HW5, OS

Dr Zali

Dey, 1403

Sepehr Ebadi

9933243

۱)

الف)

Mutual exclusion : یعنی زمانی که یک ریسورس مشترک را در هرلحظه از زمان فقط به یه پروسس بتونیم بدیم.

Hold and wait : در شرایطی که پروسسی یک ریسورسی را گرفته باشد و در همین حال درخواست زده باشد برای گرفتن ریسورس دیگری.

No preemption : ما نتونیم ریسورسی را از پروسسی بگیریم و خود پروسس با اختیارش خود ش فقط بتواند ریسورس را ازاد کند.

circular wait : اگر n تا پروسس داشته باشیم انگاه پروسس ۱ منتظر ریسورسی باشد که دست پروسس ۲ است و پروسس ۲ منتظر ریسورسی باشد که دست پروسس ۳ است و ...

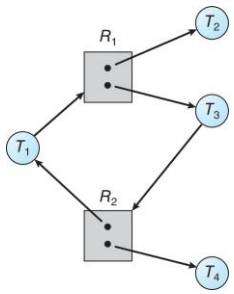
ب)

در شکل زیر با اینکه دور داریم ولی بن بست رخ نمیدهد زیرا:

چون ترد ۴ به ریسورس دیگری درخواست نزده پس هر وقت تمام شود ترد۴ ازاد میکند ریسورس ۲ را پس الان میتونیم ریسورس ۲ را به ترد ۳ بدیم.

و از انطرف الان ترد ۳ ریورس های ۱ و ۲ را دارد و روی ریسورس دیگری هم درخواست نزده پس هروفت تمام شود این دو ریسورس را ازاد میکند و الان میتونیم ریسورس ۱ را به ترد ۱ بدیم.

پس نمیتونیم بگیم شرط تشخیص بن بست داشتن دور در گراف است.



ج)

در بحث مقابله با بن بست روش هایی داریم که

روش اول این است که سیستم را جوری طراحی بکنیم که اصلا هیچ وقت وارد ددلاک نمی شویم. که خودش به دو روش انجام میشود.

پیشگیری :اون فرضیات را اجتناب میکنیم. ۴ شرطی که در قسمت الف توضیح داده شد.

اجتناب : وقتی پروسس ها درخواست ریسورس میکنند به نحوی به انها پاسخ بدیم که دچار ددلاک نشوند.

الگوریتم بنکر جزو الگوریتم های اجتناب از بن بست است زیرا در این الگوریتم :

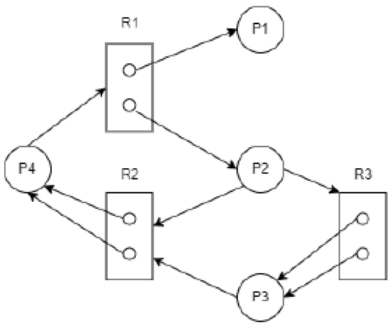
در اینجا اطلاعات اضافه ای که باید بدونیم اینه که هر پروسسی به چند تا از اینستنس های ریسورهاش حداکثر نیاز داره.

حال اگر پروسسی درخواستی روی ریسورسی بده سیستم باید چک کند اگر پاسخ بده ایا سیستم به استیت safe میره یا unsafe.

پس در واقع در موقع اختصاص منابع داریم چکی انجام میدیم که پس میشود جزو دسته اجتناب ها.

۲)

در شکل داشتن حلقه واضح است که وجود دارد در شکل دوم حلقه ها مشخص شده .



اما برای بن بست داشتن میایم و بررسی میکنیم :

p1 ، r1 را دارد اما بعدش درخواستی به ریسورسی نزده پس وقتی کارش تمام شد

نمونه ای از r1 را ازاد میکند و اکنون p4 میتونه r1 را بگیره.

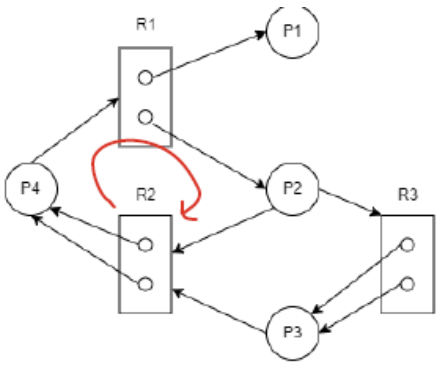
حال p4 بهش سه تا ریسورس های r1 و دو نمونه از r2 بهش اختصاص داده شده

و درخواست دیگری ندارد پس وقتی کارش تمام شد این سه تا را ازاد میکند که در

این صورت p2 میتونه یه نمونه از r2 را بگیره و p3 هم میتونه نمونه دیگه ای از r2 را بگیره.

