«به نام خالق آرامش»

نام کتاب: سیستم عامل (بخش دوم)

نام نویسنده: فرشید شیرافکن

تعداد صفحات: ۱۲۴ صفحه

تاریخ انتشار: ــــــــــــــــ







| | |
|---|---|
| (1) cout<<"1"; | (3) wait(S); |
| (2) signal(S); | (4) cout<<"3"; |
| (6) wait(Q); | (5) signal(Q); |
| (7) cout<<"2"; | (9) wait(S); |
| (8) signal(S); | (10) cout<<"4"; |

**مثال**

نحوه اجرای فرایندهای P1 و P2 چگونه باشد، تا مقدار نهایی برابر a=10 , b=6 شود؟

(مقدار اولیه سمافور s برابر یک و مقدار اولیه متغیرهای a,b نیز برابر یک می باشند.)

| P1 | P2 |
|---|---|
| a=a+2; | a=3; |
| b=b+1; | **wait(s);** |
| **signal(s);** | b=a+b; |
| b=b+1; | **wait(s);** |
| | a=a+b; |

پاسخ:

ابتدا سه دستور اول P2 اجرا می شود. بعد از اجرای این سه دستور داریم: a=3,b=4,s=0

سپس سه دستور اول P1 اجرا شده و بعد از اجرا خواهیم داشت: a=5,b=5,s=1

مجددا به P2 بر گشته و دو دستور بعدی آن اجرا شده و داریم: a=10,b=5, s=0

در نهایت به P1 رفته و بعد از اجرای دستور باقی مانده خواهیم داشت: a=10,b=6

■

**مثال**

با فرض اینکه مقدار اولیه سمافور S برابر 8 ، سمافور P برابر 3 وسمافور Q برابر 1 باشد، حداکثر چند فرایند پشت هر سمافور قرار می گیرد؟

```
.
wait(S);
wait(P);
wait(Q);
.
signal(Q);
signal(P);
signal(S);
.
```

پاسخ:

از n فرایند، حداکثر 8 فرایند می تواند از wait(S) عبور کند و بقیه (n-8) فرایند در صف سمافور S می خوابند. از 8 فرایندی که از سمافور S عبور کرده، حداکثر 3 فرایند می تواند از سمافور P عبور کند و 5 فرایند در صف P می خوابند. در نهایت از 3 فرایندی که از سمافور P عبور کرده، حداکثر 1 فرایند از wait(Q) عبور کرده و 2 فرایند در صف Q می خوابند.

■

### مسئله تولیدکننده و مصرف کننده

یک تولیدکننده یا بیشتر، نوعی داده را تولید و آنها را در بافری به اندازه n قرار می دهند. یک مصرف کننده، این اقلام را یکی یکی از بافر برمی دارد. سیستم باید از همپوشانی اعمال بافر جلوگیری کند، یعنی در هر زمان مصرف کننده یا تولید کننده می تواند به بافر دسترسی داشته باشد.

این راه حل از سه سمافور استفاده می کند:

**۱- سمافور mutex :**

سمافوری برای رعایت شرط انحصار متقابل است، تا تولید کننده و مصرف کننده به طور همزمان به بافر دسترسی نداشته باشند. (با مقدار اولیه 1)

**۲- سمافور full :**

سمافوری برای شمارش تعداد خانه های پر بافر (با مقدار اولیه 0)

**۳- سمافور empty :**

سمافوری برای شمارش تعداد خانه های خالی بافر (با مقدار اولیه n)

```
void producer (void){
  int  item;
  while(TRUE)     {
       item= produce( );
       wait(empty);
       wait(mutex);
       insert(item);
       signal(mutex);
       signal(full);    }
}
```

```
void consumer(void){
    int item;
    while(TRUE)    {
       wait(full);
       wait(mutex);
       item=remove( );
       signal(mutex);
       signal(empty);
       consume( );     }
}
```

اگر در مسئله تولید کننده- مصرف کننده ، بافر نامحدود باشد، دیگر نیازی به سمافور empty نیست. در نتیجه دستور wait(empty) از تولید کننده و دستور signal(empty) از مصرف کننده حذف می شود.

### مسئله غذا خوردن فیلسوف ها

پنج فیلسوف دور یک میز دایره ای نشسته اند. هر فیلسوف یک بشقاب ماکارونی دارد. بین هر جفت از بشقاب ها، یک چنگال قرار دارد.

هر فیلسوف برای خوردن از دو چنگال طرفین بشقاب استفاده می‌کند. زندگی هر فیلسوف از دو دوره متناوب خوردن و فکر کردن تشکیل شده است. زمانی که هر فیلسوف گرسنه می شود، سعی می کند دو چنگال سمت چپ و راست خود را بردارد. اگر موفق شد، برای مدتی غذا می خورد و سپس چنگال ها را زمین می گذارد و به فکر کردن ادامه می دهد. مسأله تغذیه فیلسوفان علاوه بر **انحصار متقابل** (که در یک زمان دو فیلسوف نمی توانند از یک چنگال استفاده کنند)، باید جوابگوی **بن بست** و **گرسنگی** نیز باشد.

راه حل: هرفیلسوف ابتدا چنگال چپ و سپس چنگال راست را بر می دارد. بعد از تغذیه یک فیلسوف، دو چنگالی که استفاده می کرد را روی میز گذاشته و دیگران می توانند استفاده کنند.

```
semaphore room=4;
semaphore fork[5]={1};
void philosopher (int i){
  while(TRUE){
    think( );
    wait( room );
    wait( fork[i] );
    wait( fork[(i+1) % 5] );
    eat( );   ناحیه بحرانی
    signal ( fork[(i+1) % 5] );
    signal ( fork[i] );
    signal ( room );
  }
}
void main( ){
  parbegin (p(0),p(1),p(2),p(3),p(4)));
}
```

سمافور room با مقدار اولیه 4 ، برای این است که اجازه ورود به بیش از چهار نفر داده نشود. اگر حداکثر چهار فیلسوف نشسته باشند، حداقل یک نفر به دو چنگال دسترسی خواهد داشت. اگر از room استفاده

نمی شد، و اجازه ورود همزمان هر پنج نفر، داده می شد، همه آنها چنگالهای چپ خود را برداشته و دیگر چنگال اضافی نمی ماند که کسی بتواند چنگال راست خود را بردارد. بنابراین بن بست رخ می داد.

در کتاب تننباوم، راه حل زیر داده شده است. در این راه حل، از یک آرایه به نام state استفاده می کند که وضعیت جاری فیلسوف(فکر کردن(0) ،گرسنگی(1) و خوردن(2)) را نگهداری می کند. یک فیلسوف در صورتی که هیچ یک از فیلسوفان چپ و راستش، در حال خوردن نباشند، می تواند غذا بخورد. این راه حل بن بست ندارد.

```
#define  LEFT    (i-1) % 5
#define  RIGHT   (i+1) % 5
typedef  int  semaphore;
semaphore  mutex=1;
semaphore  s[5];
int  state[5];
void  philosopher (int i){
  while(TRUE){
     think( );
     take_forks(i);
     eat( );
     put_fork(i);
    }
}
void  take_forks(int  i){
  wait(mutex);
  state[i]= 1;
  test(i);
  signal(mutex);
  wait(s[i]);
}
void  put_forks(int  i){
  wait(mutex);
  state[i]=0;
  test( LEFT );
  test( RIGHT );
  signal(mutex);
}
void  test(int  i){
  if( state[i]==1  &&  state[LEFT]!=2  &&  state[RIGHT]!= 2 ) {
     state[i]=2;
     signal(s[i]);
    }
```

}

## مسئله خوانندگان و نویسندگان

در این مسئله، ناحیه داده ای(مثل فایل) وجود دارد که بین تعدادی از فرایندها مشترک است. فرایند های خواننده می خواهند از این ناحیه بخوانند و فرایندهای نویسنده می خواهند در آن بنویسند.

شرایط این مسئله:

۱- هر تعداد از خوانندگان می توانند به صورت همزمان از فایل بخوانند.

۲- در هر زمان تنها یک فرایند ممکن است در این فایل بنویسد.

۳- هنگامی که نویسنده ای در حال نوشتن است، هیچ خواننده ای نمی تواند فایل را بخواند.

برای مثال یک سیستم رزرواسیون هواپیمایی را در نظر بگیرید که تعداد زیادی فرایند در آن برای نوشتن و خواندن با یکدیگر رقابت می کنند. چند فرایند می توانند به طور همزمان پایگاه داده را بخوانند ولی اگر یک فرایند در حال به روز رسانی پایگاه داده باشد، فرایندهای دیگر حتی خوانندگان،نباید به پایگاه داده دسترسی داشته باشند.

این مسئله دارای سه حالت است:

۱) خوانندگان اولیت دارند. (تا زمانی که خواننده ای وجود دارد، به خواننده اجازه ورود می دهیم)

۲) نویسندگان اولویت دارند.

۳) خوانندگان و نویسندگان بدون اولیت هستند.

**حالتی را بررسی می کنیم که خوانندگان اولویت دارند:**

```
typedef int semaphore;
semaphore mutex=1;
semaphore w=1;
int rc=0;


void writer( )
{
  while(TRUE)
  {
     wait(w);
     writing( );
     signal(w);
  }
```

```
}


void  reader( )
{
   while(TRUE)
   {
      wait(mutex);
       rc = rc+1;
       if (rc == 1)  wait(w);
       signal(mutex);

       reading( );

       wait(mutex);
       rc = rc-1;
       if (rc == 0)  signal(w);
       signal(mutex);
    }
}
```

**روال نویسنده:**

اولین نویسنده، با عمل wait(w) ، اجازه دسترسی پیدا کرده (چون مقدار اولیه w برابر 1 است)، ولی با صفر شدن این سمافور،اجازه دسترسی نویسندگان و خوانندگان دیگر گرفته می شود. در این روال، تا هنگامی که نویسنده ای به فایل دسترسی دارد، هیچ نویسنده ای و خواننده دیگری نمی تواند به فایل دسترسی داشته باشد.

**روال خواننده:**

فرایند خواننده از سمافور w برای اعمال انحصار متقابل، از متغیر سراسری rc ، برای شمارش تعداد خوانندگان و از سمافور mutex برای اطمینان از تغییر مناسب rc استفاده می کند.

دستور افزایش rc و کنترل آن در بین wait(mutex) و signal(mutex) قرار دارد، تا تغییر rc ، انحصاری شود. همچنین بعد از پایان عمل خواندن، دستور کاهش rc و کنترل آن در بین wait(mutex) و signal(mutex) قرار دارد تا تغییر rc ، انحصاری شود.

توسط if اول، به اولین خواننده اجازه ورود داده می شود،البته در صورتی که نویسنده ای فعال نباشد.

توسط if پایانی، بررسی می کنیم که اگر خواننده فعال دیگری وجود نداشته باشد، در صورت اینکه نویسنده بلوکـه شـده ای داشته باشیم، به آن اجازه داده شود.

- در این روش شرط انحصار متقابل و شرط پیشرفت رعایت می شود ولی شرط انتظـار محـدود رعایـت نمی شود. (چون اولویت با خوانندگان است و امکان گرسنگی نویسندگان وجود دارد.)

## مانیتور

مشاهده کردید که همگام سازی فرایندها با سمافور پیچیده است. در واقع تنها مشکل سمافور در سیستمی که حافظه مشترک دارد، پیچیدگی استفاده از آن در حل مسائل همگام سازی می باشد. اگر برنامه نویس در استفاده از سمافور دقت لازم را نکند، ممکن است دچار بن بست شود.

مانیتور(ناظر) ساختاری از زبان برنامه سازی است که همان کار سمافور را انجام می دهد و کنترل آن هم ساده تر است. سمافور اغلب توسط سیستم عامل پشتیبانی می شود و می تواند توسط زبان برنامه سازی نیز پشتیبانی شود، اما مانیتور باید فقط توسط زبان برنامه سازی پشتیبانی شود.

ساختار مانیتور در زبانهای برنامه سازی به صورت برنامه کتابخانه ای پیاده سازی شده است. این به افراد اجازه می دهد تا قفل های مانیتور را روی هر شیئی بگذارند. مانیتور مولفه ای نرم افزاری، مشتمل بر یک یا چند رویه، دنباله ای از مقدارگذاری های اولیه و داده های محلی است.

**ویژگی های اصلی مانیتور:**

۱ – متغیرهای داده ای محلی مانیتور، تنها برای رویه های خود مانیتور قابل دسترس بوده و هیچ رویه دیگری به آنها دسترسی ندارد.

۲ – یک فرایند با احضار یکی از رویه های مانیتور، وارد آن می شود.

۳ – در هر زمان تنها یک فرایند می تواند در مانیتور در حال اجرا باشد، فرایندهای دیگری که مانیتور را احضار کرده اند تا فراهم شدن مانیتور، معلق خواهند بود. (برقراری انحصار متقابل)

## متغیرهای شرطی(condition variable)

مانیتور با استفاده از متغیرهای شرطی که تنها از داخل مانیتور، قابل دسترس هستند، از همگام سازی حمایت می کند. برای این کار از دو تابع کتابخانه ای cwait(condition) و csignal(condition) که ساختاری از زبان برنامه سازی هستند، استفاده می کنند. در بعضی از منابع، از c اول این دستورها استفاده نمی شود.

اگر فرایندی cwait(c) را صدا بزند، پشت شرط c به خواب می رود. اگر فرایندی csignal(c) را صدا بزند، یک پیغام بیدار باش برای فرایندهایی که پشت شرط c خوابیده اند می فرستد که فقط یکی از آنها توسط زمانبند سیستم بیدار می شود.

- متغیرهای شرطی، شمارنده نیستند و مانند سمافور نمی توانند سیگنال ها را برای استفاده آینده، ذخیره کنند.
- امتیازی که مانیتورها نسبت به سمافورها در همگام سازی فرایندها دارند، این است که تمام اعمال همگام سازی در محدوده مانیتور است. بنابراین کشف خطاها و تعیین اینکه همگام سازی صحیح انجام شده است، ساده تر می باشد.
- کامپایلر(نه برنامه نویس) انحصار متقابل را به صورت خودکار در مانیتور برقرار می سازد.

## حل مسئله تولیدکننده- مصرف کننده توسط مانیتور

برای حل مسائل به کمک مانیتور، نواحی بحرانی را در داخل مانیتور تعریف می کنیم. برای بخش هـایی در مسئله که نیاز به همگام سازی وجود دارد، از متغیرهای شرطی کمک می گیریم.

```
monitor   ProducerConsumer;
  condition  full,empty;
  integer  count;

procedure   enter;
begin
      if  count = N then  cwait (full);
      enter_item;
      count := count + 1;
      if  count=1 then  csignal(empty);
end;

procedure   remove;
begin
      if count = 0 then  cwait (empty);
      remove_item;
      count := count - 1;
      if count=N-1  then  csignal(full);
end;

procedure  producer;
begin
     while  true  do
      begin
         produce_item;
         ProducerConsumer.enter;
      end
end;

procedure consumer;
begin
    while  true  do
      begin
           ProducerConsumer.remove;
           consume_item;
```

```
        end
    end;
```

- زبانهای برنامه سازی مانند C و پاسکال استاندارد، مانیتور را پشتیبانی نمی کنند.
- در عمل همگام سازی مانیتورها هم ممکن است اشتباه رخ دهد. مثلاً اگر هر یک از اعمال csignal در ناظر bounded buffer حذف شوند، در این صورت فرایندهایی که وارد صف شرط مربوط به آن می شوند، تا ابد معطل خواهند بود.
- در زبان جاوا، اگر نخی اجرای متدی که در معرفی آن از واژه کلیدی synchronized استفاده شده را شروع کند، هیچ نخ دیگری اجازه ندارد شروع به اجرای هیچ یک از متدهای synchronized درون آن کلاس کند. جاوا متغیرهای شرطی ندارد و به جای آنها دو رویه wait و notify دارد که وقتی این دو رویه در متدهای synchronized به کار روند، مشکل رقابتی پیش نمی آید.
- بر اثر اجرای csignal ممکن است فرایند دیگری که از قبل wait شده بود، فعال گردد و در نتیجه بیش از یک فرایند در یک زمان در مانیتور فعال شود. برای پرهیز از فعالیت همزمان دو فرایند درون یک مانیتور، Hansen پیشنهاد کرد که فرایندی که csignal را انجام داده باید فورا از مانیتور خارج شود. بنابراین یک دستور csignal می تواند فقط به عنوان آخرین دستور در رویه مانیتور به کار رود.

### تبادل پیام (Message Passing)

وقتی فرایندها با یکدیگر محاوره می کنند، نیازهای همگام سازی و ارتباط باید تامین شود. فرایندها نیاز به همگام سازی دارند تا انحصار متقابل اعمال گردد. ممکن است فرایندهایی که با یکدیگر همکاری می کنند نیاز به تبادل اطلاعات داشته باشند. یک رویکرد در فراهم کردن این دو عمل، تبادل پیام است. عمل واقعی تبادل پیام به شکل دو اولیه زیر ارائه می شود. این اولیه ها، فراخوان سیستمی هستند که می توان آنها را در رویه های کتابخانه ای قرار داد:

**1 - send** (destination , message)

**2- receive** (source , message)

فراخوان send، پیامی را از آدرسی که در فضای آدرس فرایند مبدا قرار دارد را برداشته (message) و به فرایند مقصد(destination) ارسال می کند. فرخوان recevie، یک پیام را از فرایند مبدا (source) دریافت کرده و آن را در آدرس مشخص شده (message) قرار می دهد.

دو نوع آدرس دهی وجود دارد:

۱- **مستقیم**: پیامها مستقیما از فرستنده به گیرنده فرستاده می شود.

۲- **غیر مستقیم**: فرستنده پیام را به یک صف مشترک(صندوق پستی) ارسال می کند و گیرنده پیام را از صندوق پستی بر می دارد.(سیستم عامل برای هر فرایند، بافری به نام mailbox ایجاد می کند.)

- اگر بین فرستنده ها و گیرنده ها رابطه چند به یک وجود داشته باشد، به صندوق پستی، "درگاه" می گویند.
- پیام در یک قالب متداول از دو بخش، سرآمد(حاوی اطلاعاتی درباره پیام) و بدنه(حاوی خود پیام) تشکیل شده است.

### همگام سازی به کمک تبادل پیام

می توان به کمک send و receive امکان هماهنگی بین فرایندها را ایجاد کرد. تبادل پیام بین دو فرایند متضمن سطحی از همگام سازی بین آنها می باشد.

زمانی که فرایندی، send را اجرا می کند، دو امکان وجود دارد:

الف- فرایند فرستنده تا دریافت پیام مسدود می شود.

ب- فرایند فرستنده، بدون توجه به عمل انجام شده، اجرایش ادامه می یابد.

زمانی که فرایندی recive را اجرا می کند، دو امکان وجود دارد:

الف- اگر پیامی قبلا فرستاده شده، این پیام دریافت شده و اجرا ادامه می یابد.

ب- اگر پیام منتظری وجود نداشته باشد، فرایند تا رسیدن پیام مسدود می ماند و یا به اجرا ادامه می دهد. بنابراین فرستنده و گیرنده هر دو می توانند مسدود شونده یا مسدود نشونده باشند.

**پیاده سازی انحصار متقابل توسط تبادل پیام**

از تبادل پیام می توان برای اعمال انحصار متقابل استفاده کرد. فرض می کنیم کهsend مسدود نشونده و receive مسدود شونده است. یعنی وقتی فرایندی می خواهد اطلاعاتی را دریافت کند، با اجرای receive، بلوکه می شود تا اطلاعات برایش فرستاده شود.

مجموعه ای از فرایندهای همزمان در صندوق پستی mutex شریکند و می توانند از این صندوق پستی برای ارسال و دریافت پیام استفاده کنند.

```
const  n=  تعداد فرایندها  ;
void  p(int  i)
{
 while(true)
  {
     receive(mutex , msg);
     /*critical section */
     send(mutex , msg);
    /*non critical section */
   }
}

void  main( )
{
   create_mailbox(mutex);
   send(mutex , null);
   Par begin( P(1) , P(2), …,P(n) );
}
```

صندوق پستی با پیامی با محتوای تهی مقدار گذاری اولیه شده است. فرایندی که می خواهد وارد بخش بحرانی خود شود، ابتدا سعی می کند پیامی دریافت نماید. اگر صندوق پستی خالی باشد، مسدود می گردد. هنگامی که فرایندی پیام را به دست می آورد، بخش بحرانی خود را انجام داده و سپس پیام را به صندوق پستی بر می گرداند. در نتیجه این پیام نقش نشانه ای را بازی می کند که از فرایندی به فرایند دیگر منتقل می شود.

اگر بیش از یک فرایند عمل دریافت را همزمان انجام دهند، در این صورت:

۱- اگر پیامی باشد، تنها به یک فرایند داده شده و بقیه مسدود می شوند.

۲- اگر صندوق پستی خالی باشد، تمام فرایندها مسدود می گردند. موقعی کـه پیـامی فـراهم مـی شود، تنها یکی از فرایندهای مسدود فعال شده و پیام به همان فرایند داده شود.

این فرض ها در تمام امکانات تبادل پیام، برقرار است.

## حل مسئله تولیدکننده- مصرف کننده توسط تبادل پیام

می توان توسط تبادل پیام، راه حلی برای مسئله تولید کننده و مصرف کننده با بافر محدود ارائه داد. این برنامه علاوه بر علائم، داده ها را نیز عبور می دهد. تولید کننده با تولید داده ها، آنها را به عنوان پیام به صندوق پستیmc می فرستد. مصرف کننده تا هنگامی که حداقل یک پیام در صندوق وجود دارد، می تواند مصرف کند. بنابراین صندوق پستی mc مانند یک بافر عمل می کند.

راه حل:

یک صندوق پستی برای تولید کننده(mp)و یک صندوق پستی برای مصرف کننده(mc) ایجاد می شود:

```
create_mailbox(mc);
create_mailbox(mp);
```

سپس صندوق پستی تولید کننده، به اندازه ظرفیت بافر، با پیام تهی پر می شود:

```
for(i=0 ; i<capacity; i++)
   send(mp,NULL);
```

سپس دو تابع تولید کننده و مصرف کننده به صورت همروند صدا زده می شود:

```
void producer( )
{
   message m1;
   while(TRUE)
    {
       receive(mp,m1);
       m1=produce( );
      send(mc,m1);
    }
}
```

```
void consumer( )
{
   message m2;
   while(TRUE)
    {
      receive(mc,m2);
      consume(m2);
      send(mp,null);
    }
}
```

زمانی که فرایندهای فرستنده و گیرنده پیام بر روی یک ماشین اجرا می شوند، مسئله کارایی پیش می ایدبه عبارتی کپی کردن پیام ها از یک فرایند به فرایند دیگر همیشه آهسته تر از انجام عملیات سمافور و یا ورود به یک مانیتور است.

گیرنده می تواند به محض دریافت پیام از فرستنده، یک پیام تصدیق(acknowledgment) به فرستنده بفرستد و او را از دریافت پیام با خبر سازد.

در یک سیستم توزیعی با حافظه اختصاصی برای رعایت انحصار متقابل، از راه حل تبادل پیام استفاده می شود.

**(مهندسی کامپیوتر- دولتی ۷۹)**

۱- دو فرایند P1 و P2 زیر به صورت هم روند اجرا می شوند. در صورتی که مقدار اولیه متغیر سراسری a صفر باشد، بعد از اجرای کامل دو فرآیند، کدام یک از گزینه های ذیل نادرست می باشد؟

(امکان اجرای آنها به صورت Interleaved نیز وجود دارد. یعنی در هر لحظه از اجرای فرایند، امکان وقوع وقفه و سوئیچ به فرایند دیگر وجود دارد.)

| p1 | p2 |
|---|---|
| a=1; | b=a;<br>c=a; |

۱) هر یک از مقادیر a وb وc یک می باشد.

۲) مقادیر b وc صفر می باشد و مقدار a یک است.

۳) مقادیر a و c هر کدام یک می باشد و مقدار b صفر است.

۴) مقادیر a وb هر کدام یک می باشد و مقدار c صفر است.

پاسخ: جواب گزینه ۴ است.

بعد از اجرای کامل دو فرایند، مقادیر این متغیرها نامشخص است و حالت های زیر ممکن می باشد:

۱) ابتدا p1 به طور کامل اجرا شده و سپس p2 اجرا شود. در این حالت a=1,b=1,c=1 .

۲) ابتدا p2 به طور کامل اجرا شده و سپس p1 اجرا شود. در این حالت a=1,b=0,c=0 .

۳) دستور اول از p2 اجرا شده و به p1 سوئیچ شود. بعد از اجرای p1 به p2 سوئیچ شده و دستور بعدی آن اجرا شود. در این حالت داریم a=1,b=0,c=1 .

گزینه ۴ نادرست است، چون وقتی مقدار b یک می شود که دستور b=a بعد از دستور a=1 اجرا شود. بعد از اجرای این دو دستور، نوبت به اجرای دستور c=a می باشد که مقدار c یک می شود.

**(مهندسی IT- دولتی ۸۴)**

۲- در یک سیستم هم روند، هر پروسس برای ورود به بخش بحرانی، تابع enter-region و پس از خروج از بخش بحرانی تابع leave-region آمده در زیر را صدا می کند. کدام گزینه صحیح است؟

```
#define  FALSE  0
# define  TRUE   1
```

```
# define N  2
int  turn;
int  intersted[N];   /* all values initially 0(FALSE)   */
void enter-region(int process){
  int other;
 other =1 - process;
  interested[process] = TRUE;
  turn = process;
  while (turn == process  &&  interseted[other] );
}
void  leave-region(int process){
   interested[process] = FALSE;
}
```

۱) امکان گرسنگی وجود دارد (starvation)  ۲) امکان بن بست وجود دارد. (deadlock)

۳) انحصار متقابل تأمین می شود.  ۴) انحصار متقابل تأمین نمی شود.

پاسخ: جواب گزینه ۳ است.

این تست همان راه حل پیترسون در کتاب Tanenbaum ، است. در نتیجه هر سه شرط "انحصار متقابل، انتظار محدود و پیشرفت" را تامین می کند. و مشکل بن بست و گرسنگی ندارد.

در این راه حل، هر فرایند قبل از ورود به ناحیه بحرانی، تابع enter_region و پس از خروج، تابع leave_region را فراخوانی می کند.

به طور مثال برای process=0 داریم:

```
P0(void)
{
  while(TRUE)
   {
     enter_region(0);
     critical-section( );
     leave_region(0);
   }
}
```

**(مهندسی IT- دولتی ۸۳)**

۳- الگوریتم زیر یک راه حل نرم افزاری برای حل مسئله ناحیه بحرانی است. در این الگوریتم:

**(IT- دولتی ۸۳)**

```
var  c1, c2 : boolean;
```

```
turn: integer;
c1 :=TRUE;
c2 :=TRUE;

cobegin
P1:
   loop
       c1:=FALSE;
       turn:=1
       while (not  c2) and (turn=1) do;
       CS1;
       c1:=TRUE;
    end loop
P2:
   loop
       c2:=FALSE;
       turn=2;
       while (not  c1) and  (turn=2) do;
       CS2;
       c2:=TRUE;
     end loop
coend
```

۱) این الگوریتم صحیح می باشد.

۲) Dead lock وجود دارد.

۳) Starvation وجود دارد.

