«بـــــــه نام خـــالق آرامـــش»

نام کتاب: سیستم عامل (بخش اول)

نام نویسنده: فرشید شیرافکن

تعداد صفحات: ۱۲۲ صفحه

تاریخ انتشار: ــــــــــــــــــ

کافئین بوکلی

CaffeineBookly.com

# سیستم عامل

مؤلف : فرشید شیرافکن

دانشجوی دکترای بیوانفورماتیک دانشگاه تهران

ناشر: سازمان علمی آموزش فرادرس

بزرگترین پلتفرم آموزش آنلاین ایران

وب: www.faradars.org

**تقدیم به:**

روح پاک پدرم

*- فرشید شیرافکن*

**سخن ناشر**

در عین تمام نقدهای وارد شده به کنکور، هنوز راه حلی عملی که در جمیع جوانب، بهتر از سبک کوتاه و چند گزینه‌ای سؤالات باشد؛ ارائه نشده است. همین موضوع، کنکور را به ویژه کنکور کارشناسی ارشد به عنوان یک آزمون متمرکز و سراسری، از اهمیت دوچندانی برخوردار می‌کند.

یکی از آسیب‌های همراه با این آزمون سراسری این است که فضای رقابتی آن با ایجاد مؤسسات گوناگون، به سرعت از فضای یک رقابت علمی تبدیل به فضای رقابت اقتصادی می‌شود؛ به گونه‌ای که هزینه سرسام آور کلاس‌ها، دوره‌ها و منابع مرتبط با آزمون، از عهده بسیاری از دانشجویان خارج می‌شود. دانشجویانی که در عین استعداد تحصیلی بالا، در میدان رقابت مالی تحمیلی، در عین تمام شایستگی‌های خود، قدرت ادامه مسیر را از دست می‌دهند یا به نتیجه‌ای که در فضای مساوی مالی برای همه باید به آن می‌رسیدند، دست نمی‌یابند.

یکی از اهداف و آرمان‌های فرادرس به عنوان بزرگ‌ترین پروژه آموزش دانشگاهی اجرا شده بر بستر وب کشور، ایجاد دسترسی همگانی و یکسان به آموزش و دانش؛ مستقل از جغرافیا، زمان و سطح مالی دانشجویان بوده است. سیاست کاری فرادرس در راستای این آرمان، انتشار آموزش‌های ویدئویی تخصصی و دانشگاهی رایگان و یا بسیار کم هزینه، با تدریس مجرب‌ترین اساتید داخل و خارج کشور بوده است.

ما با انتشار رایگان این کتاب (به همراه نزدیک به ده کتاب رایگان دیگر) یکی از گام های دیگر خود را در راستای آرمان فرادرس برداشتیم. کتاب حاضر که حاصل نزدیک به یک دهه تدریس و پژوهش و تألیف مؤلف و مدرس فرادرس می‌باشد؛ در عین هزینه‌های بالای تألیف و آماده‌سازی، به جای انتشار و فروش؛ با تأمین مالی و سرمایه‌گذاری فرادرس به عنوان ناشر، به صورت کاملاً رایگان منتشر می‌شود. ما در گام‌های بعدی نیز تلاش خواهیم کرد که تا هر جا بتوانیم، حتی شده یک کتاب مرجع دیگر و بیشتر را با پرداخت هزینه، آزادسازی کرده و به صورت رایگان منتشر کنیم.

مؤلفین و ناشرینی که تمایل به واگذاری حق انتشار کتاب خود به فرادرس را دارند، می توانند با ایمیل ebooks@faradars.org مکاتبه نمایند. ما این کتاب‌ها را با پرداخت هزینه تألیف به مدرس و ناشر، به صورت رایگان منتشر خواهیم کرد تا همه دانشجویان مستقل از سطح مالی، به منابع مفید آزمون دسترسی داشته باشند. همچنین اگر ایده و نظری در خصوص کتاب‌های رایگان فرادرس داشته باشید، خوشحال می‌شویم که آن را با ایمیل ebooks@faradars.org مطرح نمایید.


**سازمان علمی آموزش فرادرس**

**بزرگترین پلتفرم آموزش آنلاین ایران**

**وب: www.faradars.org**

دانلود رایگان مجموعه کتب ارشد کامپیوتر http://faradars.org/computer-engineering-exam

## 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

### درباره مدرس

مهندس فرشید شیرافکن کارشناس ارشد مهندسی کامپیوتر گرایش نرم‌افزار است و در حال حاضر دانشجوی دکترای بیوانفورماتیک دانشگاه تهران هستند. ایشان از مدرسین نمونه در زمینه ارائه و آموزش دروس دانشگاهی انتخاب شده‌اند.

ایشان مشاور کنکور هستند و بیش از ۳۰ کتاب در زمینه کنکور رشته کامپیوتر تألیف نموده‌اند. ایشان در حال حاضر به عنوان یکی از برترین مدرسین فرادرس از جهت کمیت و کیفیت دروس ارائه شده، نزدیک به ۲۰ عنوان درسی را در قالب آموزش ویدئویی از طریق فرادرس منتشر کرده‌اند. این مجموعه دروس تا کنون مورد استفاده ده‌ها هزار دانشجوی سراسر کشور قرار گرفته اند.

