دانشگاه آزاد اسلامی واحد زنجان

سیستم عامل

مهدی بازرگانی

گردآورنده: رعنا رزاقی

ویراستار: مژگان فضلی- سحر نجفی

زمستان ۹۲

www.mbzir.com

فهرست مطالب

فصل اول

1-تعریف سیستم عامل: ۱

۲- چند برنامگی ***(moltiprogramming)*** ۲

۳-یادآوری معماری یک سیستم کامپیوتری: ۵

فصل دوم

۱-وقفه (***Interapt***) : ۶

۲-انواع وقفه ۸

۳-وظایف سیستم عامل در انجام مکانیسم وقفه: ۱۰

۴-سلسله مراتب حافظه (عنصرذخیره سازی): ۱۲

۵-تایمر ۱۴

۶-عملیات داخلی سیستم عامل در پس گرفتن پردازنده از یک فرایند و دادن به فرایند دیگر ۱۴

۷-تفاوت بین اجرای سیستم عامل و اجرای برنامه های کامپیوتری ۱۵

۸-حالت های سیستم کامپیوتری حاوی سیستم عامل ۱۷

۹-حفاظت از حافظه ۱۹

۱۰-اجزای سیستم عامل ۲۱

۱.مدیریت فرایند ۲۱

۲.مدیریت حافظه اصلی ۲۲

۳.مدیریت دیسک ۲۲

۴.مدیریت ***I/O***.....۲۳

www.mbzir.com

۵.مفسر فرمان ۲۶

۱۱-ماشین مجازی ۲۷

فصل سوم

۱-مفهوم مدیریت فرایند ۲۹

۲-حالت های فرایند ۳۰

۳-مدل های فرایند ۳۴

۴-زمانبندها ۳۷

۵-الگوریتم های زمان بند (نوبت بند) ۳۹

۶-روش های الگوریتم های زمان بند ۴۰

-بدون قبضه کردن ۴۰

۱.***FCFS*** ۴۰

۲.***SPN*** ۴۲

۳.***HRRN*** ۴۳

-باقبضه کردن ۴۶

۱.***SRT*** ۴۶

۲.***RR*** ۴۷

سوالات فصل سوم ۵۱

فصل چهارم

۱- ***Inter process communication*** یا ***process synchronication*** ۵۸

۲-مسئله ناحیه بحرانی ۶۶

www.mbzir.com

۳-مشخصات راه حل های ناحیه بحرانی ۶۷
۴-مسائل ناحیه بحرانی ۶۸
۱.تلاش اول ۶۸
۲.تلاش دوم ۶۹
۳.تلاش سوم ۷۰
۵-مروری بر جلسات قبل ۷۲
۶-استفاده از ***TSL*** ۷۴
۷-استفاده از دستورالعمل ***Swap*** ۷۶
۸-سمافور ۷۸
۹-مسئله تولیدکننده مصرف کننده با بافر محدود ۸۳
۱۰-مسئله خوانندگان نویسندگان ۸۴
۱۱-مروری بر جلسه قبل ۸۶
۱۲-مسئله غذا خوردن فیلسوفان ۸۸
۱۳-اشکالات سمافور ۹۰
۱۴-بن بست ۹۲
۱۵-شرایط بن بست فرایند ها ۹۳
۱۶-راه حل مسئله بن بست ۹۶
۱۷- مروری بر جلسه قبل ۹۷
۱۸-الگوریتم بانکداران ۱۰۱
۱۹-الگوریتم تشخیص بن بست ۱۰۷
۲۰- مروری بر جلسه قبل ۱۰۹

www.mbzir.com

۲۱-***ROLL back*** ۱۰۹

سوالات فصل چهارم ۱۱۴

فصل پنجم

۱-مدیریت حافظه ۱۲۰

۲-مفاهیم مدیریت حافظه ۱۲۰

۳-روش های مدیریت حافظه ۱۲۲

۱.روش ایستا ۱۲۲

۲.روش پویا ۱۲۳

۴-الگوریتم تخصیص حافظه در روش پویا ۱۲۴

۵-روش رقابتی ***(buddy)*** ۱۲۵

۶-صفحه بندی ***(paging)*** ۱۲۶

۷-حافظه کاشه ***(cache)*** ۱۳۳

۸-مروری بر جلسه قبل ۱۳۴

۹-مدیریت حافظه قطعه بندی ۱۳۴

۱۰-مدیریت حافظه قطعه بندی صفحه بندی ۱۳۷

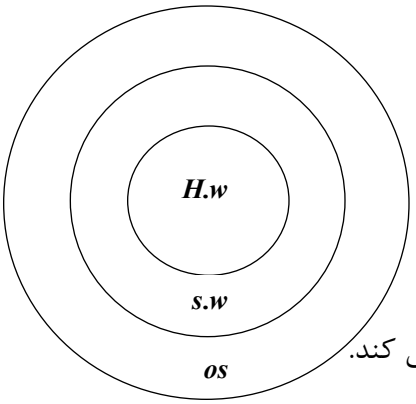
۱۱-حافظه مجازی ۱۳۹

سوالات فصل پنجم ۱۴۲

www.mbzir.com

فصل اول

۱-تعریف سیستم عامل:



۱- ***os*** برنامه ای است که واسط بین سخت افزار و کاربر است.

۲- سیستمی ترین نرم افزار یک سیستم کامپیوتری است.

۳- برنامه ای است که استفاده از کامپیوتر را برای کاربر تسهیل می کند.

۴- سیستم عامل تخصیص دهنده ی منابع است (مدیر منابع یک سیستم کامپیوتری است).

به اصطلاح به منبع ***Resourse*** گوییم که همیشه ***Resourse***های ما محدود هستند. منابع مختلفی داریم:

- **منابع سخت افزاری:** حافظه، فضای دیسک، شماره ی پورت***I/O*** (تجهیزات ورودی خروجی)، زمان پردازنده
- **منابع نرم افزاری:** فایل، رکورد، فیلد

پردازنده مخفف ***cpu*** یعنی ***Central processing unit*** (واحد پردازشگر مرکزی) است، زمانی می تواند واحد مرکزی باشد که مهم باشد و تمرکز قدرت در آن وجود داشته باشد یعنی محدودیت دارد؛ چون یکی است برای رسیدن به آن رقابت وجود خواهد داشت (زمان پردازنده هم یکی است)

Cpu یک دانه است.

↓

محدود است

↓

رقابت در آن وجود دارد

www.mbzir.com

۵- برنامه ای است که همواره بر روی یک سیستم کامپیوتری در حال اجرا است و هر برنامه دیگر کاربردی تقلی می شود.

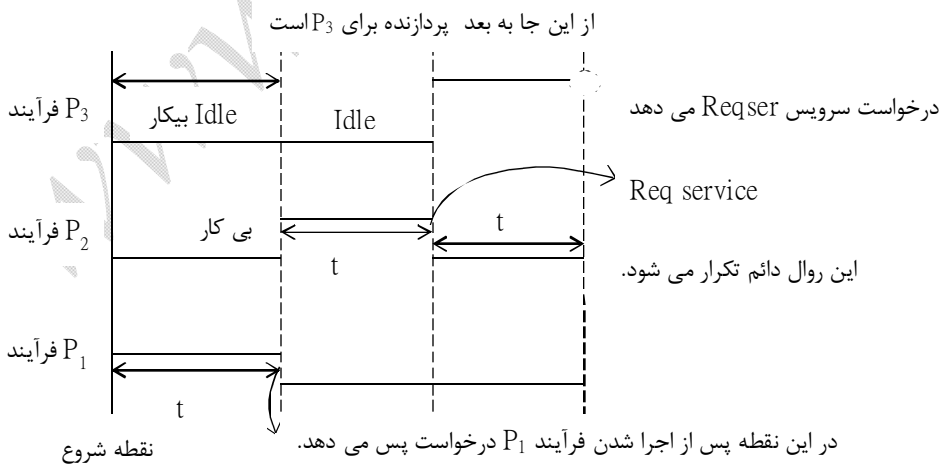
در کل ***os*** دو کار اصلی انجام می دهد
- ایجاد سادگی برای کاربر
- ایجاد شرایط برای عملکرد مؤثر سیستم

۲-چند برنامگی (*MultirPogramming*) : چند تا برنامه به صورت همزمان اجرا شوند.

پردازنده:
- عملکرد شخصی دارد. اجرای دستورالعمل مشخص
- پردازنده کنترل این که در چه زمانی چه عملی انجام شود را دارد.

آیا امکان دارد در یک زمان مشخص دو کار همزمان با هم انجام شوند؟ نه خیر، مانند انسان که امکان ندارد در یک زمان در دو مکان باشد. پس همواره نمی توانیم بیشتر از یک کار را انجام ذهیم، یعنی در یک زمان مشخص یک دستورالعمل بیشتر نمی تواند اجرا شود.

مثال) فرض کنیم ۳ فرآیند داریم:

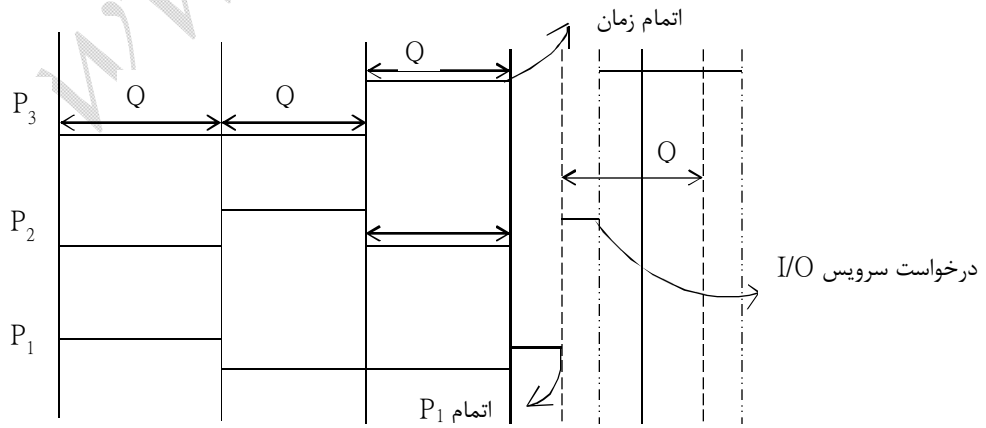


از دید ناظر بیرونی فعالیت فرآیندها به صورت پیوسته است؛ در صورتی که از دیدگاه پردازنده در هر بازه ای از زمان یک و فقط یک فرآیند اجرا می شود (نزد ***cpu*** است).

مثال) یک فیلم در ۶۰ سال پیش ساخته شده است. هنگام دیدن آن فیلم چون سرعتش کند است خودمان متوجه عکس ها در فیلم می شویم چون سرعت سیستم عامل ر ا کم کرده ایم. فیلمی که الان توسط آن عمس ها ساخته می شود نیز همان فیلم است فقط سرعت رد شدن عکس ها از جلوی چشممان را افزایش داده ایم. ما در این درس سرعت سیستم عامل را کند کرده ایم تا بدانیم در هر لحظه چه کاری انجام می دهد.

تا زمانی که بازه های زمانی کوچک باشند عملاً هیچ اتفاقی نمی افتد؛ اما اگر احیاناً بازه ی زمانی بیکاری یک فرآیند طولانی شود، آن وقت کاربر بیرونی متوجه قطع شدن فرآیند می شود.زمانی چند برنامگی را می توانیم اجرایی کنیم که هر فرآیند در زمان مشخصی پردازنده را به فرآیند دیگر تحویل دهد. اما چگونه؟ ← سربار محاسباتی روش بسیار بالاست؛ باید از اشتراک زمانی (***Time Sharing***) استفاده کنیم.

ترکیب اشتراک زمانی و چند برنامگی باعث می شود مشکل قبلی به وجود نیاید چرا که هر فرآیند یک سهم زمانی خاصی برای اجرا دارد.



www.mbzir.com

چه زمان تمام شود و با چه درخواست سرویس دهد، کار فرآیند تمام می شود.

تضمین می کنیم بعد از $2Q$ نوبت به فرآیند می رسد؛ اگر n فرآیند داشته باشیم بعد از $(n-1)Q$ نوبت به آن فرآیند خواهد رسید. فرض کنیم P_3 اتمام زمان کرده است و نوبت به P_1 رسیده است که P_1 دارد اجرا می شود؛ وقتی به نقطه رسید اتمام P_1 است و از همان نقطه به بعد P_2 شروع می‌شود.

پس در مدل اشتراک زمانی پس از پایان یافتن سهم زمانی، هر فرآیند از حالت اجرائی خارج می شود و نوبت را به فرآیندهای در صف می دهد.

خروجی چند برنامگی و اشتراک زمانی یک ایده ای را مطرح می کند به نام همروندی.

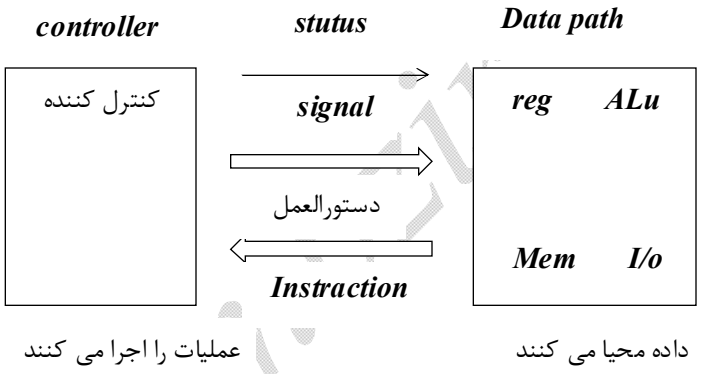
Time Sharing + Multi programmin = **همروندی فرآیندها** (***concurreny***)

www.mbzir.com

۳-یادآوری معماری یک سیستم کامپیوتری:

ما به پردازنده، کنترلر می گوییم که در داخلردازنده چند ثبات و ... قرار داشتند ارتباط آن ها با قسمت های مختلف از طریق گذرگاه بود. انواع ثبات ها عبارت بودند از ***Pc, IR , TR , DR , AR , Ac*** و ...

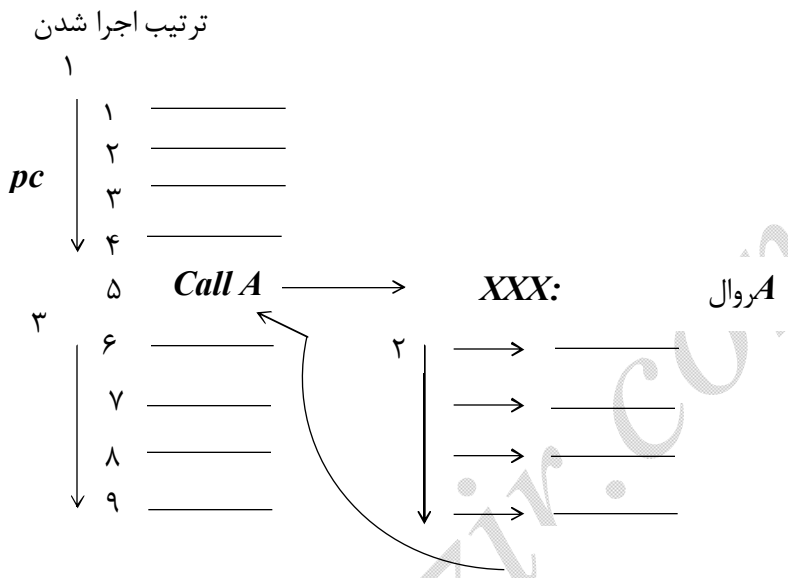
کنترل کننده یک ***cheap*** کلی است: ***Datapath*** (مسیر داده) دارای اجزایی مانند زیر است: ***I/o, Memory, ALu, Register***



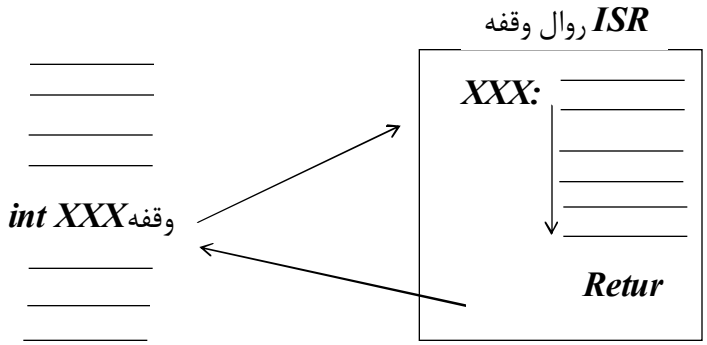
رجیسترها:
- دیده می شوند (از دیدگاه برنامه ی کاربردی) ← مانند ***AR , DR***
- دیده نمی شوند (از دیدگاه برنامه ی کاربردی) ← مانند مدیریت حافظه، وضعیت پردازنده(***psw***)، ***IR***

فصل دوم

۱- وقفه (*Interapt*) :



وقفه بسیار شبیه دستورالعمل ***call*** است ما یک برنامه را نوشتم. ***Pc*** هر بار خط ها را دانه دانه اجرا می کند وقتی به ***call*** رسید اجرای فرمان دیگر بعد از ***call*** نیست و اجرا به خط ششم نمی رود و انتقال می یابد به یک آدرس دیگر و اجرا می کند و اگر ***Return*** داشت بر می گردد به خط ششم و اگر نداشت دیگر بر نمی گردد.



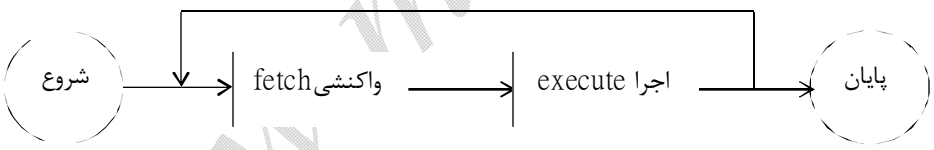
www.mbzir.com

ISR (Interupt service Routine)

وقفه پایان کار را نشان می دهد نه شروع کار را،در واقع نتیجه به بدنه ی اصلی برنامه بر می گردد. آن چه مهم است نتیجه ی وقفه است. زمان اجرای دستور ***call*** از قبل تعیین شده است، اکا زمان اجرای وقفه از بیرون تحمیل می شود یعنی کاربر می تواند یک وقفه ی سخت افزار می دهد (مثلاً ***shutting down*** کند).

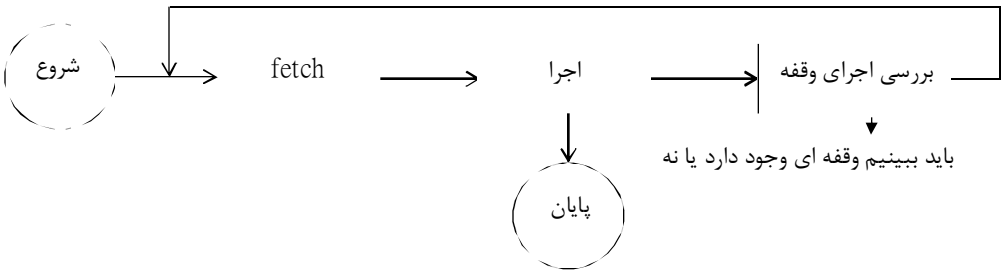
وقفه:
- سخت افزاری (***NMI***) ← در همان لحظه باید انجام شوند مانند زمان پردازنده،
- وقفه‌های کاربردی،وقفه خطاهای سخت افزاری
- نرم افزاری ← می توان به آن ها فرصت داد مانند درخواست برای اجرا که می تواند بعد از یک مدت کوتاهی اجرا شود

روال (سیکل) یک دستورالعمل خاص:



سیکل دستورالعملInstrnoction cycle

اگر وقفه اتفاق بیافتد روال کمی تغییر می کند:



www.mbzir.com

۲- انواع وقفه

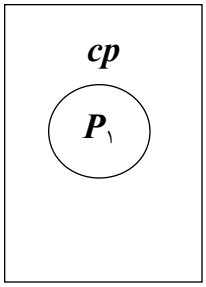
۱- وقفه ی ورودی / خروجی: مرسوم ترین نوع وقفه ها است فرض کنیم از پورت کمپورد چیزی را بخوانیم. ارتباط ا با دنیای خارج از وقفه ی ورودی / خروجی انجام می گیرد. وقفه های ورودی / خروجی در پایان دستورالعمل اجرا ی شوند.(ارسال اطلاعات به کارت شبکه)

۲- وقفه ی تایمر (زمان سنج): تایمر یک فلزی از جنس کوارتس است و کارش تولید فرکانس است؛ وسیله ای برای کنترل عملیات داخل پردازنده است؛ در واقع قابلیت تنظیم دارد. ما خودمان در نظر می گیریم که مثلاً وقفه هایمان ۱ میلی ثانیه ۱ میلی ثانیه باشند. پس از اتمام سیکل زمانی وقفه صادر می گردد. (مثلاً تمام کارهای ما با زمان سنج کشور تنظیم می گردد از جمله رفت و آمد و ...).

بنابراین تنظیم کلیه ی فعالیت های سیستم عامل با تایمر است.

۳- وقفه ی اجرای برنامه (وقفه های داخلی پردازنده): مثلاً یک برنامه ای را در ***cpu*** اجرا می کنیم. برنامه ی P_1 در آن اجرا می شود. ***Pc*** به یک خطا می رسد و وقفه در آن جا اجرا می شود.

یعنی یک عمل غیر مجاز اتفاق افتاده است، در واقع یعنی دیگر امکان ادامه ی اجرا توسط پردازنده ی نیست و باید وقفه اتفاق نیفتد.(تقسیم بر صفر،خطای سرریز شدن)



www.mbzir.com

۴- وقفه ی نقض سخت افزار: سخت افزار در مقابل خطاها تنها کاری که می تواند انجام دهد این است که وقفه دهد.

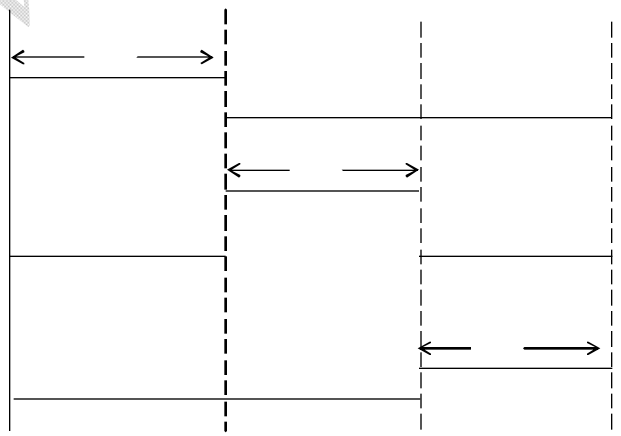
باید سخت افزار طوری طراحی شده باشد که بروز اشکال را خودش اعلام کند مثلاً خرابی ***Memory***، قطع برق، ***aultin memory***.

- وقفه های ۱ و ۲ خارجی هستند یعنی از بیرون این وقفه ها اعمال می شوند
- وقفه های ۳ و ۴ درونی هستند یعنی از درون اعمال می شوند.
- وقفه ی ۱ و ۲ حتماً در پایان دستورالعمل اجرا می شوند.
- وقفه ی ۳ و ۴ مفهومی برای اجرا در پایان دستورالعمل ندارد.

هر برنامه (فرآیند) به جز سیستم عامل؛ برنامه ی کاربردی است.

دو خاصیت مهم در سیستم عامل های امروزی ← ***Multi programming***، ***timesharing***

یکی از عملیات مهم وقفه اجرای مکانیسم سهم زمانی است.



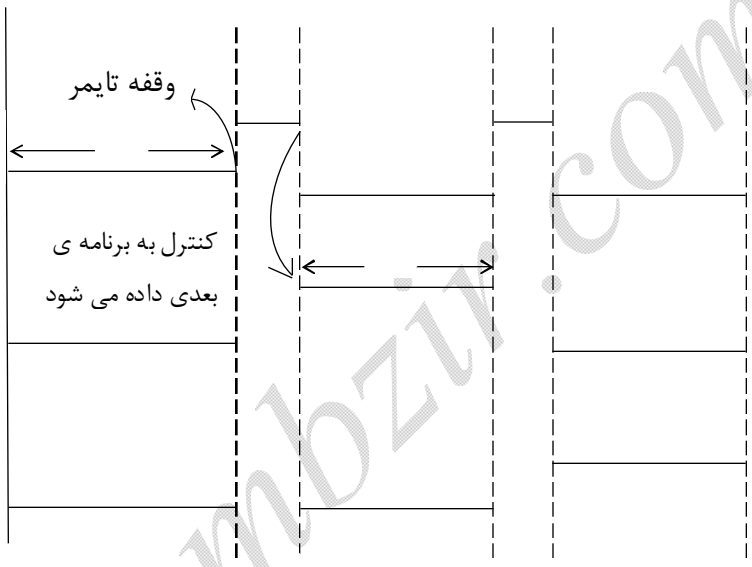
www.mbzir.com

۳- وظایف سیستم عامل در انجام مکانیسم وقفه:

۱. تعیین نوبت بعدی

۲. تنظیم تایمر برای مدت q

۳. انتقال کنترل به برنامه (فرآیند) بعدی



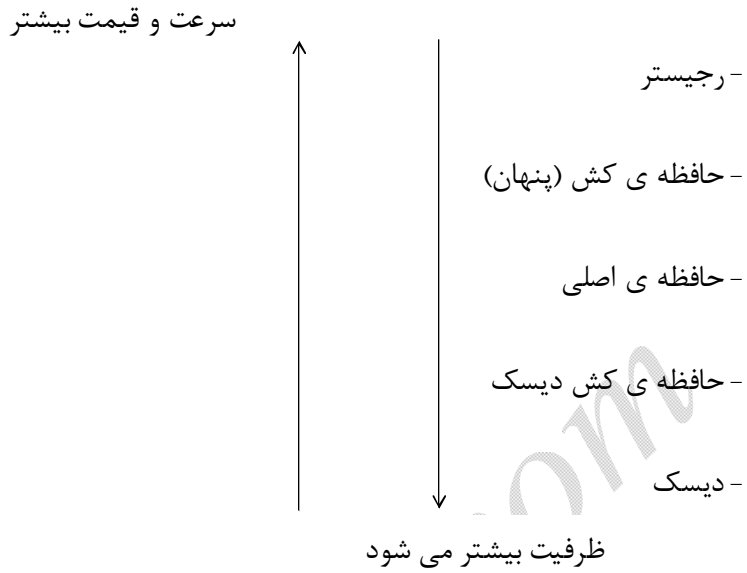
فرض کنیم از قبل P_1 داشته اجرا می شده است و به اندازه ی q بوده است (یعنی هیچ وقفه ی درونی انجام نشده است و فقط وقفه ی زمان سنج اتفاق افتاده است). حال فرض کنیم کنترل به P_2 داده می شود. باز در زمان خیلی کم کارها توسط سیستم عامل انجام می گیرد که در مقابل q خیلی زمان کمی است. قبلاً فکر می کردیم همه ی کارها پیوسته انجام می گیرند، بعد دیدیم که ***timesharing*** این طوری نیست چون سرعت بالاست؛ حال می گوییم سیستم عامل به عنوان برنامه باید پردازنده را بر انجام یک سری کار در اختیار بگیرد. وقتی سیستم عامل پردازنده را در اختیار گرفت نوبت بعدی را مشخص می کند و تنظیم تایمر برای مدت q را ***set*** می کند.

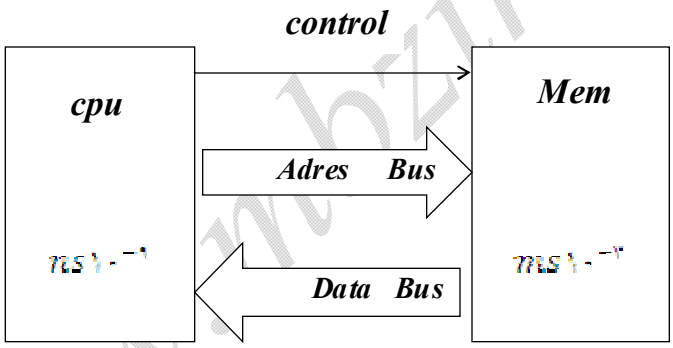
www.mbzir.com

در زمان اجرای P_1 سیستم عامل هیچ کاری نمی کند. ۲ اتفاق برای فرآیند قبل از رسیدن به وقفه ی تایمر می تواند رخ دهد که این دو اتفاق حالت های ۳ و ۴ وقفه هستند. سیستم عامل نقش تنظیم کننده را بر عهده دارد و در حین اجرای فرآیند کاری نمی تواند انجام دهد.

www.mbzir.com

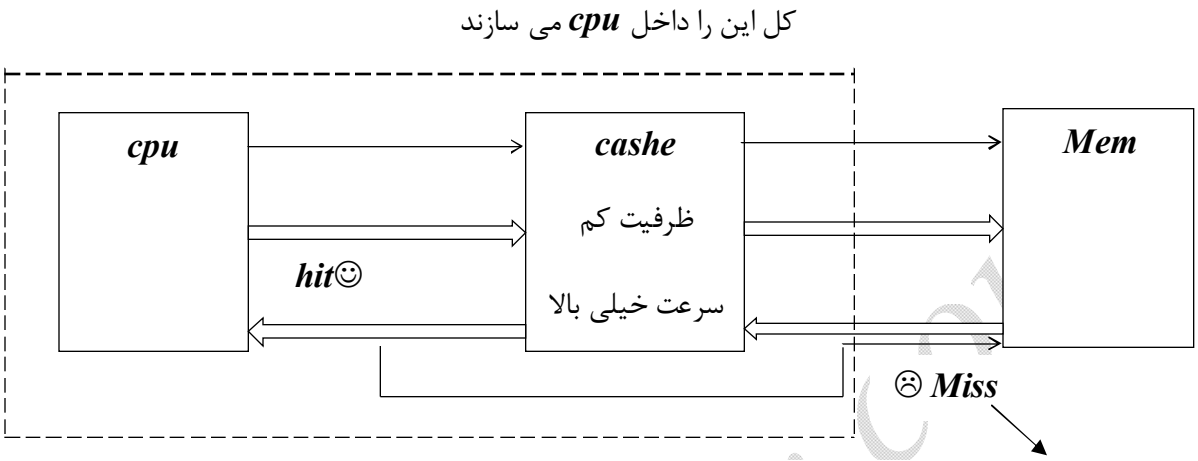
۴- سلسله مراتب حافظه (عنصرذخیره سازی):





ارتباط ***cpu*** و ***Memory*** بسیار تنگاتنگ است، یعنی در هر لحظه با هم کار می کنند. یک ارتباط داده ای روی باس دارند، یک ارتباط آدرس دارند و یک ارتباط کنترل باش است که سیگنال های کنترلی را می دهد. ***Cpu*** یک ماهیتی است که در حد نانوثانیه کار می کند ولی ***Memory*** یک ماهیتی است که در حد میلی ثانیه کار می کند. $\left(\frac{10^{-3}}{10^{-9}} = 10^{-6}\right) 10^{6}$ برابر ***Memory*** کندتر است. راندمان (خروجی یک سیستم) با عملکرد هر دوی این دو بیرون می آید پس چون ***Memory*** کندتر است پس راندمان را ***Memory*** مشخص می کند. پردازنده سریع و حافظه کند

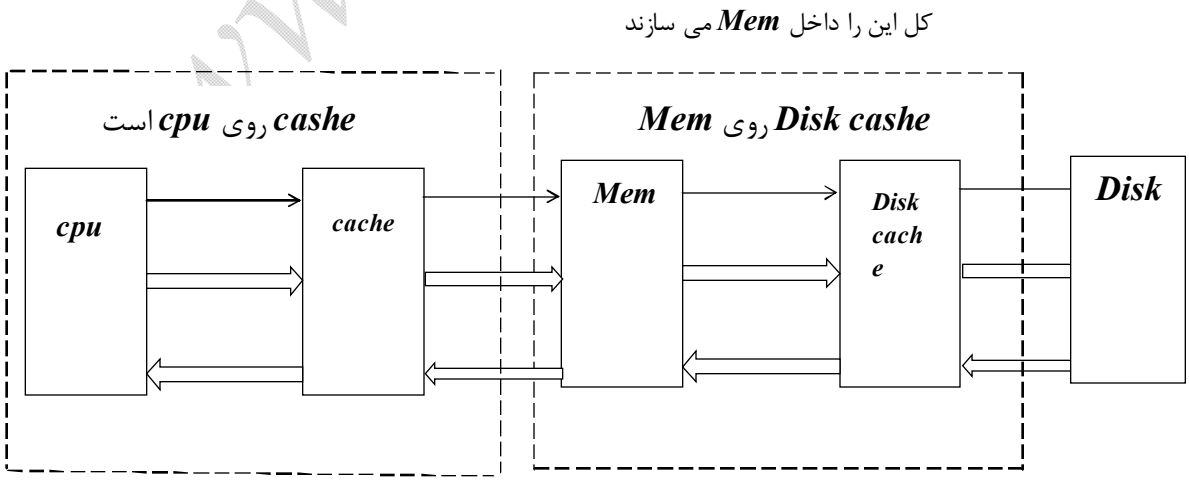
است پس از ***cashe*** استفاده می کنند که بین این دو قرار می گیرد ولی ظرفیت آن بسیار کم است.



اگر در حافظه ی ***cashe*** نباشد ***Miss*** اتفاق می افتد

در حافظه ی ***cashe*** ظرفیت کم و سرعت خیلی بالا است بنابراین خیلی از اطلاعات را در ***cashe*** قرار می دهند. ۹۸٪ اطلاعات ***cpu*** به جای ***Mem*** در ***cashe*** قرار دارد. اگر اطلاعاتی که ***cpu*** می خواهد در ***cashe*** باشد ☺؛ اگر نباشد باید به ***Mem*** برویم ☹ .

همین اتفاق نیز بین حافظه ی اصلی و دیسک رخ می دهد:



Mem در حد ***ms*** کار می کند ولی دیسک در حد ***s*** کار می کند، پس از دیسک کش استفاده می کنند که ظرفیت آن نسبت به دیسک خیلی کمتر است.

۵- تایمر:

سیستم عامل برای دادن پردازنده به فرآیند دیگر ابتدا خودش پردازنده را در اختیار بگیرد، بعد آن را به فرآیند دیگر بدهد، پس سیستم عامل چه عملیاتی را برای در اختیار گرفتن فرآیند انجام خواهد داد؟

شرایط به کار افتادن سیستم عامل
- وقفه
- در خواست از طرف برنامه ی کاربردی

۶- عملیات داخلی سیستم عامل در پس گرفتن از یک فرآیند و دادن به فرآیند دیگر:

← نقطه ی *

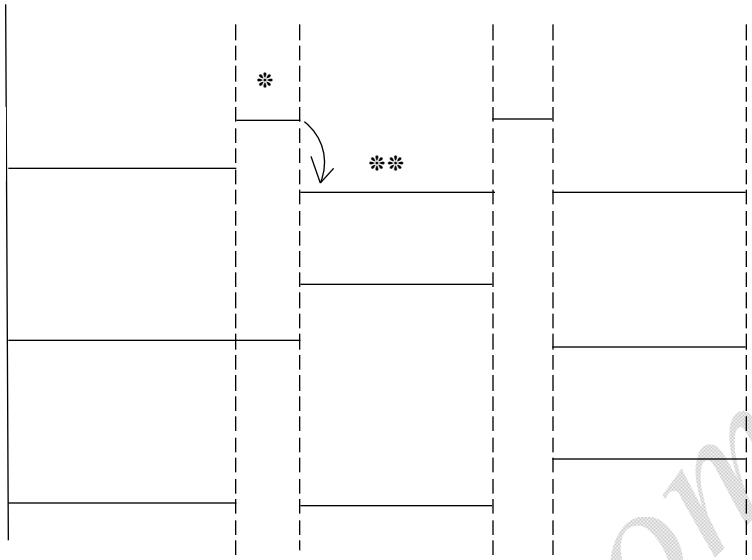
۱. ذخیره کردن حالت فرآیند P_1

۲. تعیین نوبت بعدی

۳. بار کردن حالت P_2

۴. باز گرداندن کنترل به p_2← نقطه ی **

www.mbzir.com



کار تایمر: تولید و ایجاد وقفه

پس وظیفه ی مهم تایمر نگهداشتن سهم زمانی و تولید وقفه است.

۷- تفاوت بین سیستم عامل و اجرای سایر برنامه های کاربردی:

اجرای ***os*** ← اختیارات زیاد است نیاز به دستورالعمل های ممتاز داریم.