حال p3 دو نمونه از r3 و یک نمونه از r3 را گرفته و درخواست دیگری ندارد پس زمانی که کارش تمام شد این سه ریسورس را ازاد میکند و اکنون p2 میتونه r3 را بگیرد.

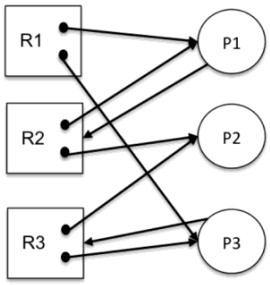
پس بن بست نداریم.



۳)

الف)

خیر نداریم. زیرا p1 ، r1 و r2 را دارد و روی r2 دوباره درخواست زده پس باید منتظر بمونه تا r2 ازاد بشه. در حالیکه p2 ، r2,r3 را دارد و درخواست دیگری هم نزده پس وقتی کارش تمام شد این ریسورس ها را ازاد میکند. حال p1 میتونه r2 را دوباره بگیرد.



p3،r1,r3 را دارد و دوباره روی r3 درخواست زده که وقتی p2 کارش تمام شد این ریسورس را ازاد کرد و الان ازاده و میتونه p3 این را بگیرد. پس بن بست نداریم.

ب)

در این صورت بله بن بست داریم زیرا : خود r1 به p1,p3 اختصاص یافته اکنون و از طرفی هم p3 فقط روی r3 درخواست زده و همینطور از طرفی r3 الان به p2,p3 اختصاص یافته پس همه پروسس ها گرو کشی کردند و پروسسی نیست که بتونه کارش راتمام بکنه تا ریسورس هایش را ازاد کنه.

ج)

خیر بن بست نداریم زیرا : p1 الان همه درخواست هایش را گرفته بود و درخواستی دیگر نداشت و بعد از اتمام کارش ریسورس هایش را ازاد میکرد در اینصورت p2 که روی r1 درخواست زده بود میتونه بگیره این ریسورس را و بقیه موارد هم شبیه قسمت الف حل میشد.

د)

خیر بن بست نداریم زیرا : شبیه به همان قسمت الف منابع ازاد میشدن به ترتیب : وقتی p2 ،r2,r3 را ازاد میکند p1 ،r2 را میگیره و کارش تمام که شد r1,r2 را ازاد میکند اکنون p3 میتونه r3 را بگیره و کارش که تمام شد دوباره r1,r3 را ازاد میکنه و حالا p4 میتونه r1 را بگیره. پس بن بست نداریم.

۴)

الف)

ماتریس مکس – ماتریس الوکیشن = ماتریس نید

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	1	1	3	2	1	1	1	2	0	0	6	4	4	2
P_1	1	1	0	0	1	2	0	2	0	1	0	2				
P_2	1	0	1	0	3	2	1	0	2	2	0	0				
P_3	0	1	0	1	2	1	0	1	2	0	0	0				

ب)

بله زیرا مثلا اگر به ترتیب به پروسس های ۰ تا ۳ پاسخ بدیم همه انها پایان می یابد اجراشون و ریسورس ها به اندازه میباشد در هر مرحله و شکل زیر را برای این تریتب اجرا داریم :

p0 p1 p2 p3

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	1	1	3	2	1	1	1	2	0	0	6	4	4	2
P_1	1	1	0	0	1	2	0	2	0	1	0	2	7	6	4	2
P_2	1	0	1	0	3	2	1	0	2	2	0	0	7	7	4	4
P_3	0	1	0	1	2	1	0	1	2	0	0	0	9	9	4	4
													11	9	4	4

ج)

بله میتوان زیرا با ترتیب اجرای مثلا پروسس های ۰ تا ۳ همه پروسس ها پاسخ داده میشه و ریسورس هایشون را ازاد میکنند.

شکل زیر را برای این ترتیب اجرا خواهیم داشت :

