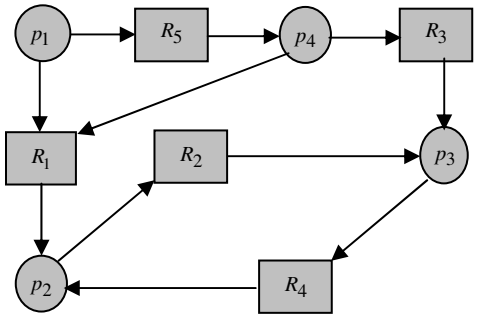
جلسه هشتم

3- روش تشخیص بن بست و بازیافت سیستم :

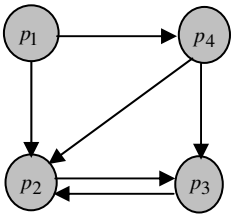
تشخیص بن بست برای حالت یک نمونه از هر منبع

در این روش از روی گراف تخصیص منبع، گراف انتظار($wait-for\ graph$) را بدست می آوریم. برای بدست آوردن گراف انتظار، گره های منبع را از گراف تخصیص حذف کرده و کمان های مناسبی را باهم ترکیب می کنیم.

مثال. فرض کنید گراف تخصیص منابعی به شکل زیر باشد، اولا گراف "انتظار برای" را رسم کنید ثانیا مشخص نمائید که سیستم دچار بن بست شده است یا نه.



گراف تخصیص منبع



گراف "انتظار برای"

چون حلقه وجود دارد سیستم دچار بن بست شده است

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

تشخیص بن بست برای حالت چند نمونه از هر منبع

روش گراف انتظار برای این حالت قابل استفاده نیست. در حالتی که هر منبع چند نمونه دارد می بایست از روشی شبیه بانکداران استفاده کنیم. در این روش به جای ماتریس $Need$ از ماتریس $n\times m$ به نام $Request$ استفاده می شود که نیاز های 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 میدهیم($work=Available$)اگر به ازای $i=0\cdots n$ ، $Allocation\ (i)\neq 0$ باشد پس $finish[i]=fals$ و گرنه $finish[i]=True$

2- به ازای $i=0\cdots n$ ، یک i چنان پیدا کنید که $finish\ [i]=fals$ باشد و $Request[i]>work$ (سطر مربوط به پردازه 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 ($work+=Alocation$)، زیرا پردازهای پیدا شده که با منابع موجود میتواند اجرایش را کامل کند و پس از خاتمه اجرا، منابع در اختیارش را آزاد سازد، دوباره برو به مرحله 2 .

4- اگر پردازه 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 پردازه ها می باشد.

مثال. سیستمی با 5 پردازه $p_1\cdots p_5$ 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 شودیا خیر؟

وضعیت اولیه سیستم

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

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

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

	A	B	C
Work	0	0	0

	p_1	p_2	p_3	p_4	p_5
finish	*f*	*f*	*f*	*f*	*f*

	A	B	C
Work	0	1	0

	p_1	p_2	p_3	p_4	p_5
finish	*T*	*f*	*f*	*f*	*f*

	A	B	C
Work	3	1	3

	p_1	p_2	p_3	p_4	p_5
finish	*T*	*f*	*T*	*f*	*f*

	A	B	C
Work	5	1	3

	p_1	p_2	p_3	p_4	p_5
finish	*T*	*T*	*T*	*f*	*f*

	A	B	C
Work	7	2	4

	p_1	p_2	p_3	p_4	p_5
finish	*T*	*T*	*T*	*T*	*f*

	A	B	C
Work	7	2	6

	p_1	p_2	p_3	p_4	p_5
finish	*T*	*T*	*T*	*T*	*T*

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

ب. در این صورت ماتریس **Request** به شکل زیر می باشد. هر که می توان 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 منابع $(0,0,0) \xrightarrow{p_1} (0,1,0)$

زمان فراخوانی الگوریتم تشخیص بن بست:

سئوال مهم این است که الگوریتم تشخیص بن بست را چه زمان هائی باید اجرا کنیم.

در یک حالت حد، میتوان در هر بار درخواستی که سریعا قابل اعطاء نمی باشد، این الگوریتم را اجرا کنیم. بدین ترتیب علاوه بر اینکه می توانیم مجموعه پردازش های موجود در بن بست را تشخیص دهیم، پردازش خاصی که باعث بن بست شده است نیز مشخص می شود. ولی از طرف دیگر با توجه به زمانگیر بودن الگوریتم های تشخیص 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. مثلا در هر ساعت یکبار، یا هر بار که بهره وری CPU به زیر 40% برسد، یا هر بار که بار سیستم کم است. اگر این الگوریتم در زمانهای دلخواهی 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 تشخیص 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- خاتمه دادن به پردازش ها:

☐تمام پردازه های درگیر بن بست خاتمه داده شوند، که این راه حل هر چند سربار ناشی از اجرای متعدد الگوریتم های تشخیص بن بست را ندارد ولی ممکن است سیستم متحمل هزینه زیادی شود، چرا که ممکن است پردازه 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 پردازه انتخاب شده و خاتمه داده میشود، سپس الگوریتم کشف بن بست فراخوانی می شود. اگر سیستم دچار بن بست باشد، مجددا پردازه دیگری انتخاب شده و خاتمه 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. انتخاب یک پردازه جهت خاتمه دادن می بایست بر اساس هزینه کمینه باشد، که هزینه کمینه می تواند به فاکتور های زیر بستگی داشته باشد.

1- نوع پردازه(دسته ای یا محاوره ای)

2- مدت زمانی که از اجرای پردازه سپری شده است

3- مدت زمانی که اجرای پردازه باقی مانده است

4- تعداد و نوع منابعی که در اختیار پردازه است

5- درصد استفاده پردازه 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 مشخص گردد.

الف. انتخاب منبع و پردازش های مورد نظر

ب. باز گرداندن به عقب(Rollback) یعنی پردازشی که منبع او پس گرفته شده، باید به حالت امنی به عقب برگردانده شود، تا بعدا از آن حالت مجددا اجرایش را از سر بگیرد.

ج.قحطی زدگی یعنی باید تضمین کرد که منابع همواره از یک پردزاش خاص بازپس گرفته نشود،چرا که در آن صورت اجرای آن پردازش مرتبا 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.

□ جالب 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 ارزانتر 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 تنهائی 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 پردازه ها پس گرفت؛ زیرا به محض کمبود حافظه یکسری از پردزاه ها به حافظه جانبی منتقل می شوند.
- حافظه جابه جاپذیر (پشتیبان): تخصیص از پیش، میتواند انجام پذیرد چرا که حداکثر نیاز های ذخیره سازی از قبل می تواند مشخص باشد.
- منابع پردازش: از طریق اجتناب

پایان جلسه هشتم