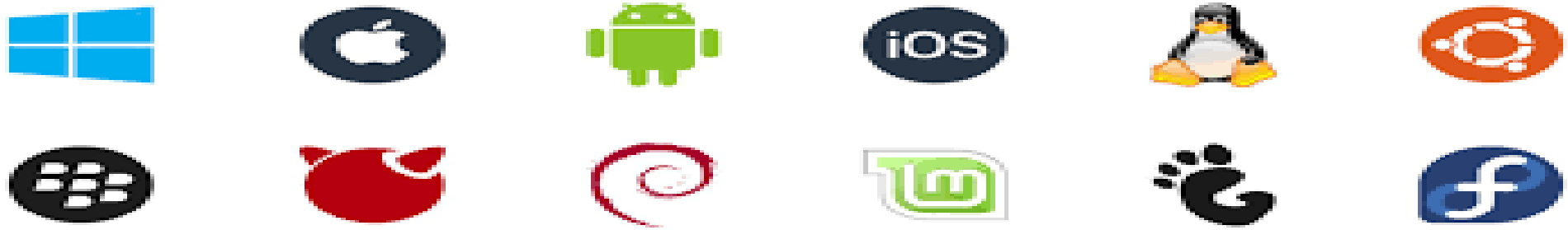



# سیستم های عامل

## Operating Systems

میلاد سلطانی

# فصل ششم

- روش صفحه بندی نیازی
- جایگزینی صفحه
- الگوریتم های جایگزینی صفحه
  - ✓ الگوریتم FIFO - الگوریتم بهینه (Optimal)
  - ✓ الگوریتم LRU - الگوریتم NRU
  - ✓ الگوریتم NRU - الگوریتم سالمندی (Aging)
  - ✓ الگوریتم صفحه ساعت – الگوریتم LFU
  - ✓ الگوریتم MFU
- تخصیص قاب ها
  - ✓ تخصیص مساوی و متناسب
  - ✓ تخصیص سراسری و محلی
- کوبیدگی (Thrashing)
- اندازه صفحه

## الگوریتم های جایگزینی صفحه

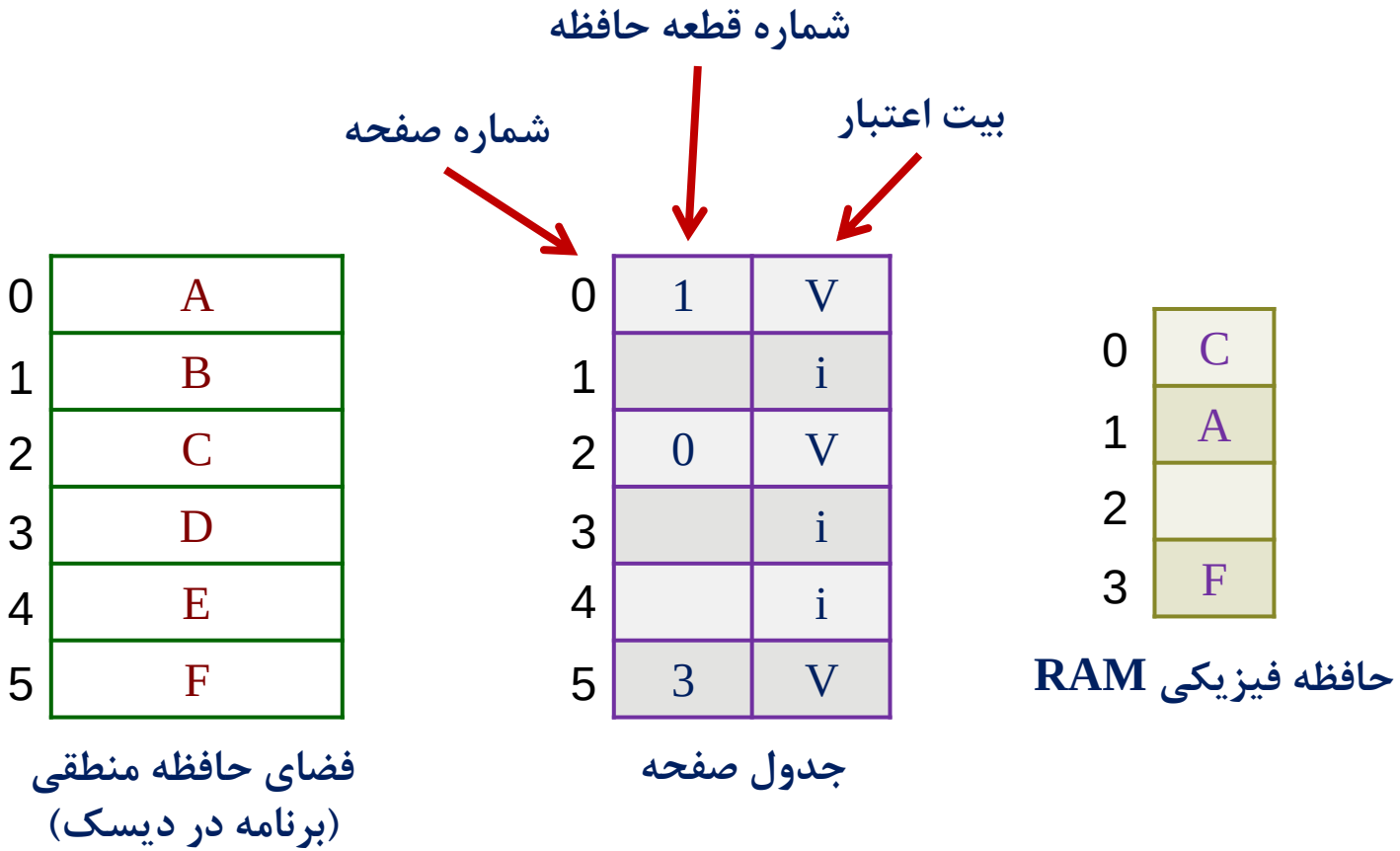
# روش صفحه بندی نیازی (Demand Paging)

- این روش ترکیبی از تکنیک صفحه بندی و روش مبادله (Overlay) می باشد.

- در روش مبادله کل فرآیند بین حافظه و دیسک مبادله می شود، ولی در روش صفحه بندی نیازی فقط صفحاتی از برنامه که به آنها نیاز است به حافظه آورده می شوند.