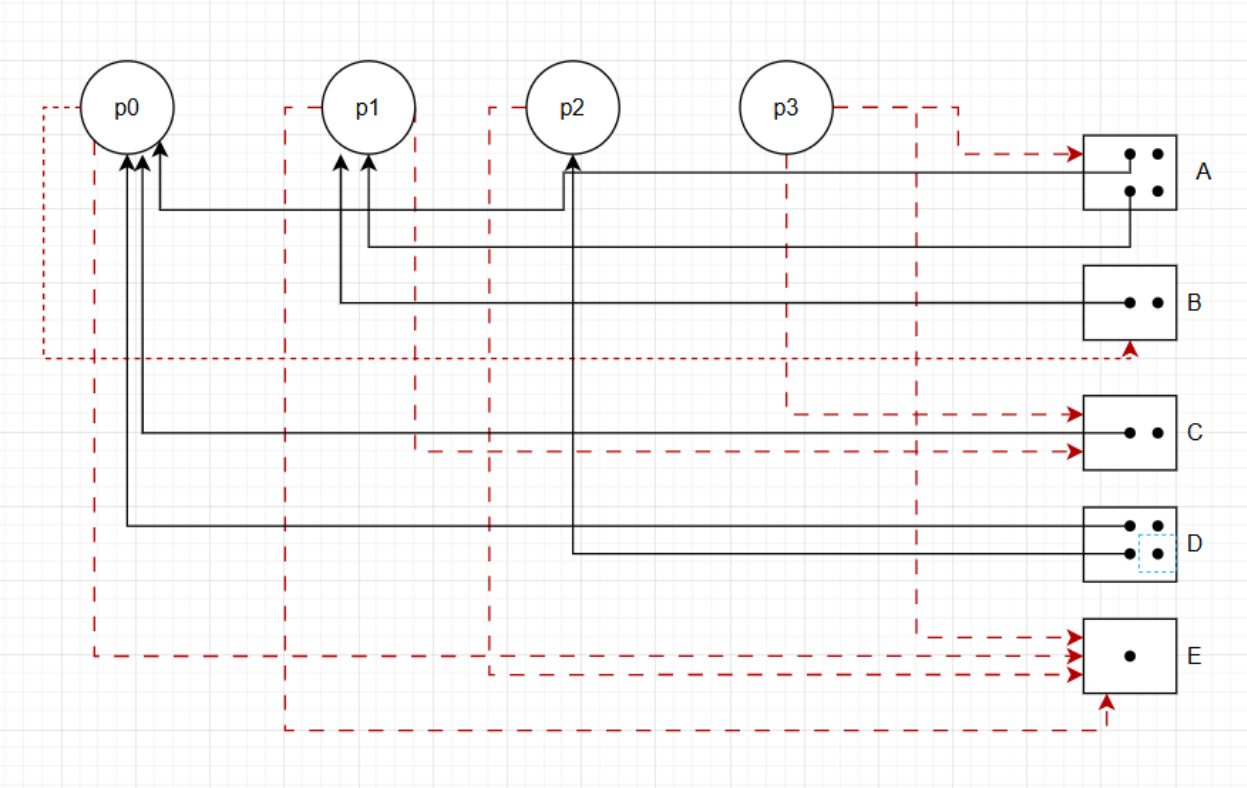
در ابتدا باید ماتریس available اپدیت بشه مقدارش با کم کردن مقدار اولیش از پروسسی که میخواهیم بهش پاسخ بدیم.

p0 p1 p2 p3

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	1	1	3	2	1	1	1	2	0	0	**4**	**3**	**4**	**2**
P_1	1	1	0	0	1	2	0	2	0	1	0	2	5	5	4	2
P_2	1	0	1	0	3	2	1	0	2	2	0	0	5	6	4	4
P_3	0	1	0	1	2	1	0	1	2	0	0	0	7	8	4	4
													9	8	4	4

۵)

الف)