اجرای برنامه های کاربردی ← اختیارات کم است و نیازی به دستورالعمل های ممتاز ندارد.

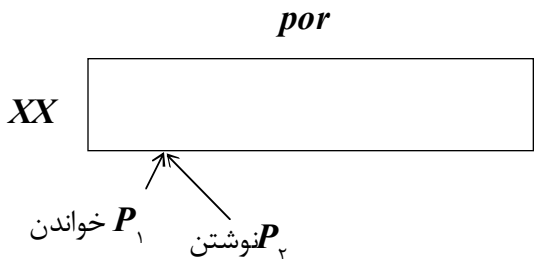
دستورالعمل های ممتاز ← این دستورالعمل ها مخصوص ***cpu*** است.

(***privilyeIntruction***)

P_1: ***IN XX*** از آدرس ***XX*** ورودی بخوان

P_2: ***out XX*** به آدرس ***XX*** بنویس

www.mbzir.com



این دو در پورت (P_1و P_2) از حال هم بی خبرند و هر دو به ***port*** دستیابی دارند و با هم دارند زمانبندی می شوند. به این اتفاق هرج و مرج می گویند.

به خاطر هرج و مرج، فرآیندها اجازه ی اجرای کار با دستورالعمل های ممتاز را ندارند. برای اجازه داشتن درخواست ها به سیستم عامل داده می شود و اگر سیستم عامل صلاح دانست (درخواست ها را در داخل صف قرار می دهد)، فرآیندها اجرا می شوند.

تفاوتی بین اجرای سیستم عامل و یک برنامه کاربردی وجود دارد. اختیارات سیستم عامل بیشتر از برنامه های کاربردی است چون ***os*** نظارت بیشتری دارد. چرا برنامه های کاربردی از دستورالعمل های ممتاز استفاده نمی کنند؟ فرض کنیم برنامه های کاربردی بتوانند فرآیندها را اجرا کنند؛ چون پورت ***IN*** و پورت ***out*** از هم خبری ندارند پس در آن پورت خاص هرج و مرج داده ای رخ می دهد. پس تمامی برنامه ها برای اجرای دستورالعمل های ممتاز از طریق سیستم عامل انجام می شوند که سیستم عامل دستورالعمل های را گرفته؛ صف بندی می کند و سپس نوبت دهی می کند. یکی از دستورالعمل های ممتاز دستکاری قسمت های حافظه است که به شدت سیستم عامل جلوی آن را می گیرد.

www.mbzir.com

۸-یک سیستم کامپیوتری (حاوی سیستم عامل) ۲ مد (حالت) دارد:

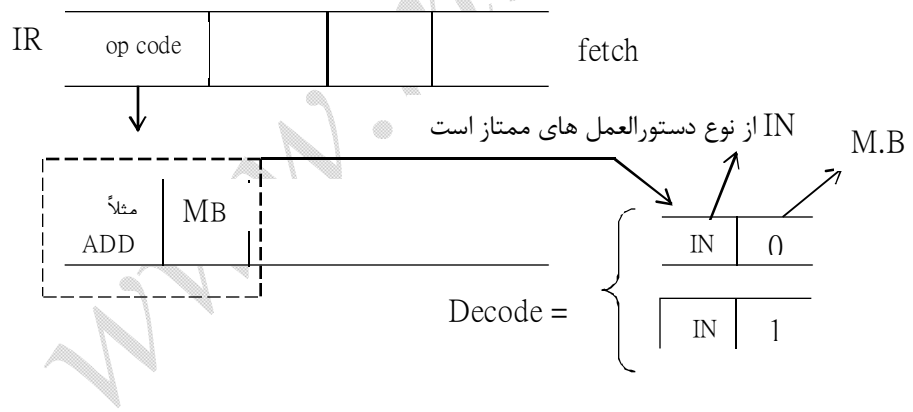
مدهای دوگانه (***dual mode***)

- مد سیستم ***sustem mode*** ←مخصوص ***os*** است،اختیارات زیاد است،اجرای ***p.I*** مجاز است.
- مد کاربر ***user mode*** ←مخصوص ***App***ها است، اختیارات کم است، اجرای ***p.I*** مجاز است.

سخت افزار باید متوجه شود که سیستم در کدام مد قرار دارد و آیا اجازه هر دستورالعملی را در مدها دارد یا نه؟

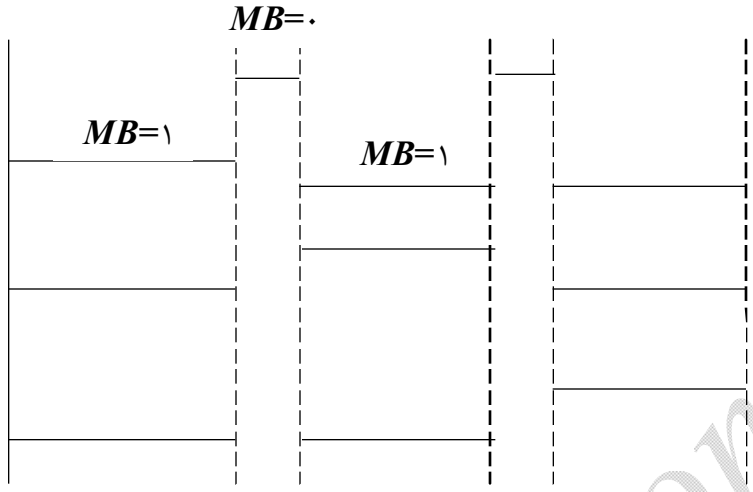
برای متوجه شدن سخت افزار از ***Mode Bit*** استفاده می کنیم که یا ۰ است یا ۱

- ۰ ***System mode***
- ۱ ***System mode***



معمولاً در فاز ***fetch*** این رجیسترها را واکنشی می کنیم؛ در فاز ***Decode*** تک تک این ها را می شناسیم. حال کدام دستورالعمل مجاز است؟ ***IN۰*** یا ***IN۱*** ؛ ***IN۰system mode*** باشد که در ***mode bit , system mode*** صفر است.

www.mbzir.com



تغییر بین ۰=***MB*** و ۱=***MB*** چگونه اتفاق می افتد؟ چگونه سیستم عامل این کار را انجام می دهد؟ سیستم عامل وقتی اجرا می شود ۰=***MB*** است و وقتی برنامه ی کاربردی مانند P_1، P_2 و P_3 اجرا می شوند ۱=***MB*** است. زمانی که صفر به ۱ تبدیل می شود ۱ ← ۰ کنترل در اختیار خود سیستم عامل است و سیستم عامل چون می تواند دستورالعمل های ممتاز را انجام دهد پس به راحتی می تواند این تنظیم را انجام دهد. زمانی که ۱ به صفر تبدیل می شود ۰ ← ۱ سخت است چون کنترل در اختیار کاربردی است.

شرایط به کار افتادن سیستم عامل:
- وقفه
- درخواست برنامه ی کاربردی از سیستم عامل (وقفه ی نرم افزاری)
- درخواست برای اینکه ***MB*** از یک به صفر تبدیل شود

۲ نتیجه دارد

نتیجه ی ۱: وقفه باعث تغییر ***MB*** می شود.

نتیجه ی ۲: سیستم عامل در اثر وقفه (به مفهوم عام) به کار می افتد.

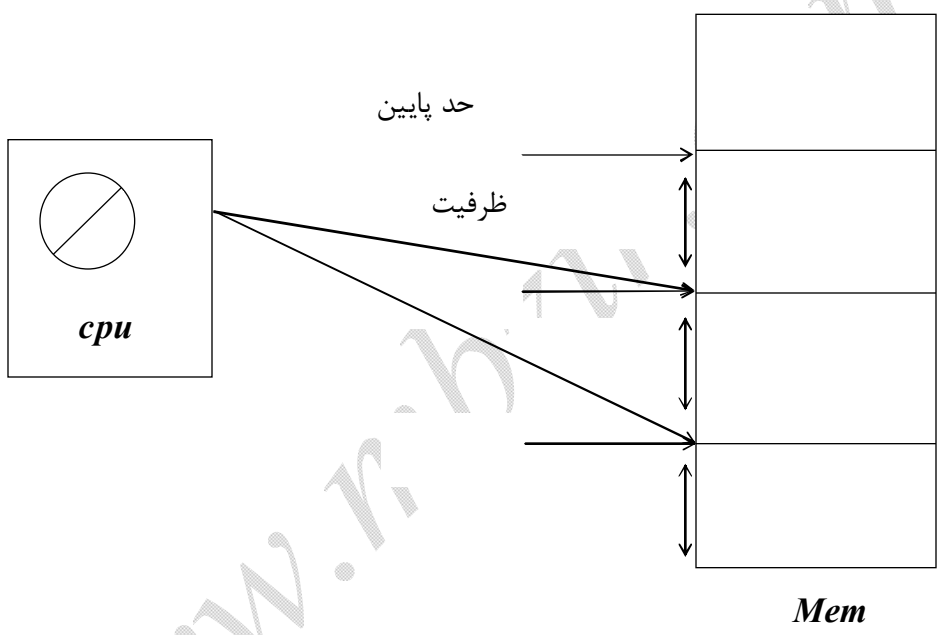
www.mbzir.com

۹- حفاظت از حافظه:

سیستم عامل باید قادر باشد مکانیزم هایی داشته باشد که فرآیندها به محدوده ی هم تجاوز نکنند.

مدیریت حافظه بر عهده ی سیستم عامل است.

سیستم عامل نباید اجازه دهد که P_1 به محدوده ی P_2تجاوز کند یا P_2 به محدوده ی P_1 تجاوز کند.

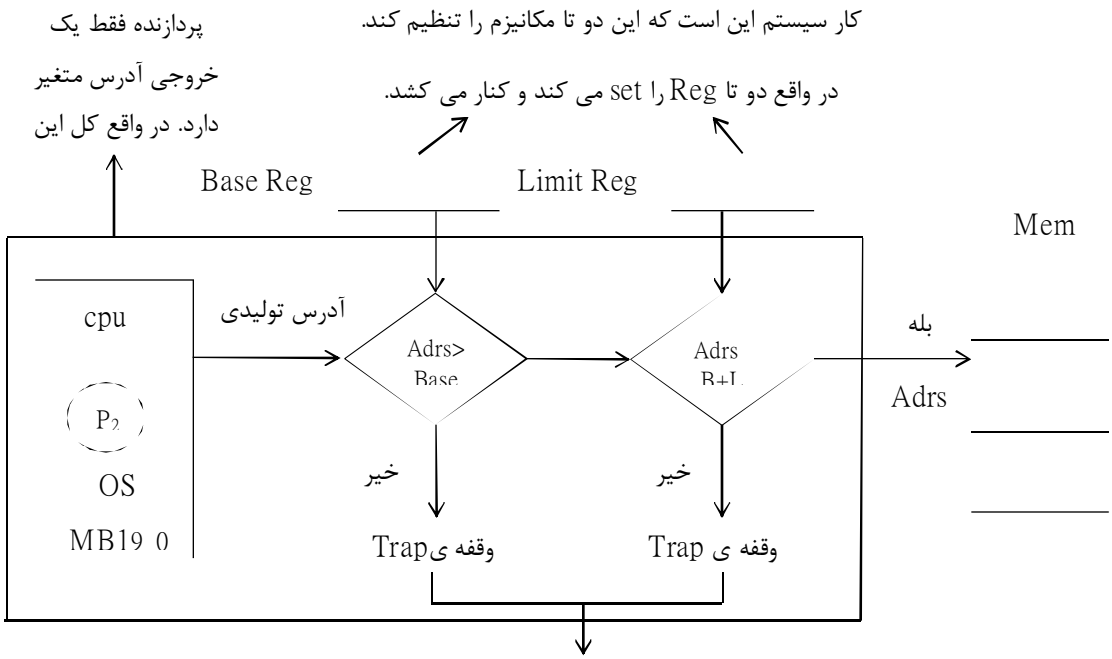


برای حفاظت از حافظه به ازای هر فرآیندی ۲ رجیستر داریم: $\left\{ \begin{array}{l} \textit{Limit} \\ \textit{Base} \end{array} \right.$

هر فرآیندی که می خواهد اجرا شود مثلاً P_1، P_1 می رود در *cpu*، *Buse* و *Limit* با P_1 مقداردهی خواهد شد. بعد نوبت به P_2 می رود.

Base Reg	$\cancel{P_1}$ P_2
Limit Reg	$\cancel{P_1}$ P_2

سیستم عامل برای هر فرآیند یک حد پائین و یک ظرفیت می گذارد.



وقفه اتفاق افتاد یک خطای غیر مجاز (وقفه از نوع) یعنی سیستم عامل از کار می افتد و مد از به می رود. دیگر کنترل در اختیار P_2 نیست؛ پس P_2 را از پردازنده بیرون آورده و به جای آن سیستم عامل می نشیند

۱۰-اجزای سیستم عامل

۱. مدیریت فرآیند (***process Management***)

۲. مدیریت حافظه ی اصلی (***Memory Management***)

۳. مدیریت دیسک (***Disk Management***)

۴. مدیریت ***I/O*** (***I/o Management***)

۵. شبکه سازی (***Networking***)

۶. مفسر فرمان (***command Interpretor***)

۱- مدیریت فرآیند (*process Management*): در فرآیند یک ***process*** ویک ***pragramm*** داریم.

وظایف سیستم عامل در خصوص مدیریت فرآیند:

۱. ایجاد (***create***) و حذف (***delete***) فرآیند

۲. معلق سازی فرآیند و شروع دوباره ی فرآیند: با دو عمل انجام می گیرد ***Resumption*** (شروع دوباره) و ***Sun pestion*** (معلق)

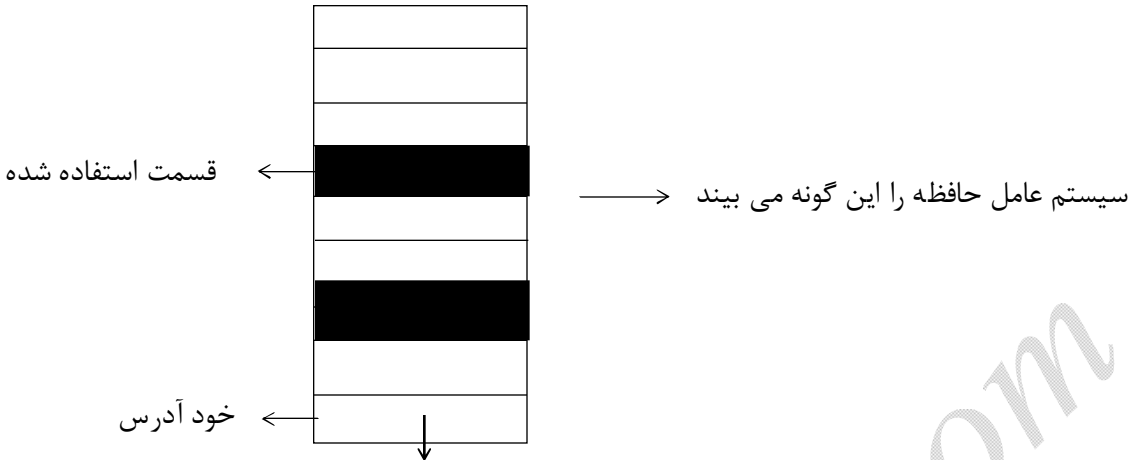
۳. همگامی فرآیندها (***concorrency***)

۴. مکانیزم هایی برای ارتباط فرآیندها ارائه می دهد.

۵. مکانیزم هایی برای حل مسئله بن بست ارائه می دهد.

www.mbzir.com

۲- مدیریت حافظه ی اصلی (*Memory Management*):



آرایه ی خطی از کلمات ***RAM: Linear Array of words=***

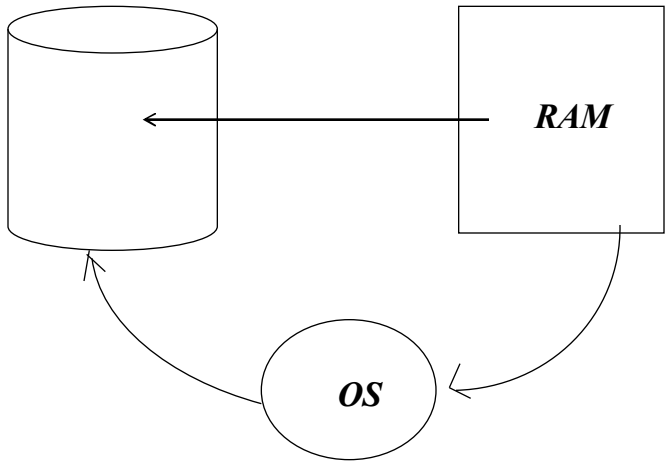
وظایف سیستم عامل در خصوص مدیریت حافظه ی اصلی:

۱. آخرین وضعیت قرارگیری فرآیندها در حافظه
   - پیگیری فضاهای استفاده شده در حافظه
   - پیگیری فضاهای آزاد
۲. تخصیص دادن و پس گرفتن به موقع حافظه از فرآیند

۳- مدیریت دیسک (*Disk Management*): بسیار شبیه به حافظه ی اصلی است از باب وضعیت تخصیص دیسک، پیگیری وضعیت فضاهای آزاد، پیگیری نوبت بندی دسترسی به دیسک.

وظایف سیستم عامل در خصوص مدیریت دیسک
- ۱. پیگیری وضعیت تخصیص دیسک
- ۲. پیگیری وضعیت فضاهای آزاد
- ۳. پیگیری نوبت بندی دسترسی به دیسک

www.ibzir.com

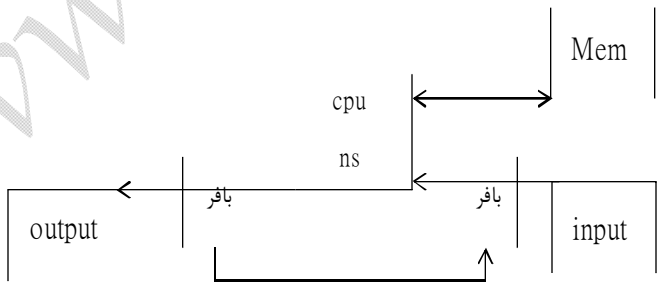


حافظه دیسک با حافظه ی ***RAM*** تفاوت دارد. ***RAM*** حافظه ای از جنسی الکترونیک است. حافظه ی دیسک از نوع مکانیکی است. حافظه ی ***RAM*** هیچ حرکتی در آن نیست ولی دیسک دائم به جلو و عقب می رود و حرکت می کند و می چرخد. بنابراین شیوه های دستیابی تفاوت می کند.

در ***RAM*** موقعیت خانه های حافظه بابت دستیابی تفاوتی نمی کند.

در ***Disk*** بسته به موقعیت خانه ی مورد نظر از بابت دوری و نزدیکی به ***head*** تفاوت می کند (مانند ***CD*** که به دور محوری می چرخد).

۴- مدیریت *I/O* (*I/O Management*):



برای کنترل هماهنگی سرعت

پردازنده از کانال هایی به عنوان ورودی داده گرفته (از ***input***) و به ابزاری به نام خروجی داده می دهد (به ***cutput***) نا داده را در ***bus*** بگذارد. عمده ی مشکل این است که سیستم عامل با این

www.mbzir.com

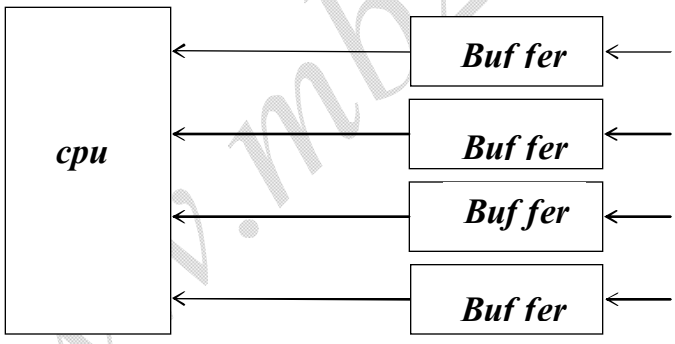
ورودی ها و خروجی ها چه باید بکند؟ چون ابزار آلات ورودی / خروجی کند هستند، از سخت افزاری به نام بافر استفاده می کنیم. هر ابزار ورودی / خروجی یک بافر دارد. بافرهای برای کنترل هماهنگی سرعت بین پردازنده و ابزارآلات ورودی / خروجی استفاده می شوند. (سرعت سخت افزارها متفاوت است و برخی تند و برخی کند است.)

بافر از جنس حافظه است و برای این که پردازنده منتظر نماند از بافر استفاده می کنیم چه در ورودی و چه در خروجی.

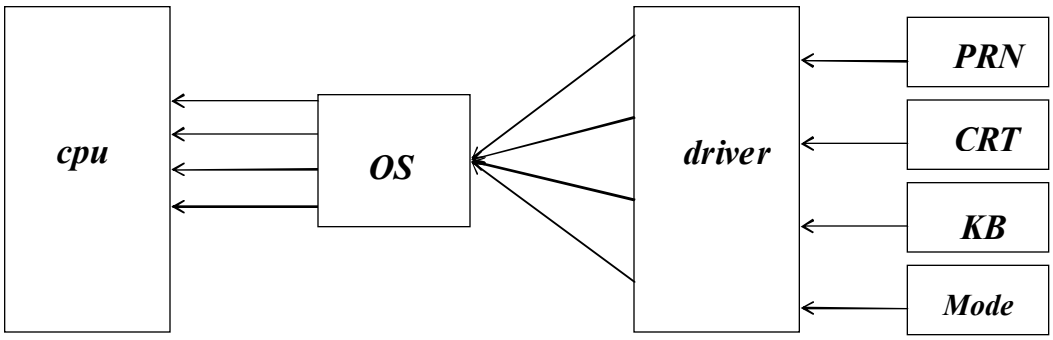
وظایف سیستم عامل در خصوص مدیریت *I/O*:

۱. اداره کردن بافر پورت ها ← برای هر پورتی ما یک بافری داریم. بافرها باید کنترل شوند.

۲. صف بندی درخواست های ورودی / خروجی (برحسب اولویت ها)



۳. مدیریت درایوها (گرداننده ها)



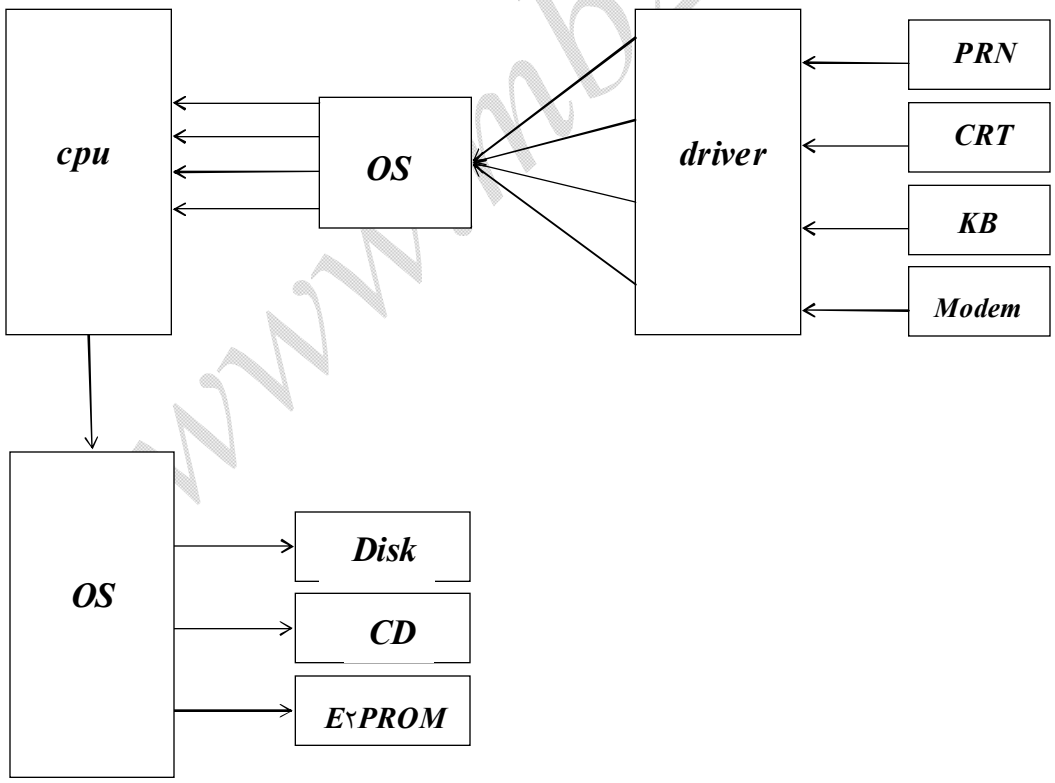
www.mbzir.com

همه ی درخواست ها بعد از ***driver*** به سیستم عامل داده می شود بنابراین مدیریت آن ها به وسیله ی سیستم عامل انجام می گیرد.

مدیریت فایل: همواره جایگاه فایل بر روی رسانه ی ذخیره سازی ثانویه (دیسک مغناطیسی، دیسک نوری، فلشی مموری و ...) است. نوع ذخیره سازی در ***CD*** با اطلاعات نوری است ولی در فلش مموری مثل مدل ***RAM*** ذخیره سازی می شود ولی پایا (ماندگار) است.

با توجه به این که فایل بر روی رسانه های ذخیره سازی گوناگون به صورت های مختلفی قرار می گیرد و از لحاظ فیزیکی بر روی هر رسانه (***Media***) به شکل مختلفی قرار می گیرد، اما از لحاظ منطقی (دید کاربر) کاملاً یکسان به نظر می رسد.

به بیان دیگر دید منطقی یکسانی از اطلاعات فایل به کاربر می دهد



www.mbzir.com

۵- مفسر فرمان (*command Interpretor*): سیستم مفسر فرمان جزئی از سیستم عامل نیست

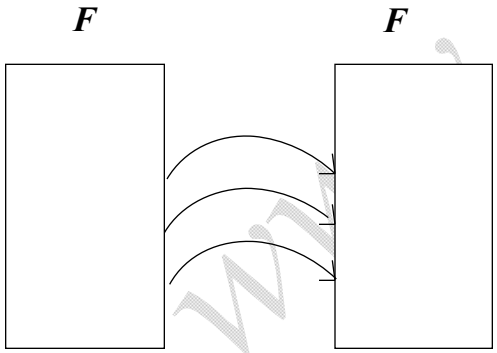
سیستم عامل مجموعه ای از فراخوانی های سیستمی برای اخذ سرویس است و همه ی فرآیندها از این سرویس ها استفاده می کنند.

سؤال: فرآیندها چگونه از سرویس ها استفاده می کنند؟ با استفاده از فراخوانی سیستمی (***system call***) می توانند از سرویس ها استفاده کنند.

سیستم مفسر فرمان به دو شکل متنی و گرافیکی می باشد.

مثلاً کپی کردن F_1 در مسیر F_2 یک سرویس است و اجرای سرویس با فراخوانی سیستمی امکان پذیر است.

خود کپی فراخوانی سیستمی نیست. سرویس $\rightarrow$ ***copy*** F_1 , F_2



ما برای انجام عمل کپی چند تا ***systemcall*** نیاز داریم:

۱. دریافت نام فایل ها (مبدأ و مقصد)

۲. باز کردن فایل مبدأ (***open file***)

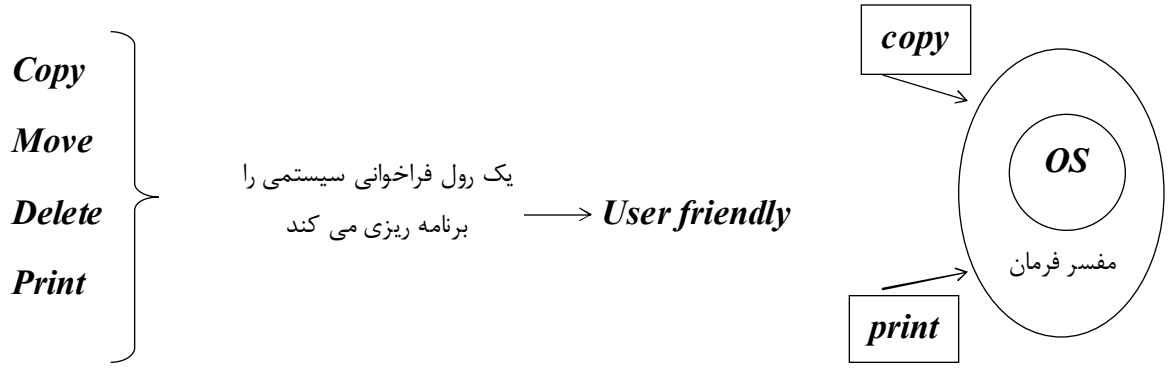
۳. ایجاد فایل مقصد (***create file***)

۴. از فایل ورودی بخوان و در فایل خروجی بنویس $\leftarrow$ { ***Read*** F_1 ; ***Write*** }

۵. بستن هر دو فایل (***Close file***)

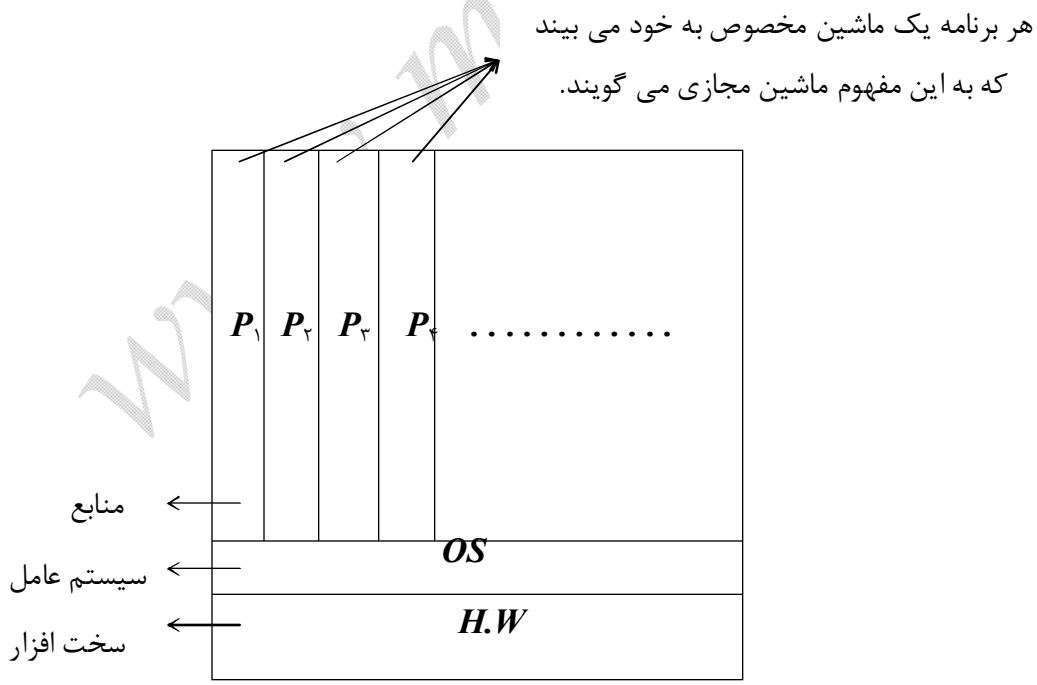
۶. پایان عامل (***Exit***)

www.mbzir.com



۱۱- ماشین مجازی

از دید من کاربر فکر می کنم که فقط فرآیند من دارد اجرا می شود در حالی که کلی فرآیند دارد اجرا می شود. در واقع سیستم عامل من را گول می زیند و من اینطوری فکر می کنم.



www.mbzir.com

هر فرآیند برای اجرا شدن نیاز به منابع دارد که این منابع را سیستم عامل در اختیار یک فرآیند قرار ی هد. هنر سیستم عامل این است که کار می کند که هر فرآیند فکر می کند منابع فقط در اختیار اوست؛ در حالی که این منابع اشتراکی اند و نمی توانند اختصاصی باشند.

هر فرآیند منابع را به صورت مستقل برای خود می بیند، یعنی شراکت در منابع توسط ***OS*** مخفی شده است.

www.mbzir.com

فصل سوم

۱- مفهوم مدیریت فرآیند:

- مفهوم فرآیند و حالات فرآیند

- همگام سازی فرآیندها

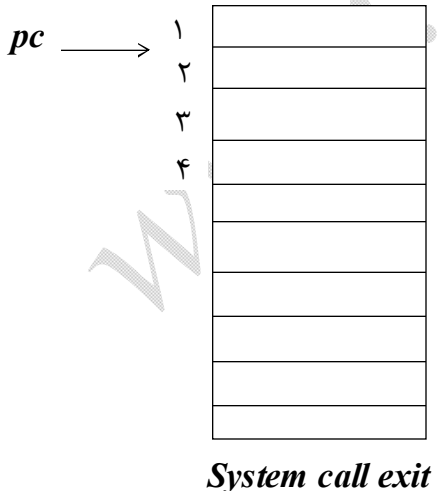
- زمان بندی (***scheduling***) فرآیندها

- الگوریتم های زمان بندی

- مقایسه الگوریتم های زمان بندی

به یک برنامه ی در حال اجرا فرآیند یا ***process*** می گوییم که با ***program*** متفاوت است.

هر برنامه شامل کدهایی است که این کدها دارند اجرا می شوند که اجرای این فرآیند به صورت ترتیبی است و این خط ها را ***pc*** نشان می دهد.

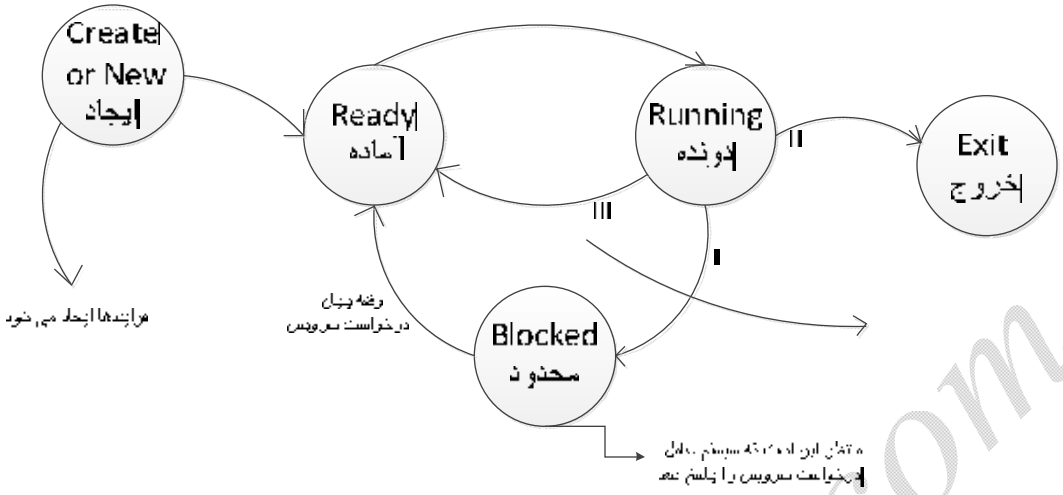


Fetch

Pc ← ***pc***+۱

در هر لحظه یک دستورالعمل از فرآیند اجرا می شود

فرآیند طول عمر دارد و در طول عمر خود از حالات مختلفی عبور می کند.



۲- حالت های فرآیند

۱. پردازنده در اختیار فرآیند است. (در حال اجرا است)

۲. فرآیندهایی که درخواست یک سرویس به ***OS*** داده اند و منتظر پاسخ دادن به سرویس آن ها هستند.

۳. فرآیندهایی که آماده ی اجرا و گرفتن پردازنده می باشند.

۴. یعنی تمام منابع داده شده به فرآیند پس گرفته می شود از سیستم خارج می شود.

۵. فرآیند های تازه ایجاد شده در این حالت قرار می گیرند.

در نقطه ی * درخواست سیستمی را خود فرآیند انجام می دهد و می گوید من دارم تمام می شود ولی در وقفه به صورت خشونت آمیز است و می گوید برو بیرون.

هر فرآیند برای خودش یک شمارنده ی برنامه دارد.

نکته: در هر حال تغییر حالت به دو شکل است:
۱. درخواست سرویس از سیستم
۲. وقفه

www.mbzir.com

سؤال) چه وقایعی می شود پردازنده را از یک فرآیند بگیریم؟

اگر پردازنده را از یک فرآیند پس بگیریم نتیجه ی کار این است که یک فرآیند جدید را وارد مرحله ی ***Running*** کنیم.

