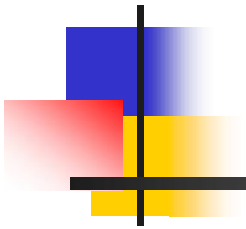
سیستم عامل

Operating Systems

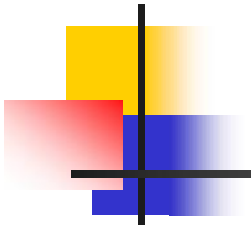
جلسه هشتم

مدرس: اسماعیل طغرایی

www.Teach.Toghraee.ir



همزمانی: انحصار متقابل و همگام سازی



مباحث این فصل:

- اصول همزمانی
- انحصار متقابل : رویکرد های نرم افزاری
- انحصار متقابل : حمایت سخت افزار
- راهنماها
- ناظرها
- تبادل پیام
- مساله خوانندگان و نویسندگان
- سؤالات دوره ای

همزمانی:

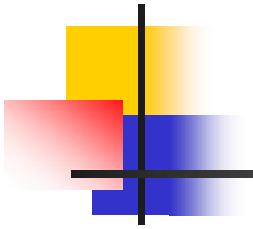
- همزمانی در سه زمینه زیر اجرا میشود:
  - کاربرد های متعدد
    - چند برنامه ای
  - کاربرد ساخت یافته
    - کاربرد ها میتوانند مجموعه ای از فرایند های همزمان باشند.
  - ساختار سیستم عامل
    - سیستم عامل مجموعه ای از نخها و فرایند هاست.

موضوعات محوری در طراحی سیستم عامل:

- **چند برنامه ای** : مدیریت فرایندهای متعدد در داخل یک سیستم تک پردازنده ای

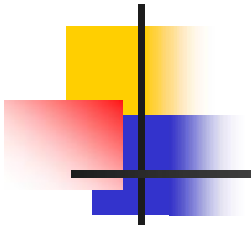
- **چند پردازشی**: مدیریت فرایندهای متعدد در داخل یک سیستم چند پردازنده ای

- **پردازش توزیعی** :مدیریت فرایندهای متعدد که روی سیستم های کامپیوتری متعدد و توزیع شده اجرا میشوند.



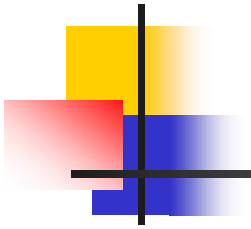
مشکلات سیستم تک پردازنده ای:

- اشتراک منابع سراسری پر مخاطره است
- مدیریت تخصیص بهینه منابع به فرایندها توسط سیستم عامل دشوار است.
- یافتن محل خطای برنامه نویسی مشکل است.



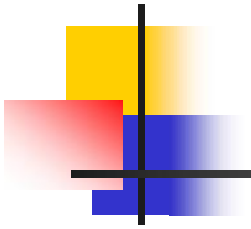
همزمانی:

- همزمانی گروهی از موضوعات طراحی را در بر میگیرد:
  - ارتباط بین فرایندها
  - اشتراک منابع و رقابت برای آنها
  - همگام سازی فعالیتهای فرایند های متعدد
  - توزیع وقت پردازنده در بین فرایند ها



یک مثال ساده:

```
void echo()
{
  chin = getchar();
  chout = chin;
  putchar(chout);
}
```



یک مثال ساده:

Process P1

```
.
chin = getchar();
.
chout = chin;
putchar(chout);
.
.
```

Process P2

```
.
.
chin = getchar();
chout = chin;
.
putchar(chout);
.
```

موارد مهم در سیستم عامل همزمانی:

- با در نظر گرفتن مساله همزمانی فرایند ها، در طراحی سیستم عامل باید موارد زیر را در نظر داشت.
  - سیستم عامل باید بتواند فرایند های فعال را دنبال کند.
  - سیستم عامل باید بتواند منابع را به فرایندها تخصیص دهد و بگیرد.
  - سیستم عامل باید داده ها و منابع هر فرایند را از دسترسی سایر فرایندها محافظت کند.
  - نتایج یک فرایند باید مستقل از سرعت پیشرفت فرایندهای دیگر باشد.