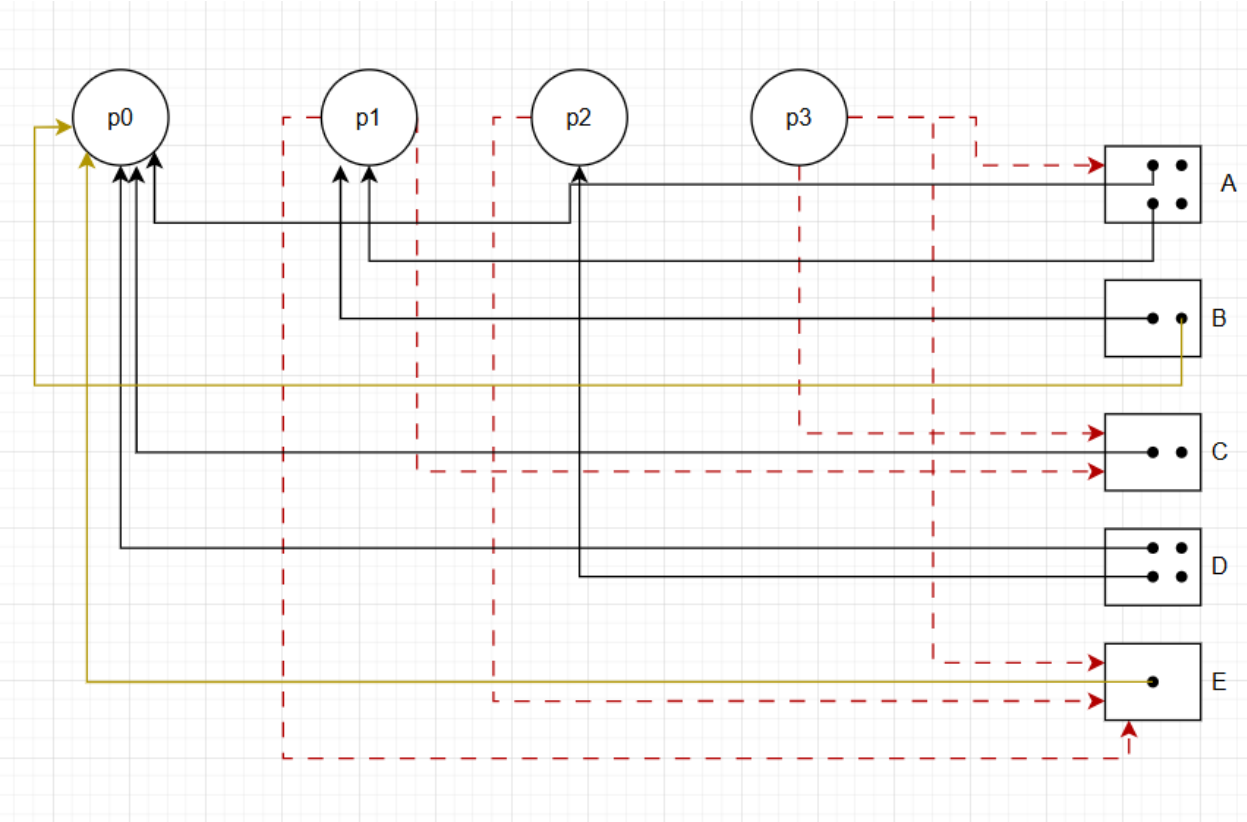
ب)

خیر بن بست نداریم زیرا همه درخواست ها پاسخ داده میشه و مشکلی برای تخصیص منابع نداریم در هر مرحله. در شکل زیر ترتیب اجرای پروسس های از ۰ تا ۳ را می بینید.

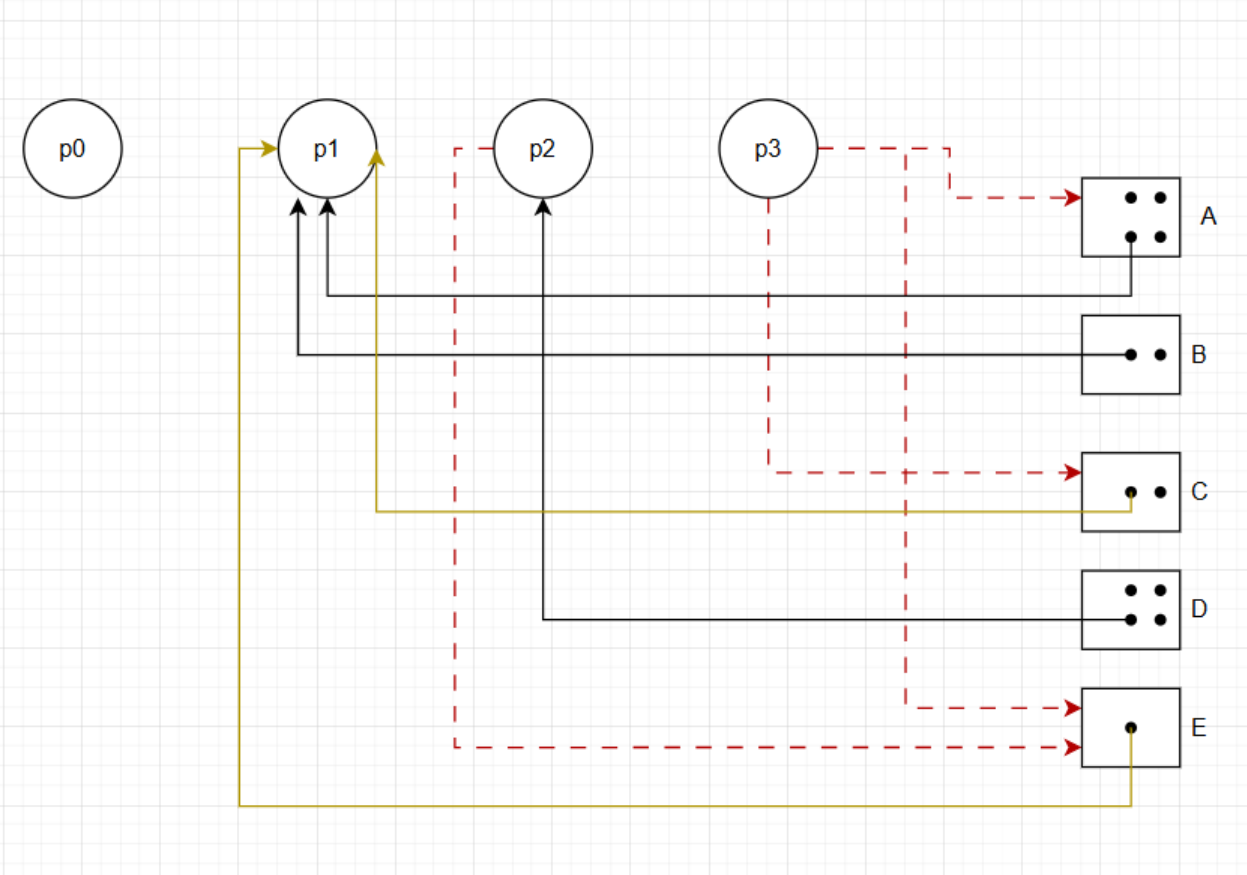
	Allocation					Request					Available				
	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
P_0	1	0	1	1	0	0	1	0	0	1	2	1	1	2	1
P_1	1	1	0	0	0	0	0	1	0	1	3	1	2	3	1
P_2	0	0	0	1	0	0	0	0	0	1	4	2	2	3	1
P_3	0	0	0	0	0	1	0	1	0	1	4	2	2	4	1
											4	2	2	4	1