سؤال) فرآیندی که هنوز ایجاد نشده است امکان این که درخواست ایجاد بدهد، به چه صورت است؟

فرآیندی که هر مرحله ***Ranning*** قرار دارد این کار را انجام می دهد، پس ایجاد یک فرآیند فقط . فقط توسط یک فرآیند دیر امکان پذیر است.

سؤال) هر فرآیند توسط سیستم عامل چگونه شناخته می شود؟ وضعیت شناسایی هر فرآیند با سیستم عامل به چه شکلی است؟

ما یک ساختار داده ای به نام ***PCB*** داریم تحت عنوان ***Process control Black*** (بلوک کنترل فرآیند)

PCB شامل چه اطلاعاتی است؟

۱. حالت فرآیند (***process statuse***)

۲. شمارنده ی برنامه (***program counter***)

۳. محتویات رجیسترها (***Registers***)

۴. اشاره گر پشته (***stack pointer***)

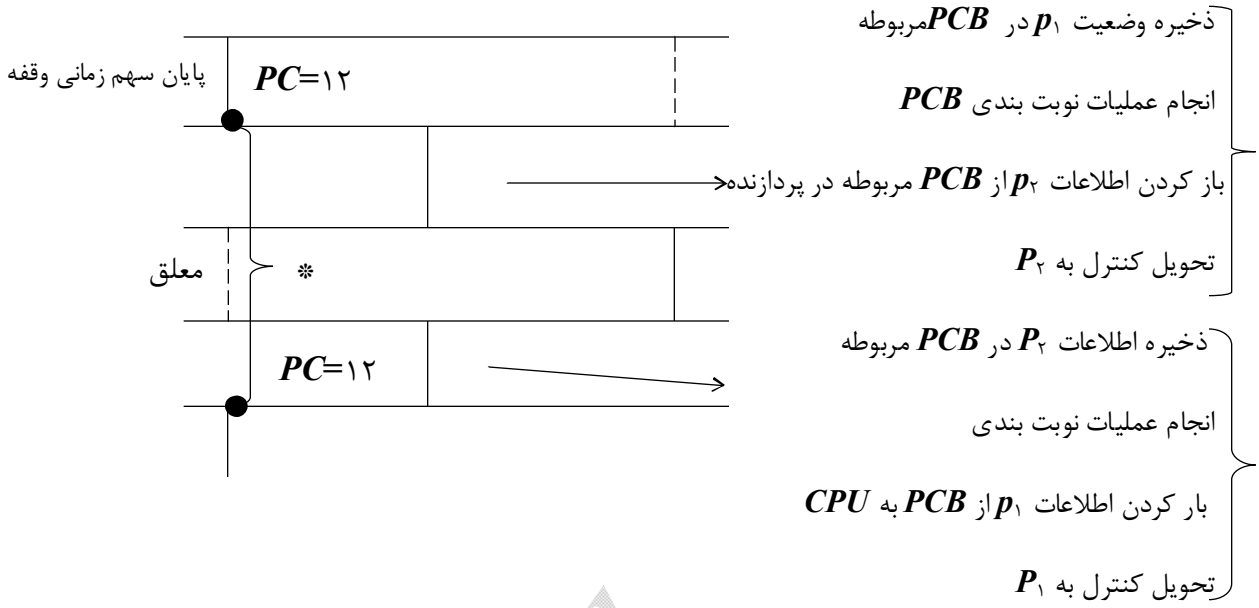
۵. اطلاعات زمان بندی پردازنده (***cpu scheduling info***)

۶. اطلاعات مدیریت حافظه (***Memory Management info***)

۷. مدیریت حسابرسی (***accounting info***)

www.mbzin.com

نکته ی مهم: ***PCB*** نماینده ی یک فرآیند نزد ***OS*** است.



* به عملیات بین دو فقط ***switchingtext*** می گوییم. (سوئیچینگ متن یا تعویض متن)

برای ورود به ***CPU*** ما باید یک صف داشته باشیم.

فرآیندها به دو دلیل از پردازنده خارج می شوند:

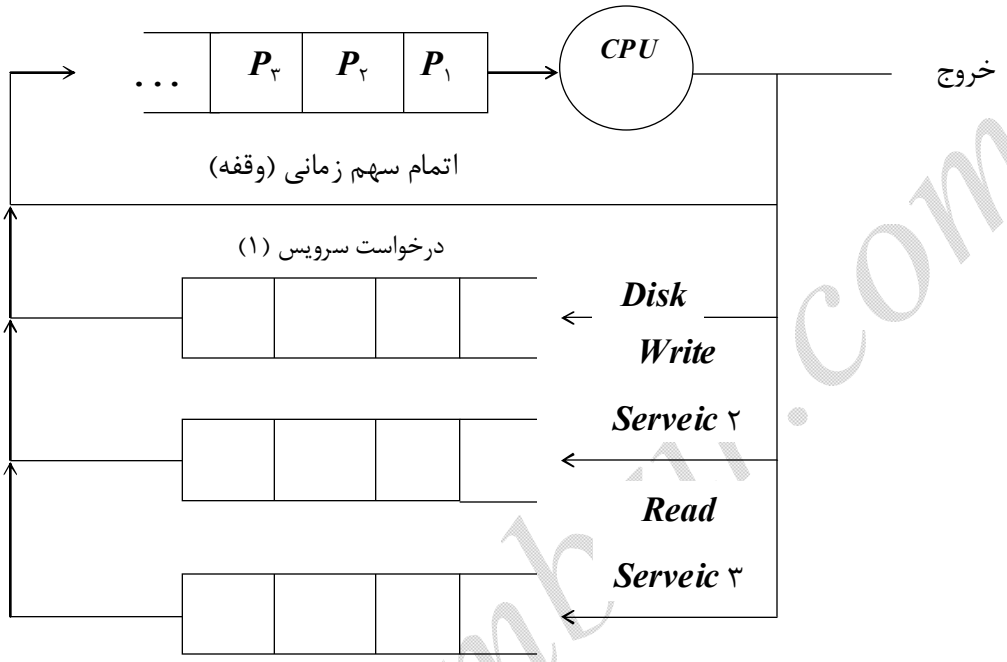
۱- اتمام سهم زمانی

۲- درخواست سرویس

www.mt

نکته : ما لزوماً صف هایمان صف های یکتا نیستند؛ یعنی فرآیندها به ازای سرویس ها مختلف صف های گوناگون دارند.

برای هر سرویس یک صف جداگانه ای داریم.

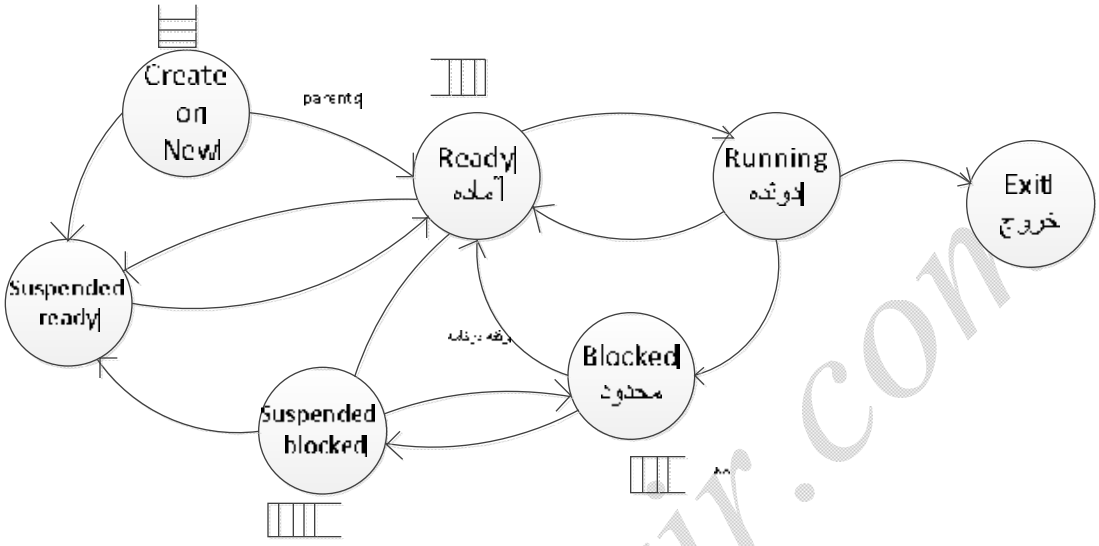


سؤال) چه چیزهایی از فرآیند در صف سرویس قرار می گیرد؟ یا فقط ***PCB*** و یا اشاره گر به ***PCB*** قرار می گیرد.

www.mbm.com

۳- مدل های فرآیند

حالات فرآیند



در ***Exit*** زمانی که فرآیند خارج می شود یا برنامه تمام شده است و یا این که سیستم عامل از ***cpu*** بیرون می اندازد.

اینجا فرآیند جدید به این صورت است که فرآیند تحت عنوان ***parent process*** می رود بیرون و ***child process*** را وارد می کند و ***child process*** زمان شروع و پایان و خیلی چیزهای دیگر را از والد به ارث می برد.

سؤال اصلی) اولین فرآیند چگونه ایجاد می شود؟ به صورت دستی ایجاد می شود. ← ***Reset*** در واژه یعنی به حالت معلولی بردن. وقتی ***Reset*** انجام می شود به اصطلاح سک ***PCB*** ساخته می شود. (***PCB*** کارت ملی فرآیند است که کلیه ی مشخصات فرآیند را از آن می توانیم بشناسیم.)

وقتی عمل ***Reset*** را انجام می دهیم این کارها هم انجام می شود:

www.mbzir.com

Boot
۱. ***program counter*** مقدار مشخص می گیرد.
۲. ***stack pointer*** مقدار مشخص می گیرد.
۳. برخی رجیسترها مقدار خاصی می گیرند.

یعنی ***Boot*** انجام می گیرد. ***Boot*** یعنی سیستم عامل از دیسک به حافظه ی اصلی منتقل می شود. بنابراین پشت صفحه ی یک ***Reset*** کردن که به صورت دستی است ایجاد فرآیند توسط فرآیند رونده (***Runn irg***) است.

سؤال) ایجاد فرآیند در خصوص تعامل کاربر با سیستم عامل مثل کیبورد و ماوس چگونه است؟

ما یک سری ***Backgraund pracess*** داریم یعنی فرآیندهای پشت صحنه که دارند کار می کنند. ***Backgraund pracess***، یعنی درخواست های کاربر را می فهمد و این باعث می شود خودش درخواست فرآیند جدید بدهد.

در نقطه ی ** یک تعداد فرآیند در ***Blacked*** صف ایستاده اند (یکی به خاطر کیبورد، یکی به خاطر ماوس و ...). در اینجا عملاً کار پردازش انجام نمی شود ولی منبع را گرفته اند (اخذ منبع) و حافظه را اشغال کرده اند. برای یک مقاطعی موقتاً این حافظه را از آن می گیریم. سیستم عامل این‌ها را در ***su spended Blocked*** قرار می دهد که این حالت در دیسک قرار دارد نه در حافظه‌ی اصلی؛ پس حافظه ی اصلی را اشغال نمی کند.

باز هم درخواست تغییر حالت با وقفه یا درخواست برنامه ی کاربردی از ***OS*** است.

سؤال) این مدت موقت باید کم باشد یا زیاد؟ اگر کم باشد در حد ***ms*** به ***Blocked*** می رود ولی اگر زیاد باشد در حد ۲و۳ ثانیه و بیشتر به ***ready*** می رود (وقفه ایجاد شده و وارد حالت آماده می‌شود.)

Suspended Ready← صف فرآیندهایی است که منتظر هستند ولی روی دیسک.

فرق ***Ready*** و ***Suspended Ready***←***Ready*** در حافظه ی ***RAM*** قرار دارد و حسن آن این است که به ***Ranning*** نزدیکتر است و صف فرآیندهایی است که وقفه ایجاد نمی شود؛ ***Suspended Ready*** در حافظه ی دیسک است و فرآیندهایی است که به مدت طولانی در صف رسیدن به ***cpu*** هستند و صف فرآیندهایی است که وقفه ایجاد می شود.

Suspendad Blocked

- سریعاً جواب گرفتیم ←***Bloked*** بعد ***ready***
- سریعاً جواب نگرفتیم ←***ready***
- بعد از مدت طولانی جواب نگرفتیم ←***suspended Ready***

فرآیند ایجاد شونده که آن را فرآیند رونده دارد تعیین می کند در دو جا می تواند برود براساس اولویت

فرآیند

- ***Ready***
- ***SuspendedReady***

حالت خود فرآیند و ضروری بودنش تعیین می کند که به ***Ready*** برود یا به ***suspended Ready***

در نقطه ی * در ***Ready*** نیز یک صف وجود دارد. فرآیندهای ته صف را که مدت زمان زیادی برای سرویس دهی می خواهند را به خاطر آزاد کردن حافظه به ***suspended Ready*** منتقل می کنیم

در همه ی قسمت های ***create***، ***suspended Ready***، ***Blocked*** و ... یک صف داریم. کنترل صف های فرآیند به وسیله ی زمان بند (***scheduler***) انجام می شود.

www.mbzir.com

۴-زمانبندها

۳ نوع زمان بند داریم:

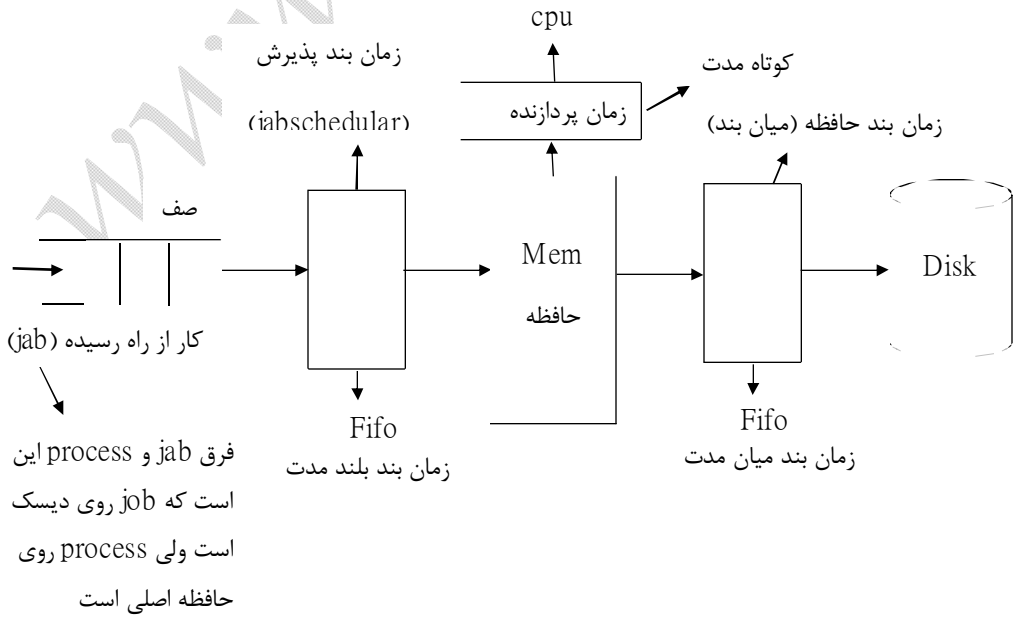
۱. زمان بند کوتاه مدت ***short time scheduler*** ← برای صف آماده است مثلاً در ***eady Queue ready***

۲. زمان بند بلند مدت ***long time scheduler*** ← ***New Queue*** در ***create***

۳. زمان بند میان مدت ***Middle time scheduler*** ← ***service queue*** در ***Blocked*** و ***uspended Blocked***

چرا دسته بندی می کنیم و براساس چه مکانیزمی این کار را انجام می دهیم؟ براساس مراجعه ی سیستم عامل به صف های است ک دسته بندی انجام می شود، یعنی سیستم عامل دائماً به صف کوتاه مدت مراجعه می کند اگر صفی در ***New*** وجود داشته باشد (که معمولاً وجود ندارد) به صورت بلند مدت خواهد بود. پس دلیل زمان بندی، مراجعه ی سیستم عامل به صف هاست.

رابطه ی ۳ زمان بند در شکل صفحه ی بعد مشخص شده است



www.mbzir.com

در بین صف و حافظه یک زمان بند پذیرش یا به اصطلاح ***job scheduler*** داریم که همان زمان بند بلند مدت است (در ***create***) یک زمان بند حافظه بین حافظه و دیسک داریم که میان مدت است و برای ***Blocked*** و ***suspended Bloched*** است. یک زمان بند هم داریم که بین ***CPU*** و حافظه است و به آن زمان بند پردازنده می گویند که کوتاه مدت بوده و برای ***Ready*** است. از جهت اهمیت و بالا بردن بهره وری و ***Rundoman*** سیستم، بهترین صف، صف کوتاه مدت است که هم کارایی سیستم و هم بهره وری پردازنده بالا می رود. پس در زمان بندی کوتاه مدت الگوریتم های مختلفی را تست می کنند. زمان بند بلند مدت و میان مدت از یک سیاست استفاده می کنند به نام ***FIFO***

FIFO یعنی سرویس دهی و نوبت حضوری است و هر کسی اول آمد اول نوبت می گیرد و اول می رود.

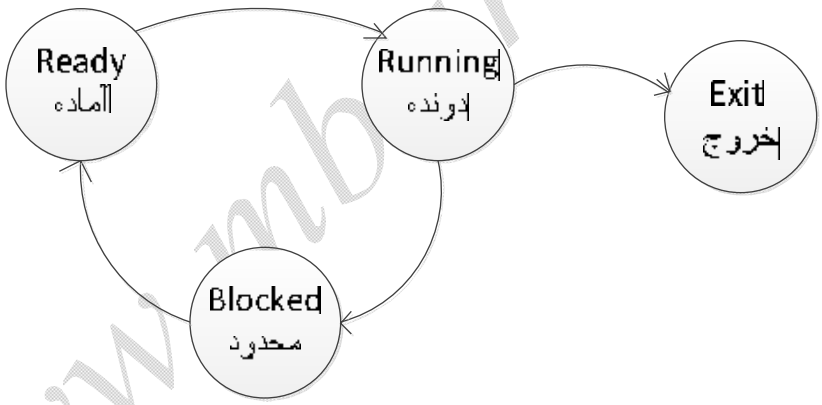
www.mbzir.com

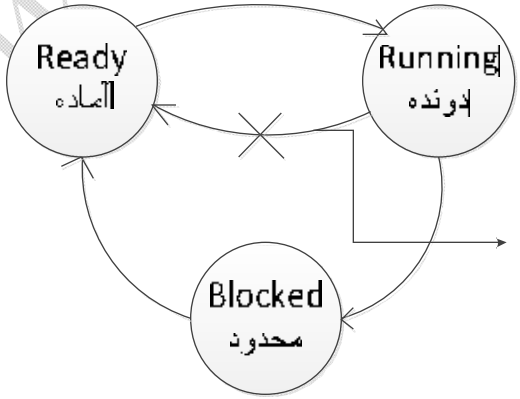
۵- الگوریتم های زمان بندی (نوبت بندی):

اگر بخواهیم سیستم عامل را از خصوص زمان بندی تقسیم کنیم به ۲ دسته تقسیم می شود:

۱. بدون قبضه کردن (پس نگرفتنی) ***Non-prcemptive*** ← فرآیند هر زمان که به اتمام رسید پردازنده را رها می کند.(زمانیکه اولویت با انجام عمل باشد استفاده میگردد.)

۲. با قبضه کردن (پس گرفتنی) ***preamptive*** ← در بازه های زمانی مختلف می توان پردازنده را از فرآیند پس گرفت.(در سیستم های تعاملی و زمانیکه اولویت با درخواست کاربر باشد استفاده میشود)





www.mbir.com

-زمان گردش یا زمان کل (*Turn Around time*):

مدت زمانی که یک فرآیند از لحظه ی ورود تا لحظه ی اتمام در سیستم طی می کند گفته می شود.

-زمان سرویس یا زمان پردازش (*service Time*):

برابر کل مدت زمانی است که فرآیند از ***cpu*** استفاده کرده است (یعنی در فاز ***Running*** بوده است).

-زمان انتظار (*waiting Time*):

برابر کل مدت زمانی است که فرآیند در صف آماده انتظار دریافت ***cpu*** را داشته است.

۶- روش های الگوریتم زمان بندی

-بدون قبضه کردن

۱. ***FCFS***

۲. ***SPN***

۳. ***HRRN***

۱.*FCFS* (***Frist come Frst Served***): هر کسی زودتر آمد زودتر هم سرویس می گیرد ← فرآیندی که زودتر ***cpu*** را درخواست کرده، زودتر آن را در اختیار می گیرد.

مثال) ۳ فرآیند P_1 و P_2 و P_3 در زمان وارد سیستم شده اند و زمان سرویس آن ها مطابق جدول رو به رو است. اگر الگوریتم نوبت بندی آن ها ***FCF*** باشد، مطلوب است میانگین زمان کار و میانگین زمان انتظار

www.mbzir.com

فرآیند	زمان ورود *Arrival time*	زمان سرویس *Service time*	زمان اتمام *FCFS*	زمان انتظار	زمان اتمام *SPN*	زمان اتمام *HRRN*
$P_۱$	۰	۱۰	۱۰	۰	۱۸	۱۸
$P_۲$	۰	۵	۱۵	۱۰	۸	۸
$P_۳$	۰	۳	۱۸	۱۵	۳	۳

زمان بند بدون قبضه کردن است. $\rightarrow P_۱$ و $P_۲$ و $P_۳$ ترتیب وارد شدن

۰ ۱۰ ۱۵ ۱۸

$P_۱$	$P_۲$	$P_۳$

$\overline{WT} = \frac{۰+۱۰+۱۵}{۳} = ۸/۳$ میانگین زمان انتظار

۰+۱۰ ۳+۱۵

$\overline{TAT} = \frac{۱۰+۱۵+۱۸}{۳} = ۱۳/۳$ میانگین زمان کل (زمان انتظار + اجرا)

حال ترتیب ورود را عوض می کنیم: $\rightarrow P_۱$ و $P_۲$ و $P_۳$

۰ ۳ ۸ ۱۸

$P_۱$	$P_۲$	$P_۳$

$$\overline{WT} = \frac{۰ + ۳ + ۸}{۳} = ۳/۷$$

$$\overline{TAT} = \frac{۳ + ۸ + ۱۸}{۳} = ۹/۷$$

ما می خواهیم $\overline{WT}$ و هم $\overline{TAT}$ کم باشد کدام ترتیب بهتر است (هر دو ***FCFS*** هستند)؟ روش و ترتیب دومی بهتر است چون ترتیب ورودی فرق کرد و فرآیندی انتخاب شد که زمانش کمتر بوده

www.mbzir.com

در ***FCFS*** کیفیت پاسخ بستگی به ترتیب ورودی دارد، ترتیب ورودی کاملاً تصادفی است، پس بنابراین کیفیت ***FCFS*** کاملاً تصادفی است.

اگر در ***FCFS*** کوتاه ترین زمان ممکن انتخاب شود بهترین خواهد بود و نتیجه ی بهتری خواهد داشت و این همان الگوریتم ***SPN*** است. پس نتیجه می گیریم الگوریتم ***SPN*** بهترین کیفیت را تضمین می کند بعنی بهترین میانگین در ***SPN*** می باشد.

۲.*SPN* (***shortest process Next***): ابتدا کوتاهترین کار انتخاب می شود. اگر ابتدا کوتاه ترین زمان را انتخاب کنیم بهترین حالت پیش خواهد آمد.

۰	۳	۸	۱۸
$P_۱$	$P_۲$	$P_۳$	

$P_۱$ ۱۰

$P_۲$ ۵

$P_۳$ ۳

مشکلات *SPN*

۱- زمان اجرای فرآیندها را بایستی از قبل بدانیم (تخمین بزنیم).

۲- همواره اولویت به فرآیندهای با زمان سرویس کم داده می شود. ← پدیده ی گرسنگی رخ می دهد (***starvation***)

۰	۳	۸	۱۰	
$P_۳$	$P_۲$	$P_۴$	$P_۵$	

$P_۴$ ۲ $P_۱$ ۱۰

$P_۱$ ۱۰ $P_۵$ ۳

این الگوریتم، الگوریتم ظالمانه ای است.

$P_۱$ دچار گرسنگی شده و امیدی به گرفتن سرویس ندارد.

www.mbzir.com

مشکل اول راه حل دارد (می توان تخمین را درست کرد) ولی مشکل دوم راه حل ندارد.

3. الگوریتم *HRRN* (*Highest Response Ratio Next*):

سیاست ما در الگوریتم ***SPN*** $\frac{1}{t_s}$ بود. یعنی اگر این کسر از همه بزرگ تر باشد در فرآیندی، آن انتخاب می شود. این کسر وقتی که مخرج کوچک تر باشد، بزرگ تر خواهد بود. $\frac{1}{t_s}$ فاکتور انتخاب

$\leftarrow \frac{\text{بیشترین میانگین زمان اجرا}}{SPN} \leftarrow$ مشکل گرسنگی

در الگوریتم ***HRRN*** $\frac{T_w}{T_s}$ را حساب می کنیم. این کسر برای هر فرآیندی که بزرگتر باشد انتخاب می شود. یعنی یا باید صورت بزرگ باشد یا مخرج کوچک باشد.

T_w کی بزرگ است ← وقتی زیاد منتظر بمانیم.

T_s کی کوچک است ← وقتی زمان سرویس کم باشد.

P_3	P_2	P_1	

P۱ ۰/۱۰ p۱
۳/۱۰

P۲ ۰/۵ p۲ ۳/۵

P۳ ۰/۳

الگوریتم ***HRRN*** بهترین نتیجه (میانگین)را تضمین نمی کند، اما مزیتی که دارد این است که این الگوریتم گرسنگی ندارد و نسبت به روش قبل روش عادلانه تری است. یعنی به بیان دیگر این الگوریتم دو پارامتر میزان انتظار بیشتر و زمان سرویس کمتر زا در انتخاب فرآیند بعدی لحاظ می کند.

www.mbzir.com

مثال) ۵ فرآیند در سیستم به نام های ***A*** و ***B*** و ***C*** و ***D*** و ***E*** موجود هستند. زمان ورود و زمان سرویس آن ها به شرح جدول رو به رو می باشد. مطلوب است میانگین زمان کل و $\frac{T_T}{t_s}$ برای الگوریتم های ***FCFS*** و ***SPN*** و ***HRRN***

Process	***A***	***B***	***C***	***D***	***E***
Arrivali time	۰	۲	۴	۶	۸
Service time	۳	۶	۴	۵	۲
FCFS ***Finish time***	۳	۳	۱۳	۱۸	۲۰
Total time	۳	۷	۹	۱۲	۱۲
$\frac{T_T}{t_s}$	$\frac{۳}{۳}=۱$	$\frac{۷}{۶}=۱/۱۷$	$\frac{۹}{۴}=۲/۲۵$	$\frac{۱۲}{۵}=۲/۴$	$\frac{۱۲}{۲}=۶$
SPN ***F.T***	۳	۹	۱۵	۲۰	۱۱
T.T	۳	۷	۱۱	۱۴	۳
$\frac{T_T}{t_s}$	$\frac{۳}{۳}=۱$	$\frac{۷}{۶}=۱/۱۷$	$\frac{۱۱}{۴}=۲/۷۵$	$\frac{۱۴}{۵}=۲/۸$	$\frac{۳}{۲}=۱/۵$
HRRN ***F.T***	۳	۹	۱۳	۲۰	۱۵
T.T	۳	۷	۹	۱۷	۷
$\frac{T_T}{t_s}$	$\frac{۳}{۳}=۱$	$\frac{۷}{۶}=۱/۱۷$	$\frac{۹}{۴}=۲/۲۵$	$\frac{۱۴}{۵}=۲/۸$	$\frac{۷}{۲}=۳/۵$

$\overline{TAT}=۸/۶$

$\bar{P}=۲/۵۶$

$\frac{T_T}{T_S}$

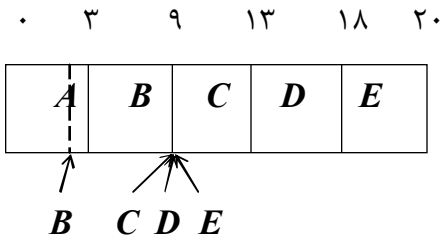
$\overline{TAT}=۷/۴$

$\bar{P}=۱/۸۴$

$\overline{TAT}=۸/۰$

$\bar{P}=۲/۱۴$

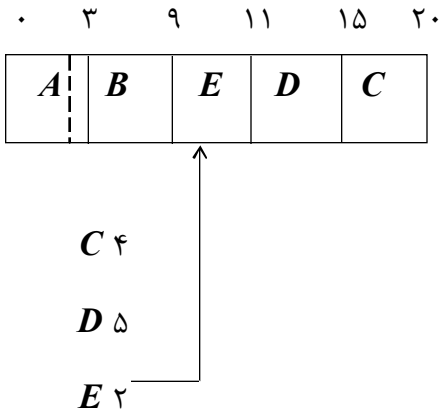
FCFS زمان بند از نوع پس گرفتنی است. (چون ***FCFS*** پس نگرفتنی است.)



www.mbzir.com

SPN

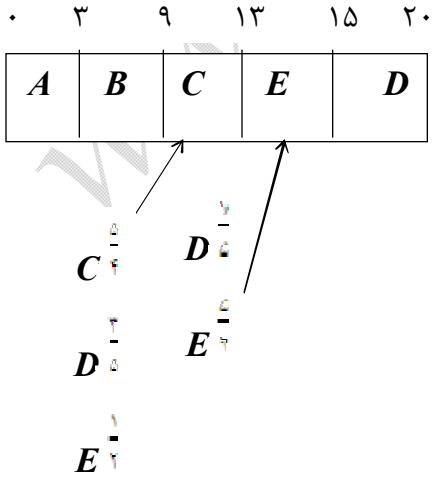
زمان بند پس نگرفتنی است و فرآیند دیگری نمی تواند وارد شود.



اول A وارد می شود. وقتی A در ۳ تمام شد فقط B که ۲ است در سیستم است، پس B وارد می‌شود. وقتی B در j تمام شد C و B و E در سیستم هستند، پس هر کدام کوچک تر بودند آن انتخاب می شود.

HRRN

نسبت $\frac{T_w}{t_s}$ در هر بار محاسبه می کنیم.



www.mbzir.com

در هر سه الگوریتم ***A*** و ***B*** اول اجرا شده اند، پس عملاً ***A*** و ***B*** تأثیری ندارندو چیزهایی که تعیین کننده بوده اند ***C*** و ***D*** و ***E*** است. پس از ***B*** به بعد، این سه فرآیند روی سیستم فشار آورده اند. بهترین میانگین زمان اجرا ***SPN*** است، بعد از آن ***HRRN*** است، بعد ***FCFS*** است که اصطلاح به ***FCFS*** عادلانه ترین الگوریتم گویند، اما میانگین زمان اجرای آن زیاد است.

-با قبضه کردن

۱. ***SRT***

۲. ***RR***

۱. ***SRT*** (***shortest Remaning Time***):کوتاهترین زمان باقی مانده

این روش دقیقاً مثل ***SPN*** است با این تفاوت که پس گرفتنی است. در ***SPN*** هر کدام که کمتر بود آن را انتخاب می کنیم. در ***SRT*** می گوییم در هر لحظه از اجرا اگر فرآیندی پیدا شد که مدت زمان کمتری برای اجرا مانده باشد، آن فرآیند انتخاب می شود در این الگوریتم در هر لحظه از اجرا اگر فرآیندی پیدا شد که زمان اجرای آن از زمان اجرای فرآیند در حال اجرا کمتر بود، می تواند پردازنده را از فرآیند پس بگیرد.

این الگوریتم یک الگوریتم تخیلی است و زیاد کاربرد ندارد (به اصطلاح گویند روش تخیلی است).

در این الگوریتم گرسنگی بدتر می شود و معمولاً روش عملی را ندارد (ارزش عملیاتی ندارد).

$$SPN + preemative = SRT$$

سؤال قبل را با روش ***SRT*** حل می کنیم

www.mbzir.com

Process	A	B	C	D	E
Finish time	۳	۹	۹	۲۰	۱۰
Total time	۳	۷	۴	۱۴	۱۲
SRT $\frac{T_T}{t_s}$	$\frac{۳}{۳} = ۱$	$\frac{۱۳}{۶} = ۲/۱۶$	$\frac{۴}{۴} = ۱$	$\frac{۱۴}{۵} = ۲/۸$	$\frac{۲}{۲} = ۶$

$\longrightarrow \overline{TAT} = ۷/۲$

$\longrightarrow \bar{P} = ۱/۵۹$

این روش از همه بهتر شد اما میزان گرسنگی آن از همه بیشتر است.

SRT

A تمام شد ***C*** تمام شد ***E*** تمام شد ***B*** تمام شد

۰ ۳ ۴ ۶ ۸ ۱۰ ۱۵ ۲۰

A	*B*	*C*	*C*	*E*	*B*	*D*

۲ ***B (۶)***

B (۶) ***C (۴)*** ***C (۲)*** ***C (۵)*** ***B (۵)***

A (۱) ***B (۵)*** ***D (۵)*** ***D (۵)***

D (۵) ***E (۲)***

2. ***RR*** (***Round Robin***)

هر فرآیند به اندازه ی یک کواشوم زمان پردازنده را در اختیار می گیرد.

فرآیندها شروع می کنند به اندازه ی ***q*** از ***cpu*** استفاده می کنند.

این روش، همان روش سیستم های سهم زمانی است.

اولویت با فرآیندی است که تازه آمده است. پس اولویت با ***B*** است بعد با ***A***؛ مگر این که مسئله بگوید زمانی که یک فرآیند آمد اولویت با فرآیند قبلی است.

R ***q***=۱

f.TE ***f.TC*** ***f.TB*** ***f.TD***

۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹ ۱۰ ۱۱ ۱۲ ۱۳ ۱۴ ۱۵ ۱۶ ۱۷ ۱۸ ۱۹ ۲۰

A	***A***	***B***	***A***	***C***	***B***	***D***	***C***	***E***	***B***	***D***	***C***	***E***	***B***	***D***	***C***	***B***	***D***	***B***	***D***

زمان غیر مفید= ***17 α***

Process		***A***	***B***	***C***	***D***	***E***
	F.T	۴	۱۸	۱۷	۲۰	۱۵
RR	***T.T***	۴	۱۶	۱۳	۱۴	۷
	$\frac{T_T}{t_s}$	$\frac{4}{3} = 1/33$	$\frac{16}{6} = 2/67$	$\frac{13}{4} = 3/25$	$\frac{14}{5} - 2/8$	$\frac{7}{2} = 3/5$

این روش بدترین روش است. اما یک نکته وجود دارد: کدام یک از روش های قبلی به فرآیندی مانند ***C*** به سرعت این فرآیند دارد پاسخ می دهد؟

FCFS← در ۹ سرویس می دهد

SPN← در ۱۱ سرویس می دهد

HRRN← در ۹ سرویس می دهد

SRT← در ۴ سرویس می دهد ← تخیلی است و ارزش عملی ندارد.

www.mbzir.com

RR ← در ۵ سرویس می دهد ← به محض ورود دارد سرویس می دهد. در ارزش ها و سیستم های عملی این روش بهتر عمل می کند. این روش بهترین زمان پاسخ را دارد (سرعت آن از همه بهتر است). اما بدترین میانگین زمان کل و بدترین میانگین سیستمی را دارد. ($\frac{T_T}{t_s}$) چون تعامل با کاربر زیاد است و کاربر منتظر سرویس دهی است این روش استفاده می شود.***q***=۴

۳	۴	۷	۱۱	۱۵	۱۷	۱۹	۲۰
A	***B***(۲)		***C***	***D***(۱)	***B***	***E***	***D***

C ***D*** ***B*** (۲)

B ***B*** ***E*** (۲)

D ***E*** ***D***(۱)

وقتی ***q*** را بزرگ می کنیم عملاً فرآیندهای کوچک نمی توانند استفاده کنند یعنی $q \to \infty$

RR→ FCFS

اگر اندازه ی کوانتوم به سمت ∞ میل کند، الگوریتم ***RR*** به سمت الگوریتم ***FCFS*** میل می کند. اگر ***q*** به سمت صفر میل می کند، الگوریتم ***RR*** به سمت اشتراک پردازنده (***cpu sharing***) میل می کند، یعنی مستقیم مساوی هر دوره ی زمانی ***CPU*** بین فرآیندهای آماده در آن دوره ی زمانی.