محاوره فرایندها:

- **بی اطلاعی فرایند ها از یکدیگر:** اینها فرایند های مستقل از یکدیگرند خواستار همکاری با یکدیگر نیستند.

- **اطلاع غیر مستقیم فرایند ها از یکدیگر:** اینها فرایند هایی هستند که لزوما از شناسه یکدیگر آشنا نیستند، ولی در دسترسی به بعضی اشیاء مثل بافر ورودی خروجی با یکدیگر مشترکند.

- **اطلاع مستقیم از یکدیگر:** اینها فرایندهایی هستند که قادرند با استفاده از شناسه، با یکدیگر ارتباط برقرار کنند و برای کار مشترک بر روی بعضی فعالیت ها ساخته شده اند.

میزان اطلاع	رابطه	تاثیری که یک فرایند روی فرایند دیگری میگذارد	مسئله بالقوه کنترل
بی اطلاعی فرایند ها از یکدیگر	رقابت	• استقلال نتایج یک فرایند از عملکرد فرایندهای دیگر • امکان تاثیر در زمانگیری فرایند	• انحصار متقابل • بن بست • گرسنگی
اطلاع غیر مستقیم فرایند ها از یکدیگر	همکاری بوسیله اشتراک	• امکان وابستگی نتایج یک فرایند به اطلاعات بدست آمده از فرایند های دیگر • امکان تاثیر در زمانگیری فرایند	• انحصار متقابل • بن بست • گرسنگی • وابستگی داده ها
اطلاع مستقیم از یکدیگر	همکاری توسط ارتباط	• امکان وابستگی نتایج یک فرایند به اطلاعات بدست آمده از فرایند های دیگر • امکان تاثیر در زمانگیری فرایند	• بن بست • گرسنگی

رقابت میان فرایند ها برای منابع:

- در مورد فرایند های رقیب با سه مساله کنترلی برخورد خواهیم داشت:
  - انحصار متقابل(بخش بحرانی)
    - در هر لحظه فقط یک برنامه اجازه دارد به بخش بحرانی خود وارد شود.
    - به عنوان مثال در هر لحظه تنها یک فرایند اجازه دارد پیامی را به چاپگر بفرستد.
  - بن بست
    - : هنگام اعمال انحصار متقابل، در صورتیکه یک فرایند کنترل منبعی را در اختیار بگیرد و در انتظار منبع دیگری برای اجرا یاشد ممکن است بن بست رخ دهد.
  - گرسنگی
    - ممکن است یکی از فرایندهای مجموعه برای مدتی نامحدود از دسترسی به منابع مورد نیازش محروم بماند، چرا که سایر فرایند ها منابع را به طور انحصاری بین یکدیگر مبادله میکنند. به این حالت گرسنگی می گویند.

ملزومات انحصار متقابل:

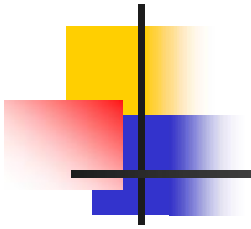
- انحصار متقابل باید اعمال گردد: از میان فرایندهایی که برای منبع یکسان یا شیء مشترکی دارای بخش بحرانی هستند، تنها یک فرایند مجاز است در بخش بحرانی خود باشد.
- فرایندی که در بخش غیربحرانی خود متوقف میشود، باید طوری عمل کند که هیچ دخالتی در عملکرد فرایند های دیگر نداشته باشد.
- برای فرایندی که نیاز به دسترسی یه یک بخش بحرانی دارد نباید امکان به تاخیر انداختن نامحدود آن وجود داشته باشد، "بن بست یا گرسنگی نمی تواند مجاز باشد.