P0 P1 P2 P3

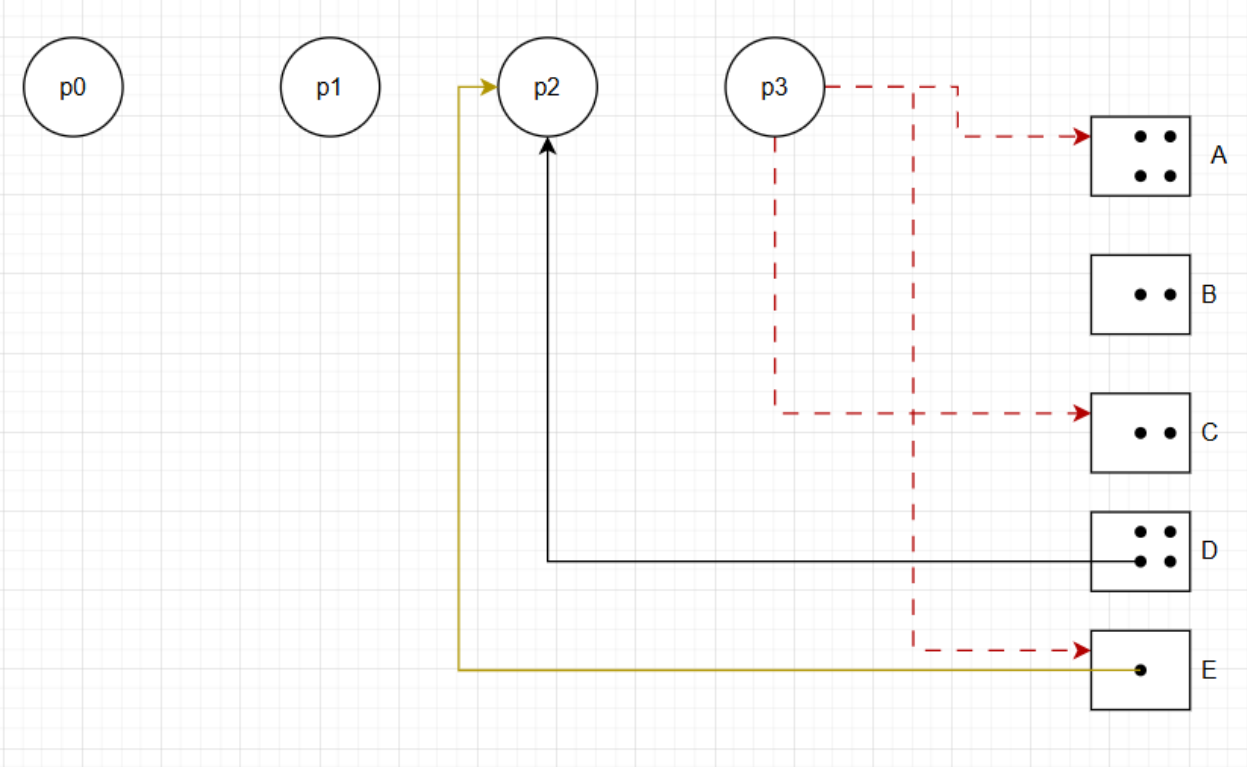
در مرحله ای که اول به P0 پاسخ دادیم : با رنگ زرد اختصاص یافته های جدید مشخص شده که خط چین های قبلی برداشته شده و جهت فلش ها هم عوض شده. و دیگر خط چین هم نیست و به صورت خط ممتد کشیده شده .