**مشاهده همه آموزش های تدریسی و تالیفی توسط مؤلف کتاب - کلیک کنید.**

## کتب رایگان دیگر از این مجموعه آموزشی

۱. آموزش برنامه نویسی C++ - کلیک کنید (+)
۲. آموزش شیء گرایی در سی پلاس پلاس – کلیک کنید (+)
۳. آموزش نظریه زبان‌ها و ماشین – کلیک کنید (+)
۴. آموزش پایگاه داده‌ها – کلیک کنید (+)
۵. آموزش ساختمان داده‌ها – کلیک کنید (+)
۶. آموزش ذخیره و بازیابی اطلاعات – کلیک کنید (+)

برای دانلود رایگان این مجموعه کتب، به لینک زیر مراجعه کنید:

**http://faradars.org/computer-engineering-exam**

| 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 (+) |

فهرست مطالب

فصل ۵ : بن بست

**شرایط بن بست**

**گراف تخصیص منابع**

**روش های رفع بن بست**

**ترمیم**

**روش های پیشگیری از بن بست**

**روش های اجتناب از بن بست**

**الگوریتم بانکداران**

**خلاصه رویکردها**

**آزمون**

فصل ۶ : مدیریت حافظه

**مدیریت حافظه ابتدایی**

**جا به جایی و حفاظت**

**مبادله**

**الگوریتم های مکان یابی و تخصیص حافظه**

**مدیریت حافظه با سیستم رفاقتی**

**روی هم گذاری** (Overlay)

**صفحه بندی** (Paging)

**حافظه مجازی**

**صفحه بندی درخواستی**

**صفحه بندی چند سطحی**

**جدول صفحه وارونه(معکوس)**

**بافرهای کناری ترجمه** (TLB)

**زمان موثر دسترسی**

**آزمون**

**الگوریتم‌های جایگزینی صفحه**

**الگوریتم بهینه (optimal)**

**الگوریتم NRU**

فصل ۱

# مفاهیم اولیه

سیستم عامل از منابع سخت افزاری پردازنده برای ارائه خدمات به کاربران استفاده می کند. بنابراین آشنایی با سخت افزار کامپیوتر، برای بررسی سیستم عامل ضروری است. اجزای سخت افزاری تشکیل دهنده کامپیوتر عبارتند از:

۱- پردازنده ۲- حافظه اصلی

۳- مولفه های ورودی و خروجی ۴- اتصالات داخلی سیستم.

### پردازنده

پردازنده از واحد محاسبه و منطق(ALU)، واحد کنترل و رجیسترها(ثبات ها) تشکیل می شود و سه گام "واکشی، رمز گشایی و اجرا" را به طور مداوم انجام می دهد. پردازنده دائما در حال کار است و فقط در موارد خاصی به مدت کوتاهی به حالت Hold می رود و کنترل گذرگاه را به کنترل کننده DMA می دهد.

**ثباتهای پردازنده**

در داخل پردازنده مجموعه ای از ثباتها وجود دارد. این ثباتها سطحی از حافظـه کـه سـریعتر و کـوچکتر از حافظه اصلی است را فراهم می کنند. ثباتهای داخل پردازنده عبارتند از:

۱- ثباتهای داده (AX,BX,CX,DX)

۲- ثبات شاخص (SI,DI)

۳- اشاره گر قطعه (CS,DS,SS,ES)

۴- اشاره گر پشته(SP)

۵- شمارنده برنامه (PC یا IP)

۶- ثبات دستورالعمل(IR)

**تذکر**: تمام پردازنده ها شامل یک یا مجموعه ای از ثباتها هستند به نام کلمه وضـعیت (PSW) کـه حـاوی اطلاعات وضعیت هستند. این ثبات، نوعا علاوه بر کـدهای وضـعیت، شـامل اطلاعـات دیگـری، مثـل بیـت فعال/غیر فعال کردن وقفه و بیت حالت کاربر/سرپرست، نیز می باشد.

**حالتهای اجرای پردازنده**

پردازنده دارای دو حالت اجرا (dual-mode) می باشد:

**۱- مد کاربر:** حالت کم امتیازتری که برنامه های کاربران در این حالت اجرا می شوند.

**۲- مد هسته(سیستم،کنترل، کرنل)** : حالت ممتازتری که دستورالعملهایی مانند تغییر ثباتهای کنترل، I/O و مدیریت حافظه، در این حالت اجرا می شوند. در این حالت نرم افزار کنترل کامل پردازنده، دستورالعملها، ثباتها و حافظه را در اختیار دارد. این سطح از کنترل برای برنامه های کاربران غیرضروری،ناامن و نامطلوب است.