۴) شرایط ناحیه بحرانی برآورده نمی شود (استفاده انحصاری)

پاسخ: گزینه ۱ جواب است.

این الگوریتم، مشابه راه حل Peterson است. در پیترسون مقدار اولیه flag ها flase بود و هر فرایند قبل از ورود به ناحیه بحرانی، آن را true و بعد از خروج آن را false می کرد. در این تست، این منطق معکوس شده است که تاثیری بر عملکرد صحیح آن نمی گذارد و مانند پیترسون درست کار می کند و همه شرایط را رعایت می کند.

**(مهندسی کامپیوتر- دولتی ۷۴)**

۴- آیا کد صوری زیر برای مسأله Critical Section بین دو فرآیند هم روند قابل قبول است؟چرا؟

(اپراتور AND به معنای اجرای هم روند است.)

```
{
 int  turn;
 boolean  falg[2];
 proc (int  i ) {
   while(TRUE)
   {
      compute;
      falg[i]:= TRUE;
      turn:= (i+1) mod 2;
      while( flag[(i+1) mod 2]  and  turn = i );
      critical-section;
      flag[i] := FALSE;
   }
 }
 turn:=0;
 falg [0] :=FALSE;
 falg [1] :=FALSE;
 proc(0)  AND  proc(1);
}
```

۱) خیر– زیرا Deadlock وجود دارد.

۲) بلی– زیرا شرط Mutual Exclusion برقرار است.

۳) خیر– زیرا شرط Mutual Exclusion برقرار نیست.

۴) بلی– زیرا Deadlock وجود ندارد.

پاسخ: جواب گزینه ۳ است.

روش به کار رفته، روش پیترسون است با این تفاوت که قبل از حلقه در turn ، مقدار معکوسی قرار داده می شود. (یعنی برای p0 مقدار 1 و برای p1 مقدار 0). که این کار باعث نادرست شدن روش می شود.

فرض کنید P0 اجرا شود و flag خود را true کرده و توسط دستور turn:=(i+1)mod 2 مقدار turn را یک می کند و چون turn برابر صفر نیست از while عبور کرده و وارد ناحیه بحرانی می شود. اگر در این زمان وقفه ای رخ دهد و به P1 سوئیچ شود، این فرایند نیز بعد از true کردن flag خودش ، با اجرای دستور turn:=(i+1)mod 2 مقدار turn را صفر کرده و چون turn برابر یک نیست ،این فرایند نیز وارد ناحیه بحرانی می شود. بنابراین شرط انحصار متقابل برقرار نیست.

■

**(مهندسی IT- دولتی ۸۸)**

۵- راه حل ناحیه بحرانی زیر را برای فرآیندهای $P_i : (i = 1,2)$ درنظر بگیرید. کدام مورد صحیح است؟

```
shared var
  turn: integer;  turn:=0;
Pi :
  while (1) {
  flag[i]:=TRUE;
  turn:= ( turn+ i ) %2 + 1;
  while( not( flag[i]) or  turn == i %2 +1 );
  Critical – Section
  flag[i]:= FALSE;
  turn:=( turn+ i) %2 + 1;
  Non Critical – Section
}
```

۱) راه حل ناحیه بحرانی کاملا صحیح است.

۲) شرط پیشرفت (Progress) تنها شرطی است که نقض می گردد.

۳) شرط انحصار متقابل (multual exclusion) تنها شرطی است که نقض می گردد.

۴) هر دو شرط انحصار متقابل و پیشرفت نقض می شوند.

پاسخ: گزینه ۴ جواب است.

مقدار flag ها در این راه حل بی اثرند. چون در بدنه هر فرایند، به جای بررسی flag فرایند مقابل، flag خود فرایند چک می شود. بعد از حذف flag ها، کد هر دو فرآیند به صورت زیر می باشد:

| P1 | P2 |
|---|---|
| turn:=0<br>**while**(1){<br>turn:= (turn+1) %2 + 1;<br>**while**( turn == 2);<br>**Critical – Section**<br>turn:=(turn+1) %2 + 1;<br>Non Critical – Section} | turn:=0;<br>**while**(1){<br>turn:= (turn+2) %2 + 1;<br>**while**( turn == 1);<br>**Critical – Section**<br>turn:=(turn+2) %2 + 1;<br>Non Critical – Section} |

اگر p1 اجرا شود، مقدار turn را به 2 تغییر می دهد و در حلقه while منتظر می ماند. اگر در این لحظه به p2 ، سوئیچ شود ،p2 نیز مقدار turn را به 1 تغییر می دهد و در حلقه while منتظر می ماند. بنابراین بن بست رخ داده و شرط انتظار محدود نقض می شود. شرط پیشرفت نیز نقض می شود، چون زمانی که p1 در حلقه while منتظر است، اگر p2 که در خارج از ناحیه بحرانی است، قصد ورود به ناحیه بحرانی را نداشته باشد، مانع پیشرفت p1 می شود. با این شرایط بدیهی است که شرط انحصار متقابل نیز نقض می شود. بنابراین هر ۳ شرط "انحصار متقابل، پیشرفت و انتظار محدود" نقض می شود.

تذکر: بهتر بود طراح محترم، در گزینه ۴ ، شرط انتظار محدود را نیز اضافه می کرد، تا داوطلب فکر نکند که منظور این است که فقط آن دو شرط که در گزینه ۴ آمده، نقض می شود.

### (مهندسی IT – دولتی ۹۱)

۶-آیا کد زیر می تواند راه حلی برای دو پردازش هم روند باشد؟

```
proc ( int  i ){
   while(TRUE){
    computeation;
    key[i] = TRUE;
    while ( key[i] )
      swap (key[i] , lock );
    ناحیه بحرانی CS
    lock = FALSE;
   }
}
lock = FALSE;
key[1] = FALSE;
key[2] = FALSE;
```

۱) راه حل صحیح نیست، زیرا انحصار متقابل(mutual exclusion) رعایت نمی شود.

۲) راه حل صحیح نیست، زیرا شرط پیشرفت برقرار نیست.

۳) راه حل صحیح نیست، زیرا تضمینی برای محدودیت زمان انتظار ندارد.

۴) راه حل صحیح است.

پاسخ: جواب گزینه ۳ است.

راه حل داده شده، انحصار متقابل و پیشرفت را رعایت می کند. ولی چون نوبت دهی برای ورود به ناحیه بحرانی کاملا تصادفی است، شرط انتظار محدود رعایت نمی شود.

### (مهندسی IT – دولتی ۹۰)

۷- پیاده سازی زیر از عملیات تجزیه ناپذیر(atomic) روی سمافورها را در نظر بگیرید:

```
typedef   struct{
  int   value;
   struct  process  *list;
 }semaphore;
```

| **wait**(semaphore  ***S**) | **signal**(semaphore  ***S**) |
|---|---|
| { | { |

| | |
|---|---|
| S −> value--;<br>if ( S −> value < 0 )<br>{<br>add this process to S −> List;<br>**block**(-(S −> value));<br>}<br>} | S −> value++;<br>if ( S −> value <= 0 )<br>{<br>remove a process P from S −> List;<br>**wakeup**(P);<br>}<br>} |

مفروضات و تعاریف زیر را نیز داریم:

block(n) : فرآیندهای بلوک شونده را به ترتیب n از کوچک به بزرگ مرتب کرده که به این ترتیب فرآیند اول با block(1) در سر صف قرار می گیرد و به ترتیب n (از کوچک به بزرگ) توسط wakeup(P) از حالت بلوک خارج می شوند.

$S->value=1$

برنامه های سیستم به شکل زیر مفروضند:

```
mutex: semaphore;
do{
   wait(mutex);
   critical-section( );
   signal(mutex);
   remainder-section
}while(TRUE)
```

کدام گزینه صحیح است؟

۱) خواص انحصار متقابل و انتظار محدود برقرارند ولی پیشرفت برقرار نیست.

۲) خواص انحصار متقابل و پیشرفت برقرارند ولی انتظار محدود برقرار نیست.

۳) خواص پیشرفت و انتظار محدود برقرارند ولی انحصار متقابل برقرار نیست.

۴) خواص انحصار متقابل برقرار است ولی پیشرفت و انتظار محدود برقرار نیست.

پاسخ: جواب گزینه ۲ است.

چون ترتیب خروج فرایندها بلوکه شده در صف، بر اساس ورود آنها نیست(سمافور ضعیف)، بنابراین امکان گرسنگی(قحطی) وجود دارد. بنابراین شرط انتظار محدود برقرار نیست.

### (مهندسی کامپیوتر- آزاد ۸۶)

۸- در صورتی که دو کد زیر برای حل مسأله کلاسیک همزمانی بافر محدود با ظرفیت n باشد، آن گاه مقدار سمافورهای x و y و z باید چه مقداری باشند؟

| تولید کننده | مصرف کننده |
|---|---|
| **while** (1) { | **while** (1) { |

| | |
|---|---|
| **wait** (z)<br>**wait** (y)<br>حذف قطعه از بافر<br>**signal**(y)<br>**signal** (x)<br>استفاده از قطعه<br>} | تولید قطعه<br>**wait** (x)<br>**wait** (y)<br>اضافه کردن قطعه به بافر<br>**signal** (y)<br>**signal** (z)<br>} |

۱) x=0 , y=1 , z=n　　۲) x=n , y=n , z=1

۳) x=n , y=1 , z=0　　۴) x=0 , y=n , z=1

پاسخ: جواب گزینه ۳ است. از x برای شمارش تعداد خانه های خالی با مقدار اولیه n ، از z برای شمارش تعداد خانه های پـر با مقدار اولیه 0 و از y برای انحصار متقابل با مقدار اولیه 1 استفاده می شود.

■

**(مهندسی کامپیوتر- دولتی ۸۴)**

۹- اگر مقادیر اولیه سمافورهای n,s به ترتیب0,1 باشد، چنانچه دو زیر روال به طور هم روند اجرا شوند، کدام یـک از گزینـه های زیر صحیح است؟

| | |
|---|---|
| procedure **producer**<br>begin<br>repeat<br>produce;<br>**wait**(s);<br>append;<br>**signal**(n);<br>**signal**(s);<br>forvere<br>end; | procedure **consumer**<br>begin<br>repeat<br>**wait**(s);<br>**wait**(n);<br>take;<br>**signal**(s);<br>**signal**(n);<br>forever<br>end; |

۱) راه حل کاملا درست است.

۲) امکان بن بست وجود دارد.

۳) امکان عدم تأمین انحصار متقابل وجود دارد.

۴) امکان دارد که consumer در حالت گرسنگی بماند و producer فعال باشد.

پاسخ: همان مسئله تولید کننده- مصرف کننده با بافر نامحدود است، ولی جای دو wait در مصرف کننده عوض شده اسـت. اگر ابتدا مصرف کننده اجرا شود، wait(s) ، s را صفر و wait(n) ، n را 1- کرده و در نتیجه مصرف کننده مسدود می شـود.

حال اگر تولید کننده اجرا شود، wait(s) ، s را 1- می کند و در نتیجه تولید کننده نیز مسدود می شود. در این لحظه چون هر دو فرایند مسدود هستند، بنابراین بن بست رخ می دهد. ■

**( مهندسی IT – دولتی ۸۵)**

۱۰- در راه حل زیر برای شام خواران فیلسوف (۵ فیلسوف)، فرض کنید اجرای روال های برداشتن دو چنگال و گذاشتن هر چنگال از ابتدا تا انتهای روال با رعایت کامل Mutual Exclusion انجام می شود.

روال take-forks(i) دو چنگال سمت راست و چپ را بررسی می کند و اگر هر دو موجود بودند برمی دارد وگرنه عمل بررسی را تکرار می کند.

روال put-fork(i) چنگال شماره i را می گذارد و از روال خارج می شود.

```
void  philosopher (int i){
  while(1)  {
     think;
     take-forks(i);
     eat;
     put-fork(i);
     put-fork((i+1)%n);
    }
}
```

کدام گزینه درست است؟

۱) دارای بن بست

۲) فاقد بن بست و گرسنگی

۳) دارای بن بست و گرسنگی

۴) فاقد بن بست ولی دارای گرسنگی

پاسخ: گزینه ۴ دست است.

در این راه حل چون توسط روال take-forks(i) یا هر دو چنگال همزمان برداشته شده و یا هیچ کدام برداشته نمی شود، بنابراین بن بست رخ نمی دهد.

در این روش امکان گرسنگی وجود دارد، چون همه چیز تصادفی است و نوبت و عدالت رعایت نمی شود و ممکن است یک فیلسوف به مدت نامعلوم گرسنه بماند.

■

**(مهندسی کامپیوتر - آزاد ۸۷)**

۱۱- کدام گزینه در مورد مانیتور درست نیست؟

۱) فقط یک روال مانیتور در آن می تواند فعال باشد.

۲) یک مانیتور هیچ گاه در بخش بحرانی خود مسدود نمی شود.

۳) پیاده سازی مانیتور در سطح کامپایلر انجام می شود.

۴) مانیتور به صورت ضمنی عملیات درخواست ورود/خروج از بخش بحرانی را انجام می دهد.

پاسخ: جواب گزینه ۲ است.

در مانیتور امکان مسدود شدن با wait بر روی متغیر شرطی، وجود دارد.

# فصل ۵

# بن بست

بن بست (Deadlock) یعنی مسدود بودن دائمی مجموعه ای از فرایندها که برای منابع سیستم رقابت می کنند یا با یکدیگر در ارتباط هستند. این مسدود بودن ادامه می‌یابد تا سیستم عامل عمل فوق العاده‌ای انجام دهد، مثلاً فرایندی را حذف کند یا مجبور به برگشت به عقب کند.

یک مثال معمول از بن بست، ترافیک است. چهار خودرو که در یک زمان به چهار راهی رسیده‌اند مفروض است. چهار ربع این چهار راه منابع محسوب می‌شوند. اگر هر چهار خودرو وارد تقاطع بشوند، هر خودرو یک ربع از چهار راه (یک منبع) را در اختیار گرفته ولی نمی‌تواند پیش برود. چون منبع مورد نیاز دوم در اختیار خودروی دیگر است و این امر باعث ایجاد بن بست خواهد شد. (هر خودرو برای عبور از چهار راه به دو ربع از چهارراه نیاز دارد.)

### شرایط بن بست

اگر چهار شرط زیر **همزمان** در سیستمی وجود داشته باشد، بن بست رخ می‌دهد:

۱- **نگهداری و انتظار** (Hold and Wait)

فرایندی وجود دارد که حداقل یک منبع را در اختیار داشته باشد(نگهداری) و منتظر به دست آوردن منبع دیگری باشد که فعلا در اختیار فرایند دیگری است.

**۲- انحصاری بودن (قبضه نشدنی)** (Non Preemption)

هنگامی که فرایندی منبعی را در اختیار دارد و نتوان آن منبع را به زور باز پس گرفت.

**۳- انحصار متقابل** (Mutual Exclusion)

در هر زمان تنها یک فرایند می‌تواند از یک منبع استفاده کند و اگر فرایند دیگری درخواست همان منبع را کند، باید منتظر بماند تا منبع آزاد شود.

**۴- انتظار چرخشی** (Circular Wait)

مجموعه ای از فرایندهای منتظر $\{P_0, P_1, ..., P_n\}$وجود دارد که $P_0$ منتظر منبعی باشد که در اختیار $P_1$ است و $P_1$منتظر منبعی باشد که در اختیار $P_2$است و به همین ترتیب، $P_{n-1}$منتظر منبعی باشد که در اختیار $P_n$ است و در نهایت $P_n$منتظر منبعی است که در اختیار $P_0$ است. در واقع چرخه ای از انتظار وجود دارد.

برقرار بودن هر چهار شرط، شروط لازم و کافی برای وقوع بن بست می باشند.

شرط انتظار چرخشی منجر به شرط نگهداری و انتظار می‌گردد.

**مثال**

سیستمی با دو منبع R1 و R2 مفروض است. اگر فرایند P1 منبع R1 و فرایند P2 منبع R2 را در اختیار داشته باشد و فرایند P1 برای تکمیل اجرا به منبع R2 و فرایند P2 به منبع R1 نیاز داشته باشد، بن بست رخ می‌دهد.

اگر در سیستمی n فرایند و m منبع (از یک نوع) موجود باشد، در صورت برقرار بودن شرط $\sum_{i=1}^{n} \text{Re}\,quest(i) < m + n$ بن بست رخ نخواهد داد.

**مثال**

کامپیوتری دارای m مورد از یک منبع می باشد و n فرایند برای در اختیار گرفتن آنها با هم رقابت می کنند. هر فرایند حداکثر به دو مورد نیاز دارد. حداکثر مقدار n که به ازای آن می توان مطمئن بود، سیستم دچار بن بست نشود، چقدر است؟

حل: چون در این مثال هر فرایند حداکثر به دو منبع نیاز دارد، مجموع کل درخواستها برای n فرایند برابر 2n می باشد و داریم:

$$\sum_{i=1}^{n} \text{Re}\,quest(i) < m+n \Rightarrow 2n < m+n \Rightarrow n < m$$

بنابراین حداکثر مقدار n که به ازای آن بن بست رخ نمی دهد، برابر m – 1 می باشد.

■

**مثال**

یک سیستم کامپیوتری دارای 6 عدد TapeDrive است که n پردازه برای دستیابی به آنها رقابت می‌کنند. هر پردازه به دو درایو نیاز دارد. این سیستم به ازای حداکثر چه ارزشهایی از n فاقد بن بست است؟

$$\sum_{i=1}^{n} \text{Re}\,quest(i) < m+n \Rightarrow 2n < 6+n \Rightarrow n < 6$$

بنابراین حداکثر پنج فرایند می‌تواند وجود داشته باشد.در این حالت اگر هرکدام از فرایندها یک منبع را در اختیار گیرند، یک منبع آزاد باقی می‌ماند که یکی از فرایندها با گرفتن آن کامل می‌شود و به همین ترتیب بقیه فرایندها کامل خواهند شد و بن بست رخ نمی‌دهد. ■

**گراف تخصیص منابع**

توسط گراف تخصیص منابع می‌توان مسئله بن بست را دقیقتر تشریح کرد. در این گراف، فرایندها با دایره و منابع با مربع نشان داده می‌شوند.

در گراف تخصیص منابع دو نوع یال وجود دارد:

**۱- یال درخواست** $P_i \rightarrow R_j$

فرایند $P_i$ نمونه‌ای از منبع $R_j$ را درخواست کرده است و منتظر آن منبع است.

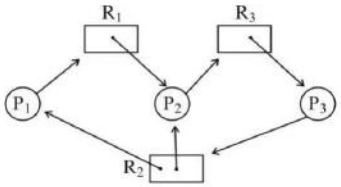
**۲- یال تخصیص** $R_j \rightarrow P_i$

نمونه ای از منبع $R_j$ به فرایند $P_i$ تخصیص داده شده است.

اگر گراف تخصیص منابع فاقد چرخه باشد، حالت بن بست وجود ندارد.

**مثال**

در گراف تخصیص منابع زیر، وضعیت بن بست را بررسی کنید؟




حل:

حالتهای فرایند در این گراف عبارتند از:

۱- فرایند P1 نمونه‌ای از منبع R2 را در اختیار دارد و منتظر نمونه‌ای از منبع R1 است.

۲- فرایند P2 نمونه‌ای از منبع R1 و منبع R2 را در اختیار دارد و منتظر نمونه‌ای از منبع R3 است.

۳- فرایند P3 نمونه‌ای از منبع R3 را در اختیار دارد و منتظر نمونه‌ای از منبع R2 است.

با توجه به این گراف، دو چرخه کمینه وجود دارد و به همین علت همه فرایندها در بن بست قرار دارند. این چرخه ها عبارتند از:

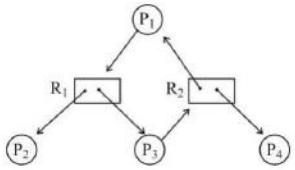
1) $P1 \to R1 \to P2 \to R3 \to P3 \to R2 \to P1$

2) $P2 \to R3 \to P3 \to R2 \to P2$

■

**مثال**

در گراف زیر وضعیت بن بست را بررسی نمایید.



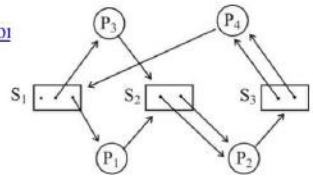

حل: با وجود چرخه $P_1 \to R_1 \to P_3 \to R_2 \to P_1$، بن بست رخ نمی دهد. چون فرایند $P_4$ می‌تواند منبع نمونه $R_2$ را آزاد کند و آنگاه منبع $R_2$ به فرایند $P_3$ داده می‌شود و چرخه از بین برود. ■

اگر چرخه ای در گراف وجود داشته باشد، ممکن است بن بست وجود داشته باشد.

**مثال**

سیستمی متشکل از چهار پردازه P1 وP2 وP3 وP4 و سه نوع منبع قابل استفاده مجدد S1 و S2 وS3 را در





نظر بگیرید. در وضعیت موجود، کدام یک از پردازه ها در بن بست قرار ندارند؟

حل: تعداد واحدهای هر منبع به ترتیب 2,2,3 است. P1 یک واحد از منبع S1 را در اختیار دارد و تقاضای یک واحد از S2 را کرده است. P2 دو واحد از S2 را در اختیار دارد و تقاضای یک واحد از S3 را کرده است. P3 یک واحد از S1 را در اختیار دارد و تقاضای یک واحد از S2 را کرده است. P4 نیز دو واحد از S3 را دراختیار دارد و تقاضای یک واحد از S1 کرده است. فرایند P4 یک واحد از S1 که موجود است را در اختیار گرفته و کامل می‌شود. بعد از کامل شدن P4، منابعی را که در اختیار داشت (دو تا s3 و یک s1) را آزاد می کند. سپس فرایند P2 کامل می‌شود، چون منابع درخواستی موجود است. بعد از P2 فرایند P1 و P3 کامل می‌شوند. یعنی هیچ فرایندی درحالت بن بست نمی‌باشد. ■

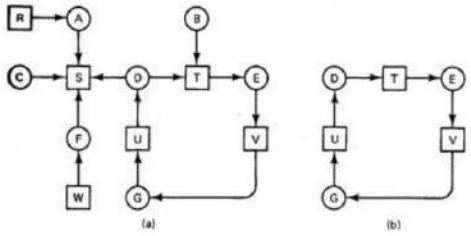
اگر هر نوع منبع دارای چند نمونه باشد، وجود چرخه الزاماً به معنای بن بست نمی‌باشد. یعنی در این حالت وجود چرخه شرط لازم برای وجود بن بست است ولی شرط کافی نیست.

**مثال**

سیستمی با 7 فرایند و 6 منبع مفروض است. با توجه به ۷ درخواست زیر، وضعیت را مشخص کنید؟

1. Process A holds R and wants S.
2. Process B holds nothing and wants T.
3. Process C holds nothing and wants S.
4. Process D holds U and wants S and T.
5. Process E holds T and wants V.
6. Process F holds W and wants S.
7. Process G holds V and wants U.

حل: گراف منابع زیر است:




مشخص می شود که حلقه DTEVGUD وجود دارد. بنابراین فرایندهای B, D,E,G در بن بست قرار دارند.

اما فرایندهای A,F,C در بن بست نمی باشند، چون منبع S می تواند به هر یک از آنها داده شود و بعد از پایان کار با این منبع، به فرایند دیگر داده شود. ■

**روش های رفع بن بست**

روشهای رفع بن بست را می توان به سه گروه عمده تقسیم کرد:

۱- **پیشگیری**(جلوگیری) **از بن بست**(Prevention)

سیستم طوری طراحی شود که از قبل امکان بن بست از بین برود، یعنی تضمین کنیم که حداقل یکی از چهار شرط بن بست رخ ندهد.

۲- **اجتناب از بن بست (Avoidance)**

در مورد درخواست‌های منبع طوری رفتار شود که حداقل یکی از چهار شرط بن بست رخ ندهد.

۳- **کشف بن بست** (Detection)

هرجا که ممکن باشد، منابع درخواستی به فرایندها داده می‌شود. سیستم عامل به طور متناوب الگوریتمی را دنبال می‌کند تا وجود شرط انتظار چرخشی را کشف کند.

**مثال**

مثالی از کاربرد الگوریتم کشف بن بست (Deadlock Detection):

سیستمی با ۳ فرایند و ۴ نوع منبع مفروض است. تعداد موجود از هر منبع در زیر آمده است:

Tape drives=4 , Plotters=2 , Printers=3 , CD ROM=1

که این موضوع به صورت A=(4 2 3 1) نشان داده می شود.

در این سیستم، فرایندها منابع زیر را در اختیار دارند:

فرایند اول = یک پرینتر، فرایند دوم= دو Tape و یک CD ، فرایند سوم= یک پلاتر و دو پرینتر

این موضوع با ماتریس زیر نشان داده می شود:

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

بعد از این تخصیص از هر یک از منابع به تعداد زیر باقی می ماند:

Tape drives =2 , Plotters=1 , Printers=0 , CD ROM =0

اما فرایندها برای اجرا نیاز به منابع دیگری دارند که در زیر مشخص شده است:

فرایند اول=دو Tape و یک CD، فرایند دوم=یک Tape و یک پرینتر،

فرایند سوم=دو Tapeویک پلاتر

که این موضوع با ماتریس زیر نشان داده می شود:

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

با توجه به اطلاعات بالا، آیا در این سیستم بن بست رخ می دهد؟

حل: فرایند اول نمی تواند اجرا شود، چون نیاز به یک CD دارد که موجود نیست. فرایند دوم نیز نمی تواند اجرا شود، چون نیاز به یک پرینتر دارد که موجود نیست. ولی فرایند سوم می تواند اجرا شود، چون به دو tape و یک پلاتر نیاز دارد که موجود است. بعد از اجرای فرایند سوم ، منابعی که در اختیار داشته(یک پلاتر و دو پرینتر)، آزاد شده و تعداد منابع آزاد به صورت زیر در می آید:

Tape drives =2 , Plotters=2 , Printers=2 , CD ROM =0

در این وضعیت، دو فرایند دیگر می توانند اجرا شوند و بنابراین بن بست رخ نمی دهد.

**ترمیم**

بعد از کشف بن بست، روشهای زیر برای ترمیم آن وجود دارد:

الف- قطع تمام فرایندهای بن بست.

ب- برگشت فرایندهای بن بست به نقاطی که از قبل برای بررسی تعریف شده و شروع مجدد تمام فرایندها.

ج- قطع پی در پی فرایندهای بن بست تاجایی که دیگر بن بست وجود نداشته باشد.

د- قبضه کردن پی درپی منابع تا جایی که دیگر بن بست وجود نداشته باشد.

### روش های پیشگیری از بن بست

روشهای پیشگیری از بن بست عبارتند از:

الف- غیر مستقیم (پیشگیری از بروز یکی از سه شرط لازم اول)

ب- مستقیم (پیشگیری از بروز شرط چهارم)

### روشهای پیشگیری برای هر یک از چهار شرط بن بست

#### ۱- نگهداری و انتظار

برای پیشگیری از رخ دادن این شرط، باید فرایند را مجبور کرد تا همه منابع مورد نیاز را یکباره درخواست کند و تا زمانی که همه منابع به او داده نشود، فرایند را مسدود کرد.