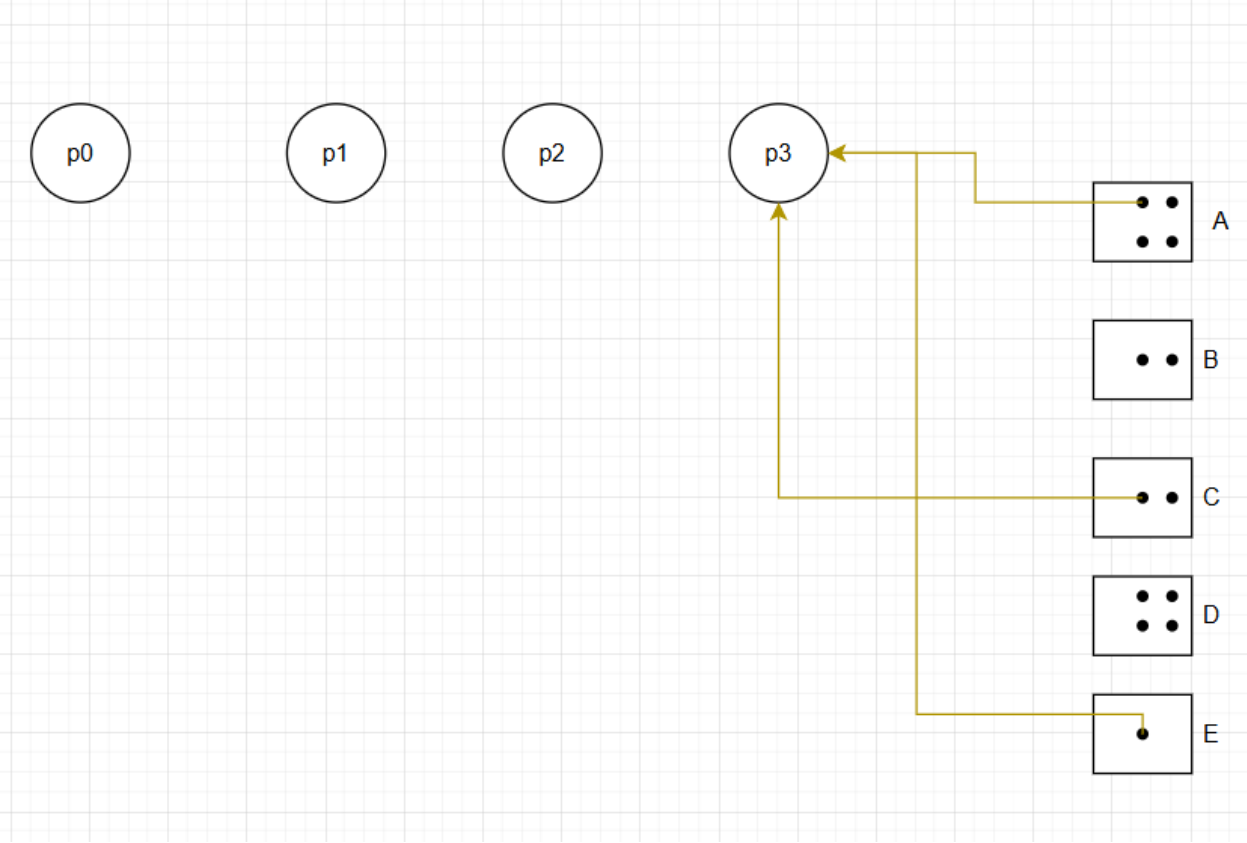
سپس وقتی P0 تمام شد ریسورس هایش را ازاد میکند و اکنون به P1 میتونیم پاسخ بدیم :



حال که P1 تمام شد ریسورس هایش را ازاد میکند و به P2 میتونیم پاسخ بدیم :



P2 هم تمام میشود و به P3 میتونیم پاسخ بدیم :


p0
p1
p2
p3
A
B
C
D
E

۶)

الف)

INVALID : زیرا طول سگمنت ۱ ، ۱۴ تا است در صورتی که به خط ۲۰ ام اشاره کرده ادرس درحالیکه اصلا ندارد سگمنت ۱ خط ۲۰ام. افست اشتباه

ب)

۱۳۲۷ + ۴۵۰ = ۱۷۷۷

ج)

INVALID : زیرا شرط اول که شماره سگمنت کوچکتر از تعداد سگمنت های پروسس باشد را ندارد.یعنی به سگمنتی اشاره میکند که اصلا همچین سگمنتی نداریم.