- در این روش در جدول صفحه باید یک بیت اعتبار valid-invalid داشته باشیم.
  - ✓ اگر معتبر بود یعنی صفحه مورد نظر هم در حافظه موجود است و هم قانونی می باشد.
  - ✓ اگر معتبر نبود یعنی یا صفحه غیر قانونی است یا صفحه قانونی است ولی هنوز روی دیسک است و به حافظه نیامده است.

# مثال :


شماره قطعه حافظه
شماره صفحه
بیت اعتبار
0 A
1 B
2 C
3 D
4 E
5 F
فضای حافظه منطقی
(برنامه در دیسک)
0 1 V
1 i
2 0 V
3 i
4 i
5 3 V
جدول صفحه
0 C
1 A
2
3 F
حافظه فیزیکی RAM

# روش صفحه بندی نیازی (Demand Paging)

- وقتی فرآیند درخواست صفحه ای را دارد که بیت اعتبار آن نا معتبر (invalid) است، یک تله خطای صفحه (Page Fault Trap) رخ می دهد که اصطلاحاً نقص صفحه نامیده می شود.

- هنگام وقوع نقص صفحه سیستم عامل عملیات زیر را انجام می دهد:

۱) بررسی می شود که آدرس منطقی درستی درخواست شده است یا خیر ؟

۲) اگر آدرس نادرست باشد، فرآیند خاتمه می یابد، اگر آدرس درست باشد اقدام به آوردن صفحه به حافظه می شود.

۳) یک قاب آزاد در حافظه پیدا می شود. سپس صفحه مورد نظر از دیسک به حافظه می آید.

۴) جدول صفحه بروز رسانی شده و اجرای فرآیند مورد نظر که متوقف شده بود، ادامه می یابد.

# روش صفحه بندی نیازی (Demand Paging)

- غالباً از ابتدای اجرای یک برنامه، تعدادی از صفحات مهم آن به حافظه آورده می شود.

- می توان از ابتدای اجرای یک برنامه هیچ صفحه ای را وارد حافظه نکرد و برای اجرای آن، هر صفحه مورد نیاز را منتقل کرد. به این روش صفحه بندی نیازی محض گفته می شود.

- در روش صفحه بندی نیازی دیسک به دو منظور استفاده می شود:
  - ✓ ذخیره دایمی برنامه ها
  - ✓ حافظه کمکی برای اجرای برنامه ها
    - وقتی فضای حافظه منطقی بزرگتر از فضای حافظه فیزیکی است، بخشی از دیسک به عنوان حافظه مجازی (پشتیبان) جزئی از حافظه RAM محسوب خواهد شد. در سیستم عامل ویندوز Windows از این تکنیک استفاده می شود.

# جایگزینی صفحه (Page Replacement)

- ممکن است هنگام انتقال یک صفحه از دیسک به حافظه، در حافظه هیچ قاب خالی موجود نباشد. در این حالت باید یکی از صفحه های حافظه را به دیسک منتقل کرده تا قاب خالی بوجود آمده و استفاده شود. به این کار عملیات جایگزینی صفحه گفته می شود.

- در عملیات جایگزینی صفحه، دو انتقال صفحه انجام می شود:
  - ✓ خروج صفحه غیر ضروری از حافظه به دیسک
  - ✓ انتقال صفحه درخواستی از دیسک به حافظه

- این سیستم زمان عملیات رسیدگی به نقص صفحه را دو برابر کرده و کارایی سیستم را کاهش خواهد داد.

# جایگزینی صفحه (Page Replacement)

- برای جلوگیری از کاهش کارایی سیستم جایگزینی صفحه، اکثر سیستم ها در جدول صفحه یک بیت تغییر M (Modify bit) یا بیت کثیف D (Dirty bit) برای هر صفحه در نظر می گیرند.

- وقتی محتوای یک صفحه که در حافظه است تغییر کند، بیت M=1 شده و در غیر اینصورت M=0 باقی می ماند.

- هنگام جایگزینی صفحه، سیستم عامل ابتدا به بیت تغییر نگاه کرده، اگر M=0 باشد نیاز برگشت به دیسک ندارد، چون با صفحه اصلی هیچ فرقی ندارد. لذا صفحه جدید درخواست شده به حافظه آمده و روی این صفحه ذخیره می شود. اینگونه زمان عملیات جایگزینی صفحه نصف می شود.

# جایگزینی صفحه (Page Replacement)

- برای پیاده سازی تکنیک جایگزینی صفحه دو مسئله باید حل شود:

۱) **مسئله روش جایگزینی صفحه**

- وقتی حافظه پر است کدام صفحه را انتخاب کرده و از حافظه خارج کنیم تا حافظه آزاد شود !

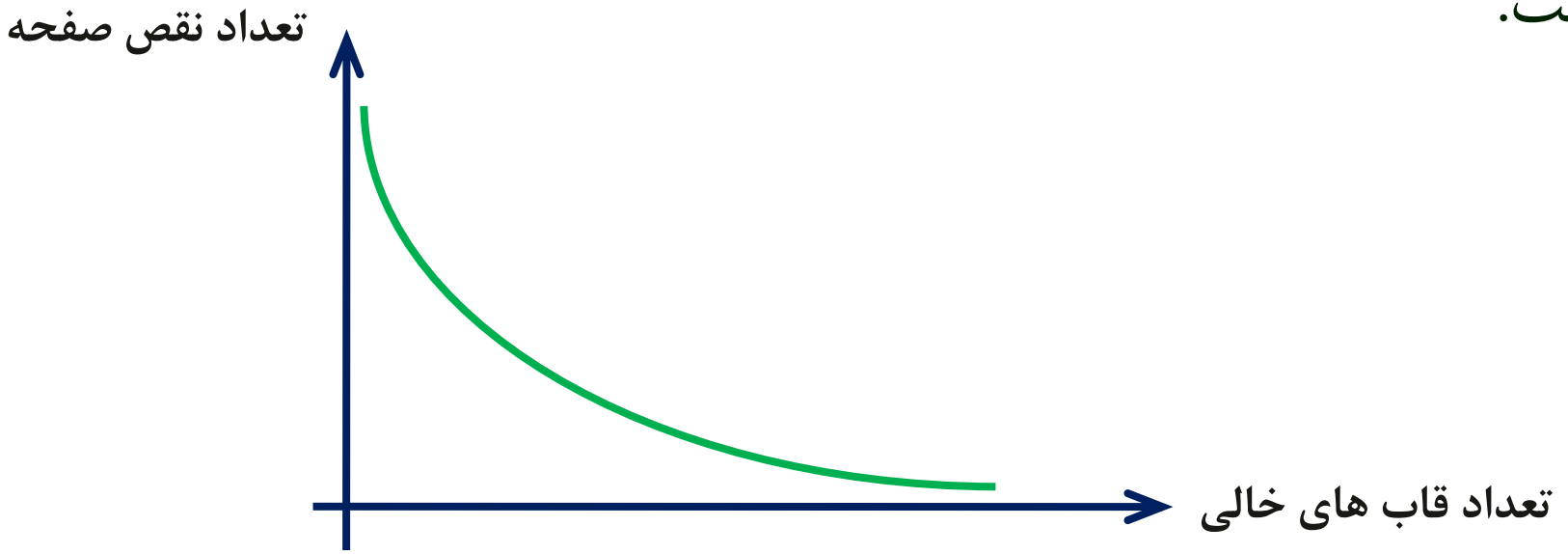
۲) **مسئله الگوریتم تخصیص قاب های حافظه**