#### ۲- انحصاری بودن (قبضه نکردن)

برای پیشگیری از بروز این شرط دو راه پیشنهاد شده است:

الف- اگر فرایندی منابعی را در اختیار دارد، درخواست جدیدش موافقت نشود. در این حالت باید منابع قبلی خود را آزاد کند و در صورت نیاز، مجدداً با منابع اضافی درخواست نماید.

ب- اگر فرایندی منبعی را درخواست کند که در اختیار فرایند دیگر است، سیستم عامل آن فرایند را قبضه کرده و منابعش را آزاد کند.(البته فرایندها نباید اولویت یکسان داشته باشند.)

#### ۳- انحصار متقابل

منابع غیرقابل اشتراک باید انحصاری باشند و نمی‌توان این شرط را رد کرد. به طور نمونه یک چاپگر نمی‌تواند همزمان بین چند فرایند مشترک باشد. و یا به یک فایل نمی توان توسط چند فرایند جهت نوشتن دسترسی پیدا کرد.

#### ۴- انتظار چرخشی

برای پیشگیری از بروز این شرط، یک ترتیب خطی از انواع منابع تعریف می‌کنیم. اگر منبعی از نوع R به یک فرایند تخصیص یابد، در ادامه تنها منابعی را می‌تواند درخواست کند که نوع آنها بعد از R قرار گرفته باشد. مثلاً اگر $i<j$ آنگاه Ri قبل از Rj است. اگر فرایند P1 منبع Ri را در اختیار داشته باشد و Rj را تقاضا کند و فرایند P2 منبع Rj را در اختیار داشته باشد و Ri را تقاضا کند، این حالت غیر ممکن است چون مستلزم $i<j$ و $j<i$ می باشد.

پیشگیری از شرط انتظار چرخشی، ممکن است موجب کند کردن فرایندها و رد کردن غیر ضروری دسترسی به منابع گردد.

**روش های اجتناب از بن بست**

برای اجتناب از بن بست دو رویکرد وجود دارد:

۱- عدم شروع فرایندی که ممکن است درخواست هایش منجر به بن بست شود.

۲- عدم پاسخ به درخواست های منبع اضافی از طرف فرایندی که با این تخصیص ممکن است منجر به بن بست شود. (الگوریتم بانکداران)

در اجتناب از بن بست، تصمیم‌گیری زیر بصورت پویا انجام می‌شود:

" اگر منبع درخواست شده، تخصیص داده شود، آیا می‌تواند منجر به بن بست شود یا نه ".

یعنی اجتناب از بن بست نیازمند اطلاع از درخواستهای آینده است.

**محدودیت های اجتناب از بن بست**

۱- تعداد منابع تخصیص باید ثابت باشد.

۲- فرایندی که منبعی در اختیار دارد نمی تواند خارج شود.

۳- حداکثر منابع مورد نیاز هر فرایند باید از قبل معلوم شود.

۴- فرایندهای مورد نظر باید مستقل باشند.

- امتیاز اجتناب از بن بست این است که قبضه کردن فرایند، لازم نمی‌باشد.(بر خلاف کشف)
- حالت امن، حالتی است که در آن حداقل یک ترتیب از فرایندها وجود دارد که می‌توانند اجرا و کامل شوند، بدون اینکه بن بست رخ دهد.
- وضعیت بن بست، یک وضعیت ناامن است،اما تمام وضعیت های ناامن،الزما ،بن بست نیستند، چون ممکن است یک فرایند به حداکثر منابع اش نیاز پیدا کند و یا حتی بخشی از منابع تخصیص یافته اش را آزاد کند.

**الگوریتم بانکداران**

در این الگوریتم که برای اجتناب از بن بست استفاده می شود از بردارها (آرایه یک بعدی) و ماتریسهای (آرایه دو بعدی) زیر استفاده می شود:

۱- بردار Resource : شامل مقدار کل هر یک از منابع.

۲- بردار Available : شامل مقدار کل هر یک از منابعی که موجود است (تخصیص داده نشده)

۳- ماتریس Claim (حداکثر نیاز): شامل درخواستهای فرایندها.

۴- ماتریس Allocation : شامل منابع تخصیص داده شده به فرایندها.

**مثال ۵-۱۴**

در سیستم زیر با چهار فرایند و سه منبع آیا حالت امن وجود دارد؟ (Resource:R1=9,R2=3,R3=6)

| | $R_1$ | $R_2$ | $R_3$ |
|---|---|---|---|
| P1 | 3 | 2 | 2 |
| P2 | 6 | 1 | 3 |
| P3 | 3 | 1 | 4 |
| P4 | 4 | 2 | 2 |

**Claim**

| | $R_1$ | $R_2$ | $R_3$ |
|---|---|---|---|
| P1 | 1 | 0 | 0 |
| P2 | 6 | 1 | 2 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

**Allocation**

حل:

با توجه به ماتریس Allocation ، مشخص می شود که تعداد نه منبع R1 ، دو منبع R2 ، پنج منبع R3 تخصیص داده شده اند. بنابراین با توجه به منابع اولیه، منابع بعد از تخصیص به صورت (R1=0,R2=1,R3=1) می باشد. سپس ماتریس NEED را از تفاضل دو ماتریس Claim و Allocation بدست می آوریم:

| فرایند | $R_1$ | $R_2$ | $R_3$ |
|---|---|---|---|
| P1 | 2 | 2 | 2 |
| P2 | 0 | 0 | 1 |
| P3 | 1 | 0 | 3 |
| P4 | 4 | 2 | 0 |

حال با توجه به ماتریس NEED و تعداد منابع آزاد بعد از تخصیص، مشاهده می شود که اجرای فرایند P1 در حال حاضر ممکن نیست، چون هیچ منبع $R_1$ آزادی نداریم. ولی منابع مورد نیاز P2 موجود است. بعد از

اجرای P2، منابعی که در اختیار داشته آزاد شده و به Available اضافه می کند:

| | | |
|---|---|---|
| 3 | 2 | 2 |
| 0 | 0 | 0 |
| 3 | 1 | 4 |
| 4 | 2 | 2 |

**Claim**

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 2 | 1 | 1 |
| 0 | 0 | 2 |

**Allocation**

| | | |
|---|---|---|
| 6 | 2 | 3 |

**Available**



توجه شود که بعد از اجرای P2، سطر مربوط به فرایند P2 در هر دو ماتریس، صفر شدند. در این حالت P1 می‌تواند اجرا شود. بعد از اجرای P1،منابع در اختیار او به بردار Available اضافه می شود:

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 3 | 1 | 4 |
| 4 | 2 | 2 |

**Claim**

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 2 | 1 | 1 |
| 0 | 0 | 2 |

**Allocation**

| | | |
|---|---|---|
| 7 | 2 | 3 |

**Available**

و بعد از اجرای P3 داریم:

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 4 | 2 | 2 |

**Claim**

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 2 |

**Allocation**

| | | |
|---|---|---|
| 9 | 3 | 4 |

**Available**



در نهایت P4 اجرا می شود و بردار Availble مانند بردار Resorce خواهد شد.

بنابراین می توان ترتیب زیر را یک حالت امن سیستم نام برد:

$$P2 \rightarrow P1 \rightarrow P3 \rightarrow P4$$

هریک از روشهای برخورد با بن بست (پیشـگیری،اجتناب و کشـف) دارای مزایـا و معـایبی هسـتند. بنابراین بهتر است از روشهای متفاوتی در شرایط متفاوت استفاده کرد.

**خلاصه رویکردها**

در رابطه با سه رویکرد پیشگیری، کشف و اجتناب طرحهای مختلفی ارائه شده که در جدولهای زیر بررسی شده اند:

| اجتناب | | |
|---|---|---|
| طرح | مزایا | معایب |
| دستکاری برای یافتن حداقل یک مسیر امن | عدم نیاز به قبضه کردن | ضرورت اطلاع از منابع مورد نیاز آینده |
| | | امکان مسدود شدن طولانی فرایندها |

| کشف | | |
|---|---|---|
| طرح | مزایا | معایب |
| احضار دوره ای برای بررسی بن بست | عدم تاخیر در آغاز فرایند | ضررهای ذاتی قبضه کردن |
| | تسهیل پردازش در حین کار | |

| پیشگیری | | |
|---|---|---|
| طرح ها | مزایا | معایب |
| درخواست یکباره تمام منبع | در مورد فرایندهایی که فعالیت شایعی را انجام می دهند خوب کار می کند. | نا کارآمدی |
| | عدم نیاز به قبضه کردن | تاخیر شروع فرایند |
| | | |

| | | |
|---|---|---|
| قبضه کردن | سهولت بکارگیری منابعی که بتوان وضعیت آنها را به سادگی ذخیره کرد. | قبضه کردن بیش از تعداد لازم |
| | | در معرض شروع شدنهای مجدد مدور |
| | | |
| مرتب کردن منابع | امکان اعمال کنترلهای زمان ترجمه | قبضه کردن نه چندان مفید |
| | عدم نیاز به محاسبه در زمان اجرا ، به دلیل اینکه مساله در طرح سیستم حل شده است. | اجازه ندادن به درخواست منابع به صورت افزایشی |

# کنکور ارشد

**(مهندسی IT – دولتی ۸۸)**

۱- سیستمی با 3 فرایند و 2 فایل read-only را در نظر بگیرید. با فرض اینکه هر فرایند حداکثر به خواندن 2 فایل نیاز داشته باشد، تعداد وضعیت های بن بست (Deadlock) حداکثر برابر کدام است؟

۱) 0 ۲) 3 ۳) 4 ۴) 5

حل: گزینه ۱ درست است.

چون فایل فقط خواندنی است، شرط انحصار متقابل برای آن مطرح نبوده و هرگز بن بست رخ نمی دهد.

**(مهندسی کامپیوتر– آزاد ۸۹)**

۲- سیستمی شامل 4 فرایند همروند و 2 منبع یکسان قابل استفاده مجدد را در نظر بگیرید به شرط آن که هر فرآیند حداکثر به 2 منبع نیاز داشته باشد، تعداد وضعیت های بن بست (deadlock states) در این سیستم به خاطر منبع مذکور حداکثر چند حالت می باشد؟

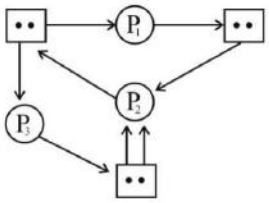
۱) 10 ۲) 5 ۳) 0 ۴) 6

حل: جواب گزینه ۴ است.

$$\binom{4}{2} = \frac{4!}{2! \times 2!} = 6$$

**(مهندسی کامپیوتر- آزاد ۸۴)**

۳- اگر گراف بن بست زیر نمایش لحظه‌ای درخواست‌های منابع و نیز تملک منابع توسط سه فرایند $P_1$ و $P_2$ و $P_3$ باشد و در صورتی که فقط با اتمام یک فرایند، منابع تملک شده توسط آن فرایند آزاد شوند، آن گاه ترتیب اتمام فرایندها از راست به چپ کدام است؟




۱) $P_1$ و $P_3$ و $P_2$ ۲) $P_1$ و $P_2$ و $P_3$

۳) $P_2$ و $P_1$ و $P_3$ ۴) هیچ کدام. زیرا سیستم در وضعیت بن بست قرار دارد.

حل: جواب گزینه ۲ است.

منبع سمت چپ بالا را $R_1$ و منبع سمت راست بالا را $R_2$ و منبع پایین را $R_3$ می نامیم. دو نمونه منبع $R_1$ در اختیار فرایندهای $P_1$ و $P_3$ می باشد. فرایند $P_2$ و $P_3$ نمی تواند ابتدا اجرا شوند، چون منبع درخواستی آنها آزاد نیست. اما $P_1$ یک نمونه از $R_2$ را که آزاد است گرفته و اجرا می شود. بعد از پایان اجرایش، منبع $R_1$ که در اختیار داشت را رها کرده و $P_2$ با گرفتن آن اجرا می شود. بعد از پایان اجرایش، منبع $R_3$ که در اختیارش بود را آزاد کرده و $P_3$ با گرفتن یک نمونه از $R_3$ اجرا می شود. بنابراین $P_1$ ، $P_2$ و در نهایت $P_3$ اجرا می شود.

■

**(مهندسی IT –دولتی ۸۷)**

۴- اگر در سیستم عاملی به هر منبع یک شماره اولویت منحصر به فردی اختصاص داده شود و از پردازش درخواست معینی با اولویت کمتر یا مساوی اولویت منبع hold شده توسط همان فرایند ممانعت به عمل آید، کدام یک از گزینه های زیر صحیح است؟

۱) این روش از بن بست جلوگیری می کند ولی احتمال گرسنگی وجود دارد.

۲) این روش موسوم به درخواست افزایش است و جهت پیشگیری از بن بست به کار می رود.

۳) این روش مبتنی بر کشف بن بست و بدین ترتیب عامل های بن بست تشخیص داده می شوند.

۴) این روش موسوم به درخواست افزایش است و به صورت دینامیکی از بن بست اجتناب می کند.

حل: جواب گزینه ۲ است.

**(مهندسی IT –دولتی ۸۴)**

۵- برای پیشگیری از بن بست از طریق شرط انتظار چرخشی، روش مبتنی بر شماره گذاری همه منابع و الزام فرایندها به درخواست منابع به ترتیب عددی (مثلا صعودی)، پیشنهاد شده است. در پیاده سازی آن با چه مشکلی روبرو خواهیم شد؟

۱) منابع سیستم را هدر می دهد.

۲) نیاز به پیش بینی آینده دارد.

۳) همه منابع قابل Spool نیست و خود فضای Spool بر روی دیسک نیز موجب بن بست می شود.

۴) هر دو مورد ۱و۲ صحیح است.

حل: جواب گزینه ۴ است.

**(مهندسی IT – دولتی ۸۴)**

۶- تحت سیستم عاملی 4 فرایند فعال و 2 منبع مدیریت می شود. وضعیت سیستم تحت جدول ذیل بیان شده است. حالت سیستم چیست؟ منابع در دسترس عبارتند از: $(R_1 = 2, R_2 = 1)$

| | $R_1$ | $R_2$ |
|---|---|---|
| $P_1$ | 9 | 5 |
| $P_2$ | 2 | 6 |
| $P_3$ | 2 | 2 |
| $P_4$ | 5 | 0 |

حداکثر نیاز

| | $R_1$ | $R_2$ |
|---|---|---|
| $P_1$ | 7 | 2 |
| $P_2$ | 1 | 3 |
| $P_3$ | 1 | 1 |
| $P_4$ | 3 | 0 |

مقادیر اختصاص داده شده

۱) امن ۲) نا امن

۳) به طور قاطع نمی توان پیش بینی کرد. ۴) بستگی دارد چه فرایندی چه منبعی را تقاضا کند.

حل: جواب گزینه ۲ است.

با تفاضل دو ماتریس داده شده، ماتریس Need به صورت زیر مشخص می شود:

| | $R_1$ | $R_2$ |
|---|---|---|
| $P_1$ | 2 | 3 |
| $P_2$ | 1 | 3 |
| $P_3$ | 1 | 1 |
| $P_4$ | 2 | 0 |

با توجه به ماتریس need و منابع در دسترس یعنی $(R_1 = 2, R_2 = 1)$، مشخص است که $P_1$ یا $P_2$ نمی توانند اجرا شوند، چون به سه $R_2$ نیاز دارند در حالی که فقط یکی موجود است. اما $P_3$ می تواند اجرا شود. بعد از اجرای $P_3$، منابعی که در اختیار داشته را آزاد می کند و منابع در دسترس $(R_1 = 3, R_2 = 2)$ می شوند. بعد از اجرای $P_3$ ، باز هم $P_1$ یا $P_2$ نمی توانند اجرا شوند، چون باز هم سه $R_2$ نداریم. بنابراین $P_4$ را اجرا می کنیم. بعد از اجرای $P_4$، منابع در دسترس $(R_1 = 6, R_2 = 2)$ خواهد شد، باز هم نمی توان $P_1$ یا $P_2$ را اجرا کرد. در نتیجه سیستم در وضعیت نا امن قرار دارد.■

**(مهندسی IT – آزاد ۸۹)**

۷- وضعیت زیر یک وضعیت.....بردار منابع: (A=5,B=5,C=2,D=4)

| | A | B | C | D |
|---|---|---|---|---|
| P1 | 2 | 1 | 1 | 4 |
| P2 | 0 | 0 | 0 | 2 |
| P3 | 1 | 1 | 1 | 1 |
| P4 | 2 | 0 | 0 | 2 |
| P5 | 0 | 1 | 0 | 0 |

ماتریس تخصیص

| | A | B | C | D |
|---|---|---|---|---|
| P1 | 4 | 4 | 2 | 5 |
| P2 | 2 | 0 | 0 | 2 |
| P3 | 1 | 1 | 1 | 2 |
| P4 | 3 | 1 | 0 | 3 |
| P5 | 1 | 1 | 0 | 1 |

ماتریس حداکثر نیاز

۱) امن است. ۲) بن بست است. ۳) نا امن است. ۴) گرسنگی است.

حل: جواب گزینه ۱ است.

با تفاضل دو ماتریس داده شده، ماتریس Need به صورت زیر مشخص می شود:

| | A | B | C | D |
|---|---|---|---|---|
| P1 | 2 | 3 | 1 | 1 |
| P2 | 2 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 1 |
| P4 | 1 | 1 | 0 | 1 |
| P5 | 0 | 0 | 0 | 1 |

با توجه به ماتریس need و منابع در دسترس یعنی $(A=5,B=5,C=2,D=4)$، مشخص است که تعداد منابع به اندازه ای است که می توان به هر یک از فرایندها به هر ترتیبی اختصاص داد. در نتیجه سیستم امن است.

یکی از این ترتیب ها:

۱- P1 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$(A=7,B=6,C=3,D=8)$

۲- P2 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$(A=7,B=6,C=3,D=10)$

۳- P3 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$(A=8,B=7,C=4,D=11)$

۴- P4 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$(A=10,B=7,C=4,D=13)$

۵- P5 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$(A=10,B=8,C=4,D=13)$

■

# فصل ۶

# مدیریت حافظه

حافظه،آرایه بزرگی از کلمات یا بایت ها است. یکی از وظایف سیستم عامل، مدیریت حافظه است. مدیریت حافظه، در سیستم چند برنامه ای باید بسیار کارآمد باشد و حافظه به نحوی باید تخصیص یابد که فرایندهای بیشتری در آن قرار بگیرند.

انواع حافظه عبارتند از: ثبات، حافظه نهان، حافظه اصلی، حافظه دیسک که در فصل اول بررسی شدند. مدیریت ثبات بر عهده کامپایلر، مدیریت حافظه نهان، سخت افزاری، مدیریت حافظه اصلی و مدیریت حافظه دیسک بر عهده سیستم عامل است.حافظه هایی که مدیریت آنها بر عهده سیستم عامل است را در این فصل و فصل های بعد بررسی خواهیم کرد.

### مدیریت حافظه ابتدایی

از آنجا که در بعضی سیستم های امروزی، از طرح های ساده مدیریت حافظه، استفاده می شود، این طرح ها را مطالعه می کنیم.

### تک برنامگی ساده

ساده ترین طرح مدیریت حافظه این است که در هر لحظه، فقط یک برنامه در حال اجرا باشد. امروزه از طرح تک برنامگی بیشتر در سیستم های توکار استفاده می شود.

سه مدل سازماندهی حافظه شامل یک سیستم عامل و یک فرایند کاربر عبارتند از:

۱- سیستم عامل در پایین حافظه و در RAM باشد.

۲- سیستم عامل در بالای حافظه و در ROM باشد.

۳- سیستم عامل در پایین حافظه و در RAM باشد. گرداننده های دستگاه در بالای حافظه و در ROM باشند. (در MS-DOS از این مدل استفاده می شود.)

- از معایب مدیریت حافظه به روش تک برنامگی، اتلاف شدید حافظه و سایر منابع است. همچنین امکان اجرای فرایند بزرگتر از حافظه اصلی وجود ندارد.

**چند برنامگی با پارتیشن ثابت و بدون مبادله**

برای اجرای چندبرنامگی می توان حافظه را به n پارتیشن تقسیم کرد. اندازه این پارتیشن ها می تواند یکسان نباشد.وقتی کاری وارد می شود، در یک صف ورودی قرار می گیرد تا در کوچکترین پارتیشنی که در آن جا می شود، قرار داده شود. مقداری از پارتیشن که توسط کار درون آن استفاده نمی شود، هدر می رود که به آن حفره (Hole) می گویند.

در یک سیستم با پارتیشن ثابت، هر پارتیشن می تواند، صف مربوط به خود را داشته باشد و یا یک صف مشترک برای همه پارتیشن ها در نظر گرفت. در حالت اول، ممکن است صف مربوط به یک پارتیشن بزرگ خالی باشد، در صورتی که صف مربوط به یک پارتیشن کوچک پر شده باشد.

مدیریت حافظه به روش پارتیشن بندی ایستا، امکان ایجاد چندبرنامگی با یک تکنیک ساده را فراهم می کند و در مقایسه با تک برنامگی از حافظه و پردازنده بهتر استفاده می شود. اما دارای معایب زیر است:

۱- تعیین تعداد و اندازه پارتیشن ها، مشکل است.

۲- درجه چند برنامگی به تعداد پارتیشن ها محدود است.

۳- نمی توان فرایند بزرگتر از بزرگترین پارتیشن را اجرا کرد.

۴- نمی توان یک فرایند را دو تکه کرد و در دو پارتیشن مجزا قرار داد.

۵- اتلاف حافظه به علت بارگذاری قسمت هایی از یک فرایند بزرگ که فعلا به آنها نیاز نیست.

۶- تکه تکه شدن داخلی

تذکر: به این علت به این تکه تکه شدن، داخلی می گویند که از حفره **داخل** فضای تخصیص یافته به فرایند ناشی می شود.

**جا به جایی و حفاظت**

در چندبرنامگی دو مشکل جا به جایی و حفاظت رخ می دهد که باید حل شوند.

**جا به جایی(Relocation)**

وقتی که برنامه اصلی با زیر برنامه های نوشته شده توسط کاربر و رویه های کتابخانه ای در یک فضای آدرس ترکیب می شوند، مشکلی که به وجود می آید این است که پیوند دهنده از کجا باید بداند که آن برنامه در زمان اجرا از چه آدرسی در حافظه شروع می شود. برای حل این مشکل، سیستم عامل و سخت افزار باید بتوانند مراجعات به حافظه را که در کد برنامه مطرح می شوند را به آدرس های فیزیکی که منعکس کننده مکان فعلی برنامه در حافظه اصلی هستند، تبدیل کند. یک راه حل نرم افزاری این است که

در همان زمان لود برنامه در حافظه، دستورات برنامه دستکاری شده و آدرس ها اصلاح شوند.

**حفاظت (Protection)**

در سیستم های چند کاربره، از حافظه متعلق به برنامه یک کاربر، در مقابل خواندن یا نوشتن برنامه کاربران دیگر باید حفاظت کرد.

یک راه حل:

حافظه به بلوک های دو کیلو بایتی که هر بلوک شامل یک کد حفاظتی چهار بیتی است، تقسیم شود. PSW نیز شامل یک کلید 4 بیتی است. همچنین فقط سیستم عامل قادر به تغییر کلید و کد حفاظتی باشد. در این صورت هر دستور فرایند در حال اجرا، که منجر به دسترسی به بلوکی از حافظه شود که کد حفاظتی آن بلوک با کلید درون PSW یکسان نباشد، متوقف شود.

**راه حل سخت افزاری دو مشکل حفاظت و جا به جایی**

می توان از دو رجیستر پایه(Base) و حد(Limit) واقع در MMU برای حل این دو مشکل استفاده کرد. هنگامی که پردازنده به فرایندی داده می شود، آدرس شروع پارتیشن در Base و طول پارتیشن در Limit قرار می گیرد. برای جلوگیری از مشکل جا به جایی، هر آدرس حافظه ای که تولید می شود، قبل از ارسال به حافظه، مقدارش به صورت خودکار با محتوای Base جمع می شود. برای مشکل حفاظت، آدرس ها با محتوای Limit مقایسه می شوند تا به فضای خارج از پارتیشن دسترسی نشود.

اگر ثبات پایه حاوی 100 و ثبات حد شامل 20 باشد، آن گاه برنامه می تواند به آدرسهایی از 100 تا خود 120 ، دستیابی داشته باشد.

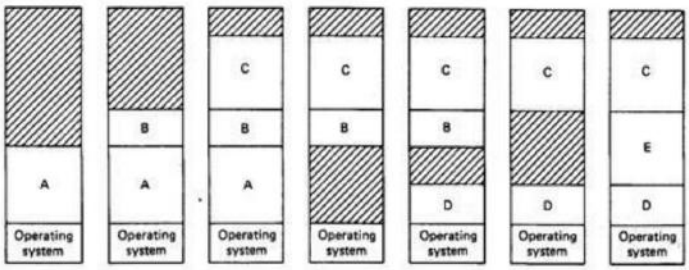
**مبادله (swapping)**

در سیستم های اشتراک زمانی در گاهی اوقات نمی توان همه فرایندهای فعال را در حافظه اصلی جای داد و باید فرایندهای اضافی به دیسک منتقل شده و بعدا به صورت پویا به داخل حافظه آورده شوند. مبادله (پارتیشن بندی پویا)، ساده ترین روشی است که اجازه مبادله بین حافظه و دیسک را می دهد. در این روش هر فرایند به طور کامل به حافظه اصلی آورده می شود(Swap in) و بعد از مدتی اجرا، به دیسک برگردانده می شود(Swap out).

مثلا در یک محیط چند برنامه ای که از الگوریتم زمانبندی RR استفاده می کند، وقتی هر فرایندی کوانتوم

زمانی خودش را تمام می کند، با فرایند دیگری مبادله می شود.

شکل زیر نحوه عملکرد سیستمی را نشان می دهد که بر اساس مبادله کار می کند. در ابتدا فقط فرایند A در حافظه است و سپس فرایندهای B و C، وارد حافظه شده اند. سپس فرایند A ، خارج شده و بعد D وارد شده و سپس B خارج شده است. در انتها، فرایند E وارد شده است.




اگر انقیاد در زمان اسمبل یا بار کردن باشد، وقتی فرایندی از حافظه بیرون رفت، هنگام برگشت به حافظه، در همان فضای قبلی بر می گردد. اگر انقیاد در زمان اجرا صورت گیرد، فرایند در محلی غیر از محل اول قرار می گیرد، زیر آدرسهای فیزیکی در زمان اجرا محاسبه می شوند.

- در پارتیشن بندی پویا، تعداد، موقعیت و اندازه پارتیشن ها متفاوت است. این انعطاف پذیری باعث بهبود بهره وری حافظه می شود.

- تخصیص و آزاد سازی حافظه در پارتیشن بندی پویا، پیچیده تر از پارتیشن بندی ایستا است.
- در پارتیشن بندی پویا، به خاطر مبادله، حفره های متعددی در حافظه به وجود می آید که باعث در رفتن حافظه می شود. به این مشکل،تکه تکه شدن خارجی می گویند. علت اینکه به آن خارجی می گویند این است که تکه تکه شدن از حفه خارج فضای تخصیص یافته به فرایندها ناشی می شود.

- برای مقابله با تکه تکه شدن خارجی از فشرده سازی (Compaction) می‌توان استفاده کرد. در این روش، حفره های کوچک با یکدیگر ادغام می شوند تا یک بلوک بزرگ از حافظه را ایجاد کنند. ساده ترین الگوریتم فشرده سازی این است که تمام حفره ها را به یک طرف برده و حفره بزرگی از حافظه آزاد تشکیل می شود. ولی این روش هزینه زیادی دارد.