Process	***A***	***B***	***C***	***D***	***E***
F.T	۳	۱۷	۱۱	۲۰	۱۹
RR ***T.T***	۳	۱۵	۷	۱۴	۱۱
$\frac{T_T}{t_s}$	$\frac{۳}{۳} = ۱$	$\frac{۱۵}{۶} = ۲/۵$	$\frac{۷}{۴} = ۱/۷۵$	$\frac{۱۴}{۵} = ۲/۸$	$\frac{۶}{۲} = ۵/۵$

$\overline{TAT} = ۱۰$

$\bar{P} = ۲/۷۱$

www.mbzir.com

$\propto \blacksquare\ context\ switching\ time$

$\propto \uparrow = over\ head \uparrow$

زمان مفید ← زمانی که پردازنده در اختیار فرآیند است نه سیستم عامل.

زمان غیر مفید ← زمانی که پردازنده در اختیار سیستم عامل می باشد.

www.mbzir.com

سوالات فصل سوم

1- کدامیک از زمانبند های روبرو کمترین TAT را دارد ؟

الف ***(*** sjf

ب***(*** SRT =SJF + PREMACTIVE

ج***(***FCFS

د***(***RRبا TQ=***1***

	AT	***ST***
P۱	۰	۸
P۲	۳	۴
P۳	۲	۹
P۴	۳	۵

۲- تعویض بین فرایند ها ***ms***۵ و در روش ***RR*** کوانتوم ***ms***۲۰ است . متوسط زمان پاسخگویی و زمان انتظار ؟

	زمان اجرا	زمان ورود	زمان اتمام	RT	WT
A	۴۰	۰	۱۲۰	۱۲۰	۸۰
B	۲۰	۰	۴۵	۴۵	۲۵
C	۵۰	۰	۱۷۵	۱۷۵	۱۲۵
D	۳۰	۰	۱۶۰	۱۶۰	۱۳۰

۰ ۲۵ ۵۰ ۷۵ ۱۰۰ ۱۲۵ ۱۵۰

A(۲۰)		B(۰)		C(۳۰)		D(۱۰)	A(۰)		C(۱۰)		D(۰)	C(۰)

۲۰ ۴۵ ۷۰ ۹۵ ۱۲۰ ۱۴۵ ۱۶۰ ۱۷۵

$\overline{TAT}$=۱۲۰+٤٥+۱۷٥+۱٦۰/٤

$\overline{WT}$=۸۰+۲٥+۱۲٥+۱۳۰/٤

www.mbzir.com

۳-۳ فرایند ***P۱,P۲,P۳*** با زمان ورود و اجرا روبرو در نظر گرفته شده ، متوسط زمان پاسخگویی؟

Sjf پس نگرفتنی

FIFO پس نگرفتنی

SRT پس گفتنی

Prionty پس نگرفتنی (اونیکی که اولویتش بالاتره انتخاب میشه)

فرایند	اولویت	ورود	اجرا	RT(sjf)	RT(FIFO)	RT(SRT)	RT(prionty)
P۱	۲	T	۴	۶	۴	۷	۴
P۲	۰	T	۲	۲	۶	۲	۵
P۳	۱	T+۳	۱	۷	۷	۴	۷

Sjf

P۲	P۱	P۳

t ***T***+۲ ***T***+۶ ***T***+۷

SRT

P۲	P۱	P۳	P۱

t ***T***+۲ ***T***+۳ ***T***+۴ ***T***+۷

Prionty

P۱	P۳	P۲

t ***T***+۴ ***T***+۵ ***T***+۷

www.mbzir.com

FIFO

P۱	P۲	P۳

t ***T***+۴ ***T***+۶ ***T***+۷

۴- زمان پاسخگویی در الگوریتم ***FCFS*** یا ***RR*** چیست ؟ ***TQ***=۱

(اولویت با فرایند قبلی است)

زمان اتمام –زمان ورود= زمان پاسخ

P	AT	ST	FT (FCFS)	FT(RR)	RT(FCFS)	RT(RR)
A	۰	۳	۳	۴	۳	۴
B	۱	۳	۶	۷	۵	۶
C	۴	۳	۹	۱۰	۵	۶
D	۶	۲	۱۱	۱۱	۵	۵

RR

۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹ ۱۰ ۱۱

A	A	B	A	B	C	B	C	D	C	D

FCFS

۰ ۳ ۶ ۹ ۱

A	B	C	D

www.mbzir.com

۵- در سیستم اشتراک زمانی با زمانبند ***RR*** تایم اسلایس ***۱ms*** ،۵ فرایند در کار کردن هستند میانگین زمان پاسخ ؟

زمان اتمام –زمان ورود= زمان پاسخ

زمان پاسخ – زمان سرویس =زمان انتظار

	AT	ST	FT
P۱	۰	۸	۲۱
P۲	۱	۱	۳
P۳	۲	۴	۱۷
P۴	۳	۶	۲۱
P۵	۴	۲	۱۱

۰ ۱ ۳ ۷ ۱۱ ۱۴ ۱۷ ۲۱

P۱	P۱, p۲	P۱,p۳,p۴,p۵	P۱,p۳,p۴,p۵	P۱,p۳,p۴	P۱,p۳,p۴	P۱,p۴

۶- ۵کار در انتظار اجرا به سر میبرند ترتیب اجرا این کار ها چگونه است؟ تا میانگین زمان برگشت ***(TAT)*** حداقل باشد.

میانگین زمان پاسخ حداقل باشد ***sjf*** است .

job	s.t
۱	۹
۲	۶
۳ ۴	۳ ۵
۵	x

X<۳: ۵،۳،۴،۲،۱

۳<x<۵: ۳،۵،۴،۲،۱

۵<x<۶: ۳،۴،۵،۲،۱

۶<x<۹: ۳،۴،۲،۵،۱

x>۹: ۳،۴،۲،۱،۵

www.mbzir.com

۷-سیستم یک پردازنده روش زمانبند ***HRRN*** است .زمان پاسخگویی این زمانبند با استفاده از جدول زیر بدست آورید :

فرایند	اجرا	ورود	اتمام	پاسخ
P*1*	*6*	*0*	*6*	*6*
P*2*	*7*	*1*	*21*	*20*
P*3*	*3*	*3*	*9*	*6*
P*4*	*3*	*5*	*12*	*7*
P*5*	*2*	*7*	*14*	*7*

۰	۶	۹	۱۲	۱۴	۲۱
P*1*	P*3*	P*4*	P*5*	P*2*	

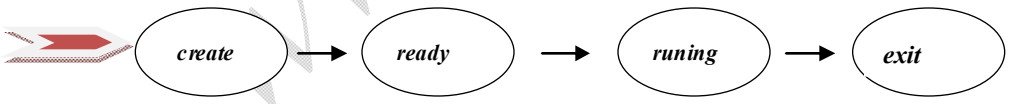
P۲ ۵/۷ ***P***۲ ۶/۷ ***P***۲ ۱۱/۷

P۳ ۳/۳ ***P***۴ ۴/۳ ***P***۵ ۷/۲

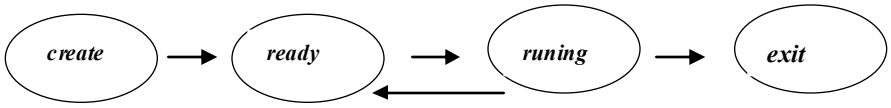
P۴ ۱/۳ ***P***۵ ۲/۲

۸- اگرزمانبند پردازنده ***FCFS*** باشد کدام شکل درست است؟

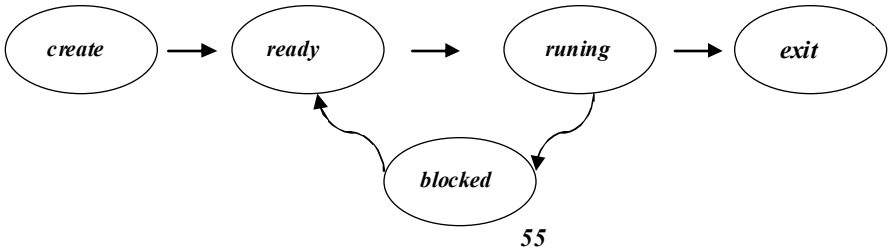
الف)



ب)



ج)



۹-سیستم تک پردازنده با زمانبند باز خورد چند سطحی با مشخصات زیر که

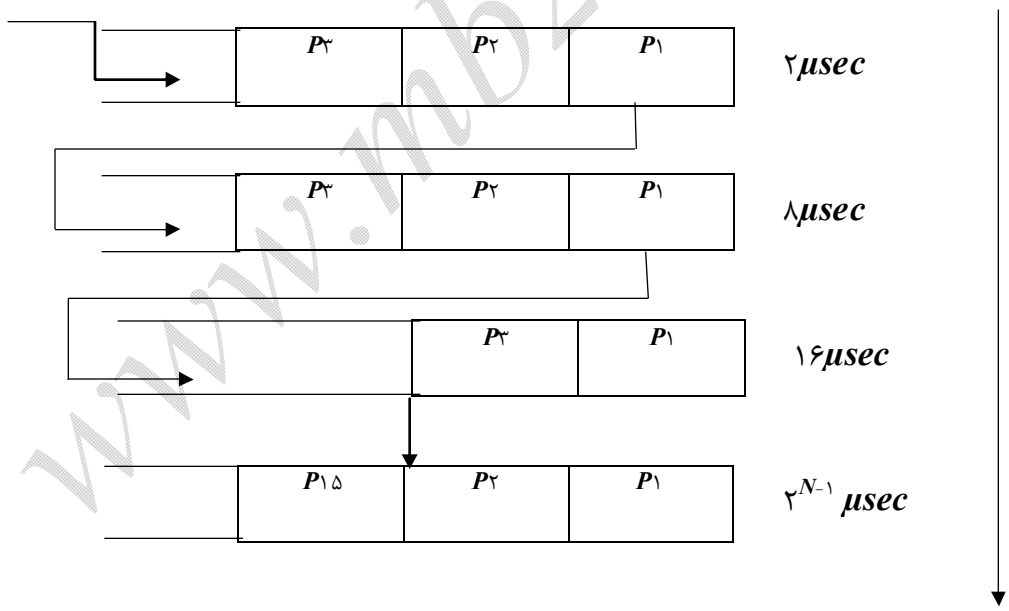
زمانبند سطح اول (صف ۰)با تکه ی زمانی ***۸μsec***

زمانبند سطح دوم (صف ۱)یا تکه ی زمانی ***۱۶μsec***

زمانبند سطح سوم (صف آخر)به صورت ***FCFS*** میباشد .

میانگین زمان برگشت و میانگین زمان انتظار در صورتیکه ۶ فرایند با مشخصات روبرو موجود باشد چیست؟

یک زمانبند پیچیده و کاربردی داریم تحت عنوان ***MLFQ*** یعنی زمانبند آن چند سطحی است.



فرایند	*A.T*	*S.T*	*F.T*	*W.T*	*R.T*
۱	۰	۴	۴	۰	۴
۲	۰	۷	۱۱	۴	۱۱

www.mbzir.com

۳	۰	۱۲	۴۷	۳۵	۴۷
۴	۰	۲۰	۵۹	۳۹	۵۹
۵	۰	۲۵	۹۲	۶۸	۹۲
۶	۰	۳۰	۹۸	۶۸	۹۸

اولویت کمتر

زمان اجرا بیشتر

۰ ۴ ۱۱ ۱۹ ۲۷ ۳۵ ۴۳ ۴۷ ۵۹ ۷۵ ۹۱ ۹۲ ۹۸

P۱(۰)	***P۲(۰)***	***P۳(۴)***	***P۴(۱۲)***	***P۵(۱۷)***	***P۶(۲۲)***	***P۳(۰)***	***P۴(۰)***	***P۵(۱)***	***P۶(۶)***	***P۵(۰)***	***P۶(۰)***

هر فرایندی که می آید صف اول به آن ***μsec*** ۸ زمان می دهد.بعد وقتی فرایند آمد در صف دوم به آن ***μsec*** ۱۶ زمان میدهد.

www.mbzir.com

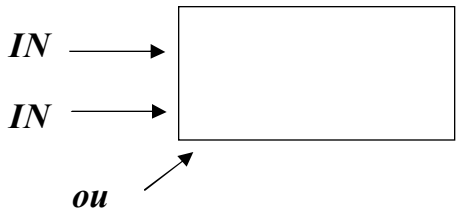
فصل چهارم

۱- *Inter process communication* یا *process synchronication*

منابع محدود اشتراکی هستند (مثل کلاس درس که یک منبع استراکی بین فرآیندهاست)، پس باید مدیریت شوند، یعنی کسی باید باشد تا بداند که چه فرآیندی از چه منبعی استفاده می کند و برا یچه زمانی است.

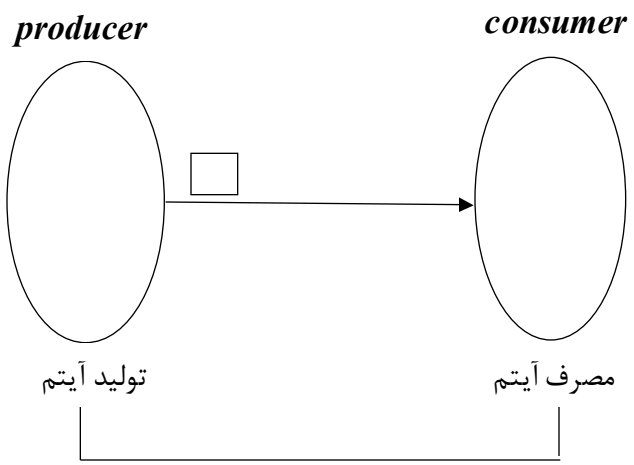
در صورت عدم مدیریت اغتشاش (هرج و مرج) پدید می آید.

اگر بخواهسم از پورت مشترک استفاده کنیم بایستی هماهنگی انجام دهیم. اتفاقی که در هرج و مرج می افتد این است که اطلاعات از بین می رود.



مثال) مصرف کننده و تولید کننده با بافر مشترک***producer & consumer***

www.mbzir.com



شرط کار این دو با هم ← سرعت هر دو یکسان باشد

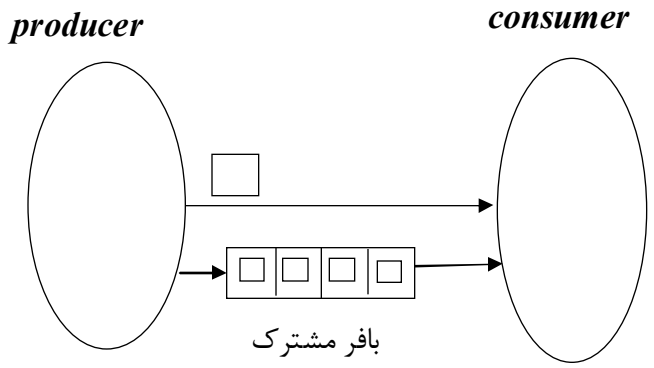
حال اگر نرخ تولید بالا برود طبیعی است که در سمت ***consumer*** مصرف نخواهد شد و برعکس.

در این حالت دو فرآیند باید هماهنگ (***synchronize***) باشند، یعنی عدم هماهنگی دو فرآیند باعث می شود که به جواب نرسیم.

هماهنگی به ازای هر آیتم تعریف می شود.

در ***level*** بعدی فرض کنیم یک بافر (بافر مشترک) قرار می دهیم:

* لزومی ندارد که سرعت فرآیند تولید کننده و مصرف کننده یکسان باشد.



۱. در تولید کننده ← شرط پر بودن بافر مهم است ← پر نیست، شروع به تولید کند.

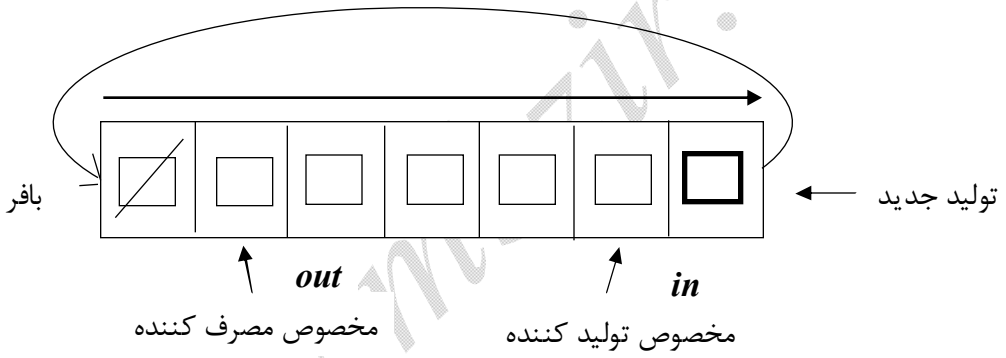
www.mbzir.com

۲. در مصرف کننده ← شرط خالی نبودن بافر مهم است ← خالی باشد دیگر نمی تواند مصرف کند.

* همگامی بهتر شد، یعنی به ازای هر آیتم نیست بلکه به ازای تعداد خانه های بافر مشترک است.

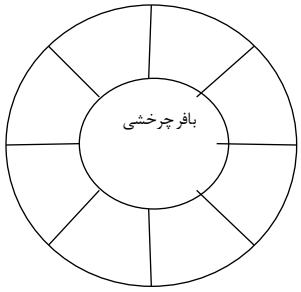
بافر مشترک همان منبع اشتراکی و محدود بین دو تا فرآیند است.

می خواهیم یک مدلی از بافر را پیاده سازی کنیم:



بافر ۲ تا ***pointer*** دارد: ***in*** که مخصوص تولید کننده است؛ ***out*** که مخصوص مصرف کننده است.

برای این که بدانیم در آخر بافر هخالی است یا نه ←



اما ابهام وجود خواهد داشت → ***in=out*** آزمایش خالی بودن بافر

in=out آزمایش پر بودن بافر

↓ پس

in=out ← واقعاً بافر خالی است.

in=out ← برای پر بودن باید طرح دیگری داشته باشیم.

in+۱=out ← ***in*** یک واحد کمتر از ***out*** باشد → حالت پر کردن

in=out → حالت خالی بودن

(in+۱) mod N =out در حالت پر بودن اگر بافر طول ***N*** داشته باشد باید در نظر بگیریم که

حال می خواهیم قطعه کدی که سمت گیرنده و فروشنده است را نوشته و ببینیم پشت صحنه چه اتفاقی می افتد؛ هماهنگی بین فرآیندها یعنی کالبد شکافی کد درون فرآیند.

پس قطعه کد تولید کننده و مصرف کننده را می نویسیم:

```
Type defitem … ;
Item Buffer [N];
int in ,out ,
    producer                          consumer
    while (True){                     while (True){
        Next p = produce-item ();         While (in==out) Do;
        While ((in+۱)Mod N==out)Do;       Next c = Buffer [out];
        Buffer [in]=nextp;                out=(out+۱)Mod N;
```

www.mbzir.com

```
in=(in+1)Mod N;                    consumer-item (Next c);
}                                  }
```

در ***producer*** تولید کننده اول تولید کرده و بعد ظرط را چک می کند.

در ***consumer*** مصرف کننده اول چک می کند و بعد مصرف می کند.

در این جا یک مشکلی وجود دارد و آن این است که همیشه یک خانه خالی می ماند (عملاً خیلی اتفاق نمی افتد که خانه هایش پر باشد).در نتیجه به جای این که با ***in*** و ***out*** کار کنیم با ***counter*** کار می کنیم.

Counter مقدار آیتم بافر را نگه می دارد.

Counter++ ← تولید کننده از این استفاده یم کند.

Counter -- ← مصرف کننده از این استفاده می کند.

قطعه کد ما این گونه خواهد شد:

```
producer                           consumer
while (True){                      while (True){
    Next p = produce-item ();        While (counter==0) Do;
    While (counter==N)Do;              Next c = Buffer [out];
    Buffer [in]=nextp;                out=(out+1)Mod N;
    in=(in+1)Mod N;                   counter--;
    counter++;                        consumer-item (Next c) ;
}                                  }
```

ما تا به حال با ۲ ***Pointer*** کار می کردیم: ***in*** و ***out*** و با این دو بافر را پیمایش می کردیم. هر بار حد فاصل بین ***in*** و ***out*** می شود تعداد ***item*** ها. ایراد کار ۱ خانه بود که این ۱ خانه در بین هزاران هزار بافر خیلی زیاد می شود؛ پس برای بهینه بودن در هزاران هزار بافر می آییم از ***counter*** استفاده می کنیم که تعداد آیتم های بافر را نگه می دارد. بعد به جای شرط پر بودن که شرط می کرد اگر پر است منتظر باش می نویسیم ***counter==N*** و به جای شرط خالی بودن ***counter==۰*** می گذاریم، اما همواره باید ***counter*** را جلو و عقب ببرسم؛ کم شدن ***counter*** به معنای مصرف است که اگر ***out*** را یکی جلو ببریم یم توانیم یکی از ***counter*** کم کنیم. زیاد شدن ***counter*** هم به معنای تولید است که اگر ***in*** را یکی جلو ببریم می توانیم ***counter*** را یکی زیاد کنیم.

این برنامه باز هم درست کار نمی کند (بعد از آمدن ***counter***)، صحت کار لاین برنامه با مشکل مواجه است؛ دلیل آن دو تا متغیر ***counter***ای است که اضافه کرده ایم.

Counter++ و ***counter—*** در ***cpu*** اجرا نمی شوند.

```
producer
while (True){
    Next p = produce-item ();
    While (counter==N)Do;
    Buffer [in]=nextp;

    in=(in+۱)Mod N:
    counter++; →  { Reg۱= counter;
                    Reg۱= Reg۱+۱;
                    counter = Reg۱;
```

```
consumer
while (True){
    While (counter==۰) Do;
    Next c = Buffer [out];
    out=(out+۱)Mod N;

    counter--; →  { Reg۲= counter:
                    Reg۲= Reg۲-۱;
                    counter = Reg۲;
    consumer-item (Next c) ;
```

} }

یک ***Macro Instruction*** شکسته شد به ۳ تا ***Register***

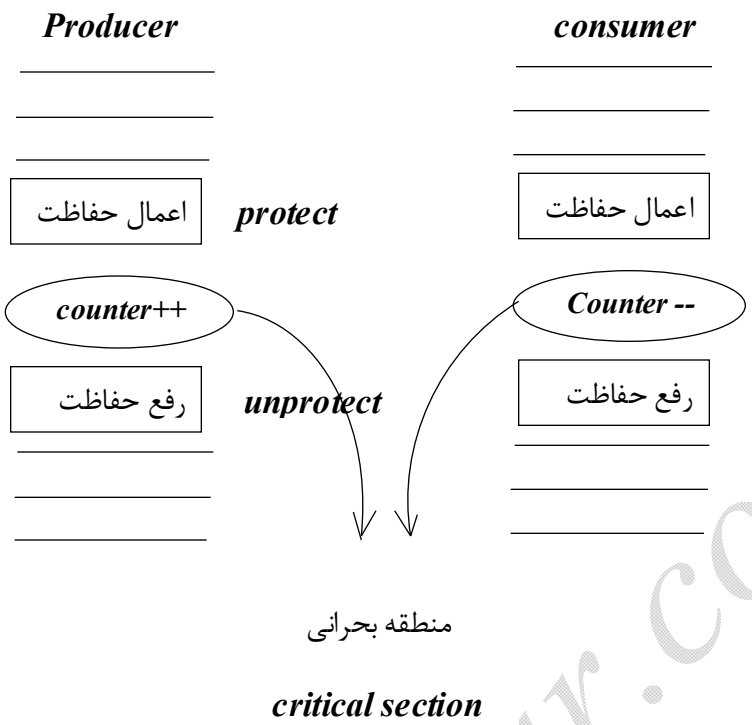
وقتی منبع مشترک و محدود باشد بایستی مدیریت شود. عدم مدیریت می شود هرج و مرج. اما چرا این مشکل پیش آمد؟ دو فرآیند تولید کننده و مصرف کننده از یک منبع مشترک که متغیر ***counter*** بود استفاده کردند.

برای جلوگیری از این کار با استفاده از حفاظ های امنیتی از منبع مشترک، مدیریت می کنیم.

بهترین روش این است که قبل از متغیر مشترک اعمال حفاظت را انجام دهیم و بعد از استفاده از متغیر مشترک رفع حفاظت می کنیم (***unprotect , protect***).

منطقه ی بین این دو خیلی خیلی حساس است. چون ممکن است وقتی آن منطقه را اجرا می کنیم سهم زمانی تمام شود و برویم بیرون.

www.mbzir.com



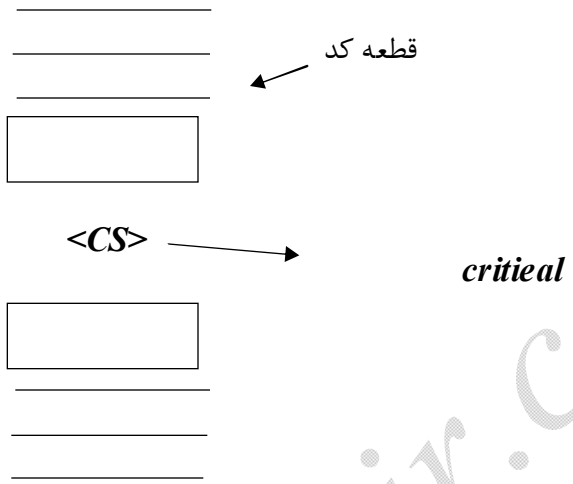
پس مکانیزم محافظت این گونه است که در زمانی که یک فرآیند با یک منبع مشترک کار دارد، هیچ فرآیند مجاور دیگری نمی تواند با آن منبع کار داشته باشد.

این مسئله که ما می آییم محافظت کنیم و در منطقه ی بحرانی اجازه ندهیم در فرآیند همزمان اجرا شوند را ***Mutual Exclusion*** انحصار متقابل گویند. یعنی فرآیندها نسبت به هم انحصاری باشند و اگر یک فرآیندی در منطزقه ی بحرانی بوذد هیچ فرآیند دیگری اجازه ی ورود به منطقه ی بحرانی را نداشته باشد.

مسئله انحصار متقابل را معانی متفاوتی کرده اند. یک معنی اش همان انحصار متقابلی است که در بالا گفتیم؛ یک معنی اش دو به دو ناسازگار است، یک معنی اش مانعه الجمعی است. ناحیه ای که ما می خواهیم انحصار متقابل را در آ؛ن رعایت کنیم و اجرایی نماییم را ناحیه ی بحرانی گویند. به مسائلی که در این ناحیه ی انحصار متقابل داریم به اصطلاح ***critical section problem***

www.mbzir.com

گویند؛ یعنی این تیپ مسائل هدفشان این است که قبل از ورود و پس از خروج چه سیاست های حفاظتی ای را اعمال کنیم (چه کدی بنویسیم؟ چه کاری کنیم که خوب کار کند؟)



www.mbzir.com

۲- مسئله ناحیه ی بحرانی

فرض کنید n فرآیند داریم $\{p_1, p_2, \ldots, p_n\}$ که هر کدام در ناحیه ی بحرانی هستند. به دنبال راه حلی هستیم که هر گاه فرآیند در ناحیه ی بحرانی باشد، هیچ فرآیند دیگری اجازه ی اجرای ناحیه ی بحرانی خود را نداشته باشد؛ به بیان دیگر استفاده از ناحیه ی بحرانی را انحصاری کنیم؟

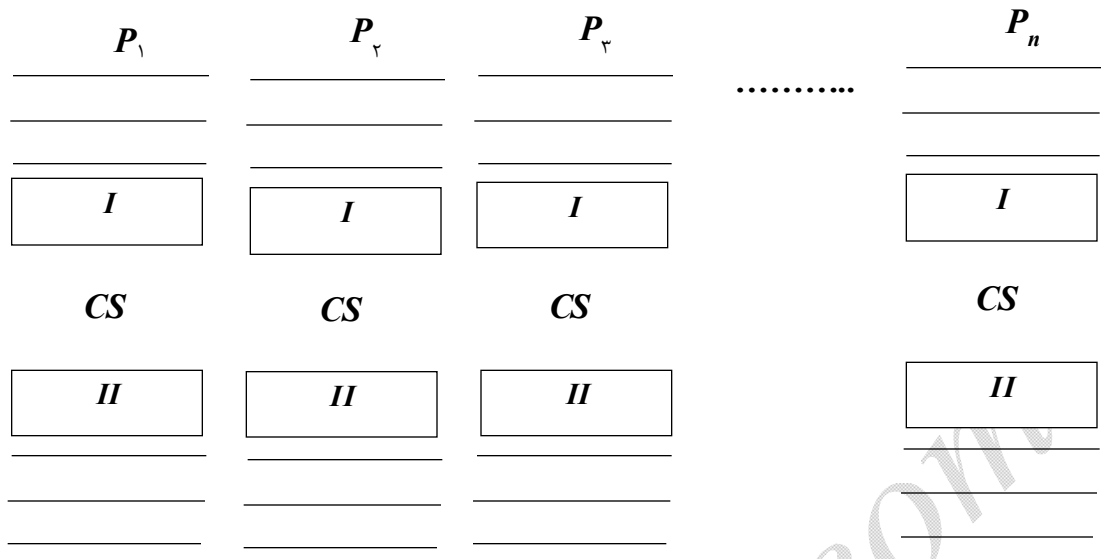
/*program Mutual Exclusionn*/

Intn /*Number for process*/

```
void p(inti){
    while (true){
                    enter-critical (i);     ← شرط ورود یا اعمال حفاظت
                    critical section [i];   ← ناحیه ی بحرانی
                    exit critical (i);      ← شرط خروج (رفع حفاظت)
                    Noncritical-section (i); ← ناحیه ی غیر بحرانی
            }
    }
Void main ()
  {
  Parbegin (P(۱),P(۲), … , P(n));
    }
```

با توجه به مفروضات، صورت مسئله در این شکل ↓

www.mbzir.com



کل مسائل ناحیه ی بحرانی به این امر می پردازند که در قسمکت قبل و بعد از ناحیه ی بحرانی چه کدی نوشته شود.

۳- مشخصات راه حل های ناحیه ی بحرانی:

۱. تضمین انحصار متقابل ***Mutual Exclusion***

۲. تضمین انتظار محدود ***Bounded Waiting***

۳. تضمین پیشرفت ***progress***

در مسائل ناحیه ی بحرانی ما معمولاً از دو فرآیند شروع کرده و به ***n*** فرآیند تعمیم می دهیم. این مسائل معمولاً در دو فرآیند خودشان را نشان می دهند؛ چون پدیده ی فرآیند یک ساختار ماده ای دارد بنابراین پیش بینی رفتارهای آینده ی آن زیاد پیچیده نیست.

۴- مسائل ناحیه ی بحرانی:

www.mbzir.com

۱.تلاش اول:

تلاش اول برای آریالای پترسن است؛ یکگ الگوریتمی است برای قبل و بعد از ورود به ناحیه ی بحرانی.

$P_۰$ $\xrightarrow[\text{متغیر مشترک بولین}]{Turn=0/1}$ $P_۱$

```
while (True){                 while (True){
    While (True !=۰);             While (True !=۱)  ← (۲) پایان سهم زمانی
    Critical-section (۰)          Critical-section (۱);
    Turn=۱;                       Turn=۰;
    Non-critical section (۰);     Non-critical section (۱);
}                             }
```

(۱)→ پایان سهم زمانی

باید بر پایه ی سناویر جلو ببریم و شرایط و ضوابط باید طوری باشند که یکی از ۳ شرط صفحه ی قبل نقض شود.(انحصار متقابل وانتظار محدود رعایت میشود)

پیشرفت را رعایت نمیکند: فرض کنیم ***turn***=۱ است (نوبت برای $P_۱$ است)

$P_۰$ $Turn = 1$ $P_۱$

```
while (True){                 while (True){
    While (True !=۰);             While (True != ۱);
    Critical-section (۰)          Critical-section (۱);
    Turn=۱;                       Turn=۰;
    Non-critical section (۰);     Non-critical section (۱);
}                             }
```

-۲ →

۰ →

-۱ →

پایان سهم زمانی →}

۲.تلاش دوم:

www.mbzir.com

سعی می کنیم مشکل راه حل اول را نداشته باشیم از متغیر ***flag*** استفاده می شود.

$P_۰$	$P_۱$
while (True){	***while (True){***
۱ ***flag[۰]= True;***	۲ ***flag[۱]= True;***
۳ ***while (flag [۱]);***	۴ ***while (flag [۰]);*** ۲
۱ ***Critical-section (۰)***	***Critical-section (۱);***
۳ ***flag[۰]= false;***	***flag[۱]= false;***
Non-critical section (۰);	***Non-critical section (۱);***
}	}

زمانی که هردو فرایند در مرحله ی اول ***T*** باشد باعث می شود که به بنبست بیافتیم و خروجی این بنبست گرسنگی می باشد.

سوال : آیا این مسئله شرط پیشرفت را رعایت میکند ؟

www.mbzir.com

۳.تلاش سوم:

ترکیب تلاش اول و دوم با استفاده از ***Turn*** و ***flag*** پیش فرض ***falg***ها همیشه ***false*** است.

$P_۰$	***fla***	۰	۱	$P_۱$
Void p(۰){	***false***	***T***	***T***	***Void p(۱){***
while (True){	***Tur***	۰	۱	***while (True){***

```
flag[۰]= True;                    ۲  flag[۱]= True;
Turn=۱;                           ۴  n=۰;
۳
while (flag [۱]& Turn =۱);        while (flag [۰] & Turn =۰);   ۶
۵  Critical-section (۰)           Critical-section (۱);
۷  flag[۰]= false;                flag[۱]= false;
Non-critical section (۰);         Non-critical section (۱);
}}                                }}
Boolean flag [۲]                  void main (){
int Turn                            flag[۰]=false
                                    flag[۱]=false
                                    parbegin (P۰ , P۱)
                                  }
```

www.mbzir.com

اعلام آمادگی

شماره ی نوبت

ترکیب ***Turn*** و ***flag*** است.

این روش هر سه شرط را تحقق می کند.

این روش با دادن نوبت به همدیگر تضمین می کند که یک فرآیند یکبار بیشتر معطل نمی ماند. برخلاف تلاش دوم روش حریصانه ای نیست. چون نوبت را دارد تعارف می کند، اما تلاش دوم حریصانه بود چرا که فوری دست خود را بلند کرده و خواست که وارد ناحیه ی بحرانی شود.

www.mbzir.com

۵- مروری بر جلسه قبل :

همانطور که گفتیم اگر منبع اشتراکی محدود باشد در آن صورت تصادف پیش می آید .

سپس به بحث راجب مسئله تولید کنندگان ، مصرف کنندگان پرداختیم.

بافر معرفی کردیم و بعد از آن به معرفی بافر چرخشی پرداختیم که اشاره گر ***in*** برای تولید کننده و اشاره گر ***out*** برای مصرف کننده بود .

سپس گفتیم فاصله ی بین ***in*** و ***out*** تعداد آیتم ها را نشان می دهد .

و چون در حالت بافر چرخشی همیشه یک خانه خالی می ماند از ***counter*** استفاده کردیم که متغییر مشترک در این دوتا بود . و گفتیم که برای تولید کننده باید ***counter ++*** بشود و برای مصرف کننده ***counter --***.

همانطور که ملاحظه کردیم ***counter*** خود شامل ***microinstruction*** هایی می باشد . سپس برای این حالت هم سناریویی طراحی کردیم که بد بود و داستان به اینجا ختم شد که ***counter*** دو مقدار پیدا کرده و در این حالت باز هم یک خانه بلااستفاده می ماند .

سپس ناحیه ی بحرانی را مطرح کردیم که دیدیم باید برای آن بحثی به نام انحصار متقابل را مطرح کنیم و باید از این ناحیه بحرانی حفاظت کنیم .

سپس یک سری مسائل را مطرح کردیم که می گفتند برای حفاظت و عدم حفاظت چه کدی بنویسیم .

و گفتیم که یک راه حل باید سه شرط داشته باشد تا به عنوان راه حل قابل قبول پذیرفته شود :

۱. انحصار متقابل

۲ . انتظار محدود

۳. تضمین پیشرفت

www.mbzir.com

سه تلاش را بیان نمودیم :

۱. تلاش اول : از ***Trun*** استفاده کردیم که مقدارش ***true*** یا ***false***بود .