ملزومات انحصار متقابل:

- هنگامی که هیچ فرایندی در بخش بحرانی خود نیست، هر فرایندی که متقاضی ورود به بخش بحرانی خود باشد، باید بدون تأخیر مجاز به ورود باشد.
- هیچ فرضی در مورد سرعت نسبی فرایندها یا تعداد آنها نمیتوان نوشت.
- هر فرایندی فقط برای مدت زمان محدوددی در داخل بخش بحرانی خود می ماند.

الگوریتم دیجسترا : تلاش اول

- هر فرایند مقدار متغیر Turn را بررسی می کند، اگر برابر شماره آن فرایند بود به بخش بحرانی خود وارد میشود.
- انتظار برای مشغولی:
  - فرایند همواره در حال چک کردن است تا ببیند آیا میتواند به بخش بحرانی خود وارد شود یا نه.
  - اگر فرایندی چه در بخش بحرانی و چه در خارج آن با شکست مواجه شود، فرایند دیگر مسدود می ماند.



الگوریتم دیجسترا : تلاش اول

- ساختاری که در بالا گفته شد ساختار **همروال** است
  - هم روال ها برای این طراحی شدند تا بتوانند کنترل اجرا را بین یکدیگر عقب و جلو ببرند.
  - این فن تنها برای ساخت دادن به یک فرایند واحد است، و **برای حمایت از پردازش همزمان کافی نیست**

الگوریتم دیجسترا : تلاش دوم

- در این روش از یک بردار دودویی استفاده میشود که در آن Flag[i] مربوط به فرایند i است.
- هر فرایند میتواند مقدار مربوط به فرایند دیگر را بیازماید، ولی نمیتواند آنرا تغییر دهد.
- هنگامی که فرایند میخواهد وارد ناحیه بحرانی خود شود ابتدا مقدار سایر فرایند ها را بررسی میکند.
- اگر هیچ فرایندی در بخش بحرانی خود نبود (یا Flag برای همه فرایند ها False بود) فرایند بلافاصله مقدار Flag خود را True میکند و وارد بخش بحرانی خود میشود. هنگام خروج فرایند مقدار Flag را به False برمیگرداند.
- در این صورت اگر فرایندی در ناحیه بحرانی خود شکست بخورد، فرایند دیگر تا ابد مسدود است.
- این روش انحصار متقابل را تضمین نمی کند.

الگوریتم دیجسترا : تلاش سوم

- فرایند P1 قبل از بررسی سایر فرایند ها مقدار پرچم خود را برای ورود به ناحیه بحرانی می نشاند.

- هنگامی که فرایند دیگری مثل P2 در ناحیه بحرانی است و پرچم فرایند P1 ، True است فرایند P1 تا زمانی که فرایند P2 از ناحیه بحرانی خارج شود در حالت مسدود میماند.

- امکان بن بست وجود دارد. هنگامی که دو فرایند Flag خود را برای ورود به ناحیه بحرانی True میکنند، در این صورت هر فرایند باید در انتظاز فرایند دیگر برای رهایی ناحیه بحرانی باشد.

الگوریتم دیجسترا : تلاش چهارم

- هر فرایند Flag خود را مقدار دهی میکند تا تمایل خود را برای ورود به ناحیه بحرانی نشان دهد. اما آماده است Flag خود را برای احترام به سایر فرایند ها تغییر دهد.
- سایر فرایند ها بررسی میشوند، اگر یکی از آنها در بخش بحرانی بود مقدار Flag به False باز میگردد و بعدا دوباره مقدار دهی میشود تا تمایل خود را برای ورود به ناحیه بحرانی نشان دهد. این چرخه تا زمان ورود ادامه دارد.

الگوریتم دیجسترا : تلاش چهارم

- دقت کنید که چرخه تست Flag میتواند به طور نامحدود گسترش یابد، اما این یک بن بست نیست چرا که بن بست زمانی رخ میدهد که چند فرایند بخواهند به بخش بحرانی وارد شوند ولی هیچ کدام نتوانند. به این حالت بن باز گفته میشود، چرا که هر تغییری در سرعت نسبی دو فرایند چرخه را شکسته و موجب ورود به ناحیه بحرانی میشود.