وقتی که حافظه به صورت پویا تخصیص داده می شود، سیستم عامل باید بداند که کدام بخش حافظه در هر لحظه، تخصیص داده شده و کدام بخش آزاد است. برای این منظور، دو روش وجود دارد:

### ۱- مدیریت حافظه با نگاشت های بیتی

نگاشت بیتی(Bitmap) راه حلی را ارائه می دهد تا بتوان به سادگی و با استفاده از مقدار ثابتی از حافظه، وضعیت پر و خالی بودن کلمات حافظه را تحت نظر داشت. در این روش، حافظه به چندین واحد تخصیص تقسیم می شود. متناظر با هر واحد تخصیص،یک بیت وجود دارد. اگر واحد متناظر آزاد باشد، این بیت 0 است و در صورت پر بودن، 1 است.

**مثال**

سیستمی از الگوریتم مدیریت حافظه نگاشت بیتی (Bitmap) به صورت زیر استفاده می‌کند:

1100101100000111

هر بیت نشان دهنده فضای پر (1) یا خالی (0) به ازاء هر 4KB حافظه اصلی است. لیست حفره ها به چه صورت است؟

پاسخ:

با نگاه به bitmap داده شده از چپ به راست، سه دسته صفر داریم :

1 100101 1, 00000 111

۱- دسته اول: دو صفر (8KB)

۲- دسته دوم: یک صفر (4KB)

۳- دسته سوم: پنج صفر (20KB)

بنابراین لیست حفره ها برابر است با: 8,4,20 .

- هر چه واحد تخصیص کوچکتر باشد، نگاشت بیتی بزرگتر خواهد بود. در مقابل، اگر واحد تخصیص بزرگ انتخاب شود، نگاشت بیتی کوچکتر خواهد بود.

- اگر اندازه فرایند، ضریبی از اندازه واحد تخصیص نباشد، در آخرین واحد مقداری از حافظه هدر می رود.

- برای انتقال یک فرایند به حافظه که به k واحد فضا نیاز دارد، مدیر حافظه باید در نگاشت بیتی جستجو کرده و k بیت متوالی 0 را پیدا کند. این عمل بسیار کند است و باعث مخالفت از طرح

Bitmap می باشد.

**۲- مدیریت حافظه با لیست های پیوندی**

در این روش،یک لیست پیوندی از قطعه های آزاد یا تخصیص یافته حافظه تشکیل می شود. مزیت این روش این است که زمانی که فرایندی خاتمه می یابد، یا مبادله می شود، این لیست به سادگی update می شود. فرایندی که پایان یافته با همسایه های خود ترکیب می شود. فیلدهای هر گره در لیست پیوندی مثال قبل شامل فیلدهای زیر است:

۱- حفره (H) یا فرایند(P) بودن

۲- آدرس شروع

۳- طول

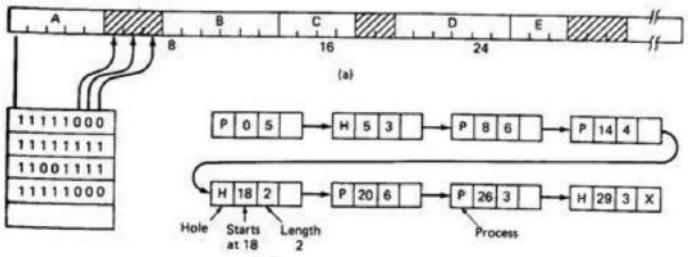
۴- آدرس گره بعدی

**مثال**

شکل زیر قسمتی از حافظه شامل 5 فرایند و 3 حفره را نشان می دهـد. منـاطق هاشـور خـورده، فضـاهای خالی هستند. این حافظه به دو روش Bitmap و لیست پیوندی نشان داده شده است.

در طرح Bitmap فضاهای خالی ، با صفر و فضاهای پر با 1 مشخص شده انـد. بـه طـور مثـال، 8 بیـت اول ، شامل پنج تا 1 و سه تا 0 است، چون پنج واحد تخصیص توسط فرایند A پر شده و بعد از آن ۳ واحد خالی می باشد.

در طرح لیسست پیوندی، فضاهای خالی با H و فضاهای پر با P مشخص شده اند.



**الگوریتم های مکان یابی و تخصیص حافظه**

وقتی فرایندها و حفره ها در یک لیست مرتب شده بر اساس آدرس قرار می گیرند، الگوریتم های مختلفی جهت تخصیص حافظه به یک فرایند وجود دارد. چند مورد از این الگوریتم ها عبارتند از:

**۱- اولین برازش (First fit)**

جستجو از ابتدای حافظه شروع شده و فرایند در اولین حفره ای قرار داده می شود که در آن جا می شود.

**۲- برازش بعدی(Next fit)**

مانند First fit است، با این تفاوت که جستجو از آخرین محل تخصیص شروع می شود.

**۳- بهترین برازش (Best fit)**

تمام لیست جستجو می شود و فرایند در کوچکترین حفره ای قرار داده می شود که در آن جا می شود.

**۴- بدترین برازش (Worst fit)**

تمام لیست جستجو می شود و فرایند در بزرگترین حفره ای قرار داده می شود که در آن جا می شود.

- الگوریتم First fit ، ساده ترین روش و معمولاً سریع ترین الگوریتم است.
- الگوریتم First fit ، بیشترین کارایی را دارد. یعنی بهره وری حافظه در آن قابل قبول است.
- در Best fit ، فضاهای حافظه پر از تکه های کوچک خالی بی مصرف می شود.
- سرعت الگوریتم های Best fit و Worst fit پایین است، چون کل لیست باید جستجو شود.
- در Next fit ، حفره های بزرگ انتهای حافظه سریع تر شکسته می شود و در ورود فرایندهای بزرگ بعدی، مشکل ایجاد می شود.
- اگر برای فضاهای خالی و فرایندها، لیست های جداگانه ای داشته باشیم، سرعت هر چهار الگوریتم ، بیشتر می شود. البته این کار باعث پیچیده شدن مکانیسم آزاد کردن حافظه می شود.
- اگر لیست فضاهای خالی بر اساس اندازه فضاها مرتب باشد، سرعت Best fit ، افزایش می یابد.

**مثال**

در یک سیستم که مدیریت حافظه با استفاده از مبادله انجام می‌شود، حافظه اصلی شامل فضاهای خالی با اندازه‌های به ترتیب 12 , 9 , 7 , 18 , 20 , 4 , 10 (چپ به راست) است. برای درخواست تکه هایی از حافظه به طور متوالی و به مقدارهای 12 و10 و 6 کیلو بایت با استفاده از روش های First fit ، Next fit ، Best fit و Worst fit کدام یک از فضاهای خالی فوق الذکر اشغال می‌شوند؟

**پاسخ: روش First fit**

تکه 12 از حفره 20 گرفته می‌شود و 8 کیلو باقی می ماند و تکه 10 از حفره 10 و تکه 6 از حفره 8 (باقیمانده از ۲۰) گرفته می‌شود:

| | 10 | 4 | 20 | 18 | 7 | 9 | 12 |
|---|---|---|---|---|---|---|---|
| 12 | 10 | 4 | **8** | 18 | 7 | 9 | 12 |
| 10 | 0 | 4 | 8 | 18 | 7 | 9 | 12 |
| 6 | 0 | 4 | 2 | 18 | 7 | 9 | 12 |

**روش Next fit:**

| | 10 | 4 | 20 | 18 | 7 | 9 | 12 |
|---|---|---|---|---|---|---|---|
| 12 | 10 | 4 | **8** | 18 | 7 | 9 | 12 |
| 10 | 10 | 4 | 8 | **8** | 7 | 9 | 12 |
| 6 | 10 | 4 | 8 | **2** | 7 | 9 | 12 |

توجه کنید که فرایند 6KB از همان فضای باقی مانده از حفره مرحله قبل استفاده کرد.

**روش Best fit:**

| | 10 | 4 | 20 | 18 | 7 | 9 | 12 |
|---|---|---|---|---|---|---|---|
| 12 | 10 | 4 | 20 | 18 | 7 | 9 | 0 |
| 10 | 0 | 4 | 20 | 18 | 7 | 9 | 0 |
| 6 | 0 | 4 | 20 | 18 | 1 | 9 | 0 |

**روش Worst fit:**

| | 10 | 4 | 20 | 18 | 7 | 9 | 12 |
|---|---|---|---|---|---|---|---|
| 12 | 10 | 4 | 8 | 18 | 7 | 9 | 12 |
| 10 | 10 | 4 | 8 | 8 | 7 | 9 | 12 |
| 6 | 10 | 4 | 8 | 8 | 7 | 9 | 6 |

**برازش سریع (Quick fit)**

در این روش برای هر دسته از فرایندها با اندازه های متداول، یک لیست جداگانه تهیه می شود. جدولی با n خانه را فرض کنید که خانه اول این جدول،اشاره گری است که به ابتدای لیست حفره های 4 کیلو بایتی، خانه دوم، به ابتدای حفره های 8 کیلو بایتی و ... اشاره می کند.

- در این روش یافتن حفره ای با اندازه مناسب، بسیار سریع است.
- در این روش حافظه، تکه تکه شده و پر از حفره های کوچک و غیر قابل استفاده می شود.

## مدیریت حافظه با سیستم رفاقتی

سیستم رفاقتی، یک تعادل قابل قبول برای فائق آمدن بر معایب طرحهای بخش بندی ایستا و پویا است. در بخش بندی ایستا تعداد فرایندهای فعال محدود است و امکان تکه تکه شدن داخلی وجود دارد و در بخش بندی پویا سربار فشرده سازی وجود دارد. در یک سیستم رفاقتی، اندازه بلاکهای حافظه توانی از 2 می‌باشند.

نحوه کار: اگر اندازه یک حفره برابر $2^k$ باشد و فرایندی به اندازه S باید به داخل آن مبادله شود، اگر S از نصف اندازه حفره بزرگتر بود، کل فضا به آن داده می شود، در غیر اینصورت،کل بلوک نصف شده و دو بلوک رفیق ایجاد می شود. این روند به صورت بازگشتی تکرار خواهد شد. در صورت آزاد شدن یک حفره،امکان ترکیب رفقای مجاور می باشد.

### مثال

شکل زیر نحوه عملکرد سیستم رفاقتی را نشان می دهد:

| | | | | | | |
|---|---|---|---|---|---|---|
| | 1024 | | | | | |
| | 512 | | | | | 512 |
| | 256 | | | 256 | | 512 |
| Request 70 | A | 128 | | 256 | | 512 |
| Request 35 | A | B | 64 | 256 | | 512 |
| Request 80 | A | B | 64 | C | 128 | 512 |
| Return A | 128 | B | 64 | C | 128 | 512 |
| Request 60 | 128 | B | D | C | 128 | 512 |
| Return B | 128 | 64 | D | C | 128 | 512 |
| Return D | 256 | | | C | 128 | 512 |
| Return C | 1024 | | | | | |

ابتدا بلوک 1024 K به دو بلوک 512K تقسیم شده و سپس بلوک اول به دو بلوک 256K و بلوک اول آن به دو بلوک 128K تقسیم می شود. که اولین درخواست(A) یعنی 70K در آن قرار می گیرد. برای پاسخ به درخواست بعدی(B) یعنی 35 K ، حفره 128 K به دو حفره 64 K تقسیم شده و B در حفره اولی قرار می گیرد. درخواست بعدی (C) یعنی 80 K ، در نیمه اول حفره 256 K قرار می گیرد. در این لحظه A آزاد شده و بعد به درخواست (D) یعنی 60K پاسخ داده می شود. با آزاد سازی B ، و بعد از آن D، سه فضای خالی کنار هم ادغام شده و یک حفره 256K را ایجاد می کنند. در نهایت بعد از آزاد سازی C، بلوک یکپارچه می شود. ■

- مزیت سیستم رفاقتی این است که چیدمان حفره ها ساخت یافته است و می توان یک ساختار درختی برای بلوک های پر و خالی ایجاد کرد و با یک الگوریتم بازگشتی، حفره مناسب را خیلی سریع پیدا کرد.

- از معایب سیستم رفاقتی، اتلاف حافظه در این روش است. زمانی که یک فرایند 70 کیلو بایتی در یک حفره 128

کیلو بایتی قرار می گیرد، فضایی به اندازه 58 کیلو بایت هدر می رود.

### روی هم گذاری (Overlay)

در گذشته، اگر حافظه تخصیص داده شده به یک فرایند از اندازه فرایند کوچکتر بود، از تکنیک Overlay استفاده می شد. در این تکنیک برنامه نویس، برنامه ها را به قسمت هایی به نام Overlay تقسیم می کند. ابتدا Overlay0 اجرا شده و بعد از پایان اجرایش، Overlay1 را صدا می زند و به همین ترتیب ادامه می یابد. در سیستم هایی که امکان نگهداری چندین Overlay به طور همزمان در حافظه وجود داشت، سیستم عامل Overlay ها را روی دیسک نگه می داشت و در مواقع لزوم، عملیات مبادله را انجام می داد. تقسیم برنامه به overlay ها و فراخوانی آنها، بر عهده برنامه نویس بود و سیستم عامل فقط عمل مبادله را انجام می دهد.

### صفحه بندی (Paging)

در این روش مدیریت حافظه، حافظه اصلی به بلوکهایی با اندازه های ثابت به نام قاب (frame) تقسیم می شود. حافظه منطقی نیز به بلوک هایی با اندازه های یکسان به نام صفحه (page) تقسیم می‌شود. وقتی یک فرایند به داخل حافظه آورده می‌شود، تمام صفحات آن به داخل قابهای موجود بار می‌شوند و یک جدول صفحه ایجاد می‌شود. جدول صفحه، محل قاب هر صفحه از فرایند را مشخص می‌کند.

**مثال**

شکل زیر تخصیص قاب های آزاد به صفحه‌های فرایند A با سه صفحه را نشان می‌دهد. جدول صفحه را مشخص نمایید.

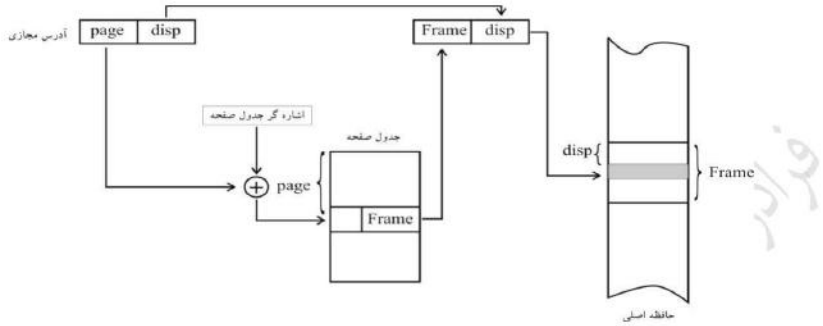
| | |
|---|---|
| | ⋮ |
| 5 | A.0 |
| 6 | |
| 7 | A.2 |
| 8 | A.1 |
| 9 | |

**پاسخ:** جدول صفحه فرایند A با سه صفحه به صورت زیر است:

| | |
|---|---|
| 0 | 5 |
| 1 | 8 |
| 2 | 7 |

- آدرس منطقی در صفحه بندی از دو قسمت تشکیل می‌شود: شماره صفحه (P) و آفست (d).
- به آفست، انحراف یا تفاوت مکان نیز می گویند.
- اندازه صفحه و اندازه قاب توسط سخت افزار تعریف می شود.
- برای ساده شدن طرح صفحه بندی، اندازه صفحه و اندازه قاب باید توانی از ۲ باشند. در این حالت آدرس نسبی و منطقی یکسان هستند.
- پردازنده به کمک جدول صفحه، یک آدرس فیزیکی تولید می‌کند که این آدرس از دو قسمت شماره قاب و آفست تشکیل شده است.

شکل زیر نحوه ترجمه آدرس در سیستم صفحه بندی را نشان می‌دهد:

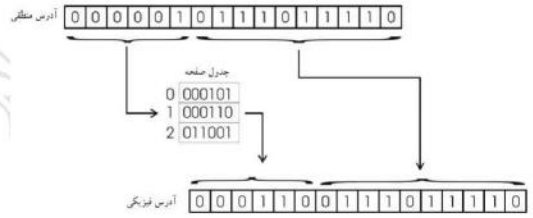



نحوه تولید آدرس فیزیکی با توجه به آدرس منطقی در شکال بالا: شماره صفحه موجود در آدرس مجازی(page) با آدرس شروع جدول صفحه، جمع شده و به آدرس بدست آمده، در جدول صفحه مراجعه می شود. در این آدرس شماره قاب (frame) متناظر با صفحه مورد نظر قرار دارد. این شماره قاب و همان آفست(disp) موجود در آدرس مجازی، آدرس فیزیکی را مشخص می کنند.

**مثال**

با توجه به آدرس نسبی 16 بیتی 0000010111011110 ، چند بیت برای شماره صفحه نیاز است؟(اندازه صفحه برابر یک کیلو بایت می باشد.)

پاسخ: چون اندازه صفحه یک کیلو بایت ($2^{10}$ بایت) است، 10 بیت برای آفست مورد نیاز است و چون آدرس 16 بیتی است، 6 بیت برای شماره صفحه باقی می‌ماند. آدرس نسبی داده شده دارای آفست (0111011110) در صفحه (000001) است. شکل زیر نحوه تولید آدرس فیزیکی از آدرس منطقی را نشان می دهد:




**تذکر**: یک برنامه می‌تواند حداکثر $2^6 = 64$ صفحه یک کیلوبایتی داشته باشد.■

**مثال**

یک فضای آدرس دهی منطقی شامل 4 صفحه است و هر صفحه حاوی 2 کلمه است. اگر این صفحات بر روی یک فضای آدرس دهی فیزیکی حاوی 8 قاب صفحه نگاشت شود، آدرس منطقی و فیزیکی چند بیتی خواهد بود؟

$4 \times 2 = 8 = 2^3$ = فضای آدرس دهی منطقی

$8 \times 2 = 16 = 2^4$ = فضای آدرس دهی فیزیکی

بنابراین آدرس منطقی 3 بیتی و آدرس فیزیکی 4 بیتی است.■

**مثال**

در یک سیستم حافظه صفحه بندی با یک جدول صفحه حاوی 64 مدخل 10 بیتی و صفحه های 512 بایتی، آدرس منطقی و فیزیکی چند بیتی است؟

اندازه حافظه منطقی برابر است با حاصل ضرب تعداد صفحات (تعداد مدخل ها)در اندازه هر صفحه:

$64 \times 512 = 2^6 \times 2^9 = 2^{15}$ بایت =اندازه حافظه منطقی

اندازه حافظه فیزیکی برابر است با حاصل ضرب تعداد آدرسها در اندازه هر صفحه:

$2^{10} \times 512 = 2^{19}$ بایت = اندازه حافظه فیزیکی

بنابراین آدرس منطقی 15 بیتی و آدرس فیزیکی 19 بیتی است. ■

**مثال**

در یک فضای آدرس دهی منطقی هر صفحه حاوی 512 کلمه است. این تعداد صفحات حافظه اصلی فضای آدرس دهی بر روی یک فضای آدرس دهی فیزیکی حاوی 8 قاب صفحه نگاشته می شود. آدرس فیزیکی حاوی چند بیت است؟

$8 \times 512 = 2^3 \times 2^9 = 2^{12}$

■

نقطه ضعف اصلی صفحه بندی **تکه تکه شدن داخلی** است. (اگر احتیاج به ناحیه بسیار کوچکی از حافظه باشد، در این صورت مقداری از فضای حافظه تلف می شود، زیرا کوچکترین واحدی از حافظه که به استفاده کننده اختصاص دارد یک صفحه است.)

## حافظه مجازی

حافظه مجازی تکنیکی است که موجب می شود فرایند بدون اینکه کاملا در حافظه باشد اجرا گردد. با استفاده از این تکنیک، می توان برنامه ای بزرگتر از حافظه فیزیکی را اجرا کرد. حافظه مجازی چندبرنامگی را به صورت موثری ممکن می‌سازد و کاربر را از محدودیتهای حافظه اصلی رها می‌کند. به کمک حافظه مجازی دیگر لزومی ندارد تمام صفحه‌ها(یا قطعه‌های) یک فرایند در حال اجرا، در حافظه اصلی قرار داشته باشند و در ابتدا یک یا چند تکه حاوی آغاز برنامه، توسط سیستم عامل به حافظه آورده می‌شود.

حافظه مجازی، حافظه منطقی کاربر را از حافظه فیزیکی تفکیک می کند. این تفکیک موجب می شود در حالتی که حافظه فیزیکی کم است، حافظه مجازی بزرگی برای برنامه نویس فراهم شود.

حافظه مجازی، برنامه نویسی را ساده می کند، زیرا برنامه نویس نگران حافظه فیزیکی و کد جایگذاری نمی باشد.

اگر در حین اجرا به آدرسی رجوع شود که در مجموعه مقیم در حافظه نباشد، یک وقفه ایجاد می‌شود و این فرایند مسدود می‌شود و تکه مورد نظر وارد حافظه می‌شود، سپس کنترل دوباره به سیستم عامل بر می‌گردد و فرایند مسدود را به حالت آماده برمی گرداند.

**تذکر**: به بخشی از فرایند که واقعاً داخل حافظه اصلی قرار دارد، مجموعه مقیم می‌گویند.

عناصر لازم برای موثر بودن حافظه مجازی عبارتند از:

۱- حمایت سخت افزاری برای به کارگیری صفحه بندی (یا قطعه بندی)

۲- حمایت نرم افزاری برای انتقال صفحه‌ها (یا قطعه‌ها) بین حافظه ثانوی و حافظه اصلی.

حافظه مجازی را می توان به روشهای زیر پیاده سازی کرد:

۱- صفحه بندی درخواستی (Demand Paging)

۲- قطعه بندی درخواستی (Demand Segmentation)

۳- قطعه بندی صفحه بندی شده (Hybrid Paging and Segmentation)

روش اول در این فصل و روش دوم و سوم در فصل بعد بررسی می شوند.

## صفحه بندی درخواستی

در صفحه بندی برای حافظه مجازی(صفحه بندی درخواستی)، مانند صفحه بندی ساده، حافظه اصلی به تکه‌های هم اندازه

به نام قاب(fram) و برنامه به صفحه‌ها (page) تقسیم می‌شود. اما بر خلاف صفحه بندی ساده، لزومی ندارد تمام صفحه‌های یک فرایند در قابهای حافظه اصلی باشند تا فرایند اجرا شود. و تنها صفحاتی از فرایند که مورد نیاز است به حافظه اصلی آورده می شوند. این روش تلفیقی از صفحه بندی و مبادله است.

آدرس هایی که توسط برنامه ها تولید می شوند(آدرس مجازی)، به واحد MMU که در درون پردازنده قرار دارد، منتقل می شوند. توسط این واحد، عمل ترجمه(نگاشت) آدرس مجازی به آدرس فیزیکی انجام می شود. این عمل مشابه مراحل ترجمه آدرس در روش صفحه بندی است، با این تفاوت که بیت حضور(Present) برای هر درایه جدول صفحه وجود دارد که حضور یا عدم حضور صفحه در حافظه اصلی را تعیین می کند.

اگر MMU به جدول صفحه مراجعه کند و بیت حضور را برای صفحه مورد نظر صفر ببیند، متوجه می شود که صفحه بر روی دیسک است. دراین حالت پردازنده در تله(trap) سیستم عامل می افتد. به این تله، خطای نقص صفحه (Page Fault) می گویند که مدیریت آن بر عهده سیستم عامل است.

وقتی سعی شود مکانی از حافظه مجازی خوانده شود که موجود نیست، فقدان صفحه رخ می دهد و منجر به صدور وقفه می گردد.

## جدول صفحه

واحد مدیریت حافظه یا MMU به کمک جدول صفحه (Page Table) ، آدرس مجازی را به آدرس فیزیکی، نگاشت می کند. در صفحه بندی مجازی، هر فرایند جدول صفحه خود را دارد. اندازه هر درایه (e) جدول صفحه معمولا چهار بایت است. جدول صفحه بر حسب اندیس شماره صفحه چیده شده است و #P یک فیلد از جدول محسوب نمی شود.

فیلدهای هر درایه از یک جدول صفحه شامل موارد زیر است:

| | |
|---|---|
| **بیت اعتبار (Valid)** | اگر بیت اعتبار برای صفحه ای 1 باشد، یعنی برنامه نویس از آدرس های درون این صفحه استفاده کرده است. اگر 0 باشد، یعنی صفحه بدون استفاده است. |
| **بیت حضور- غیاب (P/A)** | اگر این بیت 1 باشد، یعنی صفحه در حافظه قرار دارد و سایر فیلدهای این درایه قابل استفاده می باشند. به این فیلد in/out نیز می گویند. |
| **شماره قاب صفحه (F#)** | مهمترین فیلد است و هدف اصلی از نگاشت صفحه، یافتن شماره قاب صفحه است. |
| **بیت مراجعه شده (R) (Referenced)** | اگر یک مراجعه، خواندن یا نوشتن به یک صفحه انجام گیرد، این بیت برابر 1 می شود. |
| **بیت تغییر یافته (M) (Modified)** | اگر عمل نوشتن بر روی یک صفحه انجام شود، سخت افزار به صورت اتوماتیک این بیت را 1 می کند. |
| **بیتهای حفاظت** | این بیت ها نوع دسترسی مجاز را مشخص می کند. در حالت سه بیتی(rwx) ، یک |

| | |
|---|---|
| **(Protection)** | بیت برای خواندن، یک بیت برای نوشتن و یک بیت برای اجرا در نظر گرفته می شود. |
| **سن (Age)** | هر چه مقدار این فیلد برای صفحه ای کمتر باشد، نشان دهنده این است که آن صفحه مدت بیشتری در حافظه بدون مراجعه بوده است. |
| **بیت کپی در نوشتن (Copy on Write)** | این فیلد زمانی مقدار می گیرد که بیش از یک فرایند در یک صفحه شریک باشند. اگر یکی از فرایندها در صفحه بنویسد، یک کپی جداگانه برای تمام فرایندهایی که در این صفحه شریک هستند، درست می شود. |
| **بیت کش غیر فعال شده (Cashing Disable)** | توسط این بیت، می توان امکان استفاده از حافظه نهان را برای صفحه، غیر فعال کرد. ماشین هایی که از فضای I/O جداگانه برخوردارند و از I/O نگاشت شده با حافظه استفاده نمی کنند،نیازی به این بیت ندارند. |

**مثال**

یک سیستم کامپیوتری مبتنی بر حافظه مجازی با اندازه صفحه 32KB مفروض است. اگر حداکثر برنامه قابل اجرا در این سیستم 4MB باشد، تعداد مدخل های جدول صفحه کدام است؟

پاسخ:

$$\frac{4MB}{32KB}=\frac{4\times 2^{20}}{32\times 2^{10}}=2^{7}=128$$

■

**مثال**

یک سیستم کامپیوتری مبتنی بر حافظه مجازی با اندازه صفحه 32KB مفروض است. در صورتی که حجم حافظه اصلی این سیستم 512KB باشد، تعداد صفحات حافظه اصلی چه مقدار خواهد بود؟

پاسخ: $\frac{512KB}{32KB}=16$

■

**مثال**

یک سیستم حافظه صفحه بندی مجازی را در نظر بگیرید که حداکثر اندازه جدول صفحه هر فرایند در آن برابر 8 مگابایت، اندازه هر صفحه برابر 8 کیلوبایت و مدخل‌های هر جدول صفحه 16 بایتی باشند، آدرسهای

ماشینی که بتواند این حافظه مجازی را مستقیماً آدرس دهی کند، باید چند بیتی باشند؟

پاسخ:

$$e = \frac{8MB}{16} = \frac{8 \times 2^{20}}{16} = 2^{19}$$

$$2^{19} \times (8kb) = 2^{19} \times (8 \times 2^{10}) = 2^{32}$$

بنابراین آدرس باید 32 بیتی باشد. ■

- جدول صفحه در حافظه اصلی قرار می‌گیرد. چون برای هر فرایند یک جدول وجود دارد و هر فرایند می‌تواند مقدار زیادی از حافظه مجازی را اشغال کند، بنابراین تعداد مدخلهای صفحه برای هر فرایند زیاد خواهد شد و جدول صفحه فضای زیادی را اشغال خواهد کرد. بنابراین بهتر است جداول صفحه در حافظه مجازی ذخیره شوند، یعنی خود جداول صفحه نیز در معرض صفحه بندی هستند.