۲. تلاش دوم : از ***flage*** استفاده کردیم که به اصطلاح دست بالا بردن و اجازه گرفتن اطلاق می شد .

۳. تلاش سوم : ***turn*** و ***flage*** را ترکیب نمودیم . که در این حالت هیچ ایرادی وجود نداشت

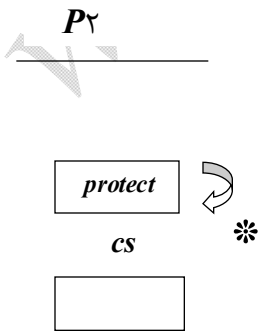
دو نکته در راه حل های قبلی وجود داشت :

۱. در راه حل های قبلی کم ترین فرض در مورد دستورالعملها وجود داشت .

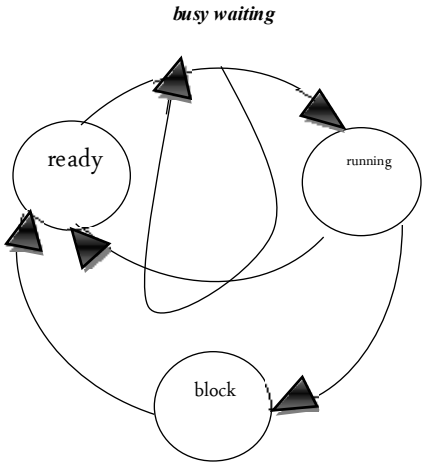
۲. راه حل های ارائه شده بیشتر بر مبنای ***busy waiting***(انتظار مشغولی) بودند . و تلاش ها نهایت کاری که می کردند این بود که اگر می خواهید وارد ناحیه بحرانی شوید در نقطه *(شکل ۲) منتظر باشیم باید این انتظار به قدری باشد که سهم زمانی تمام شود و پردازنده در دست فرآیند دیگری قرار بگیرد سپس وارد ***running*** شود .

ایراد : ایراد این مسئله این است که فرآیند، کلی زمان صرف می کند تا به ***running*** برسد ، حالا که به مرحله ***ready***رسیده است برایش وقفه صادر می شود .

بهترین روش این است که ***busy waiting***را به ***block waiting*** تبدیل شود .



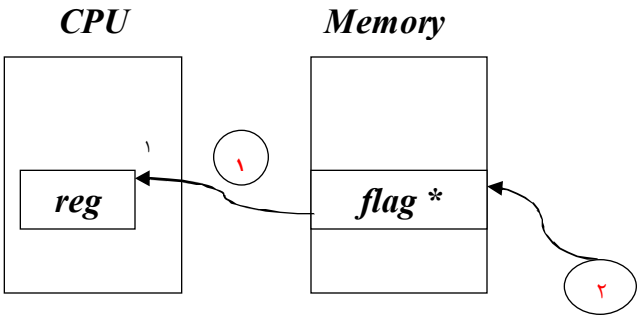
شکل ۲



شکل ۱

www.mbzir.com

۶-استفاده از TSL (*Test and Set lock*):



فرض کنیم یک پردازنده و یک حافظه داریم در مرحله اول خانه ی * از حافظه را که ۱ است به یکی از ***reg*** های ***CPU*** منتقل می کند سپس یک متغییر دیگر را از بیرون وارد خانه ی آن حافظه می کند .

enter : TSL (reg ۱ , flag)

CMP(reg ۱ , ۰)

JNZ enter (jump not zero)

<cs >

exit : move flag , ۰

قطعه کد ***enter*** سه کد دارد که قبل از ورود به ناحیه بحرانی اجرا می شود .

قطعه کد ***exit*** هم بعد از ناحیه بحرانی اجرا می شود .

دستور ***TSL*** محتویات درون حافظه را در یکی از ثبات های پردازنده میی نویسد و همزمان یک عدد غیر صفر در آن خانه ی حافظه وارد می کند.

نکته : این دو عمل که ***TSL*** انجام می دهد اتمیک هستند . یعنی جدایی ناپذیرند ، یعنی هیچگاه نمی توان وسط کار فرآیند را پس بگیرد .

www.mbzir.com

P۱	P۲
----------	----------
Enter : TSL (reg ۱ , flag)	Enter : TSL (reg ۱ , flag)
int →	
CMP (reg۱ , ۰)	CMP (reg۱ , ۰)
JNZ enter	JNZ enter
<CS>	<CS>
Exit : move flag , ۰	Exit : move flag , ۰
----------	----------

در ابتدا مقدار اولیه ۰=**flag** است . صفر را انتقال می دهد به **reg۱** و یک مقدار غیر صفر وارد خانه حافظه می کند .

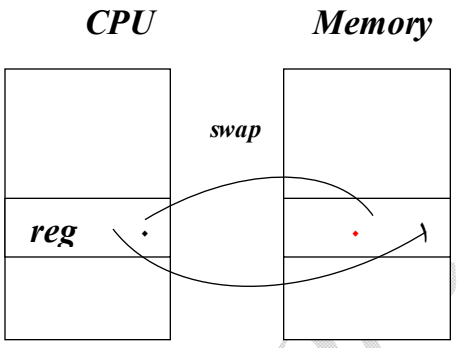
کانتر **reg۱** را منهای ۰ می کند اگر این دوبرابر باشند مقدار **flag** من ۰ خواهد شد اما اگر برابر نباشند ۰ ≠ **zero flag** می شود پس **JNZ** اجرا نخواهد شد پس وارد ناحیه بحرانی می شود و زمانی که می خواهد از ناحیه بحرانی خارج شود و ۰ را یعنی اگر صفر نبود پرش کند . ۰ = **flag**

حال مقایسه می کند مقدار **reg۱** در این روش هم هنوز مشکل حل نشده فقط یک دستور العمل **TSL** را اضافه کردیم که هر سه راه حل را هم تضمین میکند.

www.mbzir.com

۷- استفاده از دستورالعمل *swap*:

```
enter : mov reg۱ , ۱
swap reg۱ , flag
CMP reg۱ , ۰
JNZ  enter (jump not zero )
<cs >
exit : move flag , ۰
```



یک پردازنده و یک مموری داریم . مقدار اولیه ***flag*** باز هم صفر است . که می گوید یک را بریزد داخل ۱ ***reg*** سپس ***swap*** را اجرا می کند .یعنی مقدار ***flag*** و ***reg۱*** را جابجا می کند . سپس مقایسه می کند که آیا مقدار ۱ ***reg*** = ۰ است ؟ دفعه ی اول واقعا" مساوی صفر است ***JNZ*** اجرا نمی شود و وارد ناحیه بحرانی می شود .(اگر به جای ***reg۱***=۱ بود انتظار مشغولی پیش آمده و مدام در آنجا منتظر می ماند .)

دستورالعمل ***swap*** و ***TSL*** دقیقا" مشابه هم عمل می کنند با این تفاوت که در ***TSL*** مقدار غیر صفر را در ***flag*** جایگزین می کند ولی ***swap*** فقط یک را در ***flag*** جایگزین می کند . تفاوت دیگری که بین این دو دستورالعمل وجود دارد این است که دستورالعمل ***swap*** در اغلب پردازنده ها تعریف شده و پرکاربرد است .

نکته ۱: هر دو روش ***swap*** و ***TSL*** تضمین انحصار متقابل می کنند .یعنی فقط یک فرآیند وجود دارد که ***flag*** را مساوی صفر میکند .

نکته ۲: آیا این دو انتظار محدود را هم پشتیبانی می کنند؟ خیر ، زیرا بدست آوردن صفر یک پدیده ی کاملا" تصادفی می باشد .

نکته ۳ : واقعا" آن فرآیندهایی که قصد ورود به ناحیه بحرانی را دارند صفر را بدست می آورند پس شرط تضمین پیشرفت را هم دارند .

www.mbzir.com

سوال : آیا این راه حل ها نادرست می باشند ؟ ***TSL*** و ***swap*** روشهای درستی نمی باشند چرا که شرط انتظار محدود را برآورده نمی کنند . اما یک استثنا داریم معمولا" از این دو روش جاهایی استفاده می شود که ناحیه بحرانی فرآیندها خیلی کوتاه باشند .

حال باید برویم سراغ حالتی که گفتیم عمدتا " بر مبنای ***busy waiting*** هستند یعنی ***block waiting*** نیستند . که بحثی به نام سمافور ها را پیش می آورد

www.mbzir.com

۸- سمافورها (***semaphore***) :

سمافور در لغت به معنای راهنما می باشد و تعریف آن اینگونه می باشد که ساختار داده ای است شامل یک عدد صحیح و یک صف .

که سه عمل بر روی این ساختار داده تعریف می شود :

- مقدار دهی اولیه
- عمل ***Wait***
- عمل ***Signal***

ساختار آن به صورت زیر می باشد :

struct semaphore

int count

queue type queue)

مقدار دهی اولیه : به متفییر صحیح در سمافور به مقدار اولیه تخصیص می یابد .

عمل ***Wait***: یک واحد از متغییر سمافور کم می کند اگر این مقدار کم تر از صفر باشد فرآیندی که آن را فراخوانی کرده به حالت مسدود می رود . (در اینجا انتظار مشغولی را به انتظار مسدودی تبدیل کردیم .)

عمل ***Signal***: یک واحد به مقدار صحیح اضافه می کند اگر مثبت نباشد یکی از فرآیندهای مسدود شده توسط ***wait*** آزاد می شود .

اعمال ***Wait*** و ***Signal*** به صورت اتمیک اجرا می شوند . یعنی اجازه ی صدور وقفه در حین اجرای این دو عمل وجود ندارد .

www.mbzir.com

بافرض اتمیک بودن این دو عمل یک راه حلی برای حل ناحیه بحرانی مطابق الگوریتم زیر ارائه می دهیم :

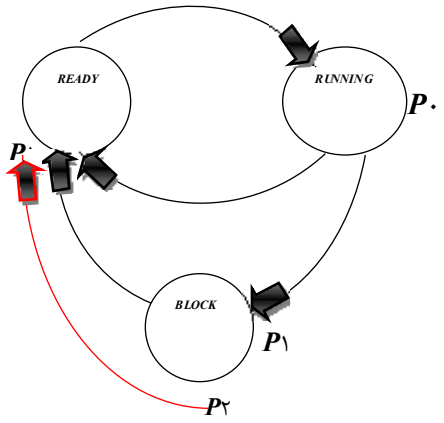
```
while (True ){
Wait (s )
critical section
signal
no - critical section
```

P۰	***P۱***	***P۲***
------	***-----***	***------***
------	***-----***	***-----***
Wait (s)	***Wait (s)***	***Wait (s)***
CS	***CS***	***CS***
Signal (s)	***Signal (s)***	***Signal(s)***
------	***-----***	***-----***
------	***------***	***------***
S=۰	***S=-۱***	***S=-۲***
S=-۱	***S=۰***	***S=۱***

ابتدا مقدار اولیه ی سمافور را برابر ۱ در نظر می گیریم .

می خواهیم ببنیم آیا این قطعه کد درست عمل می کند یا نه ؟

www.mbzir.com



P۰ میخواهد ***wait (s)*** را انجام دهد و یک مقدار از ***s*** کم می کند ***s=۰*** . چون ***s*** منفی نشد وارد ناحیه بحرانی می شود و در منطقه بحرانی وقفه سهم زمانی زده می شود . حال می رود که ***P۱*** را اجرا کند . ابتدا ***P۱*** ، ***Wait (s)*** را انجام می دهد (***S=-۱***) پس ***P۱*** می رود به حالت ***block*** حال نوبت ***P۲*** شد و ***Wait (s)*** انجام می شود . (***S=-۲***) پس این هم در حالت ***block*** قرار می گیرد .

| S | میشود تعداد فرآیندهای مسدود شده به خاطر سمافور ***S***

حال نوبت به ***P۰*** میرسد که می آید به فاز ***Signal (s)*** که یکی به ***S*** اضافه می کند و ***S=-۱*** و چون ***S*** مثبت نیست ***P۱*** از حالت ***block*** آزاد می شود و می رود به حالت ***ready*** حال ***P۱*** اجرا می شود و ***Signal (s)*** ، ***S=۰*** ، ***S*** مثبت نیست پس ***P۲*** هم وارد ناحیه بحرانی می شود و در نهایت ***S=۱*** و ما متوجه می شویم که دیگر کسی در صف نمی باشد .

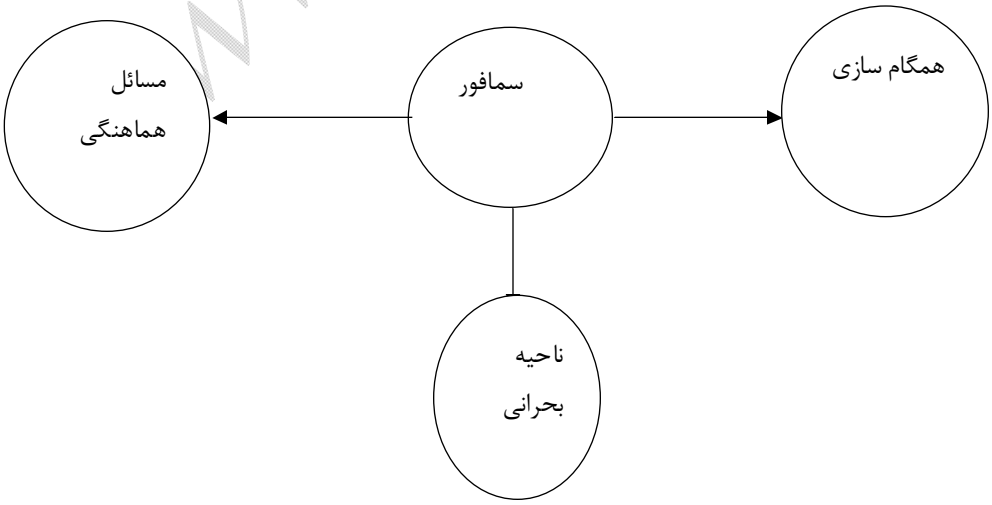
قطعه کد سمافور :

```
void wait (semaphore  s )
disable int ;
{ s.count - -;
   if (s.count <۰ )
      { place this process  in s. queue;
        block this process; }
```

```
        enable int ;
}

void signal (semaphore  s )
disable int ;
{ s.count ++;
   if (s.count <=۰ )
      {remove a  process  p from  s. queue;
       place process p on ready; }
        enable int ; }
```

این دستورالعمل ها فعلا" اتمیک نیستند . برای اتمیک بودن ***disable*** و ***enable*** را به آنها اضافه کردیم .

نکته : استفاده از سمافور ها صرفا" برای حل مسئله ناحیه بحرانی محدود نمی باشد بلکه برای بسیاری از مسائل هماهنگی ، همگام سازی ، ناحیه بحرانی و .... است .



www.mbzir.com

فرض کنید ***S***۱ حتما" قبل از ***S***۲ اجرا می شود:

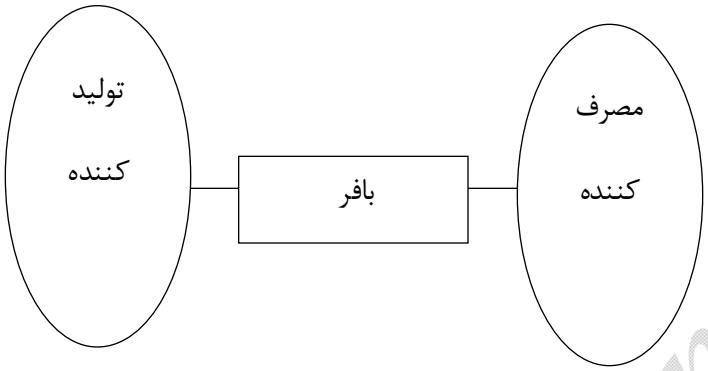
P۱	***P***۲	
----------	----------	***SYNC***= -۱
----------	----------	***SYNC***=۰
S۱	***Wait(SYNC)***	
Signal (SYNC)	***S***۲	
----------	----------	
----------	----------	
تولید کننده	مصرف کننده	

نکته : بر اساس مقدار اولیه ای که به سمافور تخصیص می یابد می توان تشخیص داد که عملکرد سمافور چیست . به فرض برای حل ناحیه بحرانی همواره مقدار سمافور برابر ۱ می باشد چراکه بایستی تضمین نماید فقط یک فرآیند در قسمت ناحیه بحرانی قرار بگیرد .

نکته : برای اتمیک کردن دستورات ***wait*** و ***signal*** بایستی از دو ***system call*** ، ***disable int*** و ***enable int*** استفاده نماییم . اما اصولا" از کار انداختن ***int*** ها کار خوبی نیست . ما سیستم هایی داریم که ***int*** های خیلی مهمی دارند .

www.mbzir.com

۹- مسئله تولید کننده مصرف کننده با بافر محدود :




```
void process (){
    while (true) {
        produce item ();
        wait (e);
        wait (s);
        App end ();
        signal (s);
        signal (n);
    }
}
```

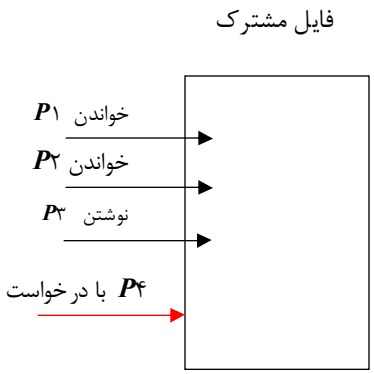
شرط پر بودن بافر

```
void costumer (){
while (true) {
        wait (n);
        wait (s);
        take ();
        signal (s);
        signal (e);
        consume _ item();
}
}
```

شرط خالی نبودن بافر

n تعداد آیتم های بافر را نشان می دهد . این روش تستی است . اگر ***s***=۱ باشد می شود تا حدودی گفت مساله ، مساله ی ناحیه بحرانی است . سمافور ***s*** برای حفظ ناحیه بحرانی است . سمافور ***e*** برای پر بودن است. سمافور ***n*** خالی نبودن بافر است .

۱۰-مسئله خوانندگان نویسندگان(*READERS AND WITERS PROBLEM*)



P۱ در حال خواندن این فایل است و ***P***۲ هم در حال خواندن این فایل مشترک است . و ***P***۳ دارد در این لحظه می نویسد . ***P***۴ با درخواست خواندن می آید . سوال این است که آیا به ***P***۴ اجازه خواندن داده می شود یا نه ؟دلیل خود را توضیح دهید .

ابتدا به این سوال پاسخ می دهیم که آیا می شود هم نوشت و هم خواند ؟

خیر ، زیرا فایلی که می نویسیم پایدار نیست (تا زمانی که نوشتن به پایان نرسیده است فایل ناپایدار است .) بنابراین خودش ناپایدار است . باید ابتدا نوشتن به پایان برسد ، بعد عمل خواندن شروع می شود .

سوال بعدی این است که آیا دو نوشتن همزمان امکان پذیر می باشد ؟

خیر ، زیرا دو فرآیند دارند یک منبع مشترک را همزمان تغییر می دهند که این ممکن نیست .

پس به ***P***۴ هم می توان اجازه ی خواندن داد با یک استراتژی . و هم می توان اجازه خواندن نداد با یک استراتژی دیگر .

دو حالت در پاسخ به این سوال پیش می آید . اول اینکه اجازه ی خواندن به ***P***۴ داده شود . دوم اینکه ***P***۱ و ***P***۲ عمل خواند نشان به اتمام برسد ، سپس عمل ***P***۳ انجام شود . سپس به ***P***۴ اجازه ی خواندن بدهیم . هر دو راه حل هایی هستند که بسته به کاربرد و استراتژی های سیستم عامل متفاوت می باشند .

www.mbzir.com

semaphore write =۱

semaphore mutex=۱

readcount =۰

Write process		***Readers process***	
----------		***----------***	
Wait (write)		***Wait (mutex);***	
int → ***Write()***	قبل از خواندن	***Read count ++;***	
Signal	بررسی حفاظت از متغییر مشترک	***If (readcount ==۱)***	اولین گروه از خوانندگان
_____		***Wait (write)***	
_____			
_____		***Signal (mutex);***	
		Read();	
		Wait (mutex);	
	بعد از خواندن	***Read count --;***	
		If(readcount ==۰)	آخرین فرآیند از گروه خوانندگان
		signal (write);	
		Signal (mutex);	
		_____	

www.mbzir.com

۱۱-مروری بر جلسه قبل :

بحث هماهنگی فرآیندها بود که نیاز بود دو فرآیند با هم هماهنگ شوند و دلیل این هماهنگی این بود که منبع محدود است . بعد یک سری تلاش ها را بیان کردیم تلاش اول ، دوم ، سوم که مزایا و معایب را بیان کردیم. سپس روشهای سخت افزاری را به نام های ***TSL*** و ***swap*** را بیان کردیم . و گفتیم ایراد روشهای سخت افزاری این بود که از ***busy waiting*** به ***block waiting*** برویم و در ***running*** منتظر نمانیم . سپس سمافور را بیان کردیم که دارای اعداد صحیح و صف می با شند . و دو عمل ***wait ,*** و ***signal*** در آن انجام می شوند . که ***wait*** کم کردن مقدار عدد صحیح بود و و اگر مقدار کم تر از صفر بود یک ***systemcall***می فرستاد به صفحه ***block***ها و آن اعداد را در صف ***ready*** قرار می داد . بعد گفتیم اینها باید اتمیک باشند و ما قبل و بعد از اینها از ***enable int*** و ***disable int*** استفاده کنیم .

و سپس به بررسی مساله مصرف کننده و تولید کننده پرداختیم و گفتیم که سمافور چند بعدی است که یکی برای ناحیه بحرانی و دیگری برای هماهنگی می باشد . و این را از مقدار اولیه و عملکرد سمافور می توانستیم تشخیص دهیم . و مساله خوانندگان و ویسندگان را هم مطرح کردیم به این صورت که یک خواننده یک فایل مشترک را می خواند و ***P***۲ هم می خواند و ***P***۳ با درخواست نوشتن می آید در ***P***۱ و ***P***۲ تضادی نداریم پس مشکلی نخواهیم داشت . و ***P***۴ با درخواست خواندن می آید و راجب این مساله بحث کردیم که به ***P***۴ اجازه بدهیم یا خیر به این صورت که گفتیم دو راه حل است ۱. اجازه ی خواندن داده می شود . ۲. اجازه ی خواندن به ***P***۴ داده نمی شود یعنی ابتدا باید کار سیستم تمام شود بعد به آن اجازه ی خواندن بدهیم .

	R	***W***
R	√	×
W	×	×

www.mbzir.com

-مسئله خوانندگان و نویسندگان

راه حل هایی برای اینکه بدانیم به کدام باید اجازه ی خواندن بدهیم و به کدام اجازه ندهیم :

semaphore write =۱

semaphore mutex=۱

readcount =۰

	Write process		***Readers process***	
	----------		***----------***	
int	***Wait (write)***	قبل از خواندن بررسی حفاظت از متغییر مشترک	***Wait (mutex);***	
→	***Write()***		***Read count ++;***	
	Signal		***If (readcount ==۱)***	اولین گروه از خوانندگان
	----------		***Wait (write)***	

	----------		***Signal (mutex);***	
			Read();	
		بعد از خواندن	***Wait (mutex);***	
			Read count --;	
			If(readcount ==۰)	آخرین فرآیند از گروه خوانندگان
			signal (write);	
			Signal (mutex);	

			----------	

www.muzin.com

اتفاقی که در مساله خواننده افتاده است به این صورت است که ***write*** می کند روی ***mutex*** و ***signal*** می کند روی ***mutex*** که این دو متغییر برای حفاظت از متغییر مشترک می باشد .

چون فرآیند خواننده است ***read count*** یکی اضافه شده است . اگر ***readcount*** = ۱ باشد یعنی اگر و فقط اگر اولین گروه از خوانندگان باشد ***wait*** میکند روی ***write*** یعنی در اینصورت تمامی فرآیندها باید روی سمافور ***write*** عمل ***wait*** انجام دهند . حال فرض کنیم ***read count*** یک نباشد. مثلا" ۵ باشد پس ***signal*** انجام شده و عمل ***read*** انجام می شود . سوال این است که چه زمانی می توان به عمل ***read*** رسید ؟ فقط زمانی که روی ***mutex*** ، ***wait*** کرده باشیم و ++ ***readcount*** شود . و عمل ***signal (mutex)*** انجام شود . ***mutex*** برای حفاظت از متغییر مشترک ***readcount*** است . بعد از خواندن ***readcount*** را یکی کم می کنیم . اگر ***readcount***=۰ باشد (یعنی آخرین فرآیند از گروه خوانندگان باشد) در آن صورت ***signal*** می کند روی ***write*** .

سوال : آیا این استراتژی ، استراتژی اول است (به ***P***۴ اجازه ی خواندن می دهد) یا استراتژی دوم (به ***P***۴ اجازه ی خواندن نمی دهد)؟

۱۲-مساله غذا خوردن فیلسوفان (شام فیلسوفان) ***Dining philosopher***:

یک میز گرد است که هر کدام از افراد نقش فیلسوف را دارند و فرض شده که این فیلسوفان چینی می باشند . و با چوب غذا می خورند . جلوی هر کدام از این فیلسوف ها بشقابی قرار داده شده است و در کنار بشقاب ها چنگال ، اما نکته ای که در اینجا وجود دارد این است که تعداد چنگال ها برای هر فیلسوف باید دو تا باشد ولی تعداد کم تر از دوتا برای هر شخص می باشد .

در این مساله فیلسوفان فرآیندهای ما می باشند . چنگال ها منابع ما می باشند زیرا فرض شده که محدود اند .

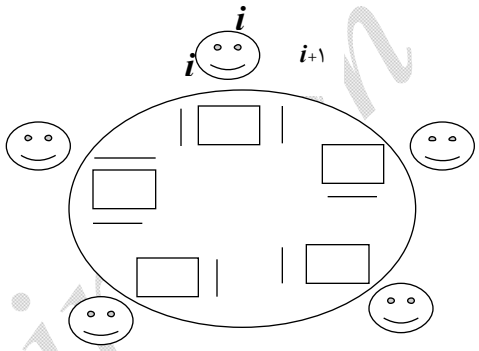
ما این نکته را هم می دانیم که فیلسوفان یا غذا می خورند یا فکر می کنند . و شرط غذا خوردن فیلسوف ***i***ام بدست آوردن چوب ***i*** ام و چوب ***i***+۱ می باشد .

برای هر منبعی سمافور تعریف می کنیم : استفاده از منابع برای فیلسوفان انحصاری می باشد .

نکته : سمافور نیازمند پشتیبانی سیستم عامل می باشد .

حال قطعه کد آن را می نویسیم :

```
 semaphore chopstick[۵];

initialization value of  chopstick[i]=۱;

void p[i]  {

while (true ){

wait (chopstick[i]);

wait (chopstick[(i+۱)mod ۵]);

eating ;

signal (chopstick[i]);

signal (chopstick[(i+۱)mod ۵]);

thinking ();

}

}

void main(){

        per _ begin (P(۰),P(۱), ....., P(۴));

}
```



می توان در یک جدول نحوه ی کارکرد این کد را هم نشان داد :

	CH[۰]	CH[۱]	CH[۲]	CH[۳]	CH[۴]
P۰	۰	۰	۱	۱	۱
P۱	×	-۱	۱	۱	۱
P۲	×	×	۰	۰	۱

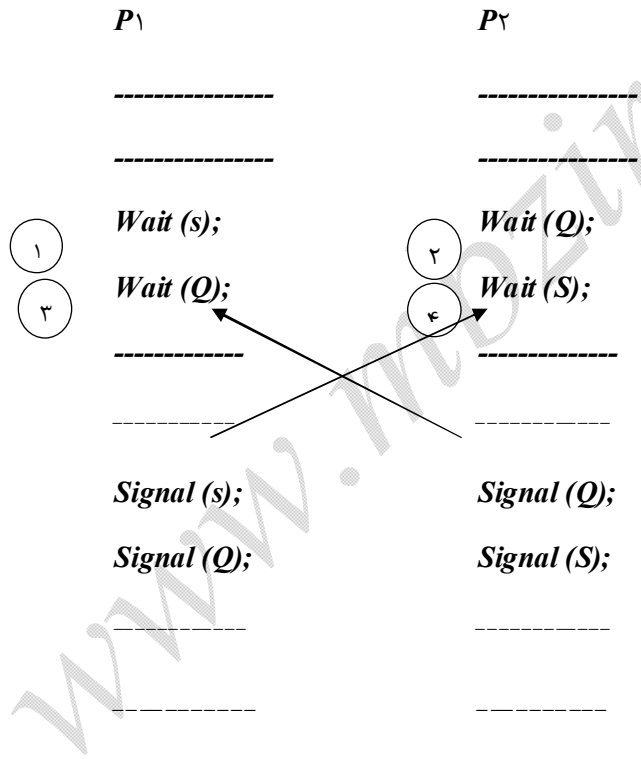
www.mbzi.ir

***P*۳**	×	×	×	−۱	۱
***P*۴**	−۱				۰

۱۳-سمافور دارای اشکالاتی هم می باشد :

۱. پدیده ی بن بست

فرض کنید قطعه کدی به شکل زیر نوشته شده است :



در اینجا ۱= ***S , Q***

	***P*۱**	***P*۲**

www.mozir.com

S	۰	۱-
Q	۱-	۰

هر دو فرآیند کاملا" مستقل هستند و به تنهایی درست کار می کنند . این سناریو را در نظر بگیرید.

۱. پایان سهم زمانی می رود به ***P***۲

۲. ***Q*** را صفر میکند ، می آید به ***P***۱

۳. ***P***۱ ، ***Q*** را ۱- می کند ، در نهایت ***P***۱ توسط ***Q*** ، ***block*** می شود .

۴. ***P***۲ ، ***S*** را ۱- میکند و ***P***۲ هم ***block*** می شود .

در این دو فرآیند هر دو ***block*** شدند . ***P***۲ منتظر است که ***P***۱ ، ***signal*** بدهد و ***P***۱ منتظر است ***P***۲ ، ***signal*** بدهد . که این باعث می شود بن بست بوجود آید . پس یکی از مشکلات سمافور ها پدیده ی بن بست است .

Signal(mutex)

Wait (mutex)

۲.مشکل دومی که وجود دارد بی مبالاتی در ترتیب نوشتن ***wait*** و ***signal*** است .

نقض

wait(s);

www.mbzir.com

۳. مشکل سومی که وجود دارد این است که قفل دائمی بوجود می آید :

Wait (s);

قفل دائمی

۴.مشکل چهارم : از یاد رفتن ***signal***

---

wait(s);

از یاد رفتن
signal

۱۴-بن بست

در استفاده از منابع فرآیندها در یک وضعیت قرار می گیرند که به آن وضعیت بن بست می گویند . در این وضعیت فرآیندها در حالتی قرار می گیرند که امیدی به رفع آنها نیست و هیچ فرآیندی امید رفتن از این شرایط را ندارد .

ما این علائم را برای نشان دادن فرآیند و منابع و ... به کار می بریم :

فرآیند

منابع

تخصیص منابع

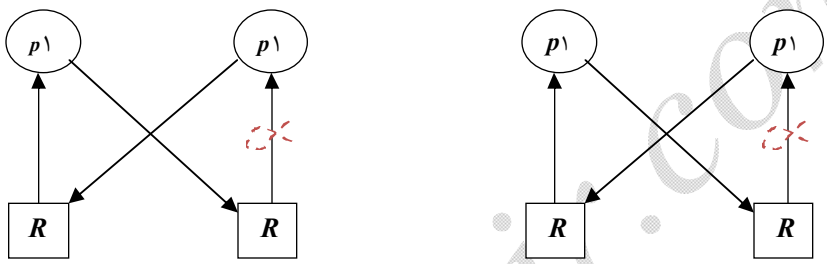
درخواست منابع

www.mbzir.com

P_1، R_1 را در اختیار دارد و P_2، R_2 را در اختیار دارد برای اتمام P_1چون درخواست داده به R_2 و ما همانطور که می بینیم R_2 هم در دست P_2می باشد و نه P_1، R_1 را آزاد می کند که

P_2 از آن استفاده کند و نه P_1 ، R_1 را آزاد می کند که P_2 از آن استفاده کند . در این حالت گوییم بن بست رخ داده است .

حال اگر P_2 دیگر از R_2 استفاده نکند این مشکل هم حل می شود .



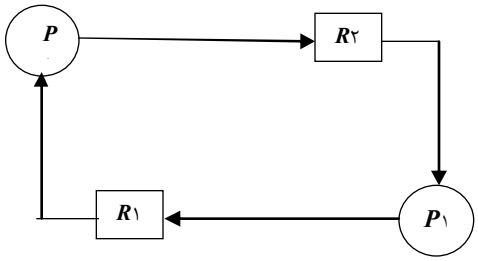
۱۵-شرایطی که فرآیند ها در بن بست قرار می گیرند :

۱. انحصار متقابل منابع نسب به هم : یعنی استفاده از منابع انحصاری می باشد .

۲. شرط نگهدار ی و انتظار (***HOLD AND WAIT***) : فرآیندی که منبع را در اختیار دارد و منتظر منبع دیگر است . متاسفانه منبع در اختیار داشته را نگهداری می کند و آزاد نمی کند .

۳. قبضه کردن منبع ممکن نباشد (***NO PREMPTIN***)

۴. انتظار چرخشی (***CRITICAL WAIT***)



www.mbzir.com

اگر بدانیم منبع نسبت به هم حالت انتظار چرخشی دارند ، به عبارتی توانستیم کشف بن بست کنیم .

کشف انتظار مشغولی به معنای کشف بن بست است و این کار از وظایف سیستم عامل است.

سه شرط اول خصوصیات ذاتی فرآیند است و برخی فرآیندها ذاتشان اینگونه است و عملا" نمی توان آنها را نقض کرد . پس اگر بتوان برای بن بست کاری کرد باید فقط و فقط شرط چهارم را نقض کنیم تا بن بست اتفاق نیفتد . اگر بن بست اتفاق بیفتد هیچ فرآیندی نمی تواند هیچ کاری بکند .

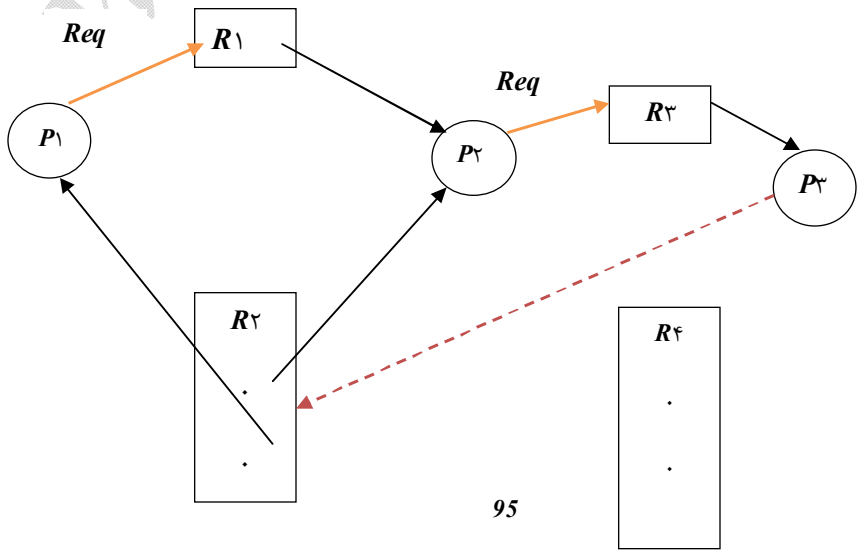
پس راه حل ما فقط شرط چهارم است .

www.mbzir.com

-مثال هایی از بن بست :

سوال ۱: آیا سیستم دراین حالت در وضعیت بن بست قرار دارد یا خیر ؟

سوال ۲: یک مسیری پیدا کنید که همه ی فرآیندها اجرا شوند ؟



پاسخ سوال اول :

خیر ، زیرا هیچ حله ای کشف نکرده ایم ، پس بن بست اتفاق نیفتاده است.

پاسخ سوال دوم:

فعلا" P۱ در حال ***wait*** است ، P۲ دو منبع R۱ , R۲ را در اختیار دارد ، یک درخواست هم به R۳داریم .