- هدف اصلی از عملیات dual-mode ، محافظت سیستم عامل از دیگر نرم افزارها می باشد.
- فرایند یا پردازش(process)، یک برنامه درحال اجرا می باشد. تفاوت برنامه و فرایند در این است که برنامه یک نهاد غیرفعال و فرایند یک نهاد فعال می باشد. معمولا برنامه بر روی دیسک به صورت یک فایل اجرایی دودویی ذخیره می شود. برنامه باید به حافظه لود شود و در داخل فرایندی قرار بگیرد تا اجرا شود. فرایند می تواند هم در حافظه اصلی و هم در حافظه جانبی قرار بگیرد.
- تنظیم زمان سیستم در مد کرنل و خواندن ساعت سیستم در مد کاربر انجام می شود.
- پردازنده به کمک بیتی که در PSW وجود دارد متوجه می شود که باید در کدام حالت (کاربر یا هسته) اجرا کند.
- وقتی کاری برای اجرا موجود نباشد، پردازنده یک برنامه شامل یک حلقه انتظار مشغول را اجرا خواهد کرد.
- عمل تغییر وضعیت از مد کرنل به مد کاربر و بلعکس از امکانات سخت افزاری است که برای کمک مستقیم به سیستم عامل طراحی شده است.

**وقفه (interrupt)**

وقفه سیگنالی است که روند عادی اجرا را تغییر می دهد. وقتی وقفه‌ای به CPU ارسال می‌شود، کار CPU متوقف شده و روال پاسخگو به وقفه (interrupt routine) اجرا می‌شود. بعد از پایان اجرای این روتین، CPU کار قبلی‌اش را ادامه می‌دهد. آدرس دستوری که هنگام اجرای آن وقفه صادر شده (آدرس برگشت) در

پشته ذخیره می شود. بعد از پایان پاسخگویی به وقفه، آدرس برگشت در شمارنده برنامه (PC) قرار می گیرد و محاسباتی که اجرای آنها به تعویق افتاده از سر گرفته می شود.

- با وجود وقفه‌ها، پردازنده می‌تواند در هنگامی که عمل ورودی/خروجی درجریان است، مشغول اجرای دستور العملهای دیگر باشد.
- سخت افزار در هر زمان می‌تواند با ارسال سیگنالی به CPU، وقفه‌ای را صادر کند. نرم افزار نیز با اجرای عملیات خاصی به نام فراخوانی سیستم (System call)، می‌تواند وقفه‌ای را صادر کند.
- وقتی وقفه‌ای رخ می‌دهد، سخت افزار کنترل را به سیستم عامل بر گردانده و ثبات‌ها و شمارنده برنامه را ذخیره می کند. همچنین نوع وقفه را نیز تعیین می کند.
- بردار وقفه، جدولی از اشاره گرها می باشد که هر اشاره گر به یک روال وقفه اشاره می کند.

## انواع وقفه‌ها

وقفه ها بر چهار نوع می باشند:

### ۱- برنامه

وقفه هایی که به دلیل بعضی شرایط حاصل از اجرای یک دستورالعمل بروز می کند:

الف- سرریز شدن محاسباتی

ب- تقسیم بر صفر

ج- تلاش برای اجرای یک دستورالعمل ماشین غیرمجاز

د- مراجعه به آدرس خارج از فضای مجاز کاربر

### ۲- زمان سنج

وقفه ای که توسط زمان سنج داخلی پردازنده تولید می شود. این وقفه به سیستم عامل اجازه می دهد، بعضی اعمال (مانند تست حافظه، چک کردن سخت افزار و یا تعیین زمان اجرای پردازنده درهر برش در سیستم استراک زمانی)را به طور مرتب انجام دهد.

### ۳- ورودی/خروجی

وقفه هایی که به وسیله کنترل کننده I/O تولید می شود، تا کامل شدن طبیعی یک عمل یا شرایط خطا را اعمال کند.

### ۴- نقص سخت افزار

وقفه هایی که با نقص سخت افزاری تولید می شود، مثل نقص برق یا خطای توازن حافظه.

**پردازش وقفه**

هنگامی که یک دستگاه I/O ، یک عمل I/O را کامل می کند، دنباله حوادث زیر اتفاق می افتد:

۱- دستگاه یک علامت وقفه برای پردازنده می فرستد.

۲- پردازنده اجرای دستورالعمل جاری را قبل از پاسخ به این وقفه، به پایان می رساند.

۳- پردازنده بروز وقفه را بررسی کرده، در می یابد که وقفه ای آمده است و علامتی مبنی بر دریافت وقفه برای دستگاه وقفه دهنده می فرستد.

۴- پردازنده برای انتقال کنترل به روال خدماتی مربوط آماده می شود. در ابتدا اطلاعات مورد نیاز برای از سرگیری برنامه جاری را ذخیره می کند. حداقل اطلاعات، محتویات ثبات وضعیت برنامه(PSW) و محل دستورالعمل بعدی(PC) می باشد. این اطلاعات می تواند در بالای پشته کنترل سیستم گذاشته شود.

۵- پردازنده شمارنده برنامه(PC) را با آدرس شروع برنامه گرداننده وقفه ای که قرار است به این وقفه پاسخ دهد، بار می کند. در این حالت پردازنده به چرخه واکشی رفته و کنترل به برنامه گرداننده وقفه منتقل می شود