۷)

تعدادی سطر هایی که جدول میتونه داشته باشه :

$$\frac{1\ KB}{4\ B} = 256$$

$$2^{10} = 1\ KB \rightarrow$$

برای ۳۴ بیت ۱۰ بیتش را داریم پس ۲۴ بیت دیگر برای ادرس دهی پیج ها نیاز داریم.

تعداد جدول هایی که نیاز داریم :

$$\frac{2^{24}}{256} = 2^{16}$$

چون هر جدول باید رد یک پیج جا شود : تعداد سطح برابر :

$$\log_{256}(2^{24}) = 2$$

پس به ۲ سطح نیاز داریم.

۸)

الف)

$$8\,KB \rightarrow 13\,bits$$
$$1\,MB \rightarrow 20\,bits$$
$$4\,MB \rightarrow 22\,bits$$
$$\frac{2^{20}}{2^{13}} = 2^{7}$$ **سطر**

ب)

$$18002H = (98306) = (0010\ 0000\ 0000\ 1000\ 0001)$$

آدرس منطقی شامل 7بیت شماره صفحه و 13بیت آفست صفحه است. پس آن را به این دو بخش تقسیم می کنیم:

۷ بیت اول : شماره صفحه = ۰۰۰۱۱۰۰

۱۳ بیت اخر : افست صفحه = ۰۰۰۰۰۰۰۰۰۰۰۱۰

صفحه –> ۰۰۰۱۱۰۰ = ۱۲

طبق جدول –> ۱۱۰

$$2^{13} * 110 = 901120$$
$$901120 + 2 = 901122$$
$$901122 = DB002H$$

ج)

برای اینکه بتونیم بهینه تر استفاده بکنیم و هدر رفت حافظه کمتری داشته باشیم در اثر internal fragmentation باید اندازه پیج ها کوچک باشند درحالیکه برای دسترسی سریع تر به حافظه و برای کارهای محاسباتی سنگین با سرعت بالاتر باید اندازه پیج ها بزرگ باشد و اجرای برنامه های سنگین و بزرگ ، برای وقتی مموری به اندازه بزرگی داریم ، در hpc ها که نیاز به پردازش داده های سنگین و با سرعت اند.

مزایا و معایب افزایش سایز پیج ها :

مزایا :

اندازه پیج تیبل کوچک تر میشه.

کاهش تعداد دسترسی های به حافظه

افزایش کارایی هنگام انتقال داده هنگام سوپینگ

ساده تر شدن مدیریت حافظه

کاهش خطای صفحه

معایب :

هدر رفت حافظه

کاهش بهرهوری برای پروسس های کوچک

افزایش سربار انتقال داده :اگر تنها بخش کوچکی از یک پیج بزرگ موردنیاز باشد، باز هم کل پیج باید منتقل شود که می‌تواند منابع سیستم را هدر دهد.

درجه مالتی پروگرمینگ سیستم را پایین اورده ایم. و هعی نیاز به سوپینگ ممکنه داشته باشیم.

اگر تعداد پروسس ها کم باشد ولی هر پروسسی نیاز باشد سریع دسترسی به حافظه داشته باشد از پیج سایز بزرگ اشتفاده میکنیم.

۹)

TLB Miss & No Page Fault

زمانی رخ میدهد که : ورودی مربوط به آدرس مورد نظر در TLB وجود ندارد و صفحه مورد نظر در حافظه فیزیکی (RAM) موجود است.

وقتی TLB نمی‌تواند آدرس منطقی را ترجمه کند، سیستم باید به جدول صفحه (Page Table) مراجعه کند. اگر صفحه در حافظه باشد، ورودی TLB به‌روزرسانی می‌شود و عملیات ادامه می‌یابد.

TLB Miss & Page Fault

زمانی که : ورودی مربوط به آدرس مورد نظر در TLB وجود ندارد و صفحه مورد نظر در حافظه فیزیکی موجود نیست و نیاز به بارگذاری آن از حافظه ثانویه (مانند دیسک) است.

وقتی TLB ورودی لازم را ندارد، سیستم به جدول صفحه مراجعه می‌کند. اگر صفحه در حافظه فیزیکی نباشد، Page Faultرخ می‌دهد و سیستم باید صفحه را از حافظه ثانویه بارگذاری کند. پس از بارگذاری، جدول صفحه و TLB به‌روزرسانی می‌شوند.