P۱ انتظار مسدود

P۲ انتظار مسدود

P۳آزاد می شود

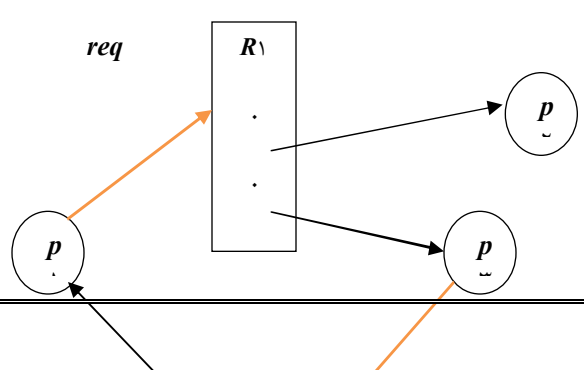
P۲ اجرا می شود و منابع تخصیص داده شده آزاد می شوند . (R۱ , R۲ و خود R۳ را آزاد می کند)

P۱ اجرا می شود و منابع تخصیص یافته آزاد می شودند .

پس این یک مسیری است که معلوم است سیستم کار می کند .

ما با P۴ کاری نداشتیم پس خیلی از منابع هستند که ما با آنها کاری نداریم . حال در مساله بعدی فرض کنید P۳به جای اینکه R۳ را آزاد کند ، درخواست ***req*** میدهد از R۲ ، آنوقت R۲,P۲,R۳,P۳,و همچنین R۲,P۱,R۱,P۲,R۳,P۳ که حلقه می شوند و در این شرایط ما هیچ کاری نمی توانیم انجام دهیم چون تمام فرآیندها در حالت انتظار مسدودی قرار می گیردند .

مثال دوم :



www.mbzir.com

req

P۳ یک ***req*** داده به ***R***۲، آیا بن بست اتفاق می افتد یا نه ؟ ظاهرا" یک ***loap*** ایجاد شده است اما دو فرآیند تعیین کننده را ما در نظر

نگرفتیم (***P***۲ **,** ***P***۴) . این دو فرآیند خیلی مهم اند .

P۲، ***R***۱ را آزاد می کند

P۱، اجرا و ***R***۱ و ***R***۲ را آزاد می کند

P۳ اجرا می شود

پس درست است که ***Loap*** ایجاد شده اما بن بست اتفاق نیفتاد.

نتیجه گیری : وجود انتظار پرخشی برای بن بست لازم است اما کافی نیست .

۱۶-راه حل های مسئله بن بست :

۱. جلوگیری از بن بست (***dead lock prevention***)

۲. اجتناب از بن بست (***deadlock avoidance***)

۳. تشخیص بن بست (***deadlock detection***)

- جلوگیری از بن بست : فقط و فقط با نقض ۴ شرط گفته شده می توان از بن بست جلوگیری کرد که تحت هر شرایطی امکان پذیر نیست . چون شرط ۱و۲و۳ ذاتی است و نمی شود کاری کرد . شرط چهارم هم این است که ***loap*** را قطع کنیم . پس تحت هر شرایطی امکان پذیر نیست .

www.mbzir.com

-اجتناب از بن بست : مدل فکرمان اینگونه می باشد که در تخصیص منبع به فرآیندها یک سری محاسباتی را انجام می دهیم . یک پیش بینی انجام می دهیم که با این تخصیص ها احتمال اینکه سیستم پس از تخصیص به حالت بن بست برسد وجود دارد یا ندارد . اگر وجود داشته باشد می گوییم تخصیص انجام نشود ، اگر وجود نداشته می گوییم تخصیص انجام شود و لی به ازای هر تخصیص باید یک تست انجام دهیم که این هم عملا" امکان پذیر نیست چون سیستم بسیار کند عمل می کند .

- تخصیص بن بست : سیستم عامل کاری ندارد که منابع چگونه استفاده می شود اما هر وقت استفاده می شود اما هر وقت خواست بداند بن بست اتفاق افتاده یا نه ؟ الگوریتم تشخیص بن بست را اجرا می کند و بعد متوجه می شود که اتفاق افتاده یا نه سپس می رود به دنبال راه حل ها که راه حل ها مختلف هستند . فقط تشخیص بن بست در اینجا ارائه می شود و راه حل را ما نمی گوییم و مثلا" یکی از راه حل ها می تواند شکستن حلقه باشد . روش سوم یک روش عملی است و ارزش کاربرد.

۱۷- مروری بر جلسه قبل :

-بن بست یک وضعیت از فرآیند است که هیچ فرآیندی امید به خروج از آن وضعیت را ندارد. ما در مدل بن بست بدبینانه به مسائل نگاه می کنیم . چند عوامل برای بن بست وجود داشت :

۱. **انحصار متقابل** (یک منبعی را انحصاری به یک فرآیند اختصاص دهیم) . ۲. **نگهداری و انتظار** (مدل فرآیند طوری است که یک منبع را آزاد نمی کند تا زمانی که کارش به پایان رسیده باشد). ۳. **قبضه کردن ممکن نیست** (منابعی هستند که نمی توانیم از آنها منبع را بگیریم) . ۴. **انتظار چرخشی** (به این معنی که فرآیند ها در سیکلی قرار گیرند که این سیکل چرخشی باشد .)

- از این چهار مورد فقط می توانیم مورد آخر (انتظار چرخشی را تغییر دهیم که این کار هم بسیار مشکل می باشد .)

-راه حل هایی برای حل مسائل بن بست ارائه نمودیم که سه دسته بودند :

۱- **جلوگیری از بن بست** :به این صورت که مکانیزمی داشته باشیم که چهار شرط بالا را نقض کنیم که این کار باعث می شود بن بست نداشته باشیم . ولی این کار امکان پذیر نمی باشد زیرا مخاطراتی را بدنبال دارد .

۲. **اجتناب از بن بست** : در این حالی باید یک مدل تفکر انتزاعی داشته باشیم و به ازای هر درخواست یک اجازه گرفته شود و سیستم عامل هر بار چک می کند که با این درخواست به حالت مطمئن می رود یا نامطمئن . مسئله ای که در این حالت وجود دارد این است که امکان دارد ۲۰ فرآیند داشته باشیم و هر فرآیند ۶۰-۷۰ نخ داشته باشد و درخواست منابع هم ۲۰۰ مورد باشد که اگر بخواهیم هر کدام را محاسبه کنیم کلی زمان نیاز دارد به همین دلیل می گویند این روشی انتزاعی و تخیلی می باشد .

۳. **تشخیص بن بست :** این روش تقریبا" عملی می باشد به این صورت که یک الگوریتمی را اجرا می کند و بن بست شناسایی می شود .

-مثال هایی برای یافتن مسیر امن :

مثال: در یک سیستم مشغول به کار در زمان t وضعیت سیستم به شرح زیر است :

سه فرآیند داریم به نام های $P۱,P۲,P۳$ که از یک منبع ۱۲ مورد دارند .

	میزان منبع در اختیار داشته	حداکثر میزان استفاده از منبع	منبع مورد نیاز
$P۱$	۵	۱۰	۵
$P۲$	۲	۴	۲
$P۳$	۴	۹	۷

سوال اول : آیا سیستم در حالت امن است ؟

سوال دوم : $P۳$ یک واحد دیگر از منبع در اختیار داشته ی خود اضافه میکند آیا در این حال باز هم مسیر امنی وجود دارد ؟

پاسخ سوال اول :

www.mbzir.com

در این جدول ما باید ستون مورد نیاز را پر کنیم .

P۱ حداکثر میزان استفاده اش از منبع ۱۰ مورد است که ۵ مورد در اختیار دارد و به ۵ مورد دیگر نیاز دارد.

P۲ حداکثر میزان استفاده اش از منبع ۴ مورد است که ۲ مورد در اختیار دارد و به ۲ مورد دیگر نیاز دارد.

p۳ حداکثر میزان استفاده اش از منبع ۹ مورد است که ۲ مورد در اختیار دارد و به ۷ مورد دیگر نیاز دارد.

همانطور که صورت مسئله گفته شده این سه فرآیند از یک منبه ۱۲ مورد دارند که باید این ۱۲ را از منابع در اختیار داشته ی آنها کم کنیم تا منبع موجود ما بدست آید .

۱۲-(۵+۲+۳) = ۳

تعداد کل منابع موجود

۳ تعداد موجودی ما است با این موجودی ما می توانیم فرآیند P۲ را اجرا کنیم . زیرا موجودی ما ۳ تا است و فرآیند P۲ فقط به ۲ مورد نیاز دارد پس کل موجودی را به فرآیند P۲ می دهیم و آن اجرا می شود . P۲ بعد از اجرا شدن آن تعداد منابعی را هم که در اختیار داشت آزاد می کند حال کل منابع موجود ما شد ۵ . با این ۵ مورد منابع می توانیم فرآیند P۱ را اجرا کنیم که P۱ هم بعد از اجرا شدن منابع در دست داشته را آزاد می کند حال کل منابع موجود ما شد ۱۰ . با این تعداد منابع هم می توانیم به P۳ منابع دهیم و آن هم بعد از اجرا منابع در دست داشته ی خود را آزاد می کند حال کل ۱۲ مورد منبع ما آزاد شد .

موجودی ۱۰ ⟶ اجرای P۱ ⟶ موجودی ۵ ⟶ اجرا ی p۲

موجودی ۱۲ ⟶ اجرای P۳

همانطور که مشاهده میشود پس از اجرای این سه فرآیند موجودی به سر جای اول خود بازگشت .

ما به مسیر P۱ ⟶ P۲ ⟶ P۳ می گوییم مسیر امن . که در آن به مشکلی بر نخوردیم .

www.mbzir.com

پس باید گفت بله مسیر امن وجود دارد .

پاسخ سوال دوم :

در این صورت جدول مورد نظر به صورت زیر خواهد شد :

موجودی کل خواهد شد :

۱۲ - (۵+۲+۳)=۲

	میزان در اختیار داشته از منبع	حداکثر میزان استفاده از منبع	منبع مورد نیاز
***P*۱**	۵	۱۰	۵
***P*۲**	۲	۴	۲
***P*۳**	۳	۹	۶

در این حالت ***P*۲**را انتخاب می کنیم و موجودی خواهد شد ۴ مورد که با این ۴ مورد موجودی نمی توانیم به هیچ کدام از دو فرآیند دیگر منبع مورد نیاز آنها را بدهیم و هیچ گزینه ای برای اجرا نخواهیم داشت .

دیگر هیچ گزینه ای برای اجرا وجود ندارد ⟹ موجودی = 4 ⟹ انتخاب ***P*۲**

در این حالت دیدیم که مسیر امنی وجود ندارد و اگر مسیر امن یافت نشود پس احتمال بروز بن بست وجود دارد .

www.ir.bzir.com

۱۸- الگوریتم بانکداران + *(Banker's algorithem)*

هرگاه m منبع داشته باشیم و n فرآیند در این صورت از الگوریتمی به نام الگوریتم بانکداران استفاده خواهیم کرد .

-الگوریتم بانکداران جهت اجتناب از بن بست

الگوریتم بانکداران برای اجتناب از بن بست ارائه می شود که کد این الگوریتم به صورت زیر است:

available [m]	از هر منبعی چه مقدار موجود است
available [j] = k	از منبع j ، k مورد موجود است
max[n × m]	حداکثر نیاز های فرآیند nام به منبع mام
max [i , j]	حداکثر نیاز های فرآیند nام به منبع mام k مورد است
allocation [n × m]	تخصیص منبع چگونه بوده است
allocation [i,j]	به فرآیند i از منبع j ، k مورد تخصیص یافته است
claim[n × m]	چه میزان نیاز داریم به حداکثر منبع خود برسیم
request i [m]	m از منبع i تقاضای فرآیند

از این رابطه می توان این موضوع را دریافت که :

claim [i , j]= max [i , j]- allocation [i , j]

requeste i یعنی اینکه درخواست ***i*** از راه می رسدو باید به این موارد وقتی که این درخواست میر سد توجه کنیم :

۱. باید بررسی شود ***request i <= claim*** اگر بله برود مرحله ۲ و اگر نه خطا صادر شود .

۲. آیا ***request i <= availabe*** اگر بله برود مرحله ۳ در غیر اینصورت برود به حالت انتظار .

۳. فرض می کنیم حالت جدید مشخص شده است :

available = available - request i

allocation = allocation + request i

claim = claim - request i

۴. آیا حالت جدید امن است ؟ اگر بله تخصیص صورت می گیرد در غیر اینصورت برود به این حالت انتظار .

ایراد های الگوریتم :

۱. ممکن است منبع داشته باشیم اما احتمال بروز بن بست بدهیم و تخصیص انجام نشود .

۲. از کجا بدانیم یک فرآیند چقدر منبع نیاز دارد ؟ (که ما معمولا" در مسیر امن این را خواهیم دید یعنی باید قطعه کد خوانده شود و منابع مورد نیازش به آن داده شود) الگوریتم بانکداران در مدل اجتناب از بن بست است ، یعنی قبل از تخصیص ، الگوریتم اجرا می شود به ازای هر درخواست . بنابراین قطعا" چندین برابر زمان اجرا بالا می رود و کارآیی ندارد . خود الگوریتم به درد نمی خورد بلکه استفاده از نتایج این الگوریتم در تشخیص بن بست انجام می شود.

مثال :

www.mbzir.com

پنج فرآیند داریم و ۳نوع منبع مختلف ، آیا حالت فعلی سیستم امن است یا نا امن است ؟

کل منابع

$A = ۱۰$

$B = ۵$

$C = ۵$

با توجه به جدول زیر به سوالات زیر پاسخ دهید :

	Allocation			Available			Claim		
	A	B	C	A	B	C	A	B	C
P۱	۰	۱	۰	۷	۳	۵	۷	۲	۵
P۲	۲	۰	۰	۳	۲	۲	۱	۲	۲
P۳	۳	۰	۲	۹	۰	۲	۶	۰	۰
P۴	۲	۱	۱	۲	۲	۲	۰	۱	۱
P۵	۰	۰	۲	۴	۳	۳	۴	۳	۱

سوال اول : آیا مسیر امن وجود دارد ؟

سوال دوم : این ***Request*** ۱ **[۰ , ۲, ۰]** از راه می رسد . آیا به این تقاضا جواب می دهید ؟

سوال سوم : این ***Request*** ۱ **[۰ , ۳, ۰]** از راه می رسد . آیا به این تقاضا جواب می دهد ؟

سوال چهارم : این ***Request*** ۱ **[۳ , ۰ , ۲]** از راه می رسد . آیا به این تقاضا جواب می دهد ؟

www.mbzir.com

پاسخ سوال اول :

فعلا ***Request*** ی نیامده و ما باید امن بودن را بررسی کنیم . ایتدا بردار ***Available*** را درست می کنیم که محاسبات به صورت زیر خواهد شد :

از ***A***، ۱۰ مورد داشتیم که ۷ مورد را تخصیص می دهیم

A = ۱۰- (۰+۲+۳+۲+۰) = ۷

از ***B***، ۵ مورد داشتیم که ۲ مورد را تخصیص می دهیم

B= ۵ - (۱+۰+۰+۱+۰) = ۳

از ***C***، ۷ مورد داشتیم که ۵ مورد را تخصیص می دهیم

C= ۷ - (۰+۰+۲+۱+۲) = ۲

حال کل موجودی خواهد شد : ***Available [۳،۳،۲]***

ابتدا باید ببنیم که ***available*** از کدام ***claim [i , j]*** کم تر است :,

Available [i , j]<= claim [i, j]

به صورت زیر عمل خواهیم نمود :

P۲انتخاب ⟶ ***AV= [۵،۳،۲]*** ⟶ ***P۴***انتخاب ⟶ ***AV= [۷،۴،۳]*** ⟶

P۵انتخاب ⟶ ***AV= [۷،۴،۵]*** ⟶ ***P۳*** انتخاب ⟶ ***AV= [۱۰،۴،۷]*** ⟶

P۱ انتخاب ⟶ ***AV= [۱۰،۵،۷]***

پاسخ سوال دوم :

باید این Request از راه رسیده را به جای فرآیند P۱ قرار دهیم و Allocation آن را تغییر دهیم :

	Allocation			***Available***			***Claim***		
	A	***B***	***C***	***A***	***B***	***C***	***A***	***B***	***C***
P۱	۰	۲	۰	۷	۳	۵	۷	۱	۵
P۲	۲	۰	۰	۳	۲	۲	۱	۲	۲
P۳	۳	۰	۲	۹	۰	۲	۶	۰	۰

www.mlzir.com

P۴	۲	۱	۱	۲	۲	۲	۰	۱	۱
P۵	۰	۰	۲	۴	۳	۳	۴	۳	۱

حال باید بررسی کنیم آیا ***[۷،۱،۵]=<[۰،۲،۰]*** چون ۲ بیشتر از ۱ است پس خطا می دهد .

پاسخ سوال سوم :

باید این ***Request*** از راه رسیده را به جای فرآیند ***P۱*** قرار دهیم و ***Allocation*** آن را تغییر دهیم :

	Allocation			***Available***			***Claim***		
	A	***B***	***C***	***A***	***B***	***C***	***A***	***B***	***C***
P۱	۰	۳	۰	۷	۳	۵	۷	۰	۵
P۲	۲	۰	۰	۳	۲	۲	۱	۲	۲
P۳	۳	۰	۲	۹	۰	۲	۶	۰	۰
P۴	۲	۱	۱	۲	۲	۲	۰	۱	۱
P۵	۰	۰	۲	۴	۳	۳	۴	۳	۱

حال باید بررسی کنیم آیا [۷،۰،۵]=<[۰،۳،۰] چون ۷ بیشتر از ۰ و ۵ بیشتر از ۰ است، پس خطا می دهد.

پاسخ سوال چهارم :

Allocation			***Available***			***Claim***		
A	***B***	***C***	***A***	***B***	***C***	***A***	***B***	***C***

www.mbzir.com

P۱	۳	۰	۲	۷	۳	۵	۴	۳	۳
P۲	۲	۰	۰	۳	۲	۲	۱	۲	۲
P۳	۳	۰	۲	۹	۰	۲	۶	۰	۰
P۴	۲	۱	۱	۲	۲	۲	۰	۱	۱
P۵	۰	۰	۲	۴	۳	۳	۴	۳	۱

بررسی می کنیم آیا ***[۳،۰،۲]<=[۴،۳،۳]*** بله ، حال باید برویم ***Available*** را بیابیم :

VA = [۱۰-۱۰, ۵-۱،۷-۷] = [۰،۴،۰]

کوچکتر باشد و سیستم به یک حات نا امن رفت ***AV*** در این حالت هم هیچ فرآیندی نداریم که از

نکته :

دو دلیل که ما از این روش اجتناب از بن بست استفاده نمی کینم این است که :

۱. سرعت را کند می کند .

۲. سرو کار داشتن با ماتریس ***Max*** الگوریتم بانکداران همواره آینده محور است یعنی همواره آینده را پیش بینی می کند .

www.mbzir.com

۱۹-الگوریتم تشخیص بن بست و روش های آن

اولا" در این الگوریتم نیاز به ماتریس ***Max*** نداریم .

ثانیا" به ازای هر درخواست اجرا نمی شود و هر زمان بخواهیم این الگوریتم را اجرا می کنیم .

ثالثا" این روش راه حل ارائه نمی دهد اما روش قبل راه حل رائه می داد .

رابعا" الگوریتم بانکداران الگوریتمی آینده محور بود .

الگوریتم تشخیص بن بست را هر زمان بخواهیم اجرا می شود چه بسا ممکن است در آن زمان چند فرآیند در بن بست باشند بعد از مدتی الگوریتم را اجرا می کنیم و در زمان های دلخواه وضعیت سیستم را بررسی می کنیم .

الگوریتم اجتناب از بن بست

الگوریتم تشخیص بن بست

www.mbzir.com

آینده محور است	در زمان حال بررسی می کند
به ازای هر درخواست اجرا می شود	در زمان مشخص اجرا می شود
داریم ***Max***	نداریم ***Max***

-روش های تشخیص بن بست :

فرض می شود فرآیندهای رونده ی فعلی در حالت خوش بینانه اگر تا آخر بروند و ادامه پیدا کنند منابع آزاد می شوند و فرآینده های مسدود ادامه پیدا می کنند به بیان دیگر ما با آینده کاری نداریم و در زمان حال تعریف می کنیم که آیا بن بست اتفاق افتده است یا نه . از یک بردار کمکی استفاده می کنیم که نام آن ***work*** است .

مرحله ۱: اگر $finish\ i = false\ ;\ allocation \neq ۰$ در غیر اینصورت $finish\ i = true$

مرحله ۲ : i را به گونه ای مشخص می کنیم که $finish\ i = false$ و $request\ i < available$ اگر i پیدا نشد به مرحله ۴ می رویم .

مرحله ۳ : $work + allocation\ i$ و $finish\ i = true$

مرحله ۴ : اگر برای $۱ < i < n$; وجود داشته باشد $finish\ i = false$ بن‌بست داریم .

تشریح مراحل بالا :

مرحله ۱ : اگر به فرآیند هیچ چیزی تخصیص نداده باشیم $finish = false$ (پس فرآیندی که به آنها منبعی تخصیص داده نشده را به حساب نمی آوریم .)

مرحله ۲ : فرآیندی را یابد که $allocation \neq ۰$ (یعنی حتما" منبعی تخصیص داده ایم).

$request\ i < allocatin$ و اگر پیدا نشد برود مرحله ۴ و اگر پیدا شد $finish\ i = True$ (یعنی تخصیص منبع انجام می شود و خارج می شود .)

مرحله ۳ : $work + allocation$ می شود . ***work*** ماتریسی است که i امین فرآیند منابع را بر می گرداند و $finish\ i = True$

www.mbzir.com

مرحله ۴ : قطعا یک فرآند نمی تواند با خود در بن بست باشد . پس چندین فرآیند وجود دارند (حداقل ۲ فرآیند) که ***finish i = false*** و این فر آیندها در بن بست می باشند . فرض کنیم

	۱	۲	۳	...۱-*N*
FINISH	*F*	*T*	*T*	...	*F*

ما نتیجه می گیریم ۱ و ۱ -*N* با هم در بن بست هستند چون در هر مرحله اگر قرار شد منبع را به آن تخصیص دهیم به محض تخصیص ***True*** میکردیم اما در صورتی که نمی شد تخصیص دهیم در این حالت می ماند و در آخر بررسی می کردیم . چند تا ***Finish i*** مانده است اگر تعداد ***F*** ها دوتا باشد بن بست خواهیم داشت ولی اگر یک ***F*** داشتیم با بن بست روبرو نخواهیم شد و بن بست نداریم چرا که بن بست خواست ماهیت تعاملی دارد و نیازمند دو فرآیند است .

۲۰- مروری بر جلسه قبل

الگوریتم تشخیص بن بست : ***Max*** ندارد و آینده محور نمی باشد .

یک بردار ***finish*** تعریف کردیم که در ابتدا همه ی ***finish*** ها ***false*** بودند سپس به تدریج این ***finish*** ها را ***true*** کردیم و دیدیم که فرآیند ۱ و ۱-*N* در بن بست اند.

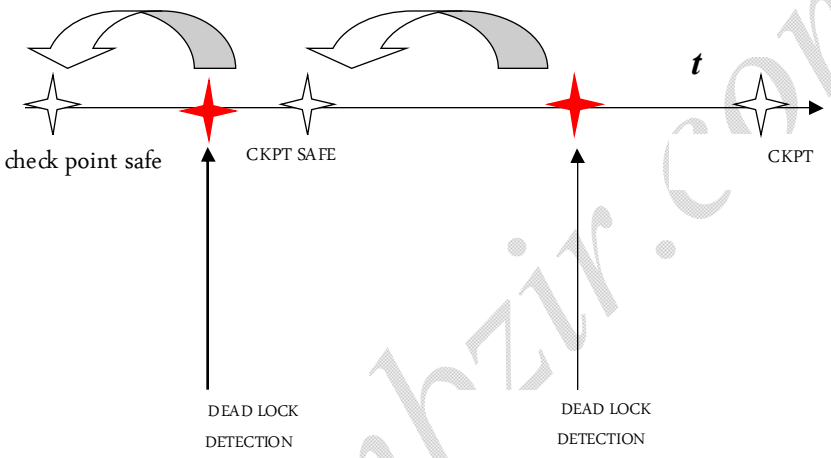
۲۱- *ROLL BACK*

حال که تشخیص دادیم این دو فرآیند در بن بست اند باید راه حلی برای این مسئله داشته باشیم که یکی از راه حل های آن ***kill*** کردن فرآیند می باشد . اگر یکی از فرآیند ها را ***kill*** کنیم بن بست رفع خواهد شد . حال اگر بخواهیم ***kill*** نکنیم زیرا این هزینه و زمان زیادی را از سیستم گرفته ایم .پس چه کار کنیم ؟ چگونه این حالت بن بست را رفع کنیم ؟

www.mbzir.com

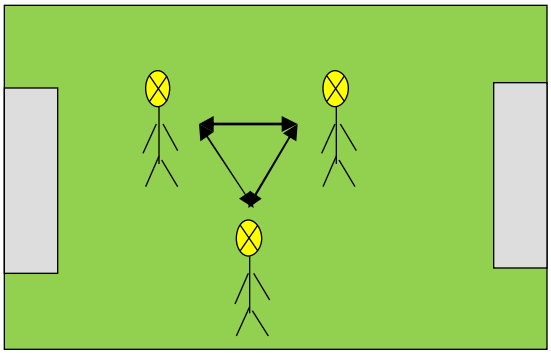
یک روش به نام ***ROLL BACK*** است . یعنی برگرداندن سیستم به حالت قبل . یعنی کارهایی که سیستم انجام می داده را ذخیره کرده ایم و زمانی که به بن بست می رسد آن را به حالت قبل بر می گردانیم .

در شکل مقابل یک ***dead lock*** رخ می دهد . فرض کنیم در قبل از آن یک ***check point*** (***CKPT***) زده ایم یعنی در این نقطه هیچ چیزی مشکلی نبوده و قبل از آن هم همینطور حال که پیش می رویم و به بن بست می رسیم به حالت قبل بر می گردیم و همینطور تا آخر ادامه می دهیم .



نکته : این به عقب برگشتن مستلزم این است که مراحل طی شده در جایی ذخیره شود که این کار هم نیزمند زمان است و هم هزینه بر است . بحث بن بست یک بحث کلی است . یعنی فقط و فقط برای فرآیندها نمی باشد . (***dead lock has global area***)

این بحث حتی در رباتیک هم وجود دارد . به این صورت که فرض کنید سه ربات فوتبالیست دارندکار می کنند و حالتی است که این سه در شرایطی قرار می گیرند که می تواند ایجاد بن بست کند بنابراین فضای کاری بسیار مهم است .



www.mhzir.com

این روش ***Rollback*** کردن را نمی توانیم برای هر فضای کاری استفاده کنیم.

مثلا" سیستم عامل ما تعاملی است یعنی کاربر دستور می دهد و منتظر پاسخ می ماند مانند سیستم عامل های ما ، حال آیا می توان مثلا" به صورت ***Rollback*** کار کنیم . مثلا" کاربر دارد فیلم نگاه می کند و ... پس بسته به شرایط می توانیم تشخیص دهیم از چه راه حلی استفاده کنیم .

www.mbzir.com

مثال :

سیستمی شامل پنج فرآیند و سه منبع ***ABC*** است . از ***A*** ، ۷مورد منبع داریم . از ***B*** ، ۲ مورد داریم . از ***C*** ، ۶ مورد داریم .

	A	*B*	*C*	*A*	*B*	*C*
*P*۱	۰	۱	۰	۰	۰	۰
*P*۲	۲	۰	۰	۲	۰	۲
*P*۳	۳	۰	۳	۰	۰	۰
*P*۴	۲	۱	۱	۱	۰	۰
*P*۵	۱	۰	۲	۰	۰	۲

سوال اول : آیا سیستم در بن بست قرار دارد ؟

سوال دوم : ***REQUEST*** ۳ ***[۰،۰،۱]*** در زمان ***t***۱ می رسد ، آیا بن بست اتفاق افتاده یا نه ؟

سوال سوم : کدامیک از این فرآیندها در بن بست قرار دارند ؟

سوال چهارم : آیا سیستم عاملی وجود دارد که راه حل برای بن بست ارائه دهد ؟

سوال پنجم : ما می دانیم که بن بست در حالت سه گانه اجرا می شود . حال فرض کنیم که بخواهیم الگوریتم تشخیص بن بست را بدهیم که در حالت ***ready*** بحث می کند . اگر بخواهیم الگوریتم تشخیص بن بست را در الگوریتم انجام دهیم چه زمانی باید این کار را انجام دهیم ؟(زمانی که شلوغ است یا زمانی که خلوت است ؟)

پاسخ سوال اول :

AV= (۷-۷،۲-۲،۶-۶) $\Longrightarrow$ ***AV= (۰،۰،۰)***

همانطور که مشاهده می شود موجودی هیچ چیز در دست نداریم

در جدول بالا همانطور که مشاهده می شود نه ***P***۱ ونه ***P***۲ هیچ کدام نیاز به منبع ندارند پس باید بسته به شرایط یکی را انتخاب کنیم .

P۱ انتخاب $\Longrightarrow$ ***AV=[۰،۱،۰]*** $\Longrightarrow$ ***P***۳ انتخاب $\Longrightarrow$ ***AV=[۳،۱،۳]*** $\Longrightarrow$

P۴ انتخاب $\Longrightarrow$ ***AV=[۵،۲،۴]*** $\Longrightarrow$ ***P***۲ انتخاب $\Longrightarrow$ ***AV=[۷،۲،۴]*** $\Longrightarrow$

P۵ انتخاب $\Longrightarrow$ ***AV=[۷،۲،۶]***

www.mbzir.com

مسیر امن وجود دارد و سیستم در بن بست قرار نمی گیرد :

$P_1 \longrightarrow P \longrightarrow P_2 \longrightarrow P_4 \longrightarrow P_5$

پاسخ سوال دوم :

در این سوال Max نداریم که بگوید احتمال بن بست داریم یا نه بلکه می گوید آیا بن بست اتفاق افتاده است یا نه .

حال اگر جواب بله بود باید به فکر خروج از بن بست باشیم و گرنه که هیچ .

$AV=[0,0,0] \Longrightarrow P_1$ انتخاب $\Longrightarrow AV=[0,1,0]$

دیگر در این حالت هیچ کدام از ***Request*** ها از ***AV*** کوچکتر نیستند پس بنابراین بن بست اتفاق افتاده است .

پاسخ سوال سوم :

همه ی آن چهار فرآیندی که مانده اند، در بن بست اند (P_2,P_3,P_4,P_5) در صورت ترسیم منابع قطعا" تشخیص حلقه می دهیم . توجه کنید که ما فقط تشخیص می دهیم نتیجه بر عهده ی طراح سیستم عامل است .

پاسخ سوال چهارم :

خیر ، حداقل سیستم عامل های مطرح این امکان را ندارند اما اگر در آینده هم قرار باشد این چنین سیستمی داشته باشیم فقط و فقط به سمت تشخیص بن بست خواهد رفت .

پاسخ سوال پنجم : زمانی که خلوت است زیرا زمانی که سیستم شلوغ باشد احتمال بروز بن بست بالا می باشد و الگوریتم تشخیص بن بست سرباری بر روی سیستم است و ما باید کاری کنیم که سربار سیستم کم شود اما الگوریتم تشخیص بن بست سربار را بالا می برد و بالا رفتن سربار به

www.mbzir.com

منزله ی کم شدن سرعت است . اما زمانی که سیستم خلوت باشد زمان مناسبی است برای الگوریتم تشخیص شاید یکی از دلایل بن بست خلوت بودن سیستم باشد زیرا هر فرآیند هر چقدر منبع بخواهد مصرف می کند و این خلوتی باعث بن بست می شود .

در این فصل مباحث مربوط به فرآیند ها را گفتیم .که شامل مواردی چون : تعامل بین فرآیندها (چه مسائلی مطرح می شود) ، سیستم عامل ها (سمافور ها) ، مساله ناحیه بحرانی ، مساله بن بست (چه است ؟ چه راه هایی برای خروج دارد ؟ شرایط چیست ؟ و اگر بخواهیم آن را رفع کنیم چه تکنیکی داریم ؟) که این مسائل فعلا بر روی کاغذ است و ارزش تجاری ندارند .

www.mbzir.com

سوالات فصل چهارم

۱- مقدار نهایی سمافور ***s*** با مقدار اولیه صفر که به صورت تصادفی ۲۵ بار ***signal*** و ۲۳ بار عمل ***wait*** روی آن شده است چیست ؟

۲=۲۳-۲۵ =جواب

۰

-۱

-۲

۲۵ تا اضافه میشه

۰

۰

۰

-۲۳

signal

۲- فرض کنید دو فرایند ***p***۱ و ***p***۲ به صورت هم روند وجود دارند . در این صورت کدام یک از موارد زیرنمیتواند خروجی دو فرایند باشد:

الف) $c^* A^* B^* D$

ب) $(AB)^* (CD)^*$

ج) $A(CD)^* D$

د) BCAD

P۱	P۲
REPEAT	REPEAT
……..	……..
PRINT'A'	PRINT 'c'
PRINT'B'	PRINT'D'
……	……
forever	forever

۳- ۳ فرایند ***P***۱,***P***۲,***P***۳ در حال اجرایند . طبق جدول زیر از چپ به راست دستورات ***P*** یا ***wait*** و دستورات ***v*** یا ***signal*** را روی ***semaphore*** عمل میکنند .مقدار اولیه ی آن ***s***=۱ است اجرا مکنند در صورتی که دو فرایند متوقف باشند و دستور ***v*** اجرا شود .فرایند ی که شماره بزرگتری دارد برای راه اندازی پیدا میکنند . حالت این ۳ فرایند پس از اجرای مجموعه دستورات زیر چیست ؟

فرایند	P۱	P۲	P۳	P۲	P۱	P۳	P۲	P۲	P۳	P۱
فرمان که اجرا میشود	P(s)	P(s)	V(s)	P(s)	V(s)	V(s)	P(s)	P(S)	V(s)	P(s)
شمارنده ی سمافور	۰	-۱	۰	-۱	۰	۱	۰	-۱	۰	-۱
بلوک شده	_	P۲	_	P۲	_	_	_	P۲	_	P۱

الف)همه ی فرایند ها در حال اجرا هستند .

ب)***P***۱,***p***۲ در حالت اجرایند و ***p***۳ در حالت ***block***

www.mbzir.com

ج) P_1, p_3 در حالت اجرایند و p_2 در حالت ***block***

د) P_3, p_2 در حالت اجرایند و p_1 در حالت ***block***

قطعا p_1 در حالت اجرا نیست . نمیتوان گفت p_2, p_3 کجا هستند .p_2, p_3 در حالت آماده اجرا و یا اجرا هستند .همیشه فقط ۱ فرایند نزد پردازنده است.

۴- دو فرایند روبرو به صورت هم روند اجرا می شوند .در صورتیکه مقدار اولیه متغیر ***a*** صفر باشد .بعد از اجرای کامل دو فرایند روبرو کدام گزینه مقادیر متغیر های ***a,b,c*** را نشان نمیدهد ؟

P۱	P۲
.....	.....
a =۱	b =a
.....	c =a
	.....

الف) b=*0*, a=c=*1*

ب) a=*1*,b=c=*0*

ج) c=*0*,a=b=*1*

د) a=b=c=*1*

۵- مسئله ***producer-consumer*** با تعدادی بافر نامحدود بین دو فرایند در نظر بگیرید.

Var m,a: semaphore;

Producer	***consumer***
Repeat	***Repeat***
Produce an item;	***P(a);***
P(m);	***P(m);***
Append the item to the buffer;	***Retrievean item from the buffer***
V(m);	***V(m);***
V(a);	***Consumer the item;***
Until false;	***Until false;***

حال اگر مقدار سمافور های ***m***=۱,***a***=۰ باشد این راه حل را چگونه ارزیابی میکنید؟

الف-کاملا درست است.

ب-امکان بن بسست وجود دارد .

ج-انحصار متقابل را تضمین نمیکند.

د-مشکل گرسنگی دارد.

نکته۱: مسائل ***Ipc*** هیچ وقت کد نویسی ندارند ولی باید خوب آن را تحلیل کنیم.

نکته ۲: ***a***=۰,***m***=۱ ما را راهنمایی میکند.

نکته۳:بافر نامحدود است پس وقتی بافر نامحدود باشد نیازمند سمافوری برای کنترل سایز بافر نیستیم .