**روشهای کاهش اندازه جدول صفحه**

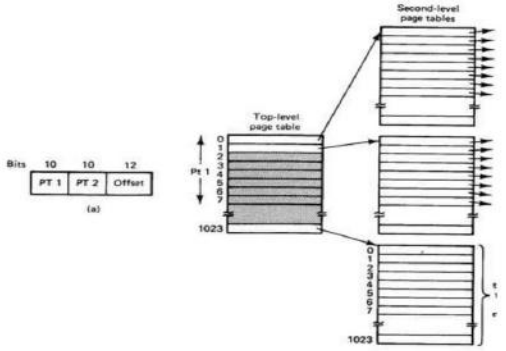
۱- جداول صفحه چند سطحی ۲- جدول صفحه وارونه(معکوس)

**صفحه بندی چند سطحی**

سیستم های مدرن از فضای آدرس منطقی گسترده ای پشتیبانی می کنند. در این محیط ها درایه های جدول صفحه زیاد خواهند شد و در نتیجه اندازه جدول صفحه بزرگ خواهد شد و چون نمی خواهیم جدول صفحه را به طور همجوار در حافظه اصلی تخصیص دهیم، باید آن را به فضاهای کوچکتری تقسیم کرد. برای این کار می توان از الگوی صفحه بندی دو سطحی استفاده کرد که خود جدول صفحه نیز صفحه بندی می شود.فرض کنید یک فرایند 20 کیلو بایتی در یک سیستم با آدرس های 32 بیتی و صفحات 4 کیلو بایتی ( $2^{12}$ بایت) داشته باشیم. بنابراین فرایند نیاز به 5 صفحه دارد. از طرفی چون آدرس 32 بیتی است و 12 بیت آن برای آفست است، پس 20 بیت برای شماره صفحه باقی می ماند. بنابراین جدول صفحه دارای $2^{20}$ درایه خواهد بود که فقط 5 درایه آن استفاده خواهد شد. پس تعداد بسیار زیادی از درایه های جدول صفحه استفاده نشده است. اگر درایه های استفاده نشده را حذف کنیم، دیگر نمی توان بر اساس P# ، جدول صفحه را اندیس دهی کرد و ترجمه آدرس کند می شود. برای حذف اغلب(نه تمامی) درایه های خالی و حفظ مکانیسم نشانی دهی بر اساس P# ، از جدول صفحه چند سطحی استفاده می کنیم. در این روش، اندیس دهی چند مرحله ای می شود.

**مثال**

سیستمی از یک جدول صفحه دو سطحی و آدرس های مجازی 32 بیتی استفاده می کند. اندازه صفحات $2^{12}$ بایت است. اولین 10 بیت آدرس به عنوان ایندکس به اولین جدول صفحه عمل می کند. در نتیجه جدول صفحه سطح اول دارای $2^{10}$ درایه است. بنابراین ایندکس سطح دوم 10 بیتی است. هر جدول صفحه در سطح دوم دارای $2^{10}$ درایه است.




**مثال**

کامپیوتری با یک فضای آدرس پذیر مجازی 18 بیتی را که صفحات آن هر یک $2^6$ بایت ظرفیت دارند، در نظر بگیرید. اندازه هر مدخل جدول صفحه 4 بایت است. به دلیل آنکه هر جدول باید داخل یک صفحه جای گیرد، یک جدول صفحه چند سطحی استفاده شده است. چند سطح مورد نیاز است؟

پاسخ:

فضای آدرس مجازی 18 بیت است که 6 بیت آن مربوط به آفست است. بنابراین 12 بیت برای شماره صفحه جا داریم. حال باید ببینیم که این 12 بیت را به چند قسمت، تقسیم کنیم.

از آنجا که هر مدخل جدول صفحه چهار بایتی است، تعداد مدخل های هر جدول صفحه که می تواند در یک صفحه جای گیرد، برابر $\frac{2^6}{4} = 2^4$ است. بنابراین اندیس های $PT_i$ برابر 4 بیت است. در نتیجه تعداد سطوح مورد نیاز برابر $\left\lceil \frac{12}{4} \right\rceil = 3$ می باشد. آدرس منطقی :

| PT1= 4bit | PT2= 4 bit | PT3= 4 bit | OFFSET= 6 bit |
|---|---|---|---|

**مثال**

فرض کنید سیستمی از فضای آدرس دهی مجازی $2^{10}$ بایت پشتیبانی می کند. در این سیستم اندازه حافظه فیزیکی قابل دسترسی $2^9$ بایت و طول هر قاب حافظه در این سیستم $2^4$ بایت می باشد. این سیستم از روش صفحه بندی برای مدیریت حافظه استفاده کرده است. با فرض اینکه هر مدخل از جدول صفحه به 11 bit به عنوان بیتهای کنترلی(بیت حضور- غیاب و ...) نیاز داشته باشد، در این صورت برای اینکه هر جدول صفحه جزئی، دقیقا در یک قاب قرار گیرد، باید حداقل از جدول صفحه چند سطحی استفاده شود؟

پاسخ: طبق اطلاعات مسئله، مشخص است که فضای آدرس مجازی 10 بیت است که 4 بیت آن مربوط به آفست است. بنابراین 6 بیت برای شماره صفحه جا داریم. حال باید ببینیم که این 6 بیت را باید به چند قسمت تقسیم کنیم. تعداد قاب ها از تقسیم اندازه فضای فیزیکی به اندازه هر قاب بدست می آید: $\frac{2^9}{2^4} = 2^5$

پس چون 32 قاب صفحه داریم، 5 بیت برای شماره قاب در جدول صفحه در نظر گرفته می شود.

همچنین 11 بیت نیز برای بیتهای کنترلی در نظر گرفته می شود. بنابراین اندازه هر درایه جدول صفحه، برابر 16 بیت(2 بایت) است. چون هر درایه جدول صفحه 2 بایت است، پس تعداد 8 درایه $(\frac{2^4}{2} = 2^3)$ می تواند در یک قاب جای بگیرد. بنابراین اندیس های $PT_i$ برابر 3 بیت است. و در نتیجه باید از جدول صفحه 2 $(\frac{6}{3} = 2)$ سطحی استفاده کرد. آدرس منطقی به صورت زیر است:

| PT1 =3 bit | PT2 =3 bit | Offset =4 bit |
|---|---|---|

## جدول صفحه وارونه(معکوس)

یکی از روش های کاهش اندازه جدول صفحه، استفاده از جدول صفحه معکوس است. جدول صفحه معکوس بر اساس شماره قاب (F#) اندیس شده است(نه شماره صفحه). هر یک از درایه ها جدول صفحه معکوس، مشخص می کند صفحه مجازی شماره چند از کدام فرایند در آن قاب صفحه قرار گرفته است.در این روش، به جای اینکه به ازای هر فرایند، یک جدول صفحه مجزا داشته باشیم، فقط یک جدول صفحه عمومی داریم.

اگر چه جدول صفحه معکوس جای کمی را اشغال می کند، ولی تبدیل آدرس مجازی به فیزیکی بسیار مشکل تر می باشد. برای سریع کردن جستجو، از یک جدول درهم (Hash Table) استفاده می شود. قسمت شماره صفحه از آدرس مجازی به کمک یک تابع درهم ساز به یک جدول درهم نگاشت می شود. جدول درهم دارای نشانگری به جدول صفحه معکوس است که شامل مدخلهای جدول صفحه می باشد. با این ساختار در جدول درهم و در جدول صفحه معکوس، برای هر صفحه حافظه حقیقی تنها یک مدخل وجود دارد.(به جای یک مدخل برای هر صفحه مجازی).

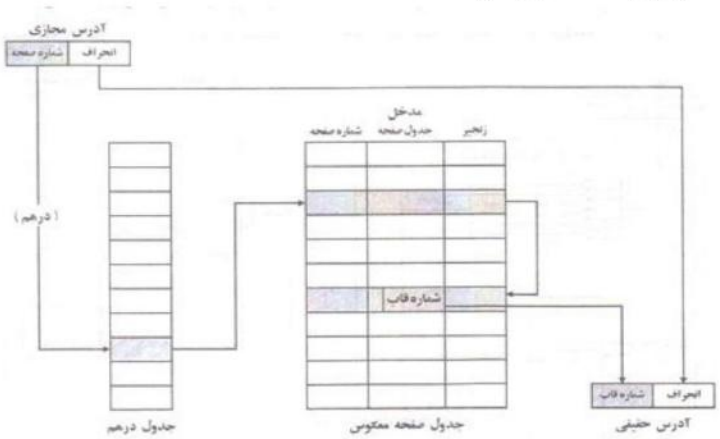
پس جدول صفحه معکوس:

۱- سبب کاهش اندازه حافظه فیزیکی جهت ذخیره سازی آن می شود.

۲- باید یک جدول صفحه خارجی نیز برای آن ذخیره شود.

۳- زمان سرویس نقص صفحه افزایش می یابد.(به دلیل ایجاد یک نقص صفحه دیگر)

شکل زیر ساختار جدول صفحه معکوس نشان داده شده است:




## بافرهای کناری ترجمه (TLB)

روشهای پیاده سازی سخت افزاری جدول صفحه عبارتند از:

۱- جدول صفحه به صورت مجموعه ای از ثباتها پیاده سازی می شود.

۲- جدول صفحه در حافظه اصلی ذخیره شود و ثبات پایه جدول صفحه (PTBR) به آن اشاره کند.

۳- استفاده از TLB (یک سخت افزار جستجوی سریع، کوچک و پنهان)

TLB : Translation Lookaside Buffers

**تذکر:** ثباتهای انجمنی TLB را میانگیرهای دم دستی نیز می نامند.

- اگر جدول صفحه بزرگ باشد، روش ۱ مناسب نمی باشد.

- در روش ۲، برای دسترسی به هر بایت، نیاز به دو دستیابی داریم (یکبار برای ورودی جدول صفحه و دیگری برای دسترسی به بایت) که این باعث کند شدن دستیابی به حافظه می شود.

**نحوه بکارگیری TLB**

در صفحه بندی تعداد اندکی از ورودیهای جدول صفحه در TLB قرار می گیرد. وقتی آدرسی تولید می شود، شماره صفحه آن با شماره صفحه موجود در ثباتهای انجمنی (که حاوی شماره صفحه و شماره قابهای متناظر با آنهاست)، مقایسه می گردد. اگر شماره صفحه در ثباتهای انجمنی پیدا شود، از شماره قاب آن برای دستیابی به حافظه استفاده می شود. اگر شماره صفحه در ثباتهای انجمنی موجود نباشد، ارجاع به جدول صفحه باید انجام گرفته و شماره صفحه و شماره قاب به TLB اضافه می شود تا در مراجعات بعدی به سرعت پیدا شود. اگر TLB پر باشد، سیستم عامل باید یکی از ورودی ها را حذف کرده و ورودی جدید را به جای آن قرار می دهد.

- هر بار که تعویض بستر (انتخاب جدول صفحه جدید) رخ دهد، باید TLB پاک شود تا فرایند بعدی که باید اجرا شود، از اطلاعات ترجمه ای نادرست استفاده نکند.

- درصد تعداد دفعاتی که شماره صفحه در ثباتهای انجمنی پیدا شود، نسبت اصابت (hit ratio) نام دارد. مثلا اگر نسبت اصابت 80% باشد، یعنی که 80% از تعداد دفعاتی که به ثباتهای انجمنی مراجعه کردیم، شماره صفحه پیدا شده است. نسبت اصابت را با H نشان می دهیم.

**زمان موثر دسترسی**

زمان موثر دسترسی، میانگین زمان واقعی است که برای یک دسترسی به حافظه اصلی مورد نیاز است. این زمان از زمان ترجمه آدرس و زمان دسترسی به کلمه مورد نظر در حافظه اصلی پس از محاسبه آدرس فیزیکی می باشد.

زمان موثر دسترسی با فرض استفاده از TLB و جدول صفحه ساده:

$$T_{Access} = T_{Translation} + T_{Mem}$$

$$T_{Translation} = H \times T_{TLB} + (1-H) \times (T_{TLB} + T_{Mem}) = T_{TLB} + (1-H) \times T_{Mem}$$

( $T_{TLB}$ : زمان دستیابی TLB ، $H$: نسبت اصابت TLB )

با فرض اینکه آدرس در TLB نباشد(یعنی H=0) رابطه به صورت زیر خواهد بود:

$$T_{Access} = T_{TLB} + 2T_{Mem}$$

زمان موثر دسترسی در صورت رخ دادن نقص صفحه با احتمال P :

$$T_{Access} = T_{Translation} + T_{Mem} + P \times T_{Disk}$$

$$T_{Translation} = T_{TLB} + (1-H) \times (T_{Mem} + P \times T_{Disk})$$

( $T_{Disk}$ : زمان انتقال صفحه از دیسک ، $P$: احتمال خطای صفحه)

با فرض اینکه آدرس در TLB نباشد و نقص صفحه حتما رخ دهد(یعنی H=0 و P=1) رابطه به صورت زیر خواهد بود:

$$T_{Access} = T_{TLB} + 2T_{Mem} + 2T_{Disk}$$

در این فرمول فرض شده که صفحه به علت تغییر باید در دیسک ذخیره شود. اگر نیازی به ذخیره نباشد از به جای 2 قبل از $T_{Disk}$ باید 1 قرار داد.

**زمان موثر دسترسی با فرض استفاده از TLB و جدول صفحه دو سطحی**

فرمت آدرس منطقی پردازنده در این حالت به صورت زیر است:

| Dir | Page | Offset |
|---|---|---|

$$T = T_{TLB} + (1 - H_{TLB})(T_1 + T_2) + T_3 + P \times T_{DISK}$$

$$T_1 = T_{Cache-Dir} + (1 - H_{Cache}) \times T_{cache-Dir\_Miss}$$

$$T_2 = T_{Cache-Pagetable} + (1 - H_{Cache}) \times T_{Cache-Pagetable-Miss}$$

$$T_3 = T_{Cache-Frame} + (1 - H_{Cache}) \times T_{Cache-Frame-Miss}$$

**بررسی حالت های مختلف:**

**۱ - اگر در مراجعه به داده اصلی در قاب صفحه دچار فقدان صفحه شویم، دو حالت رخ می دهد.**

الف- قاب خالی وجود دارد و نیاز به جایگزینی صفحه نیست: $T \cong T_{disk}$

ب- قاب خالی وجود ندارد و نیاز به جایگزینی صفحه می باشد: $T \cong 2 \times T_{Disk}$

**۲- اگر TLB اصابت کند. برای مراجعه به خود داده در حافظه دو حالت رخ می دهد:**

الف- cache اصابت کند: $T = T_{Translation} + T_{cache}$

ب- cache اصابت نکند: $T = T_{translation} + T_{cache} + T_{cache-miss-penalty}$

**۳- اگر TLB اصابت نکند، با توجه به سه مورد "دایرکتوری، جدول صفحه و قاب" در این صورت چهار حالت رخ می دهد:**

الف- در مراجعه به هر سه مورد، اصابت cache داشته باشیم:

$$T = T_{TLB} + T_{Cache-Dir} + T_{Cache-pagetable} + T_{cache-Frame}$$

ب- در مراجعه به یکی از سه مورد، cache اصابت نکند:

$$T = T_{old} + 1 \times T_{penalty}$$

(منظور از $T_{old}$، همان $T$ بدست آمده از حالت الف این حالت است)

ج- در مراجعه به دو تا از سه مورد، cache اصابت نکند: $T = T_{old} + 2 \times T_{penalty}$

د- در مراجعه هر سه مورد، cache اصابت نکند: $T = T_{old} + 3 \times T_{penalty}$

**الگوریتم‌های جایگزینی صفحه (Page Replacement Algorithms)**

در یک سیستم چند برنامه ای که از بخش بندی پویا استفاده می‌کند زمانی خواهد رسید که تمام فرایندهای موجود در حافظه اصلی در حالت مسدود قرار دارند. در این حالت سیستم عامل یکی از فرایندهای داخل حافظه اصلی را به خارج مبادله می‌کند تا فضای کافی برای فرایندی جدید یا یک فرایند آماده و معلق ایجاد شود. در این حالت سیستم عامل باید فرایندی که قرار است جایگزین شود را انتخاب کند. در واقع سیاست جایگزینی درباره انتخاب یک صفحه برای جایگزینی در زمانی که لازم است یک صفحه جدید به داخل آورده شود صحبت می‌کند.

**الگوریتم های جایگزینی عبارتند از:**

۱- بهینه ( optimal)

۲- عدم استفاده در گذشته اخیر (NRU)

۳- خروج به ترتیب ورود (FIFO)

۴- دومین شانس(Second Change)

۵- ساعت (Clock)

۶- کمترین استفاده در گذشته نزدیک (LRU)

### الگوریتم بهینه (optimal)

در این روش، صفحه ای جایگزین شود که به مدت طولانی مورد استفاده قرار نخواهد گرفت.

- الگوریتم بهینه، قابل اجرا نیست، چون نیاز به پیش بینی آینده دارد. علت مطرح شدن این الگوریتم، استفاده از آن به عنوان شاخص مقایسه می باشد.
- نام دیگر این الگوریتم، BO (Belady Optimal) می باشد.

### مثال

تعداد فقدان صفحه برای مراجعات زیر با داشتن 3 قاب صفحه در صورت استفاده از روش بهینه را مشخص کنید.

7 , 0 , 1 , 2 , 0 , 3 , 0 , 4 , 2 , 3 , 0 , 3 , 2 , 1 , 2 , 0 , 1 , 7 , 0 , 1

پاسخ:

در ابتدا به صفحه 7 مراجعه شده و چون در قابها موجود نیست، فقدان صفحه رخ می دهد و این صفحه به داخل آورده می شود. مراجعه به صفحه 0 و 1 نیز منجر به فقدان صفحه می شود.

حال به صفحه 2 مراجعه می شود. چون هر ۳ قاب پر شده است، باید یکی از صفحه های 7 , 0 و1 ، را از قابها خارج کرد، که همان صفحه 7 است،چون در آینده دورتری به آن مراجعه شده است. بعد به صفحه 0 مراجعه می شود که در قاب موجود است و بعد به 3 مراجعه می شود که جای صفحه 1 قرار خواهد گرفت،چون صفحه 1 بین صفحه های موجود در قاب در آن لحظه(یعنی 2,0,1) در آینده دورتری به آن مراجعه شده است.

این روال را ادامه می دهیم. در نهایت 9 خطای صفحه رخ می دهد.

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| F | F | F | F | | F | | F | | | F | | | F | | | | F | | |

### الگوریتم NRU

در روش NRU( Not Recently Used) ، وقتی نقص صفحه رخ می دهد، سیستم عامل بر اساس وضعیت بیت های ارجاع (R) و اصلاح (M) ، صفحات را به چهار کلاس مطابق شکل زیر، تقسیم می کند و سپس به طور تصادفی ، صفحه ای را از کلاسی که در دسته با شماره کمتری باشد، را انتخاب می کند.

| شماره کلاس | R | M |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

در واقع صفحه ای انتخاب می شود که در درجه اول از ابتدای دوره جاری،استفاده نشده است و در درجه دوم، از هنگام ورود به حافظه تغییر نکرده است. صفحه متعلق به دسته شماره 0 ، اخیرا مورد استفاده قرار نگرفته و تغییر نکرده است. به همین علت بهترین انتخاب می باشد.

**تذکر**: صفحه ای که بیت اصلاح آن یک باشد، قبل از جایگزینی باید نوشته شود.

کلاس 1، زمانی رخ می دهد که در کلاس 3 ، بیت R در یک وقفه ساعت، 0 شود. (وقفه ساعت بیت M را 0 نمی کند، چون این بیت نشان دهنده این است که آیا این صفحه باید دوباره در دیسک نوشته شود یا خیر.)

## الگوریتم FIFO

در این روش(خروج به ترتیب ورود) ، صفحه ای جایگزین شود که زمان بیشتری منتظر بوده است. برای جایگزینی، صفحه موجود در ابتدای صف را انتخاب کرده و صفحه ای که وارد حافظه شده را به انتهای صف اضافه می کنیم.

### مثال

تعداد فقدان صفحه برای مراجعات زیر با داشتن 3 قاب صفحه در صورت استفاده از روش FIFO ، چند است؟

7 , 0 , 1 , 2 , 0 , 3 , 0 , 4 , 2 , 3 , 0 , 3 , 2 , 1 , 2 , 0 , 1 , 7 , 0 , 1

پاسخ: در FIFO از یک صف استفاده می شود که طبق قانون صف، حذف از ابتدا و درج به انتها صورت می گیرد. سه مراجعه اول قابها را پر کرده و 3 نقص صفحه رخ می دهد. مراجعه به صفحه 2، موجب حذف صفحه 7 (اول صف) شده و صفحه 2 به انتهای صف اضافه می شود. مراجعه به صفحه 0 تغییری را ایجاد نمی کند چون در قاب موجود است. مراجعه به صفحه 3 موجب حذف 0 از ابتدای صف و درج 3 به انتهای صف و بروز نقص صفحه خواهد شد. در نهایت 15 خطای صفحه داریم.

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 0 | 0 | 1 | 2 | 3 | 0 | 4 | 2 | 2 | 2 | 3 | 0 | 0 | 0 | 1 | 2 | 7 |
| | 0 | 0 | 1 | 1 | 2 | 3 | 0 | 4 | 2 | 3 | 3 | 3 | 0 | 1 | 1 | 1 | 2 | 7 | 0 |

| | | 1 | 2 | 2 | 3 | 0 | 4 | 2 | 3 | 0 | 0 | 0 | 1 | 2 | 2 | 2 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F | F | F | F | | F | F | F | F | F | F | | | F | F | | | F | F | F |

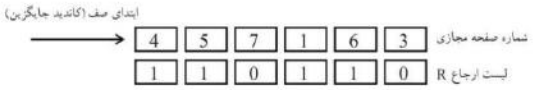
**الگوریتم دومین شانس**

الگوریتم دومین شانس، مانند FIFO می باشد. با این تفاوت که وقتی نقص صفحه رخ می دهد، اگر بیت R قدیمی ترین صفحه، 1 بود، این بیت 0 شده و صفحه به انتهای لیست منتقل می شود. با این کار به او شانس دوباره ای داده شده است.

اگر بیت R همه صفحات 1 باشد، الگوریتم دومین شانس به FIFO تبدیل می شود.

**مثال**

سیستمی از الگوریتم دومین شانس استفاده می نماید. در صورتی که به شماره صفحه مجازی 2 ارجاع شود، به جای کدام صفحه قرار می گیرد؟




پاسخ:

بیت R صفحه های 4 و5 چون 1 است، آن را صفر کرده و صفحه ها به انتهای صف منتقل می شوند. حال به صفحه 7 رسیده ایم که چون بیت R آن صفر است، صفحه را از حافظه خارج کرده و صفحه 2 را به جای آن در انتهای صف اضافه می کنیم. (بیت R صفحه دو را 1 قرار می دهیم.)

**الگوریتم ساعت**

اگر چه الگوریتم دومین شانس، الگوریتم قابل قبولی بود، اما به علت حذف صفحه از ابتدای لیست و اضافه کردن آن به انتهای لیست، روشی پر هزینه است. برای حل این مشکل از لیست چرخشی استفاده می شود. در الگوریتم ساعت، لیست پیوندی صفحات به صورت حلقوی و به شکل یک ساعت نگهداری می شود، به طوری که عقربه ساعت به قدیمی ترین صفحه اشاره می کند.

وقتی نقص صفحه رخ می دهد، به بیت R صفحه ای که توسط عقربه به آن اشاره شده، نگاه کرده که دو حالت رخ می دهد:

الف- اگر بیت R ، 0 باشد، صفحه مورد نظر خارج شده و صفحه جدید در همان مکان جایگزین شده و عقربه به جلو می رود.

ب- اگر بیت R ، 1 باشد، آنگاه 0 شده و عقربه به صفحه بعدی اشاره خواهد کرد.

**مثال**

سیستمی دارای 4 قاب صفحه است که مطابق جدول زیر صفحات مجازی در آن ها قرار دارند. اگر سیستم عامل از الگوریتم **ساعت** (Clock) استفاده نماید، در صورت وقوع نقص صفحه، صفحه جدید با کدام صفحه مجازی باید جایگزین شود؟

| شماره صفحه | زمان بارگذاری صفحه در حافظه | R |
|---|---|---|
| 1 | 50 | 1 |
| 2 | 30 | 0 |
| 3 | 20 | 1 |
| 4 | 40 | 0 |

**پاسخ:**

ابتدا عقربه بر روی قدیمی ترین صفحه یعنی صفحه 3 است. چون بیت R آن 1 است، بیت R آن 0 شده و عقربه به جلو حرکت می کند و بر روی صفحه 2 می رود(صفحه بعدی بر اساس زمان بارگذاری) . چون بیت R صفحه 2 ، صفر است، صفحه جدید با این صفحه جایگزین می شود.

الگوریتمی به نام WS-Clock هست که در آن وقتی عقربه به صفحه ای می رسد، که عضوی از مجموعه کاری (Working Set) فرایند باشد، انتخاب نمی شود و عقربه به جلو می رود. برای پیاده سازی این روش از فیلد سن (Age) جدول صفحه استفاده می شود.

(مجموعه صفحاتی که فرایند در حال حاضر از آنها استفاده می کند، مجموعه کاری نام دارد.)

### الگوریتمLRU

در روش ( LRU(Least Recently Used ، صفحه ای جایگزین می شود که برای مدت طولانی مورد استفاده قرار نگرفته است. یعنی در گذشته دورتری به آن مراجعه شده است.

### مثال

تعداد فقدان صفحه برای مراجعات زیر با داشتن 3 قاب صفحه در صورت استفاده از روش LRU چند است؟

7 , 0 , 1 , 2 , 0 , 3 , 0 , 4 , 2 , 3 , 0 , 3 , 2 , 1 , 2 , 0 , 1 , 7 , 0 , 1

پاسخ:

در ابتدا به صفحه 7 مراجعه شده و چون در قابها موجود نیست، فقدان صفحه رخ می دهد و این صفحه به داخل آورده می شود. مراجعه به صفحه 0 و 1 نیز منجر به فقدان صفحه می شود.