الگوریتم دیجسترا : یک راه حل صحیح

- در این روش هم از آرایه برداری دودویی Flag و هم از متغیر Turn استفاده میشود.
- هر فرایند برای ورود به ناحیه بحرانی ابتدا مقدار Flag خود را True میکند و سپس در انتظار مقدار Turn میماند

انحصار متقابل: حمایت از سخت افزار

- از کار انداختن وقفه ها:
  - یک فرایند تا زمانی که خدمتی از سیستم عامل را احظار نکرده و یا تا زمانی که با وقفه مواجه نشده به اجرای خود ادامه میدهد.
  - از کار انداختن وقفه، انحصار متقابل را ضمانت میکند.
  - پردازنده محدود به قابلیت در همگذاری برنامه هاست.
  - این رویکرد در معماری چند پردازشی کار نمیکند، چرا که در یک سیستم چند پردازشی در هر لحظه بیش از یک فرایند در حال اجراست.

انحصار متقابل: حمایت از سخت افزار

- دستورالعمل های ویژه ماشین:
  - این دستورالعمل ها در یک چرخه دستورالعمل واحد انجام میشوند، و در معرض دخالت دستورالعمل های دیگر نیستند.
  - در سطح سخت افزار دسترسی به یک محل از حافظه، از دسترسی به همان محل از حافظه جلوگیری میکند.

انحصار متقابل: حمایت از سخت افزار

- دستورالعمل آزمون و مقدار گذاری:

```
boolean testset (int i)
{
        if (i == 0)
          {
                i = 1;
                return true;
          }
        else
          {
                return false;
          }
}
```

انحصار متقابل: حمایت از سخت افزار

- دستورالعمل معاوضه:

```
void exchange(int register, int memory)
{
        int temp;
        temp = memory;
        memory = register;
        register = temp;
}
```

انحصار متقابل:

```
/* program mutualexclusion */
const int n = /* number of processes */;
int bolt;
void P(int i)
{
   while (true)
   {
      while (!testset (bolt))
          /* do nothing */;
      /* critical section */;
      bolt = 0;
      /* remainder */
   }
}
void main()
{
   bolt = 0;
   parbegin (P(1), P(2), . . . ,P(n));

}
```

(a) Test and set instruction

```
/* program mutualexclusion */
int const n = /* number of processes**/;
int bolt;
void P(int i)
{
   int keyi;
   while (true)
   {
      keyi = 1;
      while (keyi != 0)
            exchange (keyi, bolt);
      /* critical section */;
      exchange (keyi, bolt);
      /* remainder */
   }
}
void main()
{
   bolt = 0;
   parbegin (P(1), P(2), . . ., P(n));
}
```

(b) Exchange instruction

انحصار متقابل: دستورالعمل های ویژه ماشین

- **مزایا :**
  - برای هر تعداد از فرایندها، روی یک پردازنده و یا چند پردازنده، که از حافظه مشترک استفاده میکنند، قابل به کار گیری است.
  - ساده است و بنابراین وارسی آن آسان است.
  - از آن برای حمایت از بخش های بحرانی متعدد میتوان استفاده کرد که در آن هر بخش بحرانی میتواند با متغیر خاص خود تعریف شود.

انحصار متقابل: دستورالعمل های ویژه ماشین

- **معایب:**
  - **انتظار مشغولی وجود دارد.**
  - **امکان گرسنگی وجود دارد:** هنگامی که فرایندی بخش بحرانی خود را ترک میکند وبیش از یک فرایند در انتظار است.
  - **امکان بن بست وجود دارد:** اگر یک فرایند با اولویت پایین در بخش بحرانی خود باشد و به یک فرایند با اولویت بالاتر نیاز داشته باشد، و همینطور فرایند اولویت بالاتر در انتظار ورود به بخش بحرانی باشد بن بست رخ میدهد.