وقتی الف محقق میشود که ۳ تا شرط صحیح باشند .۱-انحصار متقابل ۲-انتظار محدود ۳-پیشرفت

گرسنگی داشته باشد یعنی پیشرفت ندارد.ما هیچ جای مسئله ***wait***و***signal*** نمیبینیم یعنی ظاهرا وجود ندارند اما واقعا هستند . یک تابع روی سمافور یا ***wait***است یا ***signal***،یعنی***p***معادل ***wait***است. ما باید بعد از تولید آیتم ***wait*** کنیم تا ببینیم مصرف شده است یا نه ؟ ***append*** یعنی تولید را میگذارد در بافر و بعد باید یکی را بیدار کند.***v(m)***همان ***signal***است.

M سمافور کنترل ناحیه ی بحرانی (***m***=۱ بود یعنی احتمالا ناحیه بحرانی است) میخواهیم حتما ***producer***انجام شود و بعد ***consumer***که همان سمافور هماهنگی است . ***a*** اول صفر است بعد کم میکند .

۶- کد زیر برای حل مسئله کلاسیک تولید کننده – مصرف کننده داده شده است . ***count***متغیر سراسری با مقدار اولیه صفر و ***n*** تعداد خانه های بافر است .راه حل زیر را ارزیابی کنید؟

Producer

```
Void producer(void){
While(true){
Produce-item();
If(count==n)sleep();
Enter-item();
```

consumer

```
Void consumer (void){
While(true){
If(count==۰)sleep();
remove-item();
Count--;
```


www.mbzir.com

الف- ممکن است تولید کننده و مصرف کننده هر دو به خواب ابدی فرو روند (بن بست)

ب-مشکل انتظار مشغولی دارد.

ج-هردو همزمان به داده مشترک دستیابی پیدا میکنند (نقض انحصار متقابل)

د-هیچ مشکلی وجود ندارد.

۷- آیا که زیربرای مسئله ناحیه بحرانی بین دو فرایند همروند قابل قبول است؟چرا؟(اپراتور ***and*** به معنای اجرای همروند است)

```
{int turn;
Boolean flag [۲];
Proc(i);
Int(i);
{while(true)
    {compute;
      Flag[i]=true;
      Turn=(i+۱)mod۲;
      While(flag[(i+۱)mod۲]&&turn==i);
      Critical-section;
       Flag[i]=false;
    }
```

الف-خیر ،زیرا***dead lock*** وجود دارد.

ب- بلی،زیراشرط ***me***برقرار است.

ج-خیر،زیرا شرط ***me*** برقرار نیست .

د-بلی،زیرا ***dead lock***وجود ندارد.

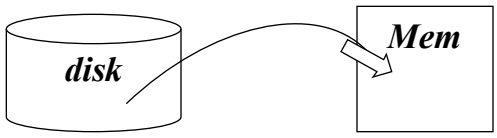
www.mbzir.com

www.mbzir.com

فصل پنجم

۱-مدیریت حافظه :

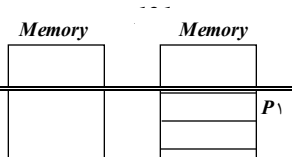
یکی از وظایف سیستم عامل مدیریت حافظه است و تخصیص حافظه به فرآیندها . فرآیند زمانی فرآیند است که به آن حافظه دهیم در غیر اینصورت یک برنامه است .



وظیفه ی سیستم عامل پیگیری فضاهای خالی و فضاهای تخصیص داده شده است .

۲-مفاهیم مدیریت حافظه :

۱- جابجایی (*Relocation*) : برای فرآیند فرق ندارد که کجای حافظه بنشیند اما به این سادگی هم نیست . مسئله این است که چطور می توان این قابلیت را در فرآیند ایجاد کرد ؟ زمانی که آدرس های تولید شده مطلق نباشند بلکه نسبی باشند.

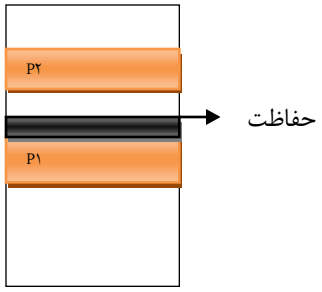


آدرس های حافظه منحصر به فرد می باشند . اما اگر فرآیندهای ما خودشان برنامه ریزی کنند و مکان مشخصی را بخواهند به این آدرس ، آدرس نسبی گوییم . زمانی می توانیم جابجایی داشته باشیم که آدرس های ما ***relative*** باشند .

در این میان این کامپایلر است که باید کار ترجمه را انجام دهد (ترجمه آدرس منطقی به آدرس فیزیکی). زمان اجرا کامپایلر وظیفه اش این است که آدرس های نسبی (منحصر به فرد نبستند و نسبت به برنامه کد نویسی شده اند) را بگیرد و به آدرس مطلق (منحصر به فرد) تبدیل کند.

۲- حفاظت (*protection*):

فرآیند ها باید به گونه ای باشند که به حد و مرز همدیگر تجاوز نکنند (یعنی نه ***P***۲ بتواند برود بالا و نه ***P***۱بتواند بیاید پایین) . یعنی مرز بندی کامل فرآیندها در حافظه ی اصلی انجام شود .

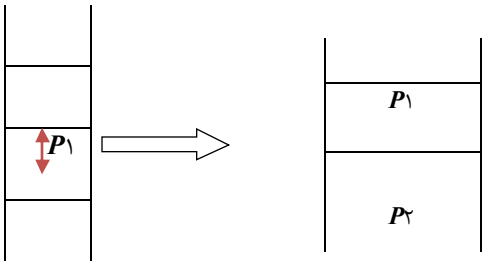


۳- سازماندهی (*organization*): دو نوع سازماندهی داریم :

۱- سازماندهی فیزیکی (***physical organization***): چیزی است که واقعا" جود دارد ممکن است فرآیندها در حافظه ی ما باشند و دقیقا" بدانیم چه اتفاقی می افتد .

www.mbzir.com

۲- سازماندهی منطقی (***logic organization***): آنی است که به نظر می آید و به نظر می آید فضا یک فضای پیوسته است .



www.mbzir.com

۳-روش های مدیریت حافظه

۱. روش ایستا ***static***

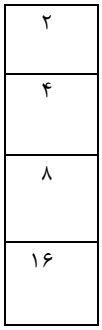
۲. روش پویا ***dynamic***

۱.روش ایستا

در این روش حافظه به بخش های ثابت از پیش تقسیم شده ، تقسیم شده است . که این روش خود شامل سه روش است :

۱. پیگیری قطعه های ***MB***۸

۲. کوچکتر از ***MB***۸ فضای باقی مانده قابل استفاده نیست . (این روش ایستا در هیچ سیستم عملی استفاده نمی شود بلکه فقط برای سیستم عامل های خاص مانند ***ECU*** ماشین کاربرد دارد)



www.mbzir.com

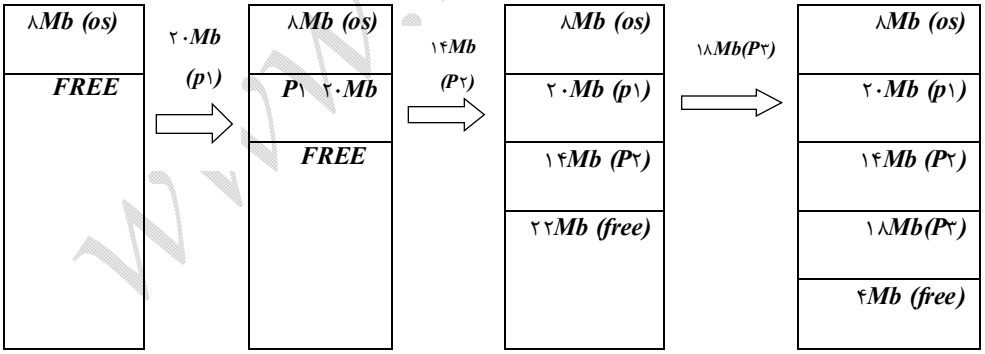
۳. لزومی ندارد بخش ها به قطعات مساوی تقسیم شود یعنی تقسیم بندی ها بر پایه ی توان ۲ ایجاد شد .

۳۲
۶۴ ***MB***

نکته : ما دو بحث داریم به نام های ۱. ***Simplicity*** ۲. ***flexibility***

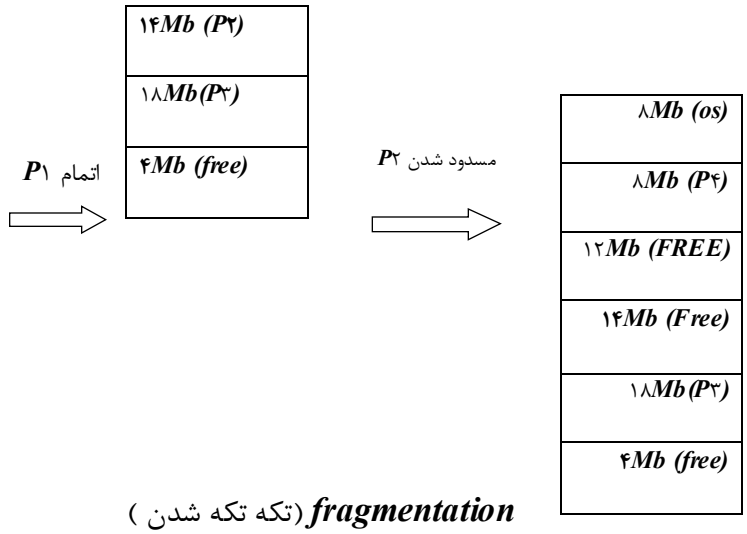
که این دو معمولا" با هم ناسازگارند و معکوس . مانند دوچرخه و ماشین. دوچرخه ساده است اما انعطاف پذیری زیاد ندارد . مثلا" سرعتش کم است ، در زمستان نمی توان از آن استفاده کرد و ..... ولی در عوض ماشین پیچیده است و انعطاف پذیری اش بالا.و هر چه پیچیدگی بالا می رود بحث (***trade off***) پیش می آید . یعنی سبک سنگین کردن .

۲.روش پویا



۸Mb (os)
۸Mb (P۴)
۱۲Mb (FREE)

www.mbzir.com



پس مشکل در اینجا ***fragmentation*** (تکه تکه شدن) است و راه حل برای اینکه این اتفاق نیفتد این است که ***defragmentation*** شود که این را در ***hard disc*** دیده ایم . ابتدا می آییم آدرس ***MB*** ۱۸ را خط به خط کپی می کنیم در فضای خالی اما این کار در حافظه ***RAM*** که به آن میگوییم ***compactation*** هزینه بر است و کاری تکراری است و نمی توانیم انجامش دهیم. ولی در ***hard***قابل اجرا می باشد زیرا دوربین ما بر روی ***hard***، بر سرعت آن نیست بلکه بر روی حجم ذخیره سازی اش است . اما در ***RAM*** کلی هزینه باید بکنیم و راندمان پایین می آید و اتلاف زمان هم داریم . نکته ای که اینجا وجود دارد این است که این درج و حذف مدام باعث شخم زدن و در اصطلاح کچلی می شود .

۴-الگوریتم تخصیص حافظه در روش پویا:

Os (۸ Mb) ***P۱ (۴Mb)***	
Free(۷NB)	***F.F***
P۲ (۱۷Mb)	
Free(۷Mb)	***B.F***
P۳(۱۰ Mb)	
Free(۱۰Mb)	***W.F***
۶۴ Mb	

www.mbzir.com

در این روش چند الگوریتم داریم :

۱. اولین برازش (*First fit*): به این صورت می باشد که از ابتدای حافظه شروع می کنیم و به اولین خانه ای که رسیدیم فرآیند را داخل آن قرار دهیم در این روش زمان جستجو زیاد است اما مناسب ترین گزینه است .

۲. بهترین برازش (*best fit*) : یک آنالیزی می کند ببیند کدام مناسب تر است . در این روش زمان جستجو زیاد به این امید که باقی مانده قابل استفاده باشد .

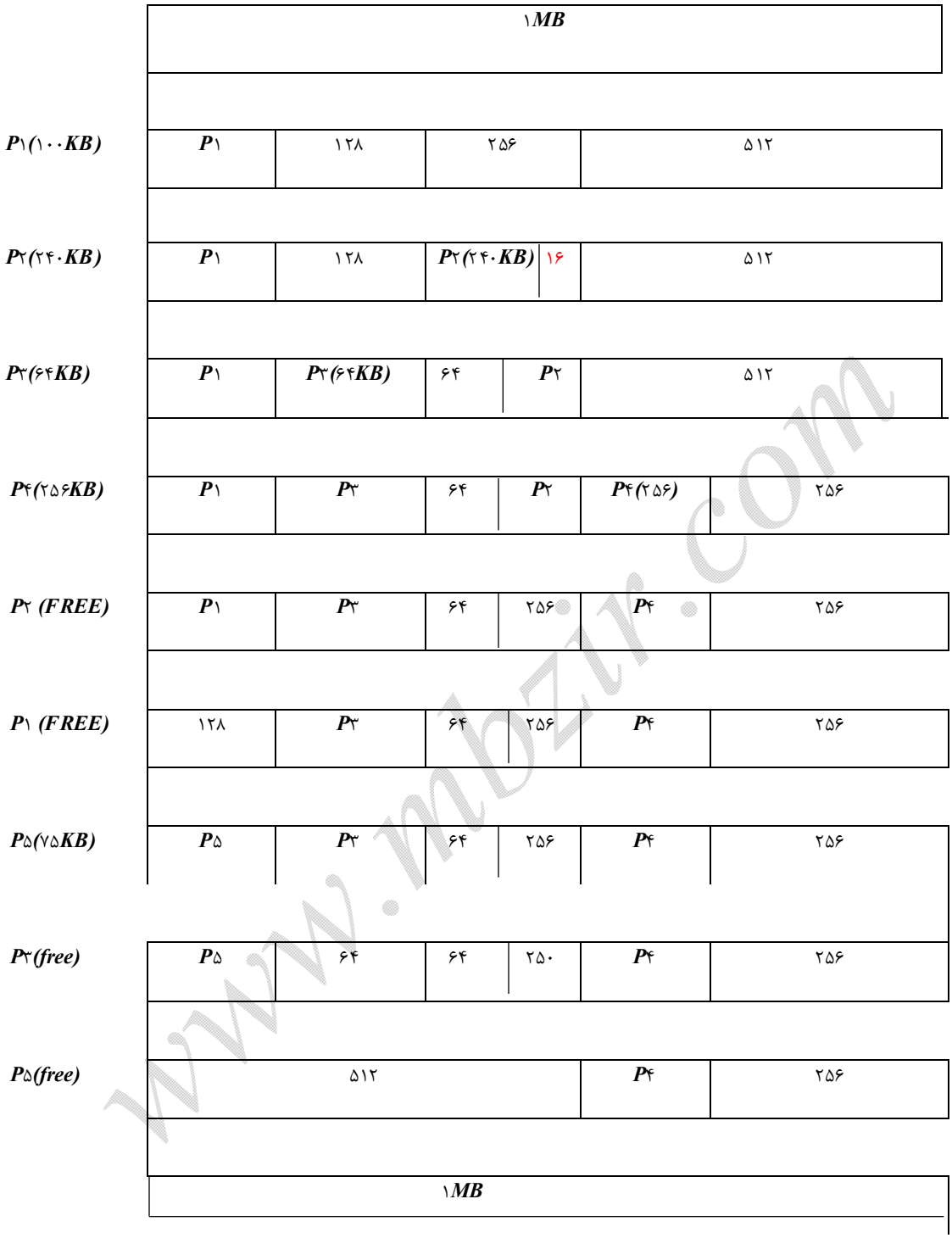
۳. بدترین برازش (*worst fit*) : نامناسب تریت را در نظر می گیریم در این روش زمان جستجو نداریم و ممکن است انتخاب ما بهترین انتخاب نباشد .

۴. برازش بعدی (*next fit*) : همان ***first fit*** است ولی هر بار از ابتدا شروع نمی کند حافظه را پیمایش کند بلکه از آخرین خانه ی تخصیص یافته فرآیندها را در خانه ها قرار می دهد . زمان جستجو هم ندارد .

ابزارسنج (*bench mark*) : ابزار سنج های مختلفی وجود دارد که این روش های بالا را بررسی کرده اند و به این نتیجه رسیده اند که ***first fit*** و ***best fit*** بهترین حالت اند. که این دو خیلی شبیه به هم عمل می کنند که در بهترین حالت ۱/۳ فضای حافظه تلف می شود که این زیاد است .

۵-روش رقابتی (*Buddy*) :

www.mbzir.com



تلفیقی از روش ایستا و پویا به صورت بازگشتی پیاده سازی می شود . در تخصیص ، فضای حافظه را به نزدیک ترین توان ۲ می دهیم .در آزاد سازی هم اگر دو قسمت رفیق آزاد باشند تلفیق صورت می گیرد. (مشکل این روش اتلاف فضای حافظه است.)

www.mbzir.com

سیستم عامل باید بداند در بحث های مدیریتی حافظه ، فضای خالی و استفاده شده چه چیزهایی هستند وباید در رقابت های سیستم عامل باشد که فضای خالی و استفاده شده را در اختیار داشته باشد .

نکته : اصولا کامپایلر آدرس نسبی تولید می کند . زمانی ما آدرس نسبی تولید می کنیم که برنامه نویس خود بخواهد آدرس مطلق داشته باشد .

۶- صفحه بندی (*paging*)

ما در سیستم صفحه بندی به دنبال این هستیم که اولا" کاملا" بتوانیم از فضایمان استفاده کنیم و ثانیا به صورت پویا باشد .

یعنی ***Dynamic segmentation*** و ***no fragmentation*** باشد .

فرض کنید سه فرآیند ***P۱,P۲,P۳*** و یک حافظه ***KB*** ۶۴ داریم . ابتدا فرآیندها را به قسمت های مستقیم تقسیم می کنیم مثلا " به قسمت های ***KB*** ۴ (این اعداد ، اعداد واقعی می باشند .)

قسمتهایی را که تقسیم می کنیم را ***Page*** (صفحه) می گویند و معمولا" در سیستم عامل های مختلف سایز آنها فرق دارد. کل هدف ما این است که ***logical view*** را به ***physical view*** تبدیل بکنیم . هر کدام از ***P۱,P۲,P۳*** باید برای اجرا شدن در فضای حافظه بنشیند .

نکته این است که هر کجا در حافظه جا داشته باشیم ، فرآیندها را قرار می دهیم . فضای کل استفاده شده و فضای تکه تکه شده وجود ندارد .

نکته : سیستم عامل باید بداند که یک فرآیند در کجای حافظه اجرا می شود . برای این منظور سیستم عامل از یک ماهیتی به نام ***Page table*** (جدول صفحه) استفاده می کند . برای هر فرآیند این جدول را دارد به عنوان مثال برای ***P۱*** ، ۵ خانه دارد .

به خاطر رفع مشکل ***FRAGMENTATION*** این تکنیک را استفاده کردیم . دقت کنیم که قرار نیست به صورت پیوسته باشد .اگر پیوسته باشد می شود همان ***Dynamic segmentation*** جلسه قبل و ***fragmentation*** ایجاد می شود .

قرار نیست قرار گیری در حافظه پیوسته باشد ، یعنی در ***paging*** ، ***defragmentation*** نداریم

در ***page table***فرآیند ***P***۱ را کاملا" پیوسته می بینیم . یعنی سمت چپ در ***page table*** فرآیند ***P***۱ ، ***logical view*** است و آن چیزی که ما در سمت راست می بینیم ***physical view*** است که آن چیزی است که واقعا" وجود دارد . آدرس هایی که تولید می شوند قطعا" منطقی هستند تا ***Rellocation*** داشته باشیم . ما باید به طریقی آدرسهای منطقی را به آدرس های فیزیکی تبدیل کنیم .

اما سوال این است که چگونه این کار را انجام دهیم ؟

یک فرآیند شامل کد ، ***data***، پشته می باشد .

- روال تبدیل آدرس منطقی به آدرس فیزیکی :

همانطور که دیدیم هر قسمت در فرآیند ***P***۱ را به ۴***KB*** تقسیم کردیم و هر قسمت در حافظه را هم به ۴***KB*** تقسیم کردیم . سوال این است که ۴***KB*** چند خانه دارد ؟ و برای ۴***KB*** چند خط آدرس نیاز داریم ؟

۴***KB*** = ۴ ×۲^۱۰= ۴×۱۰۲۴

۴*۱۰۲۴ تا خانه داریم که از صفر شروع می شود تا ۴*۱۰۲۴; در واقع ما یک خانه از حافظه را برداشتیم و خانه های آن را شمردیم .

حال می خوایم ۱۶ خانه را در حافظه آدرس دهی کنیم . برای ۱۶ خانه چند بیت نیاز داریم ؟ ۴

پس ۴بیت برای ۱۶ خانه نیاز داریم و هر خانه هم ۴*۱۰۲۴ خانه دارد :

۲^۲×۱۰^۲ = ۱۲^۲

پس برای هر خانه هم ۱۲بیت برای آدرس دهی نیاز داریم پس در کل می شود :

۱۲+۴=۱۶

پس برای حافظه اگر بخواهیم تک تک خانه ها را آدرس دهی کنیم ، ۱۶ خط آدرس نیاز داریم .

روش دوم این است که حافظه ۶۴***KB***است که برابر با : ۶۴***KB*** = ۲^۶ ×۲^۱۰ = ۲^۱۶

www.mbzir.com

پس باز هم۱۶ خط آدرس نیاز داریم .که این ۱۶ خط آدرس به شکل زیر تفکیک شده است :

۴	۱۲
شماره قاب	شماره ی خانه ی حافظه در قاب

فرض کنیم یک آدرس تصادفی را می کشیم بیرون

یک آدرس تصافی ***H*** ۴ ۵ ۱ ۷

۰۱۰۰ ۰۱۰۱ ۰۰۰۱ ۰۱۱۱

بالاخره این آدرس در یکی از خانه های حافظه است . باید مانند مدل بالا دیکد کنیم یعنی جای شماره قاب و شماره خانه را تغییر می دهیم :

۷	۱	۵	۴
شماره قاب	شماره ی خانه		

اگر در حافظه نگاه کنیم در قاب ۷ ام نوشته شده است ***P۱(۲)*** ، یعنی در این قاب شماره ی دوم فرآیند ***P۱*** قرار دارد .

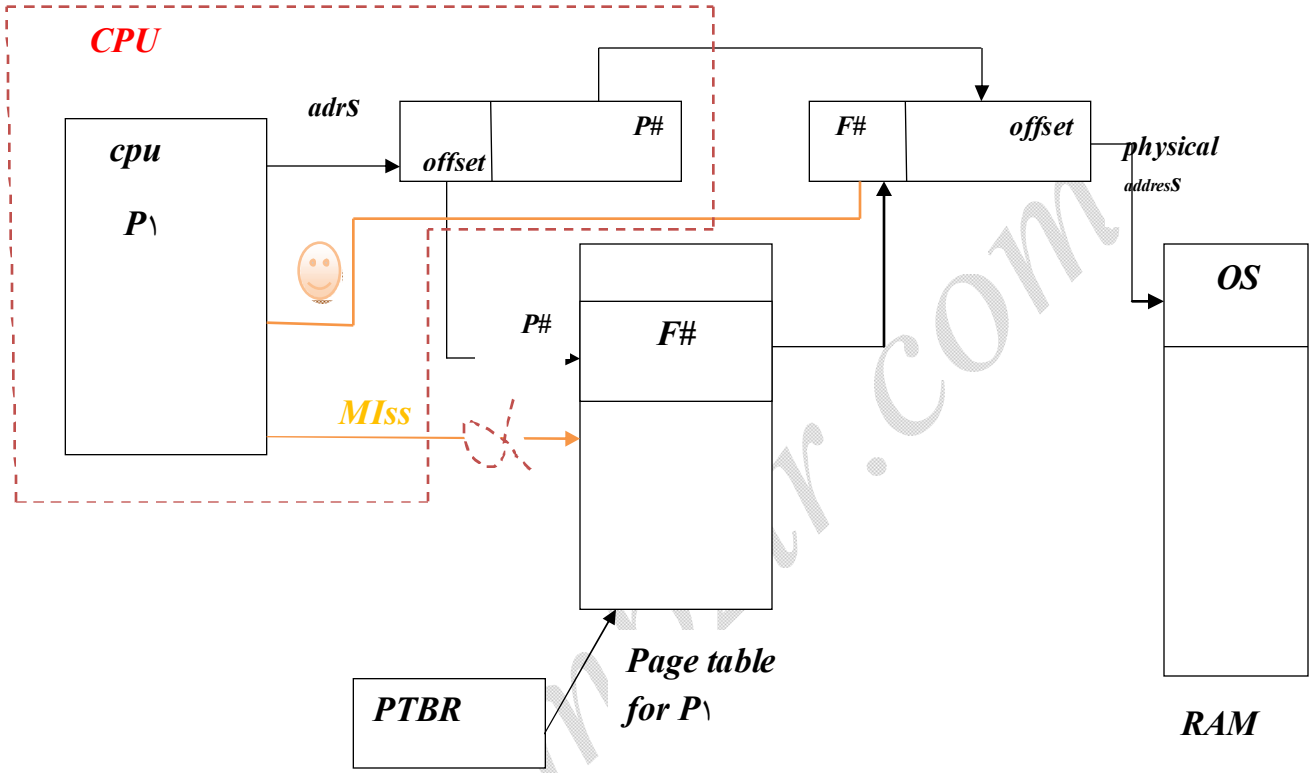
ما به دنبال خانه ی ۴ ۵ ۱ ۷هستیم . کافی است برویم در فرآیند ***P۱*** و خانه ی ۲ را نگاه کنیم ، پس قاب تبدیل به صفحه شد .

پس ۴ ۵ ۱ ۷را می نویسیم ۴ ۵ ۱ ۲.

یعنی شماره قاب را به شماره صفحه تبدیل کردیم . سایز هر دو و تعداد خانه هایشان یکسان است . فقط در ***RAM*** به صورت فیزیکی است و در دیگری به صورت منطقی است . به عبارتی باید گفت تا اینجا فقط ***Frame #***تبدیل به ***page#*** شد .

www.mbzir.com

پس این اتفاق به کمک جدول صفحه و سیستم ***paging*** اتفاق افتاد . شکل آن را رسم می کنیم تا به فهم ما بیشتر کمک کند :



در پرازنده هر بار یک فرآیندی دارد اجرا می شود ، فرض کنیم دراد اجرا می شود . آدرس تولیدی قرار بود به صورت منطقی باشد .آدرسی ما یک ***P#*** دراد و یک فاصله هم از ابتدای خانه دراد و پردازنده این آدرس را تولید کرده است و ما یک جدول صفحه هم داریم . ***P#***مراجعه می کند به جدول صفحه و از آنجا ***F#*** را بر می دارد ۱۵۴ ***OFFSET*** را دست نزدیم ، فقط ***P#*** تبدیل شد به ***F#*** و خروجی ***physical adrs*** شد و به حافظه ی ما رفت .

CPU قرار بود خروجی اش آدرس فیزیکی باشد ولی در شکل نوشته شده آدرس متطقی . اما در واقع به این شکل است که کل بخش نقطه چین در داخل ***CPU*** قرار دارد . و در حال حاضر خروجی ***CPU***، ***physical adrs*** خواهد بود .

نکته : سیستم عامل از کجا متوجه شود که فرآیند ***p*۱** را باید در جدول صفحه اش ***load*** کند ؟ یک ثباتی وجود دارد به نام ***page table base register*** . سیستم عامل می آید سهم زمانی را تنظیم میکند . یکی از چیزهایی که نیز است آن را تنظیم کند این است که کدام ***page table*** به حافظه ***load*** شود که با استفاده از ***PTBR*** این کار را انجام می دهد . این ثبات از نوع ثبات هایی می باشد که من برنامه نویس آن را نمی بینم . ***PTBR*** توسط سیستم عامل ***SET*** می شود .

پس هدف این است که آدرس منطقی را به فیزیکی تبدیل بکنیم . برنامه های من کاربر آدرس منطقی تولید می کنند و برای اینکه آدرس منطقی را به آدرس فیزیکی تبدیل کنیم با ***Offset*** کاری نداریم و فقط کافی است ***#p*** برداشته و به جای آن ***#f*** را قرار دهیم .

#f ماهیت کاملا" سخت افزاری دراد که در حافظه موجودیت دارد ولی ***#p*** موجودیت ندارد . وقتی فرآیند می خواهد اجرا شود سیستم عامل ***page table*** را برای آن ***set*** میکند و سپس کنار می رود . وقتی عمل ***set*** کردن انجام شد پردازنده می فهمد که تبدیل آدرس چگونه انجام می شود .

این روش دو حسن دارد :علاوه بر اینکه ***fragmentation*** نداریم پویایی هم دارد . البته باید گفت که یک ***fragmentation*** کوتاه دارد که به آن ***internal fragmentation*** گوییم و فقط و فقط در صفحه ی آخر بوده است و قابل چشم پوشی است .

این روش مشکلاتی هم دارد : ما دو بار مراجع به حافظه داریم و در هر بار خواندن یک بار در ***page table*** به حافظه مراجعه می کنیم . و یک بار هم مراجعه اصلی به حافظه است .

یادمان باشد ما به دنبال شماره قاب هستیم .

offset فاصله از ابتدای خانه است که به آن ***displacement*** هم می گویند .

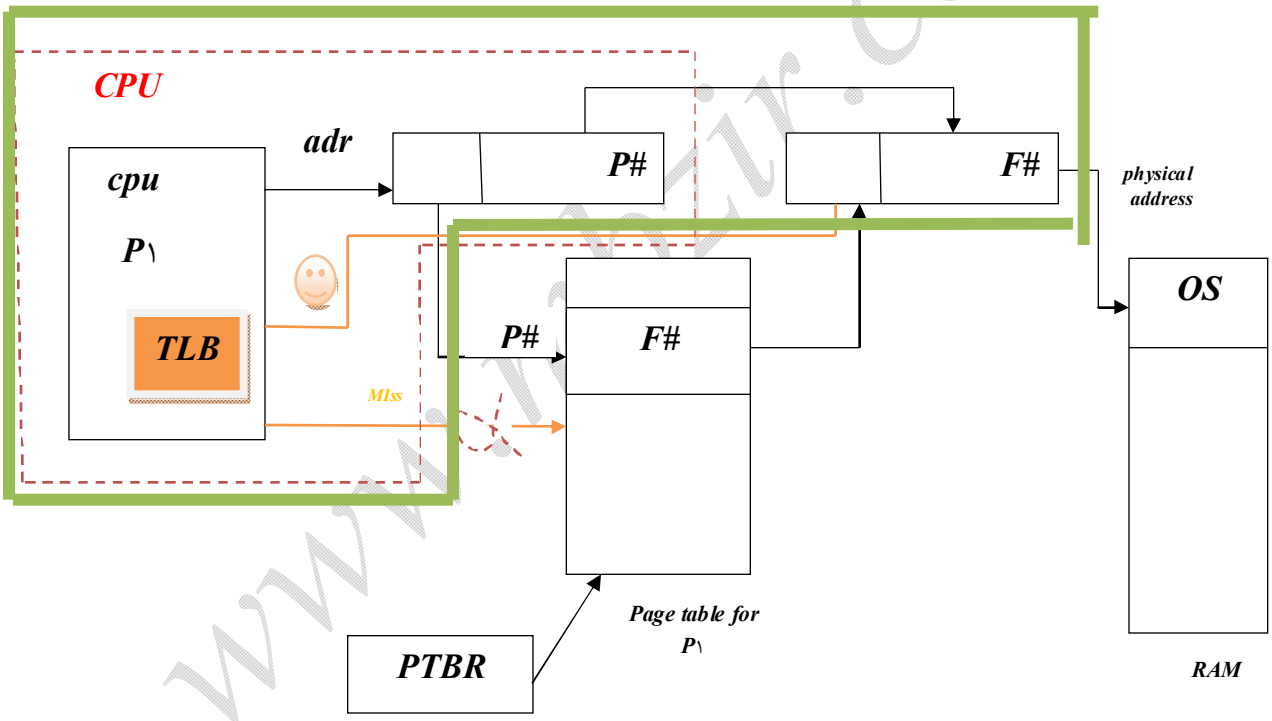
برنامه نویس آدرس منطقی تولید می کند و ***CPU*** آدرس فیزیکی .

در پردازنده هر بار یک فرآیندی دارد اجرا می شود . آدرس تولیدی قرار بود به صورت منطقی باشد . آدرس ما یک ***#P*** دارد و یک فاصله هم از ابتدای خانه دارد. پردازنده این آدرس را تولید کده است و ما یک جدول صفحه هم داریم . ***#p*** مراجعه می کند به جدول صفحه و از آنجا ***#F*** را بر

میدارد . ۱۵۴ ***offset*** را دست نزده ایم ، فقط ***P#***بدیل به ***F#*** شده و خروجی هم ***physical address*** شده و به حافظه رفت .

-معرفی بافر ***TLB***

همانطور که می دانیم پردازنده و رجیستر سرعت یکسانی ندارند و سرعت رجیستر کند تر است . راه حل کند بودن این است که یک بافر در پدازنده قرار می دهیم تحت عنوان ***TLB (translation look side buffer)*** بافر دم دستی ترجمه . نکته این است که تعداد خانه های آن خیلی کم است مثلا" ۳۲ که از ۰ تا ۳۱ می باشد که یک قسمت خیلی محدودی از جدول صفحه را قرار می دهیم در ***TLB*** و پردازنده برای هر بار مراجعه به جای مراجعه به ***page table*** به ***TLB*** مراجعه میکند .



سوال : چقدر میزان دستیابی ***CPU*** به ***TLB*** موفقیت آمیز است ؟

۹۸-۹۹٪ درخواست ما داخل ***TLB*** می باشد و پیش بینی می کند و می داند چگونه باید پر شود . در ۱-۲٪ مراجعه به حافظه داریم وناموفق خواهیم بو د.

نکته : ***TLB*** از جنس رجیستر است و از جنس حافظه نیست و میزان دسترسی به آن در حد ***ns*** می باشد مهم است که در شکل کدام قسمت ها سخت افزاری و کدام قسمت ها نرم افزاری می باشند .

تنظیم جدول صفحه نرم افزاری است و سیستم عامل این کار را انجام می دهد . ***PTBR*** نرم افزاری است و سیستم عامل این مار را انجام می دهد. تبدیل آدرس سخت افزاری است .

نکته : روح سیستم عامل هم از ***TLB*** خبر ندارد . طراح سخت افزار ***TLB*** را برای بالا بردن راندمان کارش قرار داده است . بعضی کتاب ها در شکل تفاوت دارند و می گویند بخش سبز رنگ در شکل بالا سخت افزاری است.

نکته : ***TLB*** ابتدا خالی است و به ازای هر درخواست کاربر به تدریج پر می شود .

مثال :

فرض کنید حافظه ای با زمان دسترسی ۵۰***sec*** داریم و دسترسی به ***TLB*** با زمان ***sec*** ۲ می باشد . فرض کنیمد ٪۹۸اطلاعات را می توانیم در ***TLB*** پیدا کنیم . زمان موثر دسترسی به حافظه چقدر است ؟ (***T effective***)

T eff= ٪۹۸ (۲+۵۰)+ ۰,۰۲(۲+۵۰+۵۰)=۵۳ ***nsec***

یعنی خیلی ضریب ٪۹۸ مهم است . انگار که ۵۲ بعلاوه ۱ شده و عملا" در کل یکبار مراجعه به حافظه صورت گرفته است .

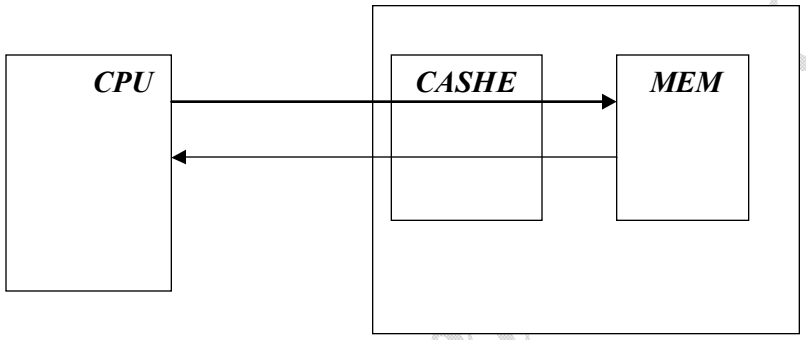
www.mbzir.com

۷- حافظه کاشه (*CACHE*)

دلیل اینکه می گوییم حافظه پنهان این است که از دید سیستم عامل مخفی می باشد . معماری زیر را در نظر بگیرید :

ضعف این معماری در حافظه است که سرعت آن بسیار پایین می باشد . و سرعت فرآیند را فرآیند کند تر تعیین می کند . پس از حافظه ی کاشه استفاده ی میکنیم .