- اگر فرآیند های متعددی در حافظه وجود دارد، به هر فرآیند چند قاب از حافظه اختصاص داده شود.

# الگوریتم های جایگزینی صفحه

- در جایگزینی صفحه هر چقدر تعداد فریم های آزاد حافظه بیشتر باشند، تعداد خطاهای نقص صفحه کاهش خواهد یافت.

- یک الگوریتم جایگزینی صفحه، هر چقدر خطای نقص صفحه کمتری را تولید کند، کارآیی بهتری خواهد داشت.



# الگوریتم FIFO

- ساده ترین روش جایگزینی صفحه است.
- این الگوریتم صفحه ای را برای خروج از حافظه انتخاب می کند که از بقیه زودتر به حافظه وارد شده است.
- برای تشخیص زمان ورود صفحه ها به حافظه، سیستم عامل از یک صف استفاده می کند که صفحه جلوی این صف کاندیدای خروج در هنگام جایگزینی صفحه می باشد.
- کارآیی این روش همواره خوب نیست.

مثال : فرض کنید یک برنامه در حال اجرا، صفحات زیر را (از چپ به راست) درخواست کند. اگر حافظه دارای 3 قاب آزاد باشد، در روش جایگزینی صفحه FIFO چند نقص صفحه رخ خواهد داد ؟

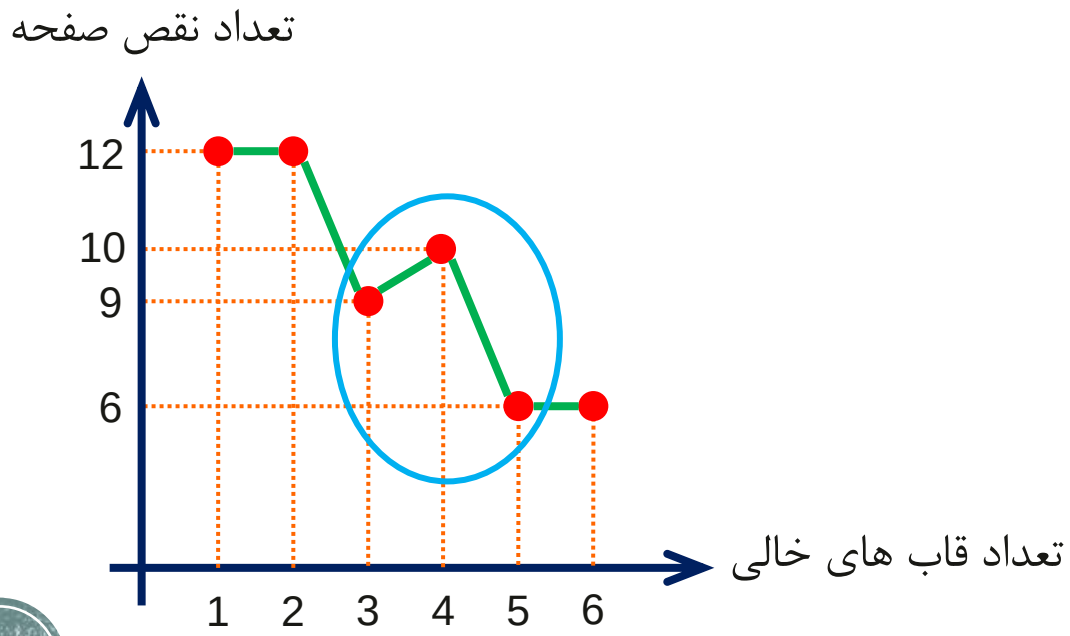
>> 7-0-1-2-0-3-0-4-2-3-0-3-2-1-2-0-1-7-0-1

| صفحه | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| قاب 1 | 7 | 0 | 1 | 2 | 2 | 3 | 0 | 4 | 2 | 3 | 0 | 0 | 0 | 1 | 2 | 2 | 2 | 7 | 0 | 1 |
| قاب 2 | | 7 | 0 | 1 | 1 | 2 | 3 | 0 | 4 | 2 | 3 | 3 | 3 | 0 | 1 | 1 | 1 | 2 | 7 | 0 |
| قاب 3 | | | 7 | 0 | 0 | 1 | 2 | 3 | 0 | 4 | 2 | 2 | 2 | 3 | 0 | 0 | 0 | 1 | 2 | 7 |
| نقص صفحه | * | * | * | * | | * | * | * | * | * | * | | | * | * | | | * | * | * |

**تعداد خطاهای نقص صفحه = ۱۵ عدد**

# الگوریتم FIFO

- الگوریتم FIFO دارای یک مشکل اساسی است که این مشکل با عنوان ناهنجاری بِلیدی (Belady's Anomaly) شناخته می شود. یعنی زمانیکه تعداد قاب های خالی حافظه افزایش می یابد، بر خلاف انتظار، تعداد خطاهای نقص صفحه افزایش می یابد.




**تمرین:**

برنامه ای برای اجرا، صفحات زیر را (از چپ به راست) درخواست کند، تعداد نقص های صفحه را با الگوریتم FIFO برای تعداد ۱ ، ۲ ، ۳ ، ۴ ، ۵ و ۶ قاب حافظه بررسی کنید.

>> 1,2,3,4,1,2,5,1,2,3,4,5

# الگوریتم بهینه (Optimal)

- این الگوریتم صفحه ای را برای خروج ار حافظه انتخاب می کند که برای بیشترین زمان آینده مورد استفاده قرار نخواهد گرفت.

- اثبات شده است که این الگوریتم نسبت به تمام الگوریتم های جایگزینی صفحه، دارای خطای نقص صفحه کمتری است.

- همانند روش FIFO دارای مشکل ناهنجاری بِلیدی نیست.

- در این روش سعی می شود همیشه "مجموعه کاری" یک برنامه در حافظه باشد.

✓ **تعریف مجموعه کاری:** تعدادی از صفحات یک برنامه است که اگر در حافظه قرار داشته باشند، برنامه با سرعت خوب و با کارآیی بالا اجرا می شود.

**مثال:** فرض کنید یک برنامه در حال اجرا، صفحات زیر را (از چپ به راست) درخواست کند. اگر حافظه دارای 3 قاب آزاد باشد، در روش جایگزینی صفحه بهینه چند نقص صفحه رخ خواهد داد ؟

>> 7-0-1-2-0-3-0-4-2-3-0-3-2-1-2-0-1-7-0-1

| صفحه | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| قاب 1 | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
| قاب 2 | | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| قاب 3 | | | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| نقص صفحه | * | * | * | * | | * | | * | | | * | | | * | | | | * | | |

**تعداد خطاهای نقص صفحه = ۹ عدد**

# الگوریتم بهینه (Optimal)

- در عمل این الگوریتم قابل پیاده سازی نمی باشد. زیرا برای سیستم عامل آینده استفاده یک برنامه از صفحاتش، از قبل مشخص نیست.

- اصولاً این الگوریتم برای مقایسه و بررسی الگوریتم های دیگر بکار می رود.

- نام دیگر این روش، الگوریتم بِلیدی یا بِلیدی اُپتیمال (BO) می باشد.

# الگوریتم LRU

- نام این روش ” اخیراً کمترین استفاده شده “ (Least Recently Used) می باشد.
- این روش در واقع تقریبی از روی بهینه است و در آن از گذشته اخیر، برای آینده نزدیک استفاده می شود.
- این الگوریتم صفحه ای را برای خروج ار حافظه انتخاب می کند که برای بیشترین زمان قبلی مورد استفاده قرار نگرفته است.
- این الگوریتم مشکل ناهنجاری بِلیدی را ندارد.

**مثال:** فرض کنید یک برنامه در حال اجرا، صفحات زیر را (از چپ به راست) درخواست کند. اگر حافظه دارای 3 قاب آزاد باشد، در روش جایگزینی صفحه LRU چند نقص صفحه رخ خواهد داد ؟

>> 7-0-1-2-0-3-0-4-2-3-0-3-2-1-2-0-1-7-0-1

| صفحه | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| قاب 1 | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| قاب 2 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
| قاب 3 | | | 1 | 1 | 1 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
| نقص صفحه | * | * | * | * | | * | | * | * | * | * | | | * | | * | | * | | |

**تعداد خطاهای نقص صفحه = ۱۲ عدد**

# الگوریتم LRU

- یک روش پیاده سازی سخت افزاری الگوریتم LRU به شرح زیر است :

✓ **با استفاده از شمارنده:** به هر سطر جدول صفحه یک شمارنده و به CPU نیز یک شمارنده داده شده، به ازای هر مراجعه به حافظه، شمارنده CPU یک واحد افزایش می یابد. وقتی صفحه درخواست می شود، عدد شمارنده CPU در مکان متناسب جدول صفحه کپی می شود. وقتی الگوریتم LRU می خواهد صفحه ای را از حافظه بیرون کند، صفحه ایی انتخاب می کند که شمارنده آن در جدول صفحه عدد کمتری باشد.

- بدترین حالت الگوریتم LRU و الگوریتم FIFO زمانی است که یک برنامه بصورت دوره ای یا چرخه ای به صفحاتش دسترسی داشته باشد و حافظه به اندازه کافی بزرگ نباشد.
  - مثلاً حافظه ۳ قاب داشته باشد و صفحات برنامه (از چپ به راست) بصورت زیر درخواست شوند :

0-2-5-6-0-2-5-6-0-2-5-6-….

# الگوریتم NRU

- نام این روش ” اخیراً استفاده نشده “ (Not Recently Used) می باشد.
- این روش، به الگوریتم دومین شانس (Second Chance) در کتاب های مرجع تَنِنباوم و سیلبِرشاتس مشهور است.
- الگوریتم NRU همانند روش FIFO دارای مشکل ناهنجاری بِلیدی است.
- این الگوریتم به دو روش متفاوت بیان شده است.
  ۱) روش مبتنی بر کتاب آقای دکتر مهرداد فهیمی
  ۲) روش مبتنی بر کتاب های تَنِنباوم و سیلبِرشاتس

# الگوریتم NRU مبتنی بر کتاب دکتر فهیمی

- این روش همان الگوریتم FIFO است که تغییر کوچکی در آن داده شده است.
- در این روش از بیت زیر استفاده می شود :

✓ **بیت مراجعه R (Reference):** هرگاه برنامه به یک صفحه دسترسی می یابد، این بیت 1 می شود.
در برخی از کتب به این بیت، بیت دسترسی A (Access) گفته می شود.

**نکته!** هرگاه این بیت 1 می شود، همانطور باقی می ماند تا سیستم عامل آن را 0 کند.

**نکته!** هر صفحه جدیدی که به حافظه آمده و درون صف FIFO قرار می گیرد، بیت مراجعه R=0 می شود.

# الگوریتم NRU مبتنی بر کتاب دکتر فهیمی

- این روش از ابتدای صف FIFO شروع به بررسی بیت R صفحات می کند.

✓ **اگر بیت R=0 باشد،** یعنی صفحه هم قدیمی است و هم از آن استفاده نشده است. لذا بلافاصله از حافظه خارج می شود.