TLB Hit & No Page Fault

زمانی که : ورودی مربوط به آدرس مورد نظر در TLB وجود دارد و صفحه مورد نظر در حافظه فیزیکی موجود است.

TLB Hit & Page Fault

این حالت ممکن نیست.

اگر TLB بتواند آدرس منطقی را ترجمه کند(TLB Hit) ، این بدان معناست که صفحه مورد نظر در حافظه فیزیکی موجود است.
بنابراین Page Fault نمی‌تواند رخ دهد. اگر صفحه در حافظه فیزیکی نباشد، TLBنمی‌تواند ورودی مناسب داشته باشد و باید TLB Missرخ دهد.

۱۰)

الف)

FIFO

Reference Stream	A	B	C	D	A	B	E	A	B	C	D	E	B	A	B
Oldest page *Newest page*	A	A B	A B C	D B C	D A C	D A B	E A B	E A B	E A B	E C B	E C D	E C D	B C D	B A D	B A D
Page fault	√	√	√	√	√	√	√			√	√	√	√	√	

ب)

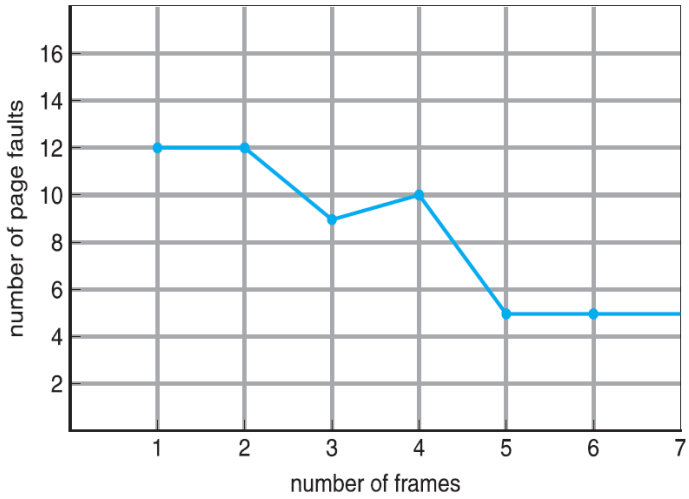
LRU

Reference Stream	A	B	C	D	A	B	E	A	B	C	D	E	B	A	B
Oldest page *Newest page*	A	A B	A B C	D B C	D A C	D A B	E A B	E A B	E A B	C A B	C D B	C D E	B D E	B A E	B A E
Page fault	√	√	√	√	√	√	√			√	√	√	√	√	

د)

انتظار داریم هر چه تعداد فریم ها را زیاد کنیم تعداد پیج فالت ها کم بشه ولی از یه جایی برعکس میشه و اینگونه نخواهد بود.

و همانطور که در نمودار مشخص است یه جایی برعکس میشه : که به این قضیه BLEADY'S ANOMALY میگویند.



۱۱)

زمانی که پروسسی پیج ها خودش را به اندازه کافی ندارد دائم درحال پیج فالت خوردنه و اگر از جانشینی محلی هم استفاده کند دائم داره فریم های خودش را جایگزین میکند وممکنه دفعه بعد به همون فریمی که در مرحله قبل ازاش کرده بود نیاز داشته باشه که این باعثت کاهش cpu utilization میشود و اگر به این دلیل کاهش cpu util داشته باشیم سیستم درجه مالتی پروگرمینگ را بالا میبره و این تازه کار را بدتر هم میکند و به پیج فالتهای بیشتری هم بخوریم و در این حالت میگیم پروسس دچار ترشینگ شده.

در هر بازه زمانی مشخص پروسس ها دارند با یه سری پیج های مشخص استفاده میکنند به این میگن لوکالیتی.

که از این میتونیم استفاده بکنیم و پیج های مشخصی که در بازه های زمانی مشخص را باهاش کار داریم را که بهشون پیج های لوکالیتی میگن را بیاریم د رفریم ها امکان خوبی را فراهم میکنند.

هر پروسسی در بازه های زمانی مشخص یه تعداد پیج های خاصی و مشخصی را باهشون کار داره که به این پیج ها میگن ورکینگ ست.

اگر مجموع ورکینگ ست های پروسس ها از مجموع مموری فیزیکال کمتر باشه مشکلی نداریم و دچار ترشینگ نمیشیم.

اما اگر بیشتر باشه ممکنه یک یا چند پروسس به ترشینگ بخورن و مدام پیج فالت بدن