حافظه کش از جنس ***register*** نیست بلکه از جنس حافظه است .اما ***TLB*** از جنس حافظه بود . سایز کش نسبت به حافظه خیلی خیلی کم تر می باشد . پس سرعتش بالاتر است . سیستم عامل فقط با مموری کر می کند و از کش هیچ اطلاعاتی ندارد . و کش هم داخل پردازنده است .



ادامه ی مثال قبل :

در صورتی که از حافظه پنهان با ضریب اصابت ۰,۹ و زمان دسترسی به آن ***nsec*** ۱۰باشد زمان موثر برای دسترسی به حافظه چقدر است ؟

Teff= ۰,۹ ×۱۰ + ۰,۱(۱۰+۵۰)=۱۵ ***nsec***

مثال سوم :

حال اگر در معماری مثال دوم سیستم ***paging*** را اضافه کنیم آنگاه متوسط زمان دسترسی به حافظه چقدر می شود ؟

paging with TLB =Teff= ۰,۹۸(۲+۱۵)+۰,۰۲(۲+۱۵+۱۵)=۱۷,۳ ***nsec***

paging without TLB =Teff=۱۵+۱۵=۳۰ ***nsec***

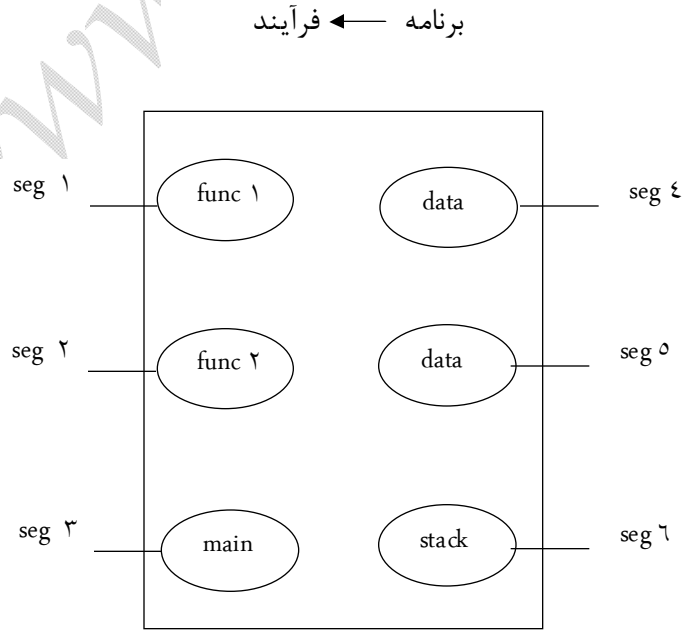
۸- مروری بر جلسه قبل

گفتیم که مکانیزم تبدیل آدرس سخت افزاری می باشد . و سیستم عامل خودش ***PTBR*** را تنظیم می کند اما ایرادش این بود که ۲ بار به حافظه مراجعه می کرد که این کار باعث می شود سرعت و کارآیی سیستم به شدت پایین بیاید . سپس یک بافر ***TLB*** در ***CPU*** قرار دادیم که تقریبا" یک بار دسترسی به حافظه صورت می گرفت . حال می خواهیم مدل دیگری از مدیریت حافظه را بیان نماییم .

۹- مدیریت حافظه قطعه بندی (***SEGMENTATION***)

این مدل اصلا" اهمیت ***paging*** را ندارد یعنی ***paging*** اهمیتش خیلی بیشتر از قطعه بندی می باشد . اما دلیل آمدن این مدل این است که مدیریت اصلی پردازنده ی اینتل ***segmentation***می باشد. در پردازنده اینتل الویت اول ***segmentation*** است و الویت دوم ***paging*** است . یعنی کاملا" اختیاری می باشد . که یک نفر می تواند ***segmentation*** را بردارد یا ***paging*** را . این مدل یک مدل قدیمی و بر مبنای ساختار ذهنی برنامه نویس می باشد .

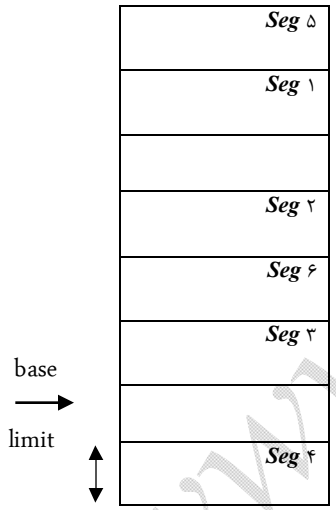
برنامه نویس یک برنامه دارد و آن را به ***segment***(قطعه های) مختلف تقسیم می کند .



فرآیند شامل : کد ، دیتا ، پشته میباشد . برنامه نویس برنامه را اینگونه برای خودش قطعه قطعه می کند . و همچنین حافظه را هم قطعه قطعه می کند (مانند شکل مقابل)

لزومی به مساوی بودن این قطعات تقسیم شده در حافظه نیست. به تعداد سایز ***segment*** این قطعا را در نظر می گیریم . و تفاوتش با ***paging***این است که آن را به قطعات مساوی تقسیم می کنیم.

به ازای هر فرآیند یک جدول قطعه داریم که جدول قطعه می خواست بررسی کند کدام صفحه در کدام قاب قرار دارد ما می خواهیم ببنیم این برنامه که تبدیل به فرآیند می شود در کجای حافظه و به چه صورت نشسته است .



باید یک ***segment table*** درست بکنیم که این جدول به ازای هر فرآیند می باشد .

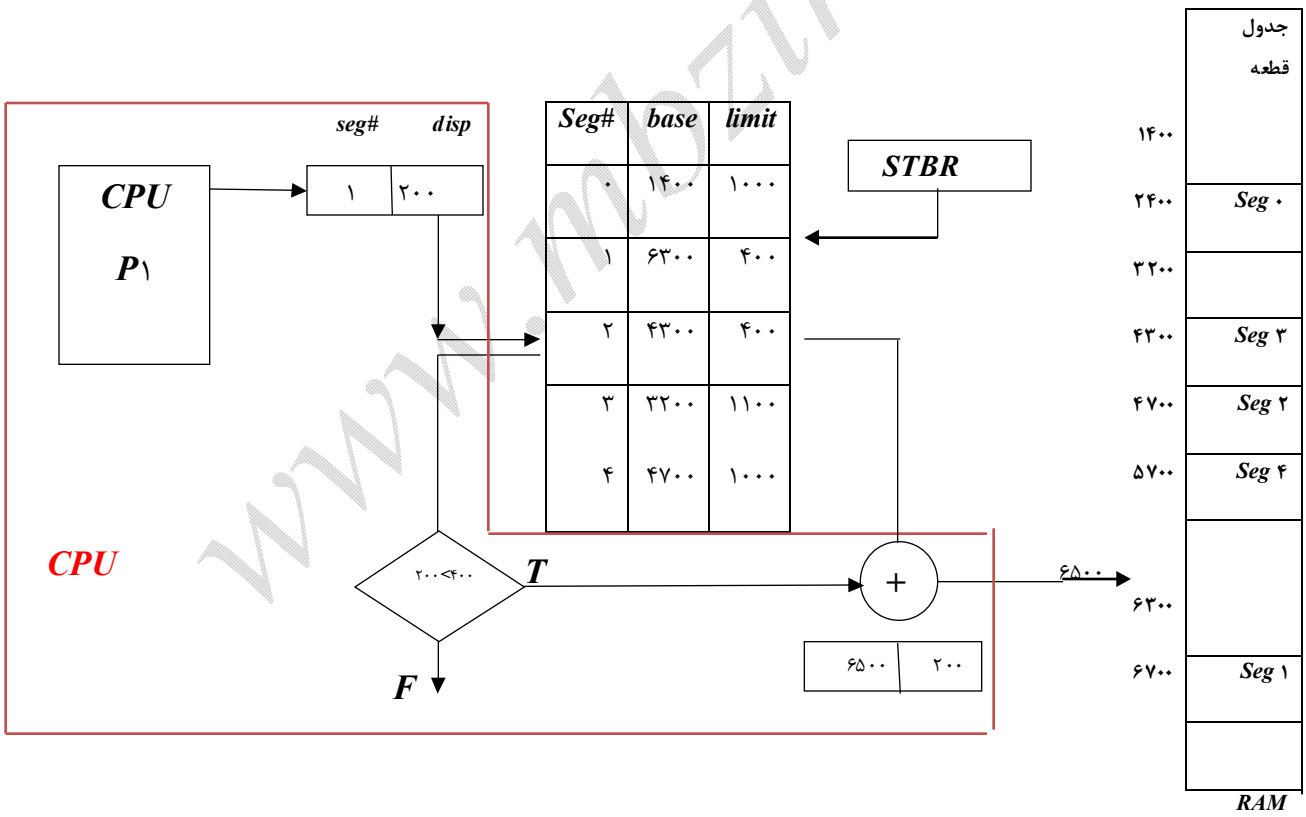
Seg#	*Base*	*Limit*
۰		
۱		
۲		
۳		

www.mbzir.com

۴		
۵		
۶		

segment table for Px

یک مثال واقعی زده و آن را تشریح می کنیم :



یک ***CPU*** داریم که فرضا" دارد ***P۱*** را اجرا می کند آدرسی که به ما می دهد شامل ***seg#*** و ***disp*** است . این انتقال پیدا می کند به جدول (این اعداد کاملا" واقعی می باشند .)

فرض کنیم آدرس ۲۰۰ ۱ تولید می شود پس ما به ۱ ***seg*** مراجعه می کنیم و قبل از این یک

checkup انجام می دهیم که آیا ۲۰۰ کوچکتر از ۴۰۰ است (باید ۲۰۰ از ۴۰۰ کوچکتر باشد) اگر اینطور بود ادامه پیدا می کند در غیر اینصورت یک وقفه غیر مجاز رخ داده . اگر درست باشد ۲۰۰ را بعلاوه ***base address*** میکند و سپس می رود به عنوان مثال در نقطه ۶۵۰۰ در حافظه می ایستد . (۲۰۰+۶۳۰۰)

نکته ۱: در سیستم ***paging*** که مدل خیلی بهتری است آدرسها یکپارچه تولید می شوند . یعنی خود سخت افزار تفکیک را انجام می داد .

اما در سیستم ***segmentation*** کمپایلر سیستم ***segmentation*** خودش دو آدرس تولید می کند . ***seg#*** و ***disp*** یعنی خود آدرس تولیدی دو بخش است .

نکته ۲: فرض کنید ***p۱*** دارد اجرا می شود در سیستم زمانبندی ***R.R*** حال که نوبت ***p۲*** شد از کجا بفهمد که نوبت او شده و سیستم عامل چگونه بفهمد که کدام جدول قطعه را بارگذاری کند ؟ ما معتقدیم که جدول قطعه در قسمت های الولیه ***RAM*** در ***OS*** است .

پس از کجا باید بداند که جدول قطعه ***P۲*** را باید بازگذاری کند ؟ اگر مدیریت حافظه ***paging*** باشد ***PTBR*** را ***set*** میکند و اگر مدیریت حافظه ***segmentation*** باشد ***STBR*** را ***set*** میکند و وقتی کنترل از سیسنم عامل خارج شد و سیستم عامل ***reg*** مورد نظر را ***set***کرد خودش کنار می کشد .

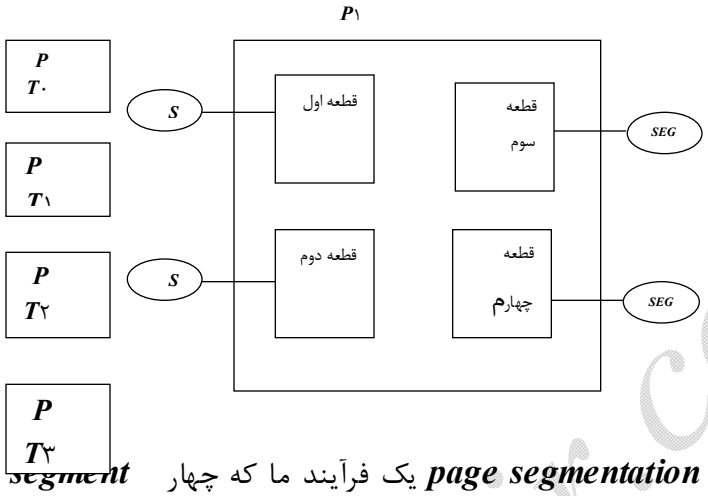
نکته ۳: این مدیریت حافظه (***segmentation***) چه ایرادی دارد ؟ مشکل ***fragmentation*** داریم اما بهتر از مدیریت حافظه پویا است چون در مدیریت حافظه ی پویا یک فرآیند را یکجا در ***RAM*** قرار می دادیم و مدیریت مان به ازای فرآیندها بود اما در اینجا ***segment*** ها را در ***RAM*** قرار می دهیم که طبیعتا" شرایط بهتر خواهد شد . اما این روش ، روش خوبی نیست و قدیمی است . روش بهتری که امروزه در پردازنده های اینتل کار می کنند مدیریت حافظه قطعه بندی صفحه بندی شده است .

۱۰- مدیریت حافظه قطعه بندی صفحه بندی شده :

این روش ترکیبی از دو روش قبل می باشد که هم باید ***PAGING*** داشته باشد و هم ***SEGMENTATION*** که پردازنده های نسل جدید این کار را می کنند .

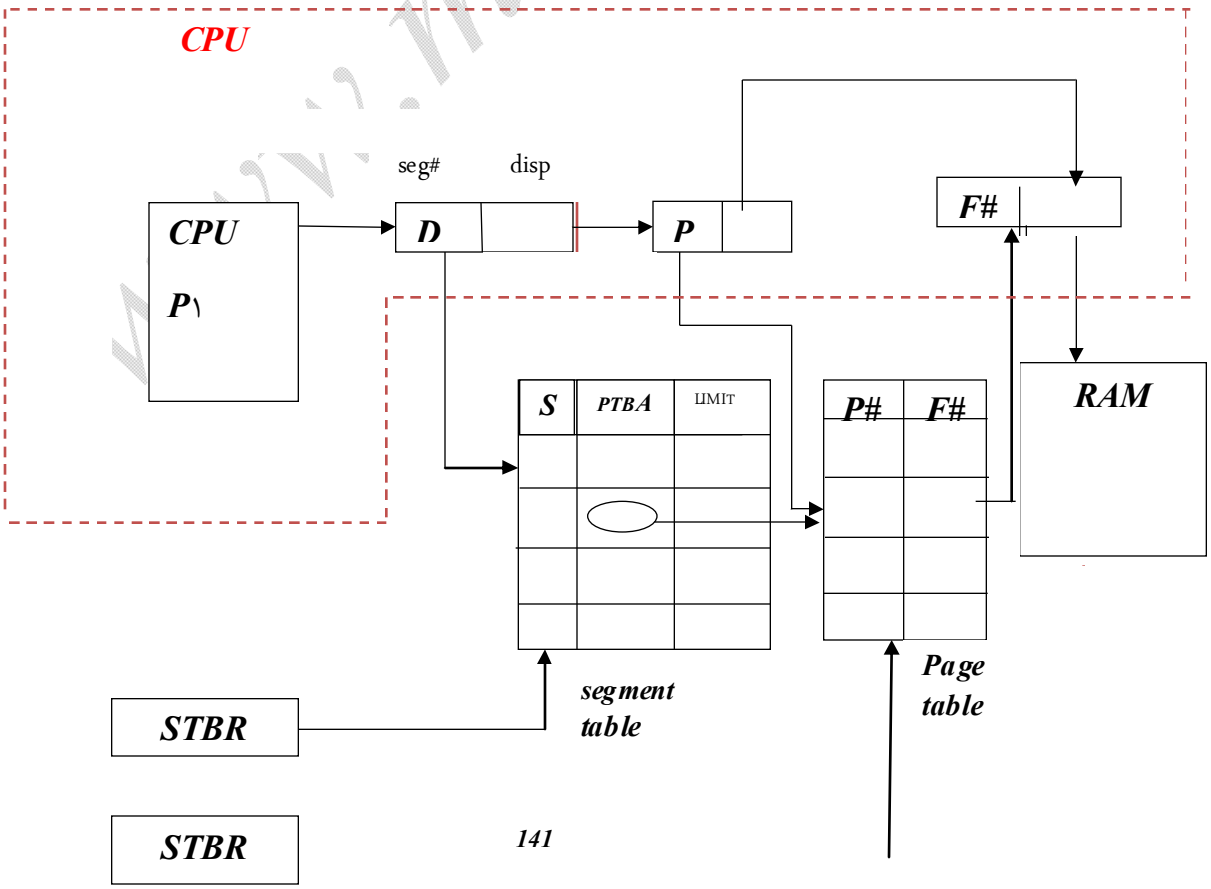
www.mbzir.com

سوال : چرا اینتل و مایکروسافت اینقدر همه گیر شده اند ؟ یکی از دلایلش این است که اصل را بر احترام بر برنامه نویسی گرفته اند . در این روش کار برنامه نویس راحت تر است و در این روش علاوه بر اینکه به ساختار ذهنی برنامه نویس توجه می شود هر قطعه به صفحات تقسیم می شود .



سوال : در سیستم ***page segmentation*** یک فرآیند ما که چهار ***segment*** دارد چند جدول صفحه نیاز داریم ؟

پاسخ : به تعداد هر ***seg*** ما جدول صفحه نیاز داریم .



حداقل ایرادی که این شکل دارد اولا" پیچیدگی آن است و همپنین سه بار دسترسی به حافظه شده است

نکته : ما می توانیم در اینتل ، هم ***Paging*** و هم ***segmentation*** داشته باشیم . اما چگونه؟ کافی است من برنامه نویس تمام برنامه هایم را در یک ***segment*** بنویسم .

در این روش دو نوع تبدیل داریم : ۱- تبدیل قطعه ۲- تبدیل صفحه

سوال : گفتیم مدیریت حافظه وظیفه سیستم عامل است پس پردازنده در اینجا چه کار میکند؟پردازنده در مدیریت حافظه نقش دارد . سخت افزار است که دارد پشتیبانی می کند . و سخت افزار است که دارد مدل مدیریت حافظه را پشتیبانی میکند پس در ابتدا سخت افزار است سپس سیستم عامل بر اساس آن نوشته می شود . اگر پردازنده ***paging*** را پشتیبانی نکند سیستم عامل کارآیی ندارد .

نکته ۱: فقط سیستم عامل تاثیر ندارد، سخت افزار است که باید مبتنی بر سیستم عامل برایش برنامه بنویسیم .

نکته ۲: اندازه صفحه را چه کسی تعیین می کند ؟ سخت افزار باید مدیریت حافظه را مدیریت کند . پس سخت افزار است که باید بگوید ***page size*** چقدر باشد .

سوال : ***TLB*** از کجا می آید و چگونه حمایت می شود ؟ و چرا وجود دارد ؟ از مشخصات سخت افزار است و سیستم عامل اصلا" از آن خبر ندارد و به خاطر بالا بردن راندمان کار طراح سخت افزار از ***TLB*** استفاده میکند .

۱۱- حافظه مجازی :

در این حالت حافظه قاب بندی می شود و در هر قاب سعی کردیم فرآیندی که می آید را در آن قاب جا می دهیم .

سوال : چند فرآیند را می توانیم در حافظه جای دهیم ؟ بستگی به حجم حافظه و حجم فرآیند دارد . پس جواب نامشخص است .

نکته : حافظه محدود است اما فرآیندها نامحدود اند . فرض کنید یک حافظه ***۴GB*** داریم و یک ***frame size*** داریم ***۴KB*** با این مشخصات چند قاب در حافظه داریم ؟

(۴GB)/(۴KB) = ۱۰^۶

یعنی ما ۱ میلیون تا قاب داریم .

وقتی رشد حافظه را داریم پیچیدگی نرم افزار را هم داریم پس باید به دنبال جاهای مختلفی در حوزه ی ذخیره سازی اطلاعات باشیم .

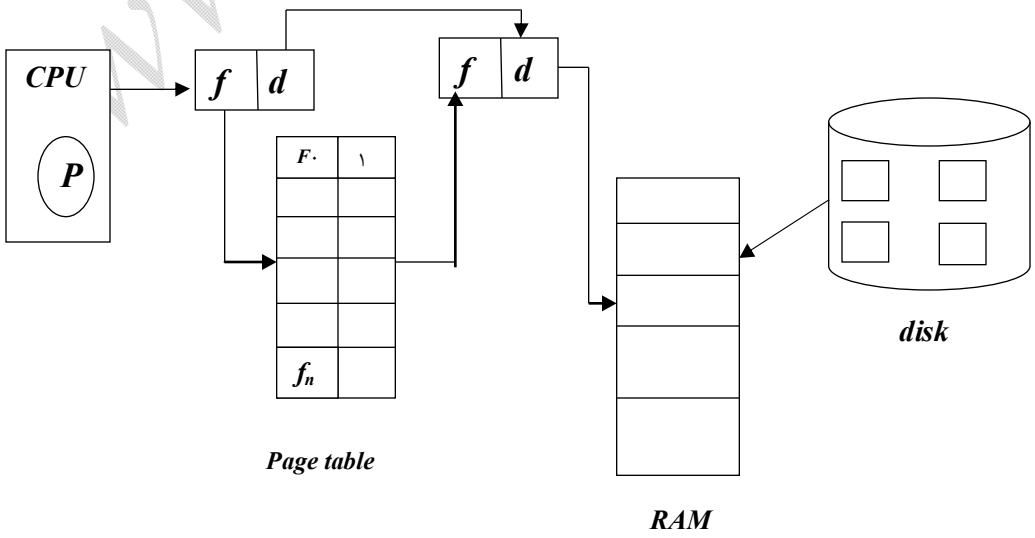
حافظه مجازی به دنبال این است که رشد ظرفیت حافظه به صورت مجازی باشد .

نکته : در اینجا به دنبال این هستیم که به صورت مجازی سایز حافظه را برای فرآیند زیاد کنیم . یعنی فرآیند احساس فضای زیادی از حافظه کند .

نکته : سیستم های امروزی همگی حافظه مجازی را پشتیبانی می کنند .

حال به توضیح حافظه مجازی می پردازیم :

دز سیستم ***paging*** حافظه را به قطعات مساوی به نام ***frame*** تقسیم میکنیم . دقیقا روی دیسک هم قطعاتی را درست کرده ایم که سعی کردیم آن قطعا سایزشان با اینها (***frame***ها) برابر با شد .



حافظه ی مجازی می گوید لزومی ندارد تمام صفحات در حافظه اصلی موجود باشد و این صفحات به تدریج کامل می شوند . اما از کجا بدانیم کدام صفحه قابش در ***page table*** آمده و کدام نیامده . یک بیت ***P/V*** در ***PAGE TABLE*** داریم این خانه ها یک محتویاتی را می گیرند اما از کجا متوجه شویم این اطلاعات ***valid***هستند یا نه ؟ باید برای هر ***frame***ای که در حافظه ی اصلی قرار می دهیم بیت ***P/V*** را ۱ قرار دهیم و باقی صفر است . در مرحله ی بعدی ۱ ***frame*** را می آوریم و ***P/V*** را ۱ میکنیم . یعنی این جدول به تدریج ظاهر می شود . ما یک اصلی به نام ***Locality***(محلی بودن) داریم طبیعت ما این است که وقتی خط ۱۰۱،۱۰۲،۱۰۳،۱۰۴ را اجرا می کنیم سپس بیاییم به خط **... , ۱۰۶ , ۱۰۵** و آنها را اجرا کنیم . ما هر باز یک سری صفحاتی را می آوریم که مربوط به صفحات قبلی است اما برخی مواقع می شود که صفحاتی را که می آوریم واقعا" محتویاتش در حافظه اصلی نیست . پس ما تا زمانی که در حافظه جا باشد مشکلی نداریم واحد های اطلاعاتی روی دیسک یا ***page***هستند یا ***fram***و لزومی ندارد تمام صفحات اصلی در حافظه باشند و ممکن است فکر کنیم که این باعث بروز مشکل می شود . اما اینطور نیست هر زمان نیاز داشتیم می آییم به حافظه اصلی زیرا قرار بود فرآیندها را گول بزنیم و فکر کنند هم به آنها حافظه داده ایم و هم واقعا" حافظه نداده‌ایم.

حل مساله این است که در صورتی که صفحه مورد نظر در حافظه نباشد چه اتفاقی می افتد ؟ ***P/V*** آن خالی (صفر) خواهد بود .

مراحلی که باید انجام بگیرد :

۱. مراجعه به جدول صفحه

۲. بروز خطای ***P.f***

۳. بکار افتادن روتین ***P.f***

۴. مشخص کردن صفحه ی مورد نظر

۵. یافتن صفحه در دیسک

۶. جایگزین کردن در حافظه

۷. اصلاح جدول صفحه

۸.بازگشت کنترل به ***Px***

نکته : وقتی خطای ***page fault*** صادر می شود یک وقفه سیستم عاملی صادر می گردد و بایستی روتین ***page fault*** را اجرا کند . و در اینصورت سیستم عامل وارد عمل می شود .

سوالات فصل پنجم

۱-تفاوت آدرس منطقی و آدرس فیزیکی چیست و کدام بهتر است ؟

آدرس منطقی مربوط به فرایند و آدرس فیزیکی مربوط به حافظه است.

آدرس منطقی :محدود از صفر تا حداکثر حافظه استفاده شده فرایند

آدرس فیزیکی :محدوده از صفر تا آخرین خانه ی حافظه ی فیزیکی

آدرس منطقی مهمتر است چون این امکان را میدهد چند فرایند را به صورت همزمان در حافظه قرار دهیم ، وفرایند ها را در بخش های مختلف حافظه جا دهیم.

۲-شرح وظیفه سیستم عامل در تعریف متن در مدیریت حافظه ***paging***چیست؟

سوییچینگ متن با تعویض متن
- ذخیره اطلاعات فرایند قبلی
- بار کردن اطلاعات فرایند جدید

۳- آدرس فیزیکی مربوط به آدرس منطقی ۳۴۰۰ چیست؟

www.mbzir.com

	base	limit
۰	۲۱۹	۶۰۰
۱	۲۳۰۰	۱۴
۲	۹۰۰	۱۰۰
۳	۱۳۲۷	۵۸۰
۴	۱۹۵۲	۹۶

Seg#	d
۳	۴۰۰

d <limit ⟸ ۴۰۰<۵۸۰

Base+۴۰۰ =۱۳۲۷+۴۰۰=۱۷۲۷

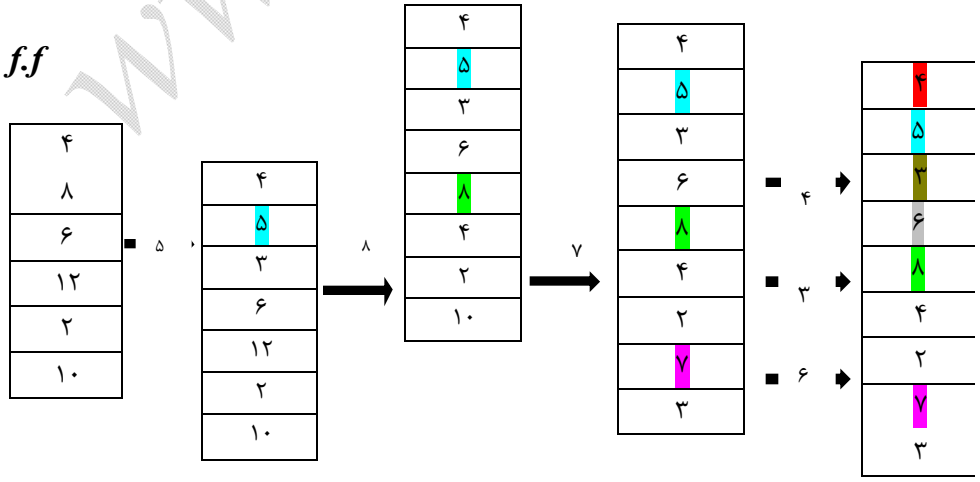
۴-فضای آدرس منطقی ۸ صفحه ***KB*** ۱ است این فضا به حافظه فیزیکی حاوری ۳۲ قاب صفحه نگاشت می شود .تعداد بیت آدرس فیزیکی؟

آدرس منطقی=***P+d*** $d = ۱kb = ۱*۲^{۱۰}$ => ***d***=۱۰

آدرس فیزیکی=***f+d*** $۳۲ = ۲^{۵}$ => ***f***=۵

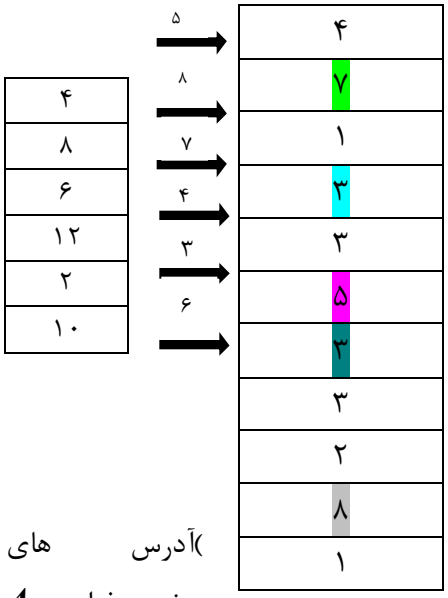
آدرس فیزیکی = ۵+۱۰ =۱۵

۵- فضاهای خالی به ترتیب ۴، ۸، ۶، ۱۲،۲،۱۰ است و در خواست ها به ترتیب ۵،۸،۷،۴،۳،۶ است .بعد از تخصیص به روش ***f.f , w.f*** فضاهای حافظه به شکل است؟



حافظه: ۰,۰,۰,۴,۲,۳

w.f



حافظه: ۴,۱,۳,۳,۲,۲

۶- در یک سیستم صفحه بندی (مدیریت حافظه)آدرس های حافظه ۱۶ بیتی سایز صفحات ***۴kb*** است . جدول صفحه فرایند ***A*** در مقایل آمده است . آدرس فیزیکی مربوط به آدرس منطقی ۱۶۰۰۰ در فرایند ***A*** کدام است ؟

0	*0*
1	*4*
2	*5*
3	*2*

$4KB = 2^2 * 2^{10} \Rightarrow d = 12$

$(16000)_{10} = 2^{13}+2^{12}+2^{11}+2^{10}+2^9+2^8$

4	*12*(d)

16= بیت آدرس

0010	*111100000000*

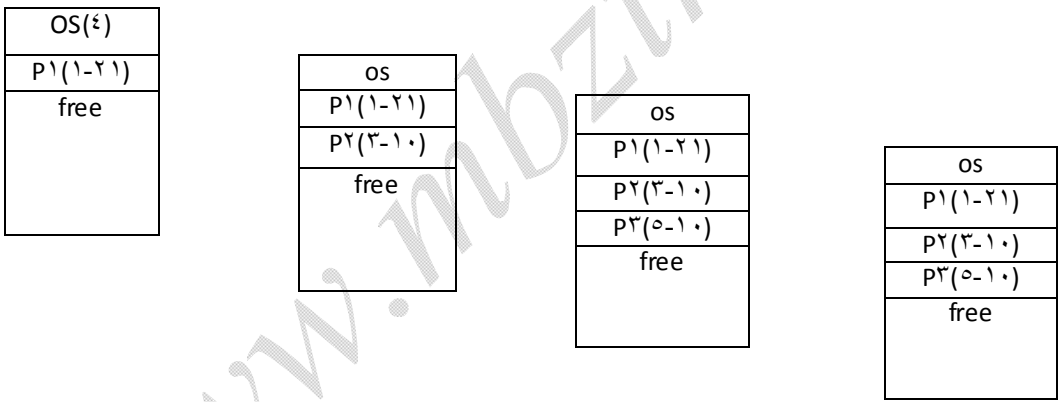
آدرس فیزیکی= ***0101111100000000***

۷- اطلاعات در خواست های تخصیص و آزاد سازی حافظه توسط فرایند را نشان می دهد . باید ***P***۱ در زمان ۱ در خواست تخصیصحافظه ***MB***۲ داشته باشد در صورت تخصیص ۲۰ واحد در زمان ۲۱ واحد حافظه تخصیص داده شده را آزاد میکند ،اگر ***OS*** فضای لازم برای تخصیص به درخواست یک فرایند را نداشته باشد آن فرایند را از بین می برد .

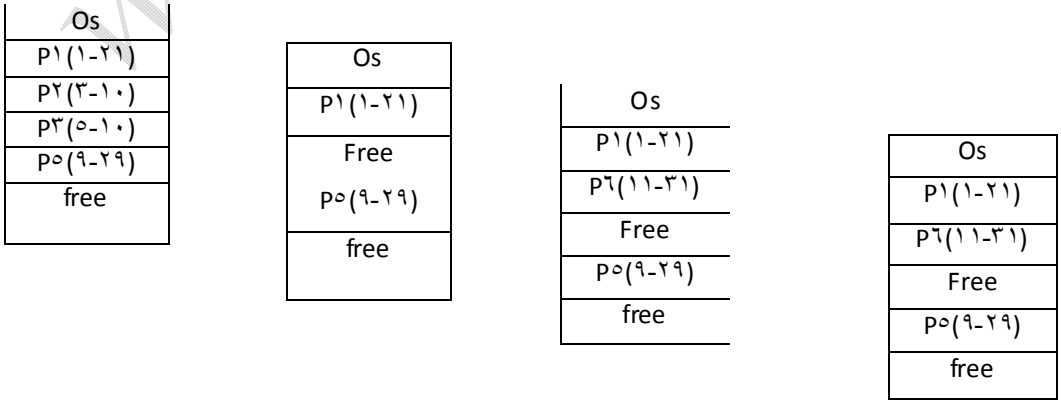
www.mbzir.com

فرض کنیدحافظه ی سیستم ***۳۶MB*** و سیستم عامل ***۴MB*** را برای خود رزرو میکند . ***FIRST FIT*** در زمان ۲۵ چند فرایند به خاطر نبودن حافظه از بین می روند ؟ ***kill p۴,p۷*** شدن

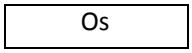
	مقدار حافظه در خواست	زمان در خواست	مدت تخصیص	
P۱	۲	۱	۲۰	۲۱
P۲	۶	۳	۷	۱۰
P۳	۱۴	۵	۵	۱۰
P۴	۱۲	۷	۲	۹
P۵	۵	۹	۲۰	۲۹
P۶	۲	۱۱	۲۰	۳۱
P۷	۱۸	۱۳	۲۰	۳۳
P۸	۱۳	۱۷	۲۰	۳۷
P۹	۳	۱۷	۲۰	۳۷



T=۱ ***T***=۳ ***T***=۵ ***T***=۷ ***Kill p۴***



T=۹ ***T***=۱۰ ***T***=۱۱ ***T***=۱۱ ***Kill p۷***



www.mbzir.com

P۱(۱-۲۱)
P٦(۱۱-۳۱)
P۸(۱۷-۳۷)
P۹(۱۷-۳۷)
Free
P۵(۹-۲۹)
free

T=۲۵

Os
free
P٦(۱۱-۳۱)
P۸(۱۷-۳۷)
P۹(۱۷-۳۷)
Free
P۵(۹-۲۹)
free

T=۱۷

ب-در زمان ۲۵ چه فرایند هایی در حافظه موجود هستند ؟

P۵,p۹,p۸,p۶

Os
free
P٦(۱۱-۳۱)
P۸(۱۷-۳۷)
P۹(۱۷-۳۷)
Free
P۵(۹-۲۹)
free

ج-اگر از روش تخصیص حافظه به صورت سیستم رفاقتی با کوچکترین واحد تخصیصی به اندازه ی ***mb***۹ استفاده شود در زمان ۲۵ چه فرایند هایی به خاطر نبود حافظه از بین می روند؟

اگر جا نشد ***kill*** میشود . ***kill***شدن در این مدل خیلی بیشتر است.

T=۱

P۱	۴***MB***	۸***MB***	۱۶***MB***

T=۳

P۱	۴***MB***	***P***۲	۱۶***MB***

www.mbzir.com

***T*=۵**

***P*۱**	***۴MB***	***P*۲**	***P*۳**	

•

•

•

www.mbzir.com