حال به صفحه 2 مراجعه می شود. چون هر ۳ قاب پر شده است، باید یکی از صفحه های 7 , 0 و1 ، را از قابها خارج کرد، که همان صفحه 7 است، چون نسبت به صفحه های 0 و1 در گذشته دورتری به آن مراجعه شده است. بعد به صفحه 0 مراجعه می شود که در قاب موجود است و بعد به 3 مراجعه می شود که جای صفحه 1 قرار خواهد گرفت،چون صفحه 1 بین صفحه های موجود در قاب در آن لحظه(یعنی 2,0,1) در گذشته دورتری به آن مراجعه شده است.

بنابراین الگوریتم LRU دارای 12 خطای صفحه است.

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
| | | 1 | 1 | 1 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
| F | F | F | F | | F | | F | F | F | F | | | F | | F | | F | | |

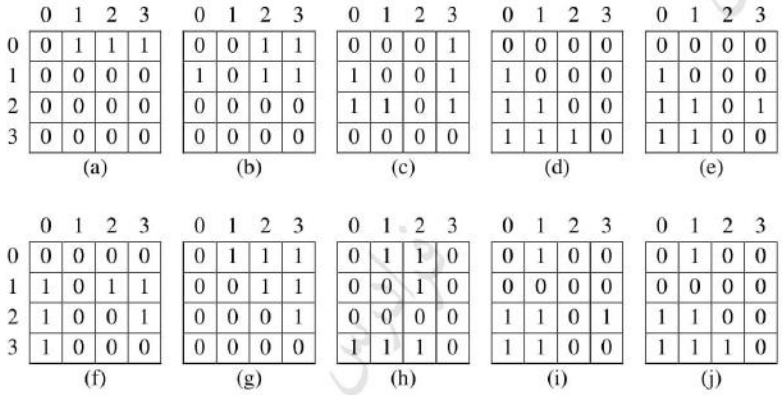
### پیاده سازی سخت افزاری LRU

پیاده سازی الگوریتم LRU ، پر هزینه است. یکی از طرح های سخت افزاری برای پیاده سازی آن، استفاده از ماتریس $n \times n$ است. در این طرح، برای یک ماشین با n قاب صفحه، سخت افزار LRU ، یک ماتریس $n \times n$ بیتی است، که در ابتدا همه عناصر آن صفر است. اگر به قاب صفحه شماره k مراجعه شود، همه بیت های سطر k ام را یک می کند و سپس همه بیت های ستون k ام، را صفر می کند. در هر لحظه، سطری که مقدار دودویی آن از همه سطرها کمتر باشد، همان صفحه مورد نظر است و نسبت به صفحه های دیگر، مدت طولانی تری بدون استفاده مانده است.

## مثال

عملکرد این الگوریتم برای 4 قاب صفحه و دسترسی به صفحات 3 2 3 2 1 0 3 2 3 0 1 2 3 2 1 0 (از چپ به است) در شکل زیر نشان داده شده است.

چون ۴ قاب داریم، پس یک ماتریس $4\times4$ بیتی با عناصر صفر در نظر می گیریم. اولین درخواست به صفحه 0 است. بنابراین ابتدا همه بیت های سطر شماره صفر را 1 می کنیم. سپس همه بیت های ستون شماره صفر را 0 می کنیم(شکل a). درخواست بعدی شماره 1 است. در ماتریس شکل a ، ابتدا سطر یک را 1 کرده و سپس ستون یک را 0 می کنیم (شکل b) . و...




## مثال

یک سیستم با 256 مگا بایت حافظه اصلی را در نظر بگیرید که از تکنیک صفحه بندی با اندازه قاب 4 کیلو بایت استفاده می کند. اگر بخواهیم الگوریتم کمترین استفاده در گذشته اخیر(LRU) را با روش ماتریس دو بعدی تقریب بزنیم، چقدر حافظه برای ذخیره سازی ماتریس نیاز است؟

پاسخ:

از آنجا که در این طرح، برای یک ماشین با n قاب صفحه، یک ماتریس $n\times n$ بیتی در نظر گرفته می شود، پس ابتدا تعداد قاب ها را محاسبه می کنیم:

$$\frac{256MB}{4KB}=\frac{2^8\times2^{20}}{2^2\times2^{10}}=\frac{2^{28}}{2^{12}}=2^{16}$$

بنابراین ماتریس مورد نیاز $2^{16}\times2^{16}$ بیتی است.

نحوه تبدیل $2^{16}\times2^{16}$ بیت به بایت:

$$\frac{2^{16} \times 2^{16}}{8} = 2^{29} = 2^{9} \times 2^{20} = 512MB$$

بنابراین به 512 مگا بایت (دو برابر اندازه حافظه اصلی)، حافظه نیاز داریم.

■

**شبیه سازی LRU در نرم افزار** (الگوریتم سالمندی)

در الگوریتم سالمندی (Aging)، از فیلد سن(age) در جدول صفحه استفاده می شود که مثلا می تواند 8 بیتی باشد. مقدار این فیلد در هنگام بارگذاری صفحه، 0 است. سیستم عامل در هر وقفه ساعت، فیلد سن همه صفحه های موجود در حافظه را یک بیت به سمت راست شیفت داده و بیت R را در بیت انتهایی سمت چپ(بیت با ارزش)، قرار می دهد. سپس فیلد R همه صفحات را 0 می کند. اگر نقص صفحه رخ دهد، صفحه ای با کمترین مقدار شمارنده برای جایگزینی انتخاب می شود.

به عنوان مثال اگر فیلد سن یک صفحه برابر 00001101 و بیت R آن برابر 1 باشد، پس از پایان پریود زمانی، ابتدا این فیلد به اندازه یک بیت به راست شیفت داده شده (0000110) و سپس بیت R در بیت انتهایی سمت چپش قرار می گیرد. پس فیلد سن برابر 10000110 خواهد شد.

به کمک فیلد سن می توان به تاریخچه مراجعات به صفحه در چند دوره اخیر پی برد. به عنوان مثال اگر فیلد سن یک صفحه 11000111 باشد، متوجه می شویم که به این صفحه در دو دوره اخیر مراجعه شده ، در سه دوره قبل از آن، مراجعه ای به آن انجام نشده و قبل از این پنج دوره، در سه دوره متوالی نیز به این صفحه مراجعه شده است.

**مثال**

اگر فیلد سن صفحه 1 برابر 00010010 و فیلد سن صفحه 2 برابر 00010000 باشد، کدام صفحه برای جایگزینی انتخاب می شود؟

پاسخ: به هر یک از دو صفحه در سه دوره متوالی اخیر، مراجعه ای نشده است. ولی به هر دو در دوره قبل از آن مراجعه شده است. اما نمی توان تشخیص داد که به کدام صفحه در آن دوره، دیرتر مراجعه شده است. در اینصورت چون به صفحه شماره 1 در سه دوره قبل از آن مراجعه شده و در مورد صفحه 2 چنین نمی باشد، صفحه 1 برای جایگزینی انتخاب می شود.■

- یکی از تفاوت های روش سالمندی با LRU این است،که نمی توان تقدم و تاخر دسترسی ها در داخل یک دوره را تشخیص داد.

- چون تعداد بیت های شمارنده در روش سالمندی محدود است، (به طور مثال 8 بیت) ، اگر مقدار شمارنده برای دو صفحه 00000000 باشد، اگر به یکی از آنها در 9 دوره قبل مراجعه شده باشد و به دیگری در 100دوره قبل، نمی توان از این موضوع آگاه شد. در اینصورت یک صفحه به صورت تصادفی انتخاب می شود.

الگوریتم های جایگزینی LFU و MFU نیز وجود دارد که به علت پایین بودن کارایی، در بیشتر منابع بیان نشده است. در صورت وقوع نقص صفحه، در LFU ، صفحه با کمترین مقدار شمارنده و در MFU ، صفحه با بیشترین مقدار شمارنده، جهت جایگزینی انتخاب می شود.

**الگوریتم بافر کردن صفحه**

در این روش، تعدادی قاب صفحه به عنوان بافر رزرو شده و صفحات درون آن از نظر جدول صفحه، غایب در نظر گرفته می شوند و با مراجعه به آنها نقص صفحه رخ می دهد. در این روش سه لیست صفحه داریم:

۱- درون حافظه ۲- درون بافر(بدون تغییر) ۳- درون بافر(تغییر یافته)

این لیست ها، به ترتیب ورود مرتب می باشند. هنگامی که صفحه ای با الگوریتم FIFO ، از ابتدای لیست اول برای جایگزینی انتخاب می شود، از لیست اول خارج شده و به انتهای لیست دوم یا سوم اضافه می شود. این صفحه واقعا از حافظه خارج نمی شود(در جدول صفحه به عنوان غایب علامت گذاری می شود)و به جای آن یک صفحه از ابتدای لیست دوم و یا سوم خارج شده و واقعا از حافظه حذف می شود.

یکی از نقطه ضعف های الگوریتم FIFO ، اخراج صفحات قدیمی پر استفاده است که در الگوریتم بافر کردن صفحه برطرف شده است. چون اگر صفحه ای با استفاده زیاد، به طور ظاهری اخراج شود، به زودی با نقص صفحه مواجه شده و به سرعت این صفحه از بافر بر گردانده می شود.

### نکات طراحی سیستم های صفحه بندی

تعداد کل قاب صفحاتی که می توان در یک سیستم به مجموعه فرایندها داد، مقداری ثابت و متناسب با اندازه حافظه اصلی است. کاهش تعداد قاب ها باعث می شود که یک فرایند نتواند مجموعه صفحاتی که در طی یک زمان از آنها استفاده می کند را در حافظه اصلی بارگذاری کند. در نتیجه تعداد نقص صفحه ها زیاد می شود. در این وضعیت، زمان CPU به جای اجرای فرایندها، صرف مبادله صفحه ها می شود و کارایی سیستم کاهش می یابد. به این پدیده، کوبیدگی (Thrashing) می گویند.

از مشکل های دیگر، می توان به این موضوع اشاره کرد که اگر در ابتدای بارگذاری یک فرایند، نتوان مجموعه کاری را تشخیص داد، باید آنقدر نقص صفحه به وجود آورد تا مجموعه کاری اش در حافظه لود شود. که باعث کند شدن سیستم می شود. بنابراین به دنبال راه حل هایی هستیم تا از این نوع مشکلات به وجود نیاید.

### پیش صفحه بندی (prepaging)

سیاست واکشی در مورد تعیین زمانی که یک صفحه باید به داخل حافظه آورده شود، دو رویکرد دارد:

۱- صفحه بندی درخواستی

۲- پیش صفحه بندی

در صفحه بندی درخواستی، فقط زمانی که مراجعه ای به مکانی از یک صفحه شود، آن صفحه به حافظه اصلی آورده می شود. ولی در پیش صفحه بندی، صفحه هایی به غیر از آنچه به وسیله خطای صفحه درخواست شده نیز به داخل آورده می شوند.

یکی از خصوصیات صفحه بندی درخواستی، رخ دادن تعداد زیادی خطای صفحه در شروع یک کار می باشد، که پیش صفحه بندی سعی به جلوگیری از این صفحه بندی زیاد دارد و از ابتدا تمام صفحات مورد نیاز فرایند را به صورت یکجا، به حافظه می آورد.

- سیاست پیش صفحه بندی می تواند یا در زمان شروع فرایند به کار گرفته شود یا هر بار که یک خطای صفحه رخ می دهد.

### مدل مجموعه کاری(working sets)

در هر لحظه زمانی(t) ، مجموعه ای از صفحات وجود دارد که در k مراجعه اخیر به حافظه، مورد استفاده واقع شده اند. به این مجموعه، مجموعه کاری می گویند و به صورت w(k,t) نمایش داده می شود. سیستم عامل با نظارت به مجموعه کاری هر فرایند، به آن قاب کافی اختصاص می دهد. اگر مجموع اندازه های

مجموعه کاری فرایندها، زیادتر از تعداد کل قاب ها شود، پردازش معلق شده و از کوبیدگی پیشگیری می شود.

**مثال**

در صورت وجود ارجاعات یک برنامه به صفحات حافظه به ترتیب زیر (از چپ به راست)

4 , 2 , 0 , 2 , 1 , 5 , 1 , 2 , 3 , 2 , 1 , 2 , 6 , 2 , 1 , 3

مجموعه کاری $w(t,k)$ که در آن t برابر زمان بین ارجاع هشتم و نهم و $k$ برابر چهار باشد، کدام است؟

پاسخ: از ارجاع بین دهم و یازدهم به اندازه ۴ ارجاع به عقب بر می گردیم:

4 , 2 , 0 , 2 , 1 , 5 , 1 , 2 , 3 , 2 , 1 , 2 , 6 , 2 , 1 , 3

بنابراین داریم: $W(t,k) = \{1,2,5\}$

■

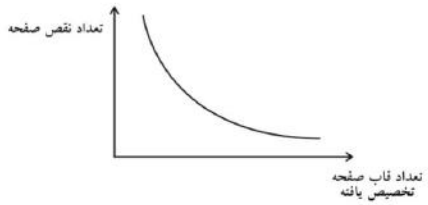
## الگوریتم فرکانس نقص صفحه (PFF)

در روش (Page Fault Frequency) PFF ، کران بالا و پایین برای نرخ خطای صفحه تعیین می شود. اگر نرخ خطای صفحه از کران بالا، بیشتر شود، قاب دیگری به آن فرایند تخصیص داده می شود و اگر نرخ خطای صفحه از کران پایین کمتر شود، قابی از فرایند پس گرفته می شود.

بنابراین PFF سعی می کند که نرخ صفحه بندی را بین دو حد قابل قبول نگه دارد تا از تفریط(کوبیدگی) و افراط(اتلاف حافظه) جلوگیری شود.

## تناقض بلیدی (Belady's anomaly)

در روش FIFO ممکن است با افزایش تعداد قابها، خطای صفحه کم نشود که به این پدیده تناقض بلیدی می گویند. تناقض بلیدی در الگوریتم های BO,LRU رخ نمی دهد. در واقع تناقض بلیدی به این معنی است که الگوریتم FIFO ممکن است از نمودار زیر، پیروی نکند:




**مثال**

دستیابی به صفحات { 4 , 3 , 2 , 1 , 4 , 3 , 5 , 4 , 3 , 2 , 1 , 5 } را در نظر بگیرید. اگر اندازه حافظه سه صفحه باشد، تعداد 9 نقص صفحه رخ می دهد و اگر اندازه حافظه را به چهار صفحه افزایش دهیم، تعداد ده نقص صفحه رخ می دهد. به این موضوع تناقض FIFO می گویند. چون انتظار داشتیم با افزایش تعداد قابهای صفحه، تعداد فقدان صفحه ها (page fault) کمتر شود ولی این چنین نشد.

پاسخ: با انباره 3 صفحه ای، 9 فقدان صفحه رخ می دهد:

| 4 | 3 | 2 | 1 | 4 | 3 | 5 | 4 | 3 | 2 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 3 | 2 | 1 | 4 | 4 | 4 | 3 | 5 | 5 |
| | 3 | 3 | 2 | 1 | 4 | 3 | 3 | 3 | 5 | 2 | 1 |
| | | 2 | 1 | 4 | 3 | 5 | 5 | 5 | 2 | 1 | 4 |
| F | F | F | F | F | F | F | | | F | F | |

و با در نظر گرفتن انباره 4 صفحه ای، 10 فقدان صفحه رخ می دهد:

| 4 | 3 | 2 | 1 | 4 | 3 | 5 | 4 | 3 | 2 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 | 4 | 3 | 2 | 1 | 5 | 4 | 3 |
| | 3 | 3 | 3 | 3 | 3 | 2 | 1 | 5 | 4 | 3 | 2 |
| | | 2 | 2 | 2 | 2 | 1 | 5 | 4 | 3 | 2 | 1 |
| | | | 1 | 1 | 1 | 5 | 4 | 3 | 2 | 1 | 5 |
| F | F | F | F | | | F | F | F | F | F | F |

■

## الگوریتم های پشته (Stack Algorithms)

الگوریتم پشته ای الگوریتمی است که در آن، مجموعه ای از صفحات موجود در حافظه برای n قاب، همیشه زیر مجموعه ای از صفحاتی است که برای n+1 قاب در حافظه خواهند بود. الگوریتم های پشته ای هیچگاه دچار تناقض بلیدی نمی شوند.

### اندازه صفحه

اندازه صفحات توانی از 2 هستند. برای تعیین اندازه صفحه باید به موارد زیر توجه شود:

**۱- اندازه جدول صفحه**

استفاده از صفحه های کوچکتر موجب افزایش اندازه جدول صفحه می شود. چون تعداد صفحات زیادتر می شود.

**۲- بهره وری**

هرچه صفحه کوچکتر باشد، بهره وری بیشتر است،چون تکه تکه شدن داخلی کمتر می شود.

**۳- زمان خواندن یا نوشتن یک صفحه**

استفاده از صفحه های کوچکتر، موجب کاهش کل زمان I/O می شود، چون محلی بودن بهبود می یابد.

کاهش اندازه صفحه موجب:

الف- کاهش تکه تکه شدن داخلی حافظه می شود.

ب- افزایش بهره وری حافظه و افزایش زمان I/O می شود.

ج- کاهش زمان سرویس نقص صفحه می شود.

تذکر: برای کمینه کردن تعداد خطای صفحه باید از صفحات بزرگ استفاده شود.

اندازه صفحه بهینه برابر $P = \sqrt{2Se}$ می باشد.(s : اندازه فرایند)(e:اندازه یک مدخل جدول صفحه)

### مثال

در محیط یک سیستم عامل که از روش حافظه مجازی برای مدیریت حافظه استفاده می‌کند، چنانچه اندزه هرفرایند 64 کیلوبایت و برای هر پروسس در جدول صفحه، 8 بایت اطلاعات ذخیره شود، اندازه صفحه بهینه در این سیستم چند بایت خواهد بود؟

$$P = \sqrt{2Se} = \sqrt{2 \times 64 \times 2^{10} \times 8} = \sqrt{2^{20}} = 2^{10}$$

### ساختار برنامه

در بیشتر موارد، کاربر از ماهیت صفحه بندی حافظه بی اطلاع است و در بعضی موارد مطلع است که موجب بهبود کارایی سیستم می شود. به طور مثال در سیستمی که اندازه صفحات ۴ کلمه است، یک برنامه نویس پاسکال می خواهد آرایه A را با صفر مقدار دهی اولیه کند.

```
Var   A : Array [1..4] [1..4] of integer;
```

می دانیم که آرایه در پاسکال به صورت سطری در حافظه ذخیره می شود. بنابراین هر سطر آرایه یک صفحه را اشغال می کند.

**الف: الگوریتم ناکارا**

```
for   j:=1  to  4   do
    for  i:=1  to  4  do
        A[i][j] := 0;
```

این کد، یک کلمه از یک صفحه و یک کلمه از صفحه دیگر را صفر می کند و این روند ادامه می یابد. اگر سیستم عامل برای کل برنامه کمتر از چهار قاب را تخصیص دهد، اجرای آن منجر به 16 خطای صفحه می گردد. $(4\times 4=16)$

**ب: الگوریتم کارا**

```
for  i=1  to  4  do
    for j=1  to  4  do
        A[i][j]=0;
```

این کد تمام کلمات یک صفحه را صفر می کند و سپس به صفحه بعدی می پردازد و به همین علت تعداد خطای صفحه برابر چهار می شود.

بنابراین، انتخاب درست ساختمان داده ها و ساختارهای برنامه نویسی می تواند موجب افزایش محلی بودن مراجعات و کاهش نرخ خطای صفحه و تعداد صفحات موجود در مجموعه کاری شود.■

**مثال**

ماتریس A[1..4][1..4] به صورت ردیفی (row–major) مفروض است. دستورات زیر عناصر این ماتریس را صفر می کند:

```
for j:=1  to 4  do
    for  i:=1  to  4  do
        A[i][j]:=0;
```

فرض کنید این برنامه در یک سیستم با مدیریت حافظه صفحه بندی بر حسب نیاز (demand paging) که اندازه قاب صفحه آن 8 کلمه است اجرا می شود. به این برنامه ۲ قاب صفحه اختصاص داده شده است که دستورات برنامه در یکی از این قابها بار شده است. قاب دیگر که ابتدا خالی است برای داده ها منظور شده است. اگر برای جایگزینی صفحات از روش LRU استفاده شود، تعداد کل فقدان صفحات چقدر است؟

(هر خانه آرایه از نوع integer است که یک کلمه از حافظه را اشغال می کند)

حل: حلقه های داده شده، ماتریس را به صورت ستون به ستون صفر می کند. از آنجا که ماتریس داده شده دو صفحه را اشغال می کند $(\frac{4\times 4}{8}=2)$، و هر دو سطر آن یک صفحه است، در هر با خواندن صفحه (دو سطر)، فقط دو خانه از یک

ستون مقدار دهی می شوند و برای مقدار دهی به هر دو خانه، یک نقص صفحه رخ می دهد و تعداد کل نقص صفحه ها برابر است با: $\frac{4\times4}{2}=8$ ■

**قطعه بندی**

در روش قطعه بندی برای مدیریت حافظه، برنامه و داده ها به تعدادی قطعه (Segment) تقسیم می‌شوند و لزومی ندارد اندازه این قطعه‌ها هم اندازه باشند. هنگامی که یک فرایند به داخل آورده می‌شود، کلیه قطعه‌های آن به داخل حافظه بار می‌شوند و جدول قطعه ایجاد می شود. هر سطر این جدول شامل آدرس شروع قطعه مورد نظر در حافظه اصلی و طول قطعه می باشد.

**نکاتی در رابطه با قطعه بندی:**

۱- آدرس منطقی در قطعه بندی از دو قسمت تشکیل یافته است: شماره قطعه و آفست.

۲- امتیاز قطعه بندی، حفاظت از قطعات و اشتراک داده ها و کد می باشد.

۳- الگوی صفحه بندی نمی تواند حافظه فیزیکی را از دیدگاه کاربر نسبت به حافظه تفکیک کند.

۴- قطعه بندی به دلیل بکارگیری قطعه‌های غیرهم اندازه، مشابه بخش بندی پویا است.

۵- تفاوت قطعه بندی با بخش بندی در این است که یک برنامه می‌تواند بیش از یک بخش را اشغال کند و لزومی ندارد این بخشها پیوسته باشند.

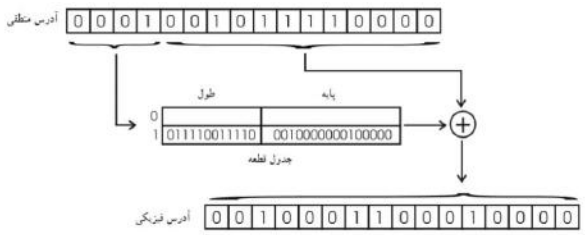
۶- قطعه بندی تکه تکه شدن داخلی را حذف می‌کند، اما دارای تکه تکه شدن **خارجی** است.

۷- تکه تکه شدن خارجی هم در بخش بندی پویا وهم در قطعه بندی وجود دارد. اما چون یک فرایند به قطعه‌های کوچکتر شکسته می‌شود، تکه تکه شدن خارجی در قطعه بندی کمتر است.

۸- در حالی که صفحه بندی از دید برنامه‌ساز مخفی است، قطعه بندی قابل رویت و عامل تسهیل سازماندهی برنامه‌ها و داده‌ها می‌باشد.

**مثال**

آدرس منطقی 16 بیتی 0001001011110000 مفروض است. نحوه تولید آدرس فیزیکی در روش قطعه بندی را نشان دهید. (آفست 12 بیتی و شماره قطعه 4 بیتی)




تذکر: آدرس فیزیکی از جمع آفست 12 بیتی با مقدار پایه بدست می آید.

تذکر : حداکثر اندازه قطعه برابر $2^{12} = 4096$ می باشد. ■

**مثال**

در یک سیستم حافظه قطعه بندی ساده، با جدول قطعه زیر، کدام آدرس منطقی (0,150) ، (2,700) فاقد آدرس فیزیکی هستند؟

| | Base | Length |
|---|---|---|
| 0 | 500 | 200 |
| 1 | 700 | 1000 |
| 2 | 400 | 600 |

پاسخ: آدرس منطقی در سیستم قطعه بندی از دو قسمت (شماره قطعه و آفست) تشکیل شده، بنابراین آدرس منطقی(2,700) آدرس فیزیکی ندارد، چون (600<700).

آدرس منطقی (0,150) ، دارای آدرس فیزیکی است، چون 150<200 . این آدرس برابر است با:500+150=650 ■

**مزایای سازماندهی بر اساس قطعه بندی**

۱- ساده شدن اداره ساختمان داده‌های رشد کننده

۲- میسر ساختن اشتراک بین فرایندها

۳- میسر شدن حفاظت

۴- امکان تغییر برنامه‌ها به صورت مستقل و ترجمه آنها بدون نیاز به پیوند و بار شدن همه مجموعه‌ها

**قطعه بندی درخواستی**

در قطعه بندی ساده، هرفرایند جدول قطعه خودش را دارد و هرگاه تمام قطعه‌های مربوط به آن فرایند به داخل حافظه اصلی بارشدند، جدول قطعه برای آن فرایند ایجاد و داخل حافظه بار می‌شود. اما در حافظه

مجازی با قطعه بندی ، لزومی ندارد تمام قطعه های یک فرایند درحافظه اصلی باشند و می‌توانند برحسب نیاز به داخل خوانده شوند. هر مدخل جدول قطعه شامل موارد زیر است:

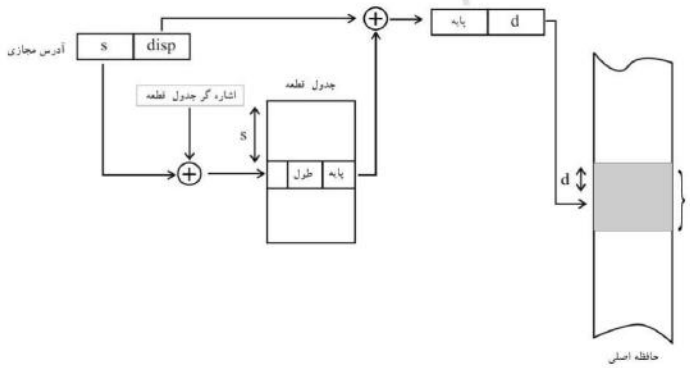
۱- طول قطعه ۲- پایه قطعه ۳- بیت حضور (P)

۴- بیت تغییر (M) ۵- بیتهای کنترلی دیگر

اگر بیت حضور فعال باشد (قطعه مورد نظر در حافظه اصلی باشد)،آدرس شروع قطعه و طول قطعه نیز در مدخل جدول در نظر گرفته می‌شود.

هنگامی که فرایندی در حال اجرا است، آدرس شروع جدول قطعه این فرایند در یک ثبات نگهداری می‌شود.

شکل زیر نحوه ترجمه آدرس در یک سیستم قطعه بندی مجازی را نشان می دهد:




**قطعه بندی صفحه بندی (Segmentation with paging)**

در سیستم ترکیبی قطعه بندی و صفحه بندی، فضای آدرس به تعدادی قطعه تقسیم می‌شود و هر قطعه نیز به تعدادی صفحه تقسیم می‌شوند. برای هر فرایند یک جدول قطعه و چند جدول صفحه وجود دارد.

