکن است حلقه های بسیاری در گراف منبع وجود داشته باشند و بدین ترتیب در حالت کلی نمی توان گفت کدام پردازش باعث بن بست شده است.

ترمیم یا بازیافت سیستم:

یک روش برای ترمیم بن بست این است که بعد از اینکه بن بست تشفیص داده شد، به اپراتور خبر داده شود که سیستم دچار بن بست شده است که در این صورت اپراتور به صورت دستی سیستم را می بایست ترمیم کند. ولی اگر خود سیستم عامل بخواهد آن را باز یافت کند دو راه وجود دارد.

1- فائمه دادن به پردازش ها:

□ تمام پردازش های درگیر بن بست فائمه داده شوند، که این راه حل هر چند سربار ناشی از اجرای متعدد الگوریتم های تشفیص بن بست را ندارد ولی ممکن است سیستم متمم هزینه زیادی شود، چرا که ممکن است پردازش هائی دستورات زیادی را اجرا کرده باشند که بدین ترتیب می بایست دوباره از ابتدا اجرا شوند.

□ روش دوم این است که هر بار یک پردازش انتقاب شده و فائمه داده میشود، سپس الگوریتم کشف بن بست فراخوانی می شود. اگر سیستم دچار بن بست باشد، مجدداً پردازش دیگری انتقاب شده و فائمه داده می شود، تا جائی که سیستم از بن بست خارج شود. اشکال این روش سربار زمانی ناشی از الگوریتم کشف بن بست می باشد. انتقاب یک پردازش جهت فائمه دادن می بایست بر اساس هزینه کمینه باشد، که هزینه کمینه می تواند به فاکتور های زیر بستگی داشته باشد.

1- نوع پردازش (دسته ای یا همواره ای)

2- مدت زمانی که از اجرای پردازش سپری شده است

3- مدت زمانی که اجرای پردازش باقی مانده است

4- تعداد و نوع منابعی که در اختیار پردازش است

5- در صد استفاده پردازش از منابع

6- تعداد منابعی که برای کامل شدن نیاز دارد

2- پس گرفتن منابع:

در این روش منابعی از یک پردازش گرفته شده و در اختیار پردازش دیگری قرار داده می شود. برای این کار باید سه موضوع مشفیص گردد.

الف. انتقاب منبع و پردازش های مورد نظر

ب. بازگرداندن به عقب (Rollback) یعنی پردازشی که منبع او پس گرفته شده، باید به حالت امنی به عقب برگردانده شود، تا بعداً از آن حالت مجدداً اجرائش را از سر بگیرد.

ج. قطعی زدگی یعنی باید تضمین کرد که منابع همواره از یک پردازش فاض بازپس گرفته نشود، چرا که در آن صورت اجرای آن پردازش مرتباً به تعویق می افتد

4- **روش صرف نظر کردن از بن بست (الگوریتم استریخ ostrich) :** در این روش در واقع هیچ عملی در مقابل بن بست انجام داده نمی شود. در صورتی که بن بست منجر به از کار افتادن سیستم شود (Hang) آنگاه سیستم به صورت دستی ری ست (reset) می شود.

□ چالب است که بدانید در اکثر سیستم عامل های امروزی مثل unix از همین روش چهارم استفاده می شود، چرا که در این سیستم ها بن بست به ندرت رخ می دهد (مثلا سالی یک بار) لذا ارزانه تر آن است که به جای روش های پر هزینه پیشگیری، اجتناب و آشکار سازی کلا از این مشکل چشم پوشی کنیم.

ترکیب روش ها در اداره بن بست

- در عمل هر یک از روش های اداره بن بست (پیشگیری، اجتناب، کشف) به تنهایی برای تمام انواع منابع مناسب نمی باشد، یک شیوه ترکیبی برای دسته های مختلف منابع مثلا می تواند به صورت زیر باشد.
- منابع داخلی سیستم مثل بلوک کنترل پردازش؛ پیشگیری از طریق ترتیب منابع
 - منابع کار (گرداننده های دیسک، نوار، چاپگر و...) : اجتناب از بن بست، چون حداکثر نیاز از قبل مشخص است
 - حافظه اصلی؛ پیش گیری از طریق پس دادن می تواند انجام پذیرد، چرا که به راحتی می توان حافظه اصلی را از پردازش ها پس گرفت؛ زیرا به ممض کمبود حافظه یکسری از پردازش ها به حافظه جانبی منتقل می شوند.
 - حافظه جابه جاپذیر (پشتیبان)؛ تفصیص از پیش، میتواند انجام پذیرد چرا که حداکثر نیاز های ذخیره سازی از قبل می تواند مشخص باشد.
 - منابع پردازش؛ از طریق اجتناب