✓ **اگر بیت R=1 باشد،** سیستم عامل آنرا 0 کرده و صفحه به انتهای صف می رود. انگار که جدیداً وارد صف شده است. بدین ترتیب شانس دومی برای ماندن در حافظه پیدا خواهد کرد.

**نکته!** صفحه ای که شانس دوم گرفته جایگزین نمی شود، مگر آنکه کلیه صفحات دیگر حافظه جایگزین شده باشند یا شانس دوم گرفته باشند.

**مثال:** فرض کنید یک برنامه در حال اجرا، صفحات زیر را (از چپ به راست) درخواست کند. اگر حافظه دارای 4 قاب آزاد باشد، در روش جایگزینی صفحه NRU چند نقص صفحه رخ خواهد داد؟ نتیجه را با روش FIFO مقایسه کنید.

>> 1-2-3-4-5-2-6-4-5-7

| صف | صفحه | 1 | 2 | 3 | 4 | 5 | 2 | × | 6 | 4 | 5 | × | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| جلوی صف | قاب 1 | 1 | 1 | 1 | 1 | 2 | 2R | 3 | 4 | 4R | 4R | 2 | 6 |
| | قاب 2 | | 2 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 5R | 6 | 4 |
| | قاب 3 | | | 3 | 3 | 4 | 4 | 5 | 2 | 2 | 2 | 4 | 5 |
| انتهای صف | قاب 4 | | | | 4 | 5 | 5 | 2 | 6 | 6 | 6 | 5 | 7 |
| | نقص صفحه | * | * | * | * | * | | | * | | | | * |

**تعداد خطاهای نقص صفحه در روش NRU = ۷ عدد**

## اجرای مثال قبل به روش FIFO

| صف | صفحه | 1 | 2 | 3 | 4 | 5 | 2 | 6 | 4 | 5 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| جلوی صف | قاب 1 | 1 | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 4 |
| | قاب 2 | | 2 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 5 |
| | قاب 3 | | | 3 | 3 | 4 | 4 | 5 | 5 | 5 | 6 |
| انتهای صف | قاب 4 | | | | 4 | 5 | 5 | 6 | 6 | 6 | 7 |
| | نقص صفحه | * | * | * | * | * | | * | | | * |

**تعداد خطاهای نقص صفحه در روش FIFO = ۷ عدد**

# الگوریتم NRU مبتنی بر کتاب سیلبِرشاتس

- در این روش از دو بیت زیر استفاده می شود :

✓ **بیت مراجعه R (Reference):** هرگاه برنامه به یک صفحه دسترسی می یابد، این بیت 1 می شود. در برخی از کتب به این بیت، بیت دسترسی A (Access) گفته می شود.

✓ **بیت تغییر M (Modified):** هرگاه محتویات یک صفحه توسط برنامه اش تغییر می کند، این بیت 1 می شود. در برخی از کتب به این بیت، بیت کثیف D (Dirty) گفته می شود.

**نکته !** هرگاه این بیت ها 1 می شود، همانطور باقی می ماند تا سیستم عامل آنها را 0 کند.

# الگوریتم NRU مبتنی بر کتاب سیلبِرشاتس

- در این روش وقتی یک فرآیند شروع می شود، سیستم عامل هر دو بیت M و R را 0 خواهد کرد.
- پس از اجرای برنامه، بیت R به صورت متناوب مثلاً هر ۲۰ میلی ثانیه، 0 می شود تا صفحه هایی که اخیراً مورد استفاده نبودند از بقیه جدا شوند.

✓ **اگر بیت M=0 و R=0 (کلاس ۰)**: صفحه اخیراً استفاده نشده و تغییر نکرده است.

✓ **اگر بیت M=1 و R=0 (کلاس ۱)**: صفحه اخیراً استفاده نشده ولی تغییر کرده است.

✓ **اگر بیت M=0 و R=1 (کلاس ۲)**: صفحه اخیراً استفاده شده است ولی تغییر نکرده است.

✓ **اگر بیت M=1 و R=1 (کلاس ۳)**: صفحه اخیراً استفاده شده است و تغییر کرده است.

# الگوریتم NRU مبتنی بر کتاب سیلبِرشاتس

- بر اساس کلاس های مختلفی که برای صفحات در نظر گرفته می شود نکات زیر در نظر گرفته می شود :

✓ صفحات کلاس 0 بهترین انتخاب برای جایگزینی هستند.

✓ صفحات کلاس 1 برای جایگزینی کاملاً مناسب نیستند، زیرا ابتدا تغییرات آنها باید روی دیسک ثبت شود و پس از آن جایگزین شوند.

✓ صفحات کلاس 3 احتمالاً به زودی مورد استفاده قرار خواهند گرفت.

✓ صفحات کلاس 4 را حتی الامکان نباید از حافظه خارج کرد.

# الگوریتم NRU مبتنی بر کتاب سیلبِرشاتس

- الگوریتم NRU بسیار ساده است و برای پیاده سازی کارآیی خوبی دارد.
- این الگوریتم در مدیریت حافظه مجازی سیستم عامل مکینتاش (Macintosh) شرکت اَپِل استفاده شده است.
- کارآیی الگوریتم NRU از الگوریتم FIFO بهتر است.

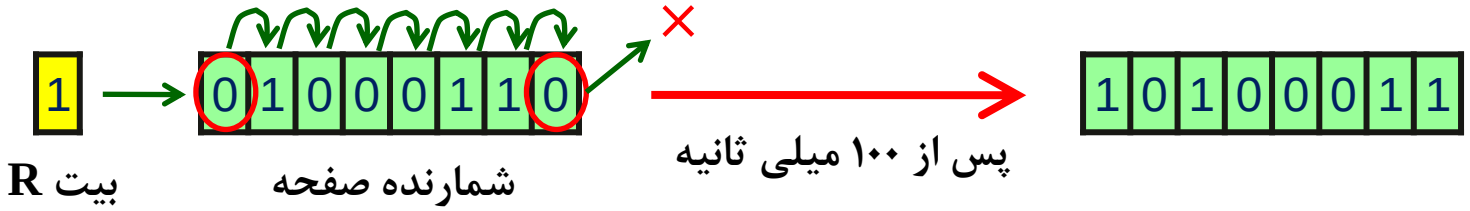
# الگوریتم سالمندی (Aging)

- این الگوریتم در واقع روش LRU را شبیه سازی می کند.

✓ در این روش یک شمارنده (اغلب ۸ بیتی) برای هر صفحه در جدول صفحه در نظر گرفته می شود. مقدار این شمارنده در ابتدا برابر 00000000 می باشد.

✓ در فاصله های مرتب مثلاً هر ۱۰۰ میلی ثانیه، شمارنده هر صفحه به راست شیفت داده شده و از سمت چپ، محتویات بیت R همان صفحه وارد شمارنده مربوطه می شود.

✓ در هنگام جایگزینی صفحه، سیستم عامل صفحه ای را جایگزین می کند که مقدار عدد شمارنده آن کمتر است.



# الگوریتم سالمندی (Aging)

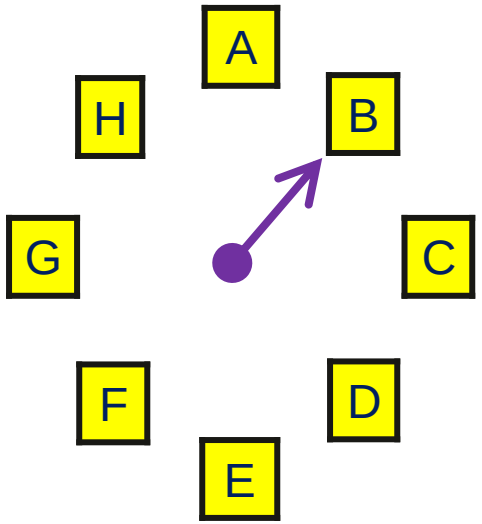
- اگر چند صفحه دارای کمترین عدد شمارنده برابر با هم باشند، می توان همه آنها از حافظه بیرون رانده یا یکی از آنها را اتفاقی یا به روش FIFO از حافظه بیرون کرد.

- اگر تعداد بیت های شمارنده 0 در نظر گرفته شود، آنگاه فقط بیت R مورد بررسی قرار خواهد گرفت که تبدیل به روش « دومین شانس» LRU می شود.

- این روش با نام الگوریتم بیت های مراجعه اضافی (Additional-Reference-Bits) در کتاب سیلبرشاتس عنوان شده است.

# الگوریتم صفحه ساعت (Clock Page)

- در الگوریتم دومین شانس NRU جابجایی صفحات زیاد مناسب نیست.
- در الگوریتم صفحه ساعت، صفحه ها مثل یک لیست حلقوی قرار می گیرند، طوریکه یک عقربه ساعت به قدیمی ترین صفحه اشاره می کند.




- ✓ اگر صفحه مورد نظر، بیت R=0 داشته باشد، از حافظه بیرون می شود و عقربه یک خانه به جلو حرکت می کند.
- ✓ اگر صفحه مورد نظر، بیت R=1 داشته باشد، بیت R=0 شده و عقربه به خانه بعدی می رود.

**نکته !** در واقع الگوریتم صفحه ساعت فقط پیاده سازی دیگر روش NRU می باشد.

# الگوریتم LFU

- نام این روش ” کمترین متناوباً استفاده شده“ (Least Frequently Used) می باشد.
  - ✓ در این روش یک شمارنده برای هر صفحه در جدول صفحه در نظر گرفته می شود.
  - ✓ هر چند لحظه بیت R مربوط به هر صفحه با مقدار این شمارنده جمع می شود.
  - ✓ هنگام خطای نقص صفحه، الگوریتم جایگزینی صفحه، صفحه ای را که شمارنده اش دارای مقدار **کمتری** است، جایگزین می کند.
- این الگوریتم همواره نتیجه مناسب را در بر ندارد.
- این روش در کتاب تَنِنباوم ”متناوباً استفاده نشده“ (Not Frequently Used) نام دارد.

# الگوریتم MFU

- نام این روش ” بیشترین متناوباً استفاده شده“ (Most Frequently Used) می باشد.
  - ✓ در این روش یک شمارنده برای هر صفحه در جدول صفحه در نظر گرفته می شود.
  - ✓ هر چند لحظه بیت R مربوط به هر صفحه با مقدار این شمارنده جمع می شود.
  - ✓ هنگام خطای نقص صفحه، الگوریتم جایگزینی صفحه، صفحه ای را که شمارنده اش دارای مقدار **بیشتری** است، جایگزین می کند.
- این روش دقیقاً معکوس روش LFU می باشد.

# بحث تخصیص قاب ها

- پس از بررسی الگوریتم های جایگزینی صفحه، نوبت به بحث تخصیص قاب می رسد.

- مشکلی که تخصیص قاب با آن مواجه است بصورت زیر می باشد :
  - ✓ " چگونه مقدار مشخصی از حافظه را بین پردازش های متقاضی تقسیم کنیم تا کارآیی سیستم بهتر شود ! "

- یا به عبارت دیگر
  - ✓ " چگونه قاب های حافظه را بین پردازش های متقاضی تقسیم کنیم تا کارآیی سیستم بهتر شود ! "

# تخصیص قاب مساوی و متناسب

- **تخصیص قاب مساوی** ساده ترین مدل اختصاص قاب ها است که m قاب را بین n فرآیند به صورت مساوی تقسیم می کند.

✓ **مثال:** اگر ۹۴ قاب آزاد حافظه و ۵ فرآیند داشته باشیم،

آنگاه به هر فرآیند ۱۸ قاب حافظه اختصاص خواهد یافت.

$$\begin{matrix} m=94 \\ n=5 \end{matrix} \rightarrow \frac{m}{n}=\frac{94}{5}\approx 18$$

- **تخصیص قاب متناسب**، قاب های موجود را به نسبت اندازه فرآیندها تقسیم می کند.

✓ **مثال:** اگر دو فرآیند یکی دارای ۱۰ صفحه و دومی با ۴۰ صفحه داشته باشیم و حافظه دارای ۲۰ قاب آزاد باشد، آنگاه به فرآیند اول ۴ قاب حافظه و به فرآیند دوم ۱۶ قاب اختصاص خواهد یافت.

$$\frac{10}{10+40}\times 20=4 \qquad \frac{40}{10+40}\times 20=16$$

تعداد صفحات هر فرآیند ← | → مجموع تعداد صفحات همه فرآیند

# تخصیص قاب سراسری و محلی

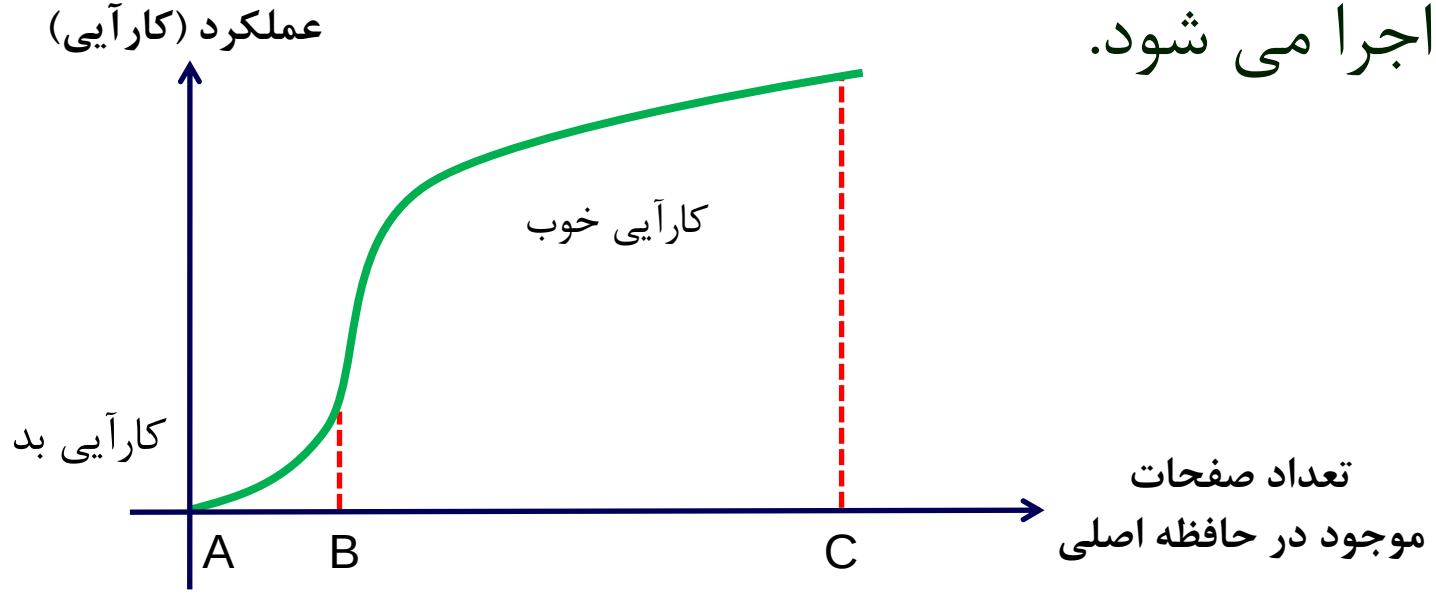
- **تخصیص قاب سراسری** از بین کل قاب های موجود یک قاب را برای جایگزینی انتخاب کرده، حتی اگر این قاب متعلق به فرآیند دیگری باشد.
  - ✓ **مشکل:** مجموعه قاب های یک فرآیند هم به رفتار صفحه بندی خود فرآیند و هم به رفتار صفحه بندی سایر فرآیندها وابسته است. یعنی مثلاً زمان اجرای فرآیند در یک اجرا ۲ ثانیه و در اجرای دیگر ۲۰ ثانیه باشد.

- **تخصیص قاب محلی** فقط از بین قاب های مخصوص به آن فرآیند برای جایگزینی صفحات آن فرآیند استفاده می کند.
  - ✓ **مشکل:** ممکن است اجرای یک فرآیند را به تأخیر اندازد، چرا که ممکن است قاب های فرآیندهای دیگر خالی باشند ولی این فرآیند قاب برای اجرا کم بیاورد.

# کوبیدگی (Thrashing)

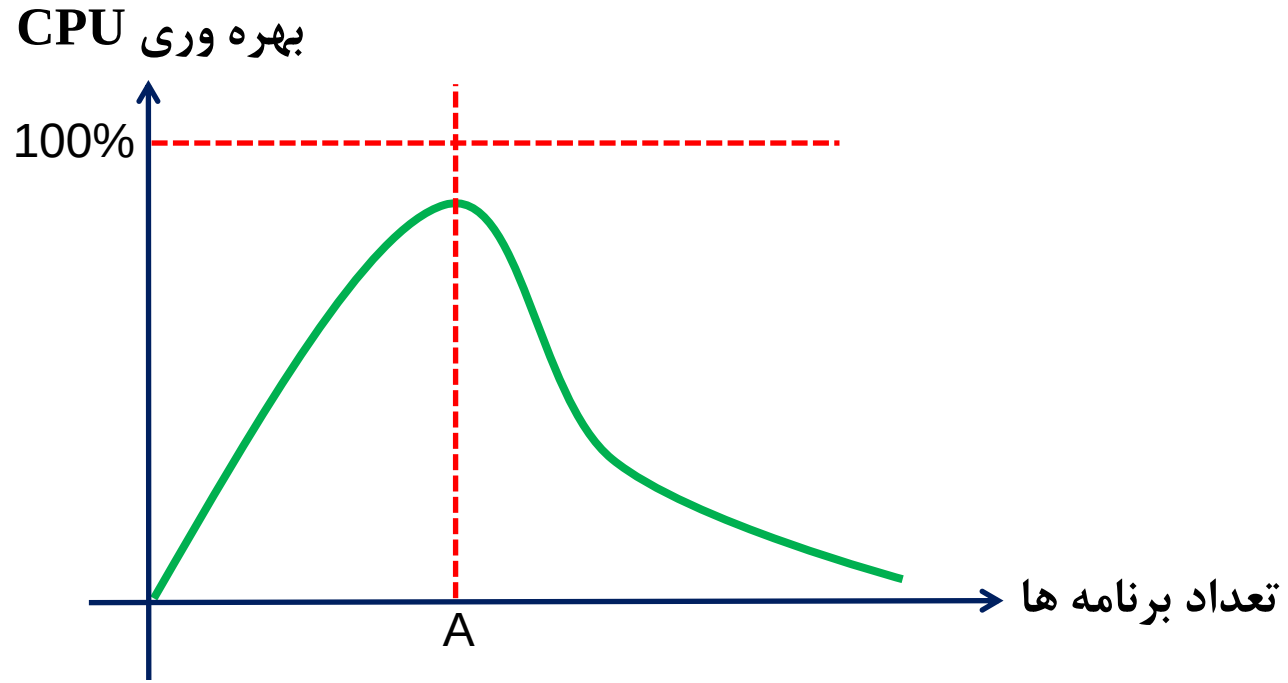
- مجموعه کاری (Work Set) هر برنامه، تعداد صفحاتی از آن برنامه است که اگر در حافظه باشد، کارآیی برنامه و سیستم خیلی بالا می رود.