آدرس مجازی در این سیستم از ۳ قسمت تشکیل شده است:

| شماره قطعه | شماره صفحه | آفست درون صفحه |
|---|---|---|

آدرس فیزیکی نیز برابر است با:

| آفست درون صفحه | شماره قاب صفحه |
|---|---|

(PTBA: Page Table Base Address)

**مثال**

یک حافظه به اندازه 8KB و با صفحات 512 بایتی را در نظر بگیرید که با روش قطعات صفحه بندی شده مدیریت می شود.

جدول قطعه به صورت زیر است:

| | PTBA | Limit |
|---|---|---|
| 0 | 0070 | 8 |
| 1 | 0078 | 8 |
| 2 | 0086 | 8 |
| 3 | 0094 | 8 |

آدرس فیزیکی متناظر با آدرس مجازی زیر را مشخص کنید.

| S# | P# | Offset |
|---|---|---|
| 11 | 101 | 1010101101 |

(لازم است بدانید در خانه به آدرس حافظه 0099 مقدار 9H ذخیره شده است.)

پاسخ:

با توجه به آدرس مجازی داده شده، چون شماره قطعه برابر 3 (همان 11 در مبنای دو) می باشد، به اندیس 3 در جدول قطعه مراجعه کرده و مقدار PTBA یعنی 0094 مشخص می شود. این مقدار را با شماره صفحه یعنی 5 (همان 101 در مبنای دو) جمع می کنیم و حاصل برابر 0099 می شود. در این آدرس، شماره قاب نوشته شده است(9H ). با قرار دادن این مقدار قبل از آفست، آدرس فیزیکی بدست می آید:

| P# | offset |
|---|---|
| 1001 | 1010101101 |

که این آدرس (یعنی 10011010101101) در مبنای شانزده برابر است با : 26AD

**مثال**

مدیریت حافظه در یک سیستم فرضی به صورت قطعه بندی صفحه بندی شده است و اندازه هر صفحه 4 کیلو بایت است، یعنی آفست 12 بیتی است. در PCB یک فرایند برای آدرس پایه جدول قطعه (STBA) مقدار 0B05H دیده می شود. اگر در این فرایند آدرس منطقی [ 01H , 4678H ] تولید شود، آدرس فیزیکی نظیر چه خواهد بود؟

(بخش اول آدرس منطقی شماره قطعه است.) (حرف H به معنی Hex است.)

هر درایه جدول قطعه سه بایتی است. بایت اول و دوم نشان دهنده آدرس پایه جدول صفحه(PTBA) و بایت سوم مسخص کننده LIMIT می باشد.

هر درایه جدول صفحه یک بایتی است و نشان دهنده شماره قاب است.

محتویات حافظه به صورت زیر است: (0B0B=0D , 0B09=07 ,0B08=0B)

پاسخ:

در مدیریت حافظه به روش"قطعه بندی صفحه بندی شده"، آدرس منطقی از سه قسمت تشکیل شده است:

| شماره قطعه | شماره صفحه | آفست |
|---|---|---|

در آدرس منطقی داده شده [01,4678] ، قسمت اول یعنی 01 ، شماره قطعه می باشد. همچنین چون آفست 12 بیتی است و چون یک عدد 12 بیتی در مبنای دو، معادل یک عدد 3 رقمی در مبنای شانزده می باشد، در نتیجه آفست برابر سه رقم آخر یعنی 678 می باشد. بنابراین شماره صفحه برابر 4 است. یعنی آدرس منطقی برابر است با:

| 01 | 4 | 678 |
|---|---|---|

از آنجا که آدرس شروع جدول قطعه برابر 0B05 ، شماره قطعه برابر 01 و هر سطر جدول قطعه 3 بایتی است، به آدرس $0B05+3=0B08$ مراجعه می کنیم. در این آدرس دو بایت اول، مشخص کننده آدرس پایه جدول صفحه( PTBA) می باشد که برابر 0B07 است. از آنجا که شماره صفحه برابر 4 و اندازه هر سطر جدول صفحه یک بایتی است، به آدرس $0B07+4=0B0B$ مراجعه کرده تا شماره قاب را پیدا کنیم، که برابر 0D است.

در نهایت با قرار دادن آفست یعنی 678 بعد از شماره قاب، آدرس فیزیکی مشخص می شود:

| شماره قاب | آفست | ⇒ | 0D | 678 |
|---|---|---|---|---|

تذکر: آفست در آدرس منطقی و فیزیکی یکسان است. (چون اندازه صفحه و اندازه قاب برابر است)

## مقایسه روشهای مدیریت حافظه

در جدول زیر، چهار نوع مدیریت حافظه با یکدیگر مقایسه شده اند:

| صفحه بندی ساده | قطعه بندی ساده | صفحه بندی حافظه مجازی | قطعه بندی حافظه مجازی |
|---|---|---|---|
| حافظه اصلی به تکه های هم اندازه به نام قاب تقسیم می شود. | حافظه اصلی تقسیم نمی شود. | حافظه اصلی به تکه های هم اندازه به نام قاب تقسیم می شود. | حافظه اصلی تقسیم نمی شود. |
| برنامه توسط مترجم یا سیستم مدیریت حافظه تقسیم می شود. | قطعه های برنامه توسط برنامه ساز به مترجم اطلاع داده می شوند.یعنی تصمیم گیری با برنامه ساز است. | برنامه توسط مترجم یا سیستم مدیریت حافظه به صفحه ها تقسیم می شود. | قطعه های برنامه توسط برنامه ساز به مترجم اطلاع داده می شوند. یعنی تصمیم گیری با برنامه ساز است. |
| بدون تکه تکه شدن خارجی | بدون تکه تکه شدن داخلی | بدون تکه تکه شدن خارجی درجه چندبرنامگی بالاتر فضای مجازی بزرگ برای فرایند | بدون تکه تکه شدن داخلی درجه چندبرنامگی بالاتر فضای آدرس مجازی بزرگ حمایت از اشتراک و حفاظت |
| تکه تکه شدن داخلی درون قاب | تکه تکه شدن خارجی گسترش به کار گیری حافظه کاهش سربار نسبت به بخش بندی پویا | تکه تکه شدن داخلی درون قاب سربار پیچیدگی مدیریت حافظه | تکه تکه شدن خارجی سربار پیچیدگی مدیریت حافظه |
| سیستم عامل باید فهرست قابهای آزاد را نگهداری کند. | سیستم عامل باید فهرست حفره های آزاد در حافظه اصلی را نگهداری کند. | سیستم عامل باید فهرست قابهای آزاد را نگهداری کند. | سیستم عامل باید فهرست حفره های آزاد در حافظه اصلی را نگهداری کند. |
| پردازنده از شماره صفحه و انحراف برای محاسبه آدرس مطلق استفاده می کند. | پردازنده از شماره قطعه و انحراف برای محاسبه آدرس مطلق استفاده می کند. | پردازنده از شماره صفحه و انحراف برای محاسبه آدرس مطلق استفاده می کند. | پردازنده از شماره قطعه و انحراف برای محاسبه آدرس مطلق استفاده می کند. |
| تمام صفحه های یک فرایند باید در حافظه اصلی باشند تا فرایند اجرا شود،مگر اینکه روی هم گذاری به کار رفته باشد. | یک فرایند به تعدادی قطعه تقسیم شده و تمام قطعه ها باید در حافظه اصلی باشند تا فرایند اجرا شود، مگر اینکه روی هم گذاری به کار رفته باشد. | لزومی ندارد تمام صفحه های یک فرایند در قابهای حافظه اصلی باشند تا فرایند اجرا شود، صفحه ها می توانند بر حسب نیاز به داخل خوانده شوند. | لزومی ندارد تمام قطعه های یک فرایند در حافظه اصلی باشند تا فرایند اجرا شود، قطعه ها می توانند بر حسب نیاز به داخل خوانده شوند. |
| | | خواندن یک صفحه به داخل | خواندن یک صفحه به داخل |

| | | | |
|---|---|---|---|
| | | حافظـه اصـلی،ممکن اسـت نیازمند نوشتن یـک صـفحه بر روی دیسک باشد. | حافظه اصلی، ممکـن اسـت نیازمند نوشتن یک قطعه بر روی دیسک باشد. |

# کنکور ارشد

**(مهندسی کامپیوتر - دولتی ۸۶)**

۱- در یک سیستم حافظه صفحه بندی ساده (Simple Paging) حافظه فیزیکی دارای $2^{24}$ بایت است. 256 صفحه فضای آدرس منطقی را تشکیل می دهد و اندازه صفحات $2^{10}$ بایت است. کدام یک از گزینه های زیر تعداد بیت های آدرس منطقی و اندازه جدول صفحه را مشخص می کند؟

۱) 18 بیت و 256 عضو
۲) 18 بیت و 16 کیلو عضو
۳) 24 بیت و 256 عضو
۴) 24 بیت و 16 کیلوعضو

پاسخ: جواب گزینه ۱ است.

تعداد درایه های جدول صفحه با تعداد صفحات فضای آدرس منطقی برابر است. بنابراین جدول صفحه دارای 256 عضو است. اندازه حافظه منطقی از حاصل ضرب تعداد صفحات در اندازه صفحه بدست می آید:

$$256 \times 2^{10} = 2^{18}$$

بنابراین تعداد بیت های آدرس منطقی برابر 18 می باشد.■

**(مهندسی کامپیوتر- دولتی ۷۱)**

۲-در یک سیستم با مدیریت حافظه Overlay، برنامه زیر اجراء می شود. این برنامه با پردازه A شروع و در انتهای این پردازه پایان می پذیرد. اندازه پردازه ها ( کیلو بایت) عبارتند از: A=5 ,B=9,C=10,D=15,E=7,F=10

اگر Overlay Driver احتیاج به 2 کیلو بایت حافظه داشته باشد، حداقل فضای مورد نیاز جهت اجرای این برنامه کدام است؟

| | | |
|---|---|---|
| procedure A;<br>.<br>call D;<br>.<br>call F;<br>.<br>end; | procedure B;<br>.<br>.<br>.<br>.<br>.<br>end; | Procedure C;<br>.<br>.<br>.<br>.<br>.<br>end; |
| procedure D;<br>.<br>call E;<br>.<br>call B;<br>end; | procedure E;<br>.<br>.<br>.<br>end; | Procedure F;<br>.<br>call C;<br>.<br>end; |

۱) 31 K ۲) 27 K ۳) 29 K ۴) 30 K

پاسخ: گزینه ۱ صحیح است.

در این روش هر زیر برنامه ای که صدا زده می شود، در حافظه فضائی به آن اختصاص می یابد و هنگامی که زیر برنامه پایان می یابد، حافظه آزاد می شود. با توجه به اینکه این برنامه با پردازه A شروع می شود، در ابتدا فضای 5K از حافظه به فرایند A داده می شود. در داخل این فرایند، فرایند D اجرا شده و 15K به آن داده می شود. در داخل فرایند D ، فرایند E اجرا شده و 7K حافظه به آن داده می شود. تا این لحظه 27K حافظه تخصیص داده شده است. با اتمام E ، فضای داده شده به آن آزاد شده و سپس با اجرای B ، 9K فضا به آن داده شده و در مجموع به 29K فضا نیاز خواهیم داشت. بعد از اتمام اجرای B، اجرای D نیز پایان یافته و F اجرا می شود. در داخل این فرایند نیز C اجرا می شود. در این حالت به 25K حافظه نیاز است. با توجه به این توضیحات، بیشترین حافظه مورد نیاز مربوط به حالت $A \rightarrow D \rightarrow B$ می باشد که به 29 کیلو بایت حافظه نیاز است. البته خود درایور نیاز به 2K دارد، بنابراین در مجموع حداقل به 31K حافظه نیاز است.

**(مهندسی IT- دولتی ۸۷)**

۳- در یک سیستم صفحه بندی ساده که جدول صفحه (page table) آن 512 عنصر16 بیتی (شامل بیت نامعتبر/ معتبر (valid/invalid) است و اندازه صفحات1 کیلوبایتی است، به ترتیب اندازه فضای آدرس فیزیکی چقدر است و آدرس فیزیکی چند بیتی است؟

۱) $26 - 2^{26}$ ۲) $25 - 2^{25}$ ۳) $24 - 2^{24}$ ۴) $16 - 2^{16}$

پاسخ: جواب گزینه ۲ است.

چون اندازه صفحه 1KB است، افست 10 بیتی است. از 16 بیت چون 1 بیت آن برای اعتبار است، 15 بیت برای شماره قاب صفحه باقی می ماند. پس تعداد بیت های آدرس فیزیکی برابر 15+10=25 می باشد. همچنین اندازه حافظه فیزیکی برابر $2^{25}$ است.

**(مهندسی کامپیوتر- دولتی ۹۲)**

۴- کدام گزینه زیر درباره جدول صفحه معکوس(inverted page-table) درست نیست؟

۱) این نوع جدول، زمان نگاشت آدرس منطقی به آدرس فیزیکی را کاهش می دهد.

۲) این نوع جدول صفحه، سبب کاهش اندازه حافظه فیزیکی جهت ذخیره سازی آن می شود.

۳) در این نوع جدول صفحه، زمان سرویس نقص صفحه(page fault) به دلیل ایجاد یک نقص صفحه دیگر افزایش می یابد.

۴) برای این نوع جدول صفحه، می بایست یک جدول صفحه خارجی نیز ذخیره شود.

حل: گزینه ۱ جواب است.

گزینه ۱ نادرست است، چون جدول صفحه معکوس، به علت نیاز به جستجو و تابع Hash ، زمان ترجمه آدرس را بالا می برد.

**(مهندسی IT- آزاد ۸۸)**

۵- در یک کامپیوتر از روش جدول صفحه معکوس شده استفاده شده است. این کامپیوتر دارای آدرس مجازی 32 بیتی، حافظه فیزیکی 64 مگا بایتی و صفحات 4 کیلو بایتی است. جدول صفحه چند مدخل (Table Entry) دارد؟

۱) $2^{8}$ ۲) $2^{12}$ ۳) $2^{20}$ ۴) $2^{14}$

پاسخ : گزینه ۴ جواب است.

در جدول صفحه وارونه، تعداد درایه های جدول صفحه برابر است با تعداد قاب های حافظه:

$$\frac{64MB}{4KB} = \frac{2^6 \times 2^{20}}{2^2 \times 2^{10}} = 2^{14}$$

■

**(مهندسی IT – دولتی ۸۸)**

۶- با فرض اینکه جدول صفحه در حافظه ذخیره شده باشد و 85% از ارجاعات به حافظه از طریق TLB انجام شود و هزینه هر ارجاع حافظه 250 نانو ثانیه و ارجاع به TLB با هزینه 5 نانو ثانیه انجام شود، با فرض عدم رخداد نقصان صفحه و عدم توازی عملیات در معماری سیستم مذکور، هر ارجاع به حافظه به طور متوسط چقدر طول می کشد؟

۱) 287.5 ثانیه ۲) 292.5 نانو ثانیه ۳) 291.75 ثانیه ۴) 505 نانو ثانیه

حل: جواب گزینه ۲ است.

همان حالت اول است:

$$T_{Translation} = T_{TLB} + (1-H) \times T_{Mem} = 5 + (0.15 \times 250) = 42.5 \ \ ns$$

$$T_{Access} = T_{Translation} + T_{Mem} = 42.5 + 250 = 292.5ns$$

■

**(مهندسی IT – آزاد ۸۴)**

۷- سیستمی علاوه بر ذخیره جدول صفحه در حافظه اصلی از TLB نیز استفاده می کند. اگر زمان خواندن از حافظه اصلی 50ns و زمان خواندن از TLB برابر 20ns باشد و درصد کارایی سیستم بدون استفاده از TLB نسبت به استفاده سیستم از TLB برابر 80 درصد باشد، آن گاه نرخ برخورد TLB چقدر است؟

۱) 10 درصد ۲) 2 درصد ۳) 80 درصد ۴) 90 درصد

حل: گزینه ۳ جواب است.

اگر از TLB استفاده نشود:

$$T_{Access} = T_{Translation} + T_{mem} = 2 \times T_{Mem} = 2 \times 50ns = 100ns$$

و اگر از TLB استفاده شود:

$$T_{Access} = [T_{TLB} + (1-H) \times T_{Mem}] + T_{Mem} = [20 + (1-H) \times 50] + 50$$

و چون درصد کارایی برابر 80% است، داریم:

$$\frac{70 + (1-H) \times 50}{100} = \frac{80}{100} \quad \Rightarrow \quad H = 0.8 \Rightarrow H = 80\%$$

■

**(مهندسی IT – دولتی ۸۴)**

۸- در یک سیستم مدیریت حافظه،اگر زمان دسترسی به حافظه 400ns و زمان دسترسی به جدول TLB ، 50ns و hit ratio در جدول TLB برابر 80٪ باشد، کدام عبارت درست است؟

۱) سیاست صفحه بندی و TLB دارای زمان دسترسی به حافظه برابر با 530ns است.

۲) سیاست صفحه بندی دارای زمان دسترسی به حافظه برابر با 850ns است.

۳) سیاست صفحه بندی دارای زمان دسترسی به حافظه برابر با 730ns است.

۴) سیاست صفحه بندی و TLB دارای زمان دسترسی به حافظه برابر با 450ns است.

پاسخ: گزینه ۱ درست است.

سیاست صفحه بندی به همراه سخت افزار TLB :

$$T_{Translation} = T_{TLB} + (1-H) \times T_{Mem} = 50 + (0.2 \times 400) = 130ns$$

$$T_{Access} = T_{Translation} + T_{Mem} = 130 + 400 = 530ns$$

گزینه ۲و ۳ نادرست است، چون در سیاست صفحه بندی بدون سخت افزار TLB :

$$T_{Translation} = T_{Mem} = 400ns$$

$$T_{Access} = T_{Translation} + T_{Mem} = 400 + 400 = 800ns$$

زمان موثر دسترسی در صورت استفاده از حافظه نهان و با در نظر گرفتن احتمال وقوع نقص صفحه:

$$T_{Access} = T_{Translation} + T_{CM} + P \times T_{Disk}$$

$$T_{Translation} = T_{TLB} + (1-H) \times T_{CM}$$

$$T_{CM} = T_{Cache} + (1-H_{Cache}) \times T_{Penalty}$$

جریمه هر عدم اصابت در حافظه پنهان : $T_{Penalty}$

**(مهندسی IT – دولتی ۸۹)**

۹- یک حافظه مجازی با این مشخصات در نظر بگیرید.

زمان دسترسی حافظه 50ns و زمان دستیابی TLB برابر 2ns ، نسبت اصابت TLB برابر 98% و احتمال خطای صفحه برای کل دسترسی ها به حافظه $2 \times 10^{-6}$ است. زمان انتقال صفحه از دیسک را 10ms فرض کنید.

برای سرعت بخشیدن به این حافظه از حافظه پنهان(cache) با این مشخصات استفاده شده است:

زمان دسترسی حافظه پنهان 10ns و نسبت اصابت حافظه پنهان 90% ، جریمه هر عدم اصابت در حافظه پنهان 100ns است.

میانگین زمان دسترسی به حافظه برای هر آدرس، به کدام یک از گزینه های زیر نزدیک تر است؟

۱) 75ns ۲) 45ns ۳) 73ns ۴) 43ns

پاسخ: جواب گزینه ۴ است.

$$T_{Translation} = T_{TLB} + (1-H) \times T_{CM} = 2 + (1-0.98) \times 20 = 2.4ns$$

$$T_{CM} = T_{Cache} + (1 - H_{cache}) \times T_{Penalty} = 10 + (1 - 0.9) \times 100 = 20\text{ns}$$

$$T_{Access} = T_{Translation} + T_{CM} + P \times T_{Disk} = 2.4 + 20 + (2 \times 10^{-6})(10 \times 10^{-3}) = 43\text{ns}$$

■

**(مهندسی IT – دولتی ۸۶)**

۱۰-در یک سیستم حافظه صفحه بندی، در یک برنامه به ترتیب به صفحات زیر رجوع شده است:

0 , 1 , 4 , 2 , 0 , 2 , 6 , 5 , 1 , 2 , 3 , 2 , 1 , 2 , 6 , 2 , 1 , 3 , 6 , 2

اگر برای این برنامه سه قاب صفحه (Page Frame) در نظر گرفته می شود و از الگوریتم جابه جائی FIFO استفاده شود، تعداد خطاهای صفحه (Page Faults) برابر است با:

۱) 10 ۲) 12 ۳) 14 ۴) 13

پاسخ: جواب گزینه ۴ است.

| 0 | 1 | 4 | 2 | 0 | 2 | 6 | 5 | 1 | 2 | 3 | 2 | 1 | 2 | 6 | 2 | 1 | 3 | 6 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 4 | 4 | 2 | 0 | 6 | 5 | 1 | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 6 |
| | 1 | 1 | 4 | 2 | 2 | 0 | 6 | 5 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 6 | 6 | 6 | 1 |
| | | 4 | 2 | 0 | 0 | 6 | 5 | 1 | 2 | 3 | 3 | 3 | 3 | 6 | 6 | 1 | 1 | 1 | 2 |
| F | F | F | F | F | | F | F | F | F | F | | | | F | | F | | | F |

**(مهندسی کامپیوتر- دولتی ۸۶)**

۱۱- حافظه اصلی کامپیوتری دارای چهار قاب صفحه می باشد. زمان بار شدن، زمان آخرین دسترسی، بیت (Reference) R و بیت M (Modify) مربوط به هر یک از صفحات در جدول زیر آمده است. اگر خطای صفحه روی صفحه مجازی شماره 4 در زمان 319 رخ دهد، تحت الگوریتم های جایگزینی LRU , NRU به ترتیب محتویات کدام یک از قاب صفحه ها ، بایستی جابجا شوند؟

| قاب صفحه | شماره صفحه مجازی | زمان بار شدن | زمان آخرین دسترسی | R | M |
|---|---|---|---|---|---|
| 0 | 2 | 125 | 278 | 0 | 1 |
| 1 | 1 | 229 | 239 | 1 | 0 |
| 2 | 0 | 119 | 271 | 1 | 0 |
| 3 | 3 | 159 | 318 | 1 | 1 |

۱) یک و دو ۲) یک و صفر ۳) دو و یک ۴) سه و صفر

پاسخ: جواب گزینه ۲ است.

در روش LRU ، صفحه ای خارج می شود که در گذشته دورتری به آن مراجعه شده است. یعنی زمان آخرین دسترسی اش از همه کمتر باشد. بنابراین صفحه شماره **1** ، که آخرین دسترسی به آن در زمان 239 بوده است، انتخاب می شود.

در روش NRU ، صفحه **2** انتخاب می شود، چون در دسته با شماره کمتری قرار دارد.

تذکر: در صورت تست، شماره قاب ها خواسته شده است. بنابراین چون صفحه 1 در قاب 1 و صفحه 2 در قاب0 قرار دارد، گزینه ۲ جواب است.

تذکر: چون در زمان رخ دادن خطای صفحه، یعنی 319 ، همه صفحه ها موجود هستند، تصمیم گیری بین همه آنها انجام گرفته است.

■

**(مهندسی کامپیوتر- آزاد ۹۰)**

۱۲- یک سیستم با 128 مگا بایت حافظه اصلی را در نظر بگیرید که از تکنیک صفحه بندی با اندازه قاب 4 کیلو بایت استفاده می کند. اگر بخواهیم الگوریتم کمترین استفاده در گذشته اخیر(LRU) را با روش ماتریس دو بعدی تقریب بزنیم، چقدر حافظه برای ذخیره سازی ماتریس نیاز است؟

۱) 16 کیلو بایت ۲) 16 مگا بایت ۳) 128 مگا بایت ۴) 32 کیلو بایت

پاسخ: جواب گزینه ۳ است.

ابتدا باید تعداد قاب ها را مشخص کرد:

$$\frac{128MB}{4KB} = \frac{128 \times 2^{20}}{4 \times 2^{10}} = 2^{15}$$

بنابراین ماتریس مورد نیاز $2^{15} \times 2^{15}$ بیتی یا 128 مگا بایتی است.

نحوه تبدیل به بایت:

$$\frac{2^{15} \times 2^{15}}{8} = 2^{27} = 2^{7} \times 2^{20} = 128MB$$

بنابراین به 128 مگا بایت (برابر اندازه حافظه اصلی)، حافظه نیاز داریم. ■

**(مهندسی کامپیوتر- دولتی ۸۴)**

۱۳- سیستمی را در نظر بگیرید که در حالت Thrashing می باشد. در این صورت کدام یک از شرایط زیر را لزوما در سیستم خواهیم داشت؟

۱) CPU در 100% اشتغال خود خواهد بود.

۲) CPU ، 100% بیکار (Idle) خواهد بود.

۳) صفحه بندی دیسک (Paging disk) در حالت 100% (صد درصد) ظرفیت خود خواهد بود.

۴) صفحه بندی دیسک (Paging disk) تقریبا 100%(صد درصد) فعال خواهد بود.

پاسخ: جواب گزینه ۴ است.

در حالت Thrashing ، نرخ نقص صفحه بسیار بالا و بهره وری CPU بسیار پایین و نزدیک به صفر (نه الزما صفر) است. پس گزینه های ۱و۲ نادرست هستند. همچنین در صد عملیات مبادله صفحات بین حافظه اصلی و دیسک بسیار بالا حتی نزدیک به 100% (نه الزما 100%) می باشد. بنابراین گزینه ۴ که از عبارت "تقریبا" استفاده کرده، صحیح می باشد.

■

**(مهندسی کامپیوتر- آزاد ۸۵)**

۱۴-اگر فرایندی اکثر زمان هایش را به جای اجرا، اختصاص به صفحه بندی دهد، این عمل چه نام دارد؟

۱) Prepaging ۲) Demand Paging ۳) Local Allocation ۴) Thrashing

پاسخ : گزینه ۴ جواب است.

**(مهندسی IT – آزاد ۸۷)**

۱۵- کدام یک از گزینه های زیر در مورد الگوریتم جایگزینی صفحه درست نیست؟

۱) الگوریتم بهینه با n+1 قاب صفحه همواره بهتر یا مساوی با الگوریتم بهینه با n قاب صفحه عمل می کند.

۲) الگوریتم LRU با n+1 قاب صفحه همواره بهتر یا مساوی با LRU با n قاب صفحه عمل می کند.

۳) الگوریتم FIFO با n+1 قاب صفحه بعضی مواقع بدتر از الگوریتم FIFO با n قاب صفحه عمل می کند.

۴) الگوریتم LRU همواره بدتر از الگوریتم بهینه عمل می کند.

پاسخ : جواب گزینه ۴ است.

گاهی ممکن است عملکرد الگوریتم LRU به خوبی عملکرد الگوریتم بهینه باشد.

گزینه دیگر درست می باشند، چون BO و LRU ناهنجاری بلیدی ندارند ولی FIFO ، ناهنجاری بلیدی دارد.■

**(مهندسی IT – آزاد ۹۰)**

۱۶- در چه صورت با پدیده Beladys Anomaly روبرو می شویم؟

۱) زمانی که الگوریتم جایگزینی صفحه از نوع Stack باشد.

۲) زمانی که رابطه مستقیم بین تعداد قاب های صفحه و تعداد نقض صفحه وجود داشته باشد.

۳) زمانی که از الگوریتم جایگزینی صفحه بهینه (Optimal) استفاده می کنیم.

۴) زمانی که مجموعه صفحات در حافظه با n قاب زیر مجموعه ای در حافظه با n+1 قاب باشد.