- اگر مجموعه کاری یک فرآیند در حافظه باشد، فرآیند بدون آنکه تعداد خطای نقص صفحه زیادی داشته باشد، اجرا می شود.



# کوبیدگی (Thrashing)

- اگر تمام صفحات یک برنامه در حافظه نباشد، زمان هایی وجود خواهد داشت که صفحات برنامه بین دیسک و حافظه مبادله می شوند. در این زمان ها می توان برنامه های دیگر را اجرا کرد تا CPU همیشه مشغول باشد. نمودار بهره وری CPU بصورت زیر خواهد بود:



# کوبیدگی (Thrashing)

- در نمودار صفحه قبل در اثر افزایش سطح چند برنامگی، بهره وری CPU بالا می رود.
  - ✓ در نقطه A برنامه اول تقاضای صفحه کرده، یک صفحه از برنامه دوم از حافظه خارج شده و صفحه درخواستی برنامه اول به حافظه می آید.
  - ✓ در نوبت اجرای برنامه دوم درخواست صفحه شده، یک صفحه از برنامه اول از حافظه خارج شده و صفحه درخواستی برنامه دوم به حافظه می آید.
- این عمل بصورت متناوب تکرار شده و در چنین شرایطی، مجموعه کاری هیچ برنامه ای در حافظه نخواهد بود و کارآیی سیستم به شدت پایین می آید.
  - ✓ این حالت نامتعادل را **کوفتگی** یا **کوبیدگی (Thrashing)** می نامند.

# کوبیدگی (Thrashing)

- پردازش در حال کوبیدگی، بیشتر زمانش را صرف مبادله صفحاتش بین حافظه و دیسک می کند بجای اینکه در حال اجرا باشد.

- برای آنکه کارآیی همواره در یک حد خوب باشد بایستی:

$$\text{اندازه RAM} \geq \sum_{i=1}^{n} \text{اندازه مجموعه کارها}$$

- یعنی جمع اندازه مجموعه های کاری n برنامه نباید از اندازه کل حافظه بزرگتر باشد.

**مثال:** اگر زمان اجرای هر دستورالعمل 100 نانو ثانیه و زمان سرویس دهی به هر نقص صفحه 5 میلی ثانیه باشد و بین هر دو نقص صفحه 50000 دستورالعمل اجرا شود، میزان بکار گیری CPU چند درصد است ؟

$$\text{زمان محاسبات CPU} = 50000 \times 100 \times 10^{-9} = 5 \text{ میلی ثانیه}$$

$$\text{درصد بکارگیری CPU} = \frac{\text{زمان محاسبات CPU}}{\text{زمان کُل}} = \frac{\text{زمان محاسبات CPU}}{\text{زمان محاسبات CPU + زمان سرویس نقص صفحه}}$$

$$\text{درصد بکارگیری CPU} = \frac{5}{5+5} = 50\%$$

# نحوه کنترل کوبیدگی

**روش اول:** با محدود کردن چند برنامگی در یک سطح بی خطر یا کنترل بار

- ✓ سطح چند برنامگی را در یک حد ثابت و بی خطر نگه داریم.
- ✓ سطح چند برنامگی بر اساس تخمین مجموعه کار تعیین گردد.
- ✓ فقط کارهایی با هم اجرا شوند که حاصل جمع مجموعه های کاری آنها از اندازه RAM بیشتر نباشد.
- ✓ به کمک تکنیک فرکانس خطای صفحه (Page Fault Frequency) سطح چند برنامگی بصورت خودکار تنظیم شود.
- ✓ هرگاه تعداد نقص صفحه از یک حد فراتر رود، سطح چند برنامگی کاهش می یابد و برعکس.

# نحوه کنترل کوبیدگی

**روش دوم:** جلوگیری از کوبیدگی توسط کنترل مداخله برنامه ها

- مداخله یعنی اینکه یک برنامه باعث شود صفحاتی از مجموعه کاری برنامه دیگر از حافظه خارج شود.
  - ✓ استفاده از تخصیص قاب محلی
  - ✓ اولویت بندی برنامه ها
    - در این حالت برنامه های با اولویت بالا می توانند صفحات برنامه های با اولویت پایین را از حافظه بیرون کنند.

**نکته !** در اکثر سیستم ها سعی می شود در ابتدا، مجموعه کاری هر برنامه به حافظه آورده شود که به آن "پیش صفحه بند" (Prepaging) می گویند.

# اندازه صفحه

- عوامل مؤثر در تعیین اندازه صفحه عبارتند از:

۱) هر چه صفحه کوچکتر باشد، تکه تکه شدن داخلی کمتر می شود.

۲) اگر صفحه بزرگ باشد، ممکن است همه محتویات آن استفاده نشود. ولی اگر کوچک باشد احتمال استفاده همه محتوای آن وجود دارد. لذا صفحه بزرگتر، فضای حافظه بیشتری را به هدر می دهد.

۳) هر چه صفحات کوچکتر باشند، تعدادشان بیشتر شده و در نتیجه حجم جدول صفحه زیاد خواهد شد.

۴) زمان تبادل صفحه بین حافظه و دیسک به اندازه صفحه بستگی ندارد. پس هر چه صفحه بزرگتر باشد، اطلاعات بیشتری در یک مبادله جابجا می شود.

**نکته !** در عمل امروزه اندازه صفحات بزرگتر شده اند چون قدرت و سرعت CPU ها و اندازه حافظه افزایش یافته است.

# سؤال ؟