پاسخ: جواب گزینه ۲ است.

زمانی این پدیده رخ می دهد که رابطه مستقیم بین تعداد قاب های صفحه و تعداد نقص صفحه وجود داشته باشد. یعنی با افزایش تعداد قاب، تعداد نقص صفحه نیز افزایش یابد. ■

**(مهندسی کامپیوتر – آزاد ۸۸)**

۱۷- کدام الگوریتم یک الگوریتم جایگزینی صفحه Stack ، محسوب نمی شود؟

۱) FIFO ۲) LRU ۳) NFU ۴) Optimal

پاسخ: گزینه ۱ جواب است.■

**(مهندسی IT – دولتی ۸۴)**

۱۸- اندازه صفحه در سیستمی با مدیریت حافظه مجازی به صورت صفحه بندی درخواستی، 256 بایت است. حافظه سیستم حاوی سه قاب صفحه (در ابتدا خالی) می باشد. هر قاب صفحه می تواند به کد یا داده انتساب شود و قاب های صفحه به اشتراک بین کد و داده استفاده می شوند. اندازه کد فرایند برابر یک صفحه است و فرض کنید که حافظه فرایند فقط از دو بخش کد و داده تشکیل می شود. اگر از روش جایگزینی FIFO استفاده شود، اجرای کد زیر منجر به چند نقص صفحه خواهد شد؟

```
X : array [1..128][1..128] of  byte
for  register int  i=1 to 128 do
   for  register int  j=1 to 128 do
          X[i][j]=0;
```

۱) 65 ۲) 8193 ۳) کمتر از 65 ۴) بیشتر از 65 و کمتر از 100

پاسخ: گزینه ۴ درست است.

هر سطر ماتریس (128 عنصر 1 بایتی) نیاز به 128 بایت حافظه دارد. بنابراین در یک صفحه 256 بایتی، دو سطر ماتریس جا می شود. بنابراین ماتریس با 128 سطر به 64 صفحه نیاز دارد. چون برنامه به صورت ردیفی، ماتریس را مقدار دهی می کند، بنابراین 64 نقص صفحه برای داده ها رخ می دهد. از طرفی چون برای کد برنامه، قاب خاصی به صورت جداگانه در نظر گرفته نشده است، صفحه حاوی کد نیز مانند صفحات حاوی داده، طبق الگوریتم FIFO خارج می شود و چون کد به طور دائم در حال اجرا است، به محض خروج دوباره نقص صفحه رخ می دهد و به حافظه بر می گردد. بنابراین چون با ورود هر سه صفحه داده، یک کد خارج شده و نقص صفحه رخ می دهد، پس تعداد $\left\lceil \frac{64}{3} \right\rceil$ یعنی 22 نقص صفحه برای کد رخ می دهد. بنابراین در کل تعداد نقص صفحه ها برابر $22+64$ یعنی 86 می باشد که از 65 بیشتر و از 100 کمتر است.■

**(مهندسی IT – دولتی ۸۴)**

۱۹-در مورد آدرس مجازی زیر در یک سیستم مدیریت حافظه که از ترکیب قطعه بندی و صفحه بندی با جداول صفحه دو سطحی بهره می برد، کدام عبارت نادرست است؟

| Segment No | PT1 | PT2 | Offset |
|---|---|---|---|
| 9 Bit | 5 Bit | 7 Bit | 13 Bit |

۱) اندازه صفحه 8K و اندازه حافظه مجازی هر فرایند 16 G است.

۲) حداکثر تعداد جداول صفحه سطح دو در هر قطعه برابر128 است.

۳) حداکثر تعداد جداول صفحه سطح یک در هر فرایند برابر 512 است.

۴) حداکثر اندازه هر قطعه برابر 32M، حداکثر تعداد صفحات در هر قطعه برابر 4096 (4K) و حداکثر تعداد صفحات در هر فرایند برابر 2M است.

پاسخ: جواب گزینه ۲ است.

گزینه ۲ نادرست است، چون حداکثر تعداد جداول صفحه سطح دو، در هر قطعه برابر $2^5 = 32$ است.

علت درستی گزینه های دیگر:

اندازه صفحه $= 2^{13} = 8K$

اندازه حافظه مجازی هر فرایند $= 2^{(9+5+7+13)} = 2^{34} = 2^4 \times 2^{30} = 16G$

حداکثر اندازه هر قطعه $= 2^{(5+7+13)} = 2^{25} = 2^5 \times 2^{20} = 32M$

حداکثر تعداد جداول صفحه سطح یک در هر فرایند $= 2^9 = 512$

حداکثر تعداد صفحات در هر قطعه $= 2^{(5+7)} = 2^{12} = 2^2 \times 2^{10} = 4K$

حداکثر تعداد صفحات در هر فرایند $= 2^{(9+5+7)} = 2^{21} = 2 \times 2^{20} = 2M$

فصل ۷

مدیریک ورودی و خروجی – مدیریت دیسک

در این فصل به نحوه اداره I/O توسط سیستم عامل می پردازیم. همچنین زمان بندی دیسک ها را مورد بررسی قرار می دهیم.

**نرم افزار I/O**

نرم افزار I/O دارای چهار لایه می باشد که بر روی سخت افزار قرار دارند. این لایه ها در شکل زیر نشان داده شده اند:

| **فرایند کاربر** |
| --- |
| **نرم افزار مستقل از دستگاه** |
| **گرداننده دستگاه** |
| **اداره کننده وقفه** |
| **سخت افزار** |

**نحوه عملکرد لایه ها در هنگام خواندن بلوکی از فایل توسط فرایند کاربر**

۱- توسط **فرایند کاربر**، سیستم عامل برای خواندن بلوک فراخوانی می شود. (فراخوانی سیستمی)

۲- بلوک در حافظه پنهان بافر توسط **نرم افزار مستقل از دستگاه جستجو** می شود.

۳- اگر بلوک در حافظه پنهان پیدا نشد، **گرداننده دستگاه** برای صدور فرمان به سخت افزار برای آوردن بلوک از دیسک فراخوانی می شود. (فرایند تا کامل شدن عملیات دیسک، بلوکه می شود.)

۴- سخت افزار بعد از پایان کار دیسک، وقفه ای ایجاد کرده و **اداره کننده وقفه** اجرا می شود تا وضعیت دستگاه را چک کرده و گرداننده دستگاه را بیدار سازد. سپس نرم افزار مستقل از دستگاه و بعد فرایند کاربر، بیدار شده و نتیجه عملیات را دریافت می کنند.

**مدیریت دیسک**

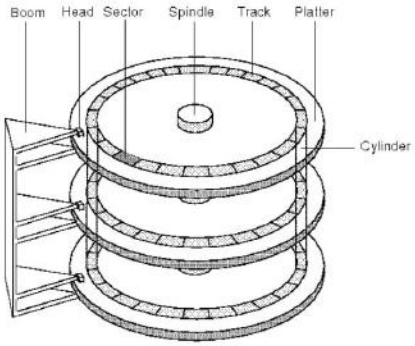
یکی از وظایف سیستم عامل، مدیریت دیسک می باشد. ابتدا دیسک مغناطیسی را تشریح می کنیم.

### دیسک مغناطیسی

دیسک مغناطیسی رسانه ای گردان با امکان دستیابی مستقیم به داده های ذخیره شده می باشد. دیسک از صفحه ای مدور و مغناطیس شونده که حول یک محور عمودی می چرخد، تشکیل شده است. رویه‌های این صفحه از غشاء فرو مغناطیسی پوشیده شده که بر روی آنها شیارهایی به صورت دایره‌های متحدالمرکز وجود دارد. شیارها از بیرون به درون با شروع از صفر شماره گذاری شده اند.

تمام شیارهای هم شعاع، تشکیل یک استوانه را می‌دهند.

شکل زیر یک دیسک پک ۳ صفحه ای را نشان می دهد. هدها به یک بازوی دیسک متصل اند که در راستای شعاع دایره حرکت می کنند.




### نرم افزار دیسک

در بخش نرم افزار دیسک، به نحوه محاسبه زمان دستیابی اطلاعات و الگوریتم های زمانبندی دیسک می پردازیم.

#### زمان استوانه جویی (Seek time)

زمانی که طول می کشد تا نوک خواندن/نوشتن به استوانه ای که داده مورد نظر در آن قرار دارد برسد. متوسط این زمان را با S نمایش می‌دهند و واحد آن میلی ثانیه است.

**زمان درنگ دوران (Rotational latency time)**

زمانی که طول می کشد تا ابتدای داده مورد نظر در اثر دوران دیسک به زیر نوک R /W برسد. که واحد آن میلی ثانیه است.

**زمان یک دور چرخش دیسک**

زمان یک دور کامل چرخش دیسک (2r) از رابطه زیر محاسبه می شود. واحد این زمان میلی ثانیه است:

$$2r = \frac{60000}{rpm}$$

که rpm سرعت چرخش دیسک است و واحد آن دور در دقیقه است.

متوسط زمان درنگ دورانی از رابطه زیر محاسبه می شود:

$$r = \frac{30000}{rpm}$$

**زمان دستیابی**

زمان دستیابی به دیسک از مجموع زمان های استوانه جویی، درنگ دورانی و انتقال محاسبه می شود.

**زمان دسترسی به فایل**

**الف- فایل بر روی n سکتورهای پراکنده ذخیره شده است.**

$$n(s + r + t)$$

s : متوسط زمان استوانه جویی

r : متوسط زمان درنگ دورانی

t : زمان خواندن یک سکتور

**ب- فایل بر روی k شیار پشت سرهم ذخیره شده است.**

$$(s + r + 2r) + (k - 1)(r + 2r)$$

این زمان تشکیل شده است از مجموع زمان خواندن شیار اول (s+r+2r) و زمان خواندن شیارهای بعدی

تذکر: متوسط زمان جستجو یعنی s ، فقط برای شیار اول در نظر گرفته می شود.

**الگوریتم های زمان بندی بازوی دیسک**

**۱- خروج به ترتیب ورود FCFS (First Come First Serviced)**

در خواست ها به ترتیب ورود به صف، اجرا می‌شوند.

**۲- ابتدا کوتاهترین زمان جستجو SSTF (Shortest Seek Time First)**

بازو به سمت درخواستی حرکت می‌کند که نزدیک ترین درخواست بعدی به مکان فعلی باشد. به عبارتی درخواستی به کمترین زمان برای حرکت بازو نیاز دارد.

**۳- مرور(آسانسور ) :Scan**

در ابتدا بازو به جهتی حرکت می‌کند که کوتاهترین زمان استوانه جویی را برای دستیابی نیاز دارد. اگر در جهت انتخاب شده به همه درخواستها پاسخ داده شد، جهت حرکت عوض می‌شود.

**۴- مرور مدور: C- Scan**

مانند روش Scan است، با این تفاوت که پس از پاسخ به آخرین درخواست در یک جهت(مثلا رو به بالا)، بازو بلافاصله به پایین ترین شماره سیلندر رفته و به سمت بالا حرکت می کند. تذکر: نام دیگر روش SCAN ، روش LOOK می باشد.

**مثال**

صف درخواستهای سیلندر به صورت ۱۴، ۲۰، ۹، ۵، ۱۲ می‌باشد و هد بر روی سیلندر ۱۰ قرار دارد. در صورت استفاده از هر یک از الگوریتم ها ی کنترل حرکت بازو، ترتیب حرکت هد را مشخص کنید؟

**FCFS** : 10 , 12 , 5 , 9 , 20 , 14 **SCAN** : 10 , 9 , 5 , 12 , 14 , 20
**SSTF** : 10 , 9 , 12 , 14 , 20 , 5 **C-SCAN** : 10 , 9 , 5 , 20 , 14 , 12

**مثال**

در یک دیسک سخت، نوک I/O HEAD روی سیلندر ۲۰ قرار دارد. اگر تقاضا برای خواندن سیلندرهای به ترتیب ۱۰، ۲۲، ۲۰، ۲، ۴۰، ۶ و ۳۸ به Driver آن وارد شود و چنانچه حرکت هدI/O بین دو سیلندر مجاور ۶ میلی ثانیه طول بکشد، در صورت استفاده از الگوریتم SSTF برای خواندن سیلندرهای، کلseek time مورد نیاز چقدر خواهد بود؟

حل: در SSTF همواره به سمت نزدیکترین سیلندر حرکت می شود:

$$20 \rightarrow 22 \rightarrow 10 \rightarrow 6 \rightarrow 2 \rightarrow 38 \rightarrow 40$$

مجموعه فاصله‌های برابر است با:

$$2 + 12 + 4 + 4 + 36 + 2 = 60$$

و چون هر حرکت ۶ میلی ثانیه طول می‌کشد، پس در کل $6 \times 60$ میلی ثانیه طول خواهد کشید. ■

**مثال**

فرض کنید در سیستمی، مدیریت دیسک از زمانبندی SSTF استفاده کند. در صورتی که جابجایی بین هر دو شیار مجاور زمانی ثابت (4ms) طول بکشد و نوک خواندن – نوشتن روی شیار 40 قرار داشته باشد، زمان جابجایی بین شیارها برای سرویس دهی به درخواستهای زیر چند میلی ثانیه است؟

ترتیب درخواستها برای شیارها(از راست به چپ) :41 , 44 , 7 , 14 , 5 , 35 , 55 , 97,100 است.

$$40 \xrightarrow{1} 41 \xrightarrow{3} 44 \xrightarrow{9} 35 \xrightarrow{20} 55 \xrightarrow{41} 14 \xrightarrow{7} 7 \xrightarrow{2} 5 \xrightarrow{92} 97 \xrightarrow{3} 100$$

که با جمع اعداد روی فلش ها، و ضرب در 4 حاصل 712 می شود. ■

- عدالت فقط در روش FCFS رعایت می شود.
- روشهای FCFS ، Scan و C-Scan ، بدون قحطی هستند.
- کارایی(میانگین زمان جستجو) در FCFS پایین است.
- مراجعات در FCFS محلی می باشد.
- حداکثر زمان پاسخ در روش C-Scan نسبت به Scan کاهش داشته است.
- الگوریتمی به نام N-Setp_Scan وجود دارد که در آن از چند صف با طول N استفاده شده و به درخواست های هر صف به روش Scan پاسخ داده می شود. در زمانی که صفی پردازش می شود، درخواست های جدید به صف های دیگر وارد می شوند.
- الگوریتم N-Setp_Scan با دو صف را الگوریتم F-Scan می گویند.
- یکی از تکنیکهای کاهش زمان درنگ دوران، استفاده از روش تداخل بلاکها (Interleaving) می باشد. در این تکنیک، بلاکها به صورت n در میان روی شیار چیده می‌شوند.

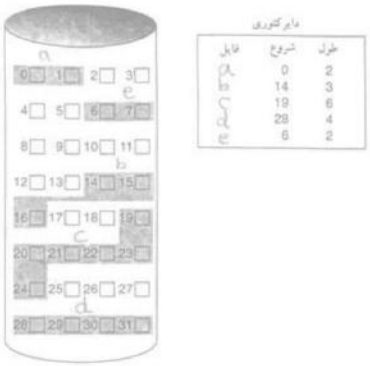
**روش های تخصیص فضای دیسک به فایل**

فضای دیسک به سه روش "پیوسته ، پیوندی و شاخصی" می تواند به فایل تخصیص داده شود. هدف روش های مختلف، رسیدن به حالتی است که هم از فضای دیسک به خوبی استفاده شود و هم دستیابی به فایل به سرعت صورت گیرد.

**۱- تخصیص پیوسته**

در این روش هر فایل تعدادی بلاک پیوسته روی دیسک را اشغال می کند و کافی است که شماره بلاک اول روی دیسک و تعداد بلاک های فایل را ذخیره کرد.

شکل زیر این نوع تخصیص را نشان می دهد:



**۲- تخصیص پیوندی**

در این روش هر فایل، یک لیست پیوندی از بلاک های روی دیسک است. بلاک ها ممکن است در هر کجای دیسک پراکنده باشند. در فهرست راهنما برای هر فایل اشاره گری به اولین بلاک فایل قرار دارد.

### ۳- تخصیص شاخصی

در روش شاخصی، اشاره گرها به بلاکهای فایل روی دیسک در یک مکان که به آن بلاک شاخص می گویند، جمع آوری می شوند. هر فایل دارای بلاک شاخص خود است که یک ماتریس از آدرسهای بلاکها است. ورودی iام در بلاک شاخص به بلاک I ام فایل اشاره می کند.

در فهرست راهنما تنها آدرس بلاک شاخص حفظ می شود. در این روش براحتی می توان دستیابی مستقیم را حمایت کرد. البته فضای بلاک شاخص تلف می شود و در بسیاری از مواقع به تمامی بلاک شاخص نیاز نمی باشد. علت پر طرفدار بودن روش شاخصی، رابطه نزدیک آن با مدیریت حافظه قطعه بندی– صفحه بندی شده است. بلاک شاخص می تواند یک جدول صفحه باشد و بلاک های فایل همانا صفحات فایل (در واقع یک قطعه) هستند.

**مشکلات تخصیص پیوسته:**

۱- یافتن فضای خالی برای یک فایل جدید. (برای یک فایل n بلاکی باید به دنبال n بلاک آزاد پشت سرهم گشت. )

۲- تعیین مقدار فضای مورد نیاز یک فایل

**مشکلات تخصیص پیوندی**

**۱- عدم حمایت از دستیابی مستقیم**

این روش فقط در رابطه با دستیابی ترتیبی خوب عمل می کند. زیرا برای رسیدن به بلاک i ، باید بلاکهای قبل از آن دستیابی شوند که هر کدام از اینها به یک خواندن از دیسک نیاز دارد.

**۲- اتلاف فضا توسط پیوند ها**

**۳- عدم قابلیت اطمینان سیستم**

با از بین رفتن تنها یکی از اشاره گرها، صدمات جدی به فایل و فضای روی دیسک وارد می گردد.

**دستیابی به فایل در تخصیص پیوسته**

**۱- ترتیبی**

سیستم فایل برای دستیابی ترتیبی، شماره بلاک روی دیسک، آخرین بلاک فایل که مورد دستیابی قرار گرفته را به خاطر می سپارد و در صورت لزوم سراغ بلاکهای بعدی می رود.

**۲- مستقیم**

برای دستیابی مستقیم به بلاک i فایل، کافی است که به بلاک b+i دستیابی صورت گیرد. (با این فرض که فایل از بلاک b دیسک آغاز شده باشد.)

**مثال**

فایلی دارای 5 بلوک از شماره 1 تا 5 می باشد. می خواهیم بلوک شماره 4 را حذف کنیم. تعداد کل نقل و انتقال دیسک در سه حالت تخصیص دیسک به فایل را مشخص کنید. (در ابتدا فایل باز است.)

حل: الف- **پیوسته**: چون همه بلوک های فایل به صورت پشت سر هم قرار دارند، بلوک 5 خوانده شده و بر روی بلوک 4 نوشته می شود. بنابراین به 2 دسترسی دیسک نیاز است.

ب- **پیوندی**: چون هر بلوک فایل حاوی اشاره گری است که آدرس بلوک بعدی را مشخص می کند، از بلوک 1 تا 4 خوانده تا آدرس بلوک 5 را به دست آوریم. سپس این آدرس را جایگزین آدرس بلوک 3 می کنیم. بنابراین به 5 دسترسی دیسک نیاز است.

ج- **شاخصی**(اندیسی): چون آدرس همه بلوک ها در یک جدول شاخص بر روی دیسک ذخیره شده، بلوک حاوی اندیس ها را خوانده و آدرس بلوک 4 را حذف کرده و بعد از به روز رسانی، بر روی دیسک باز نویسی می کنیم. بنابراین به 2 دسترسی دیسک نیاز دارد. ■

**سطوح در یک حافظه سه سطحی**

سطوح در یک حافظه سه سطحی عبارتند از:

۱- دیسک

۲- حافظه بزرگ (وسیع ولی آهسته)

۳- حافظه کوچک (سریع ولی با گنجایش محدود)

**مثال**

اگر اندازه صفحه حافظه بزرگ 4 کیلو بایت و اندازه صفحه حافظه کوچک 1 کیلو بایت باشد و زمان انتقال یک کیلو بایت برابر0.5 میلی ثانیه باشد، آنگاه:

الف- زمان انتقال 4 صفحه یک کیلو بایتی متوالی از دیسک به حافظه کوچک چند میلی ثانیه است؟

(چهار انتقال از حافظه بزرگ به حافظه کوچک )+( یک انتقال از دیسک به حافظه بزرگ )=

$= (5+2) + (4 \times 0.5) = 9\text{ms}$

ب- اگر مستقیما از دیسک به حافظه کوچک منتقل می کردیم و با توجه به اینکه اندازه صفحه باید یک کیلو بایت باشد، زمان مورد نیاز چند میلی ثانیه خواهد بود؟(زمان درنگ دورانی= 5 میلی ثانیه)

$4 \times (5 + 0.5) = 22\text{ms}$

# کنکور ارشد

**(مهندسی کامپوتر- آزاد ۹۰)**

۱- دیسکی با سرعت چرخش 12000 دور در دقیقه و متوسط زمان جستجوی 8 میلی ثانیه را در نظر بگیرید. در هر شیار 256 سکتور و در هر سکتور 512 بایت وجود دارد. فایلی به اندازه 1 مگا بایت بر روی شیارها و سکتورهای پراکنده و تصادفی ذخیره شده است. کل زمان دسترسی به این فایل چند ثانیه است؟

۱) 21.5 ۲) 68 ۳) 50 ۴) 55

پاسخ: جواب گزینه ۱ است.

زمان میانگین تاخیر چرخشی (r) :

$$2r = \frac{60000}{RPM} \Rightarrow 2r = \frac{60000}{12000} \Rightarrow r = 2.5\text{msec}$$

تعداد سکتورهای مورد نیاز برای ذخیره فایل:

$$\frac{1MB}{512} = 2048$$

زمان خواندن یک سکتور: (زمان خواندن یک شیار(2r) تقسیم بر تعداد سکتور در هر شیار)

$$\frac{5}{256} = 0.02\text{msec}$$

در نهایت زمان خواندن یک فایل با n سکتور به صورت تصادفی برابر است با:

$$2048\times(8+2.5+0.02) = 21.5\text{sec}$$

**(مهندسی کامپیوتر- دولتی ۹۰)**

۲- یک دیسک را در نظر بگیرید که شامل 100 سیلندر است(0 تا 99) . زمان لازم برای عبور هد از یک سیلندر به سیلندر مجاور یک واحد زمانی است. در زمان صفر هد بر روی سیلندر صفر است و درخواستی از گذشته وجود ندارد. شش درخواست در زمان های مختلف مطابق جدول زیر وارد می شوند.

| زمان ورود درخواست | 0 | 10 | 20 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|
| سیلندر درخواست شده | 21 | 75 | 16 | 68 | 2 | 17 |

در زمان حرکت هد به سمت یک سیلندر، ورود درخواست جدید تاثیری بر حرکت ندارد. ترتیب اجرای درخواست ها برای الگوریتم SCAN(آسانسور) چیست؟

۱) 0 21 75 16 2 17 68
۲) 0 21 75 16 17 2 68
۳) 0 21 75 16 68 2 17
۴) 0 21 75 68 16 2 17

پاسخ: جواب گزینه ۴ است.

نحوه پاسخ به درخواست ها برابر است با:

$0 \to 21 \to 75 \to 68 \to 16 \to 2 \to 17$

در لحظه صفر هد بر روی سیلندر صفر قرار دارد و تنها درخواست موجود در این لحظه، درخواست برای سیلندر 21 می باشد. بنابراین هد از سیلندر 0 به سمت سیلندر 21 می رود. در ثانیه 21، درخواست برای سیلندر75و16 رسیده، که بر طبق الگوریتم scan ، هد به سمت سیلندر 75 می رود(حرکت به سمت بالای سیلندر 21). زمانی که هد بر روی سیلندر 75 قرار دارد، درخواست برای سیلندرهای 68 و 16 وجود دارد که بر طبق الگوریتم scan ، هد به سیلندر 68 می رود(نزدیکترین سیلندر به 75). سپس به سیلندر 16 می رود. زمانی که بر روی سیلندر 16 قرار دارد، درخواست سیلندر 17 نرسیده است، بنابراین هد به سیلندر 2 می رود. در نهایت هد به سیلندر 17 می رود.

■

**(مهندسی IT- دولتی ۸۳)**

۳- به نظر شما کدام الگوریتم زمانبندی دیسک سخت، کارایی یک RAM Disk را بهینه تر می کند؟

۱) FIFO ۲) تمام الگوریتم ها منجر به کارایی یکسانی می شوند.

۳) Elevator/SCAN ۴) SSTF

پاسخ: جواب گزینه ۲ است.

در RAM Disk که از RAM برای شبیه سازی دیسک استفاده می شود، به علت نداشتن حرکت مکانیکی، زمان جستجو صفر است و بهینه تر کردن کارایی بی معنی است.

| دسته‌بندی موضوعی آموزش‌های فرادرس، در ادامه آمده است: | | | |
|---|---|---|---|
|  <br>برنامه‌نویسی |  <br>آموزش‌های دانشگاهی و تخصصی |  <br>هوش مصنوعی و یادگیری ماشین |  <br>مهندسی برق الکترونیک و روباتیک |
| برنامه نویسی – کلیک (+) | آموزش های دانشگاهی و تخصصی – کلیک (+) | هوش مصنوعی و یادگیری ماشین – کلیک (+) | مهندسی برق الکترونیک و روباتیک – کلیک (+) |
|  <br>دروس دانشگاهی |  <br>مباحث مشترک |  <br>مهارت‌های دانشگاهی |  <br>نرم‌افزارهای تخصصی |
| دروس دانشگاهی – کلیک (+) | مباحث مشترک – کلیک (+) | مهارت های دانشگاهی – کلیک (+) | نرم افزارهای تخصصی – کلیک (+) |
|  <br>مهندسی نرم‌افزار |  <br>نرم‌افزارهای عمومی |  <br>طراحی و توسعه وب |  <br>آموزش‌های عمومی |
| مهندسی نرم افزار – کلیک (+) | نرم افزارهای عمومی – کلیک (+) | طراحی و توسعه وب – کلیک (+) | آموزش های عمومی – کلیک (+) |

## منبع مطالعاتی تکمیلی مرتبط با این کتاب

**آموزش سیستم‌های عامل**


سیستم عامل یا سامانه عامل (Operating System) بدون شک مهمترین نرم‌افزار در کامپیوتر است. سیستم عامل اولین نرم‌افزاری است که پس از روشن کردن کامپیوتر مشاهده می‌شود و همچنین آخرین نرم‌افزاری خواهد بود که قبل از خاموش کردن کامپیوتر مشاهده می‌شود. سیستم عامل نرم‌افزاری است که مدیریت برنامه‌ها را به عهده گرفته و با کنترل، مدیریت و سازماندهی منابع سخت‌افزاری امکان استفاده بهینه و هدفمند آنها را فراهم کرده و بستری را برای اجرای نرم‌افزارهای کاربردی فراهم می‌کند.

آموزش سیستم عامل، توسط مهندس فرشید شیرافکن، یکی از بهترین مدرسین مسلط به این مباحث، ارائه شده است.

**مدرس:** مهندس فرشید شیر افکن

**مدت زمان:** ۱۱ ساعت

**جهت مشاهده آموزش ویدئویی این آموزش – کلیک کنید** faradars.org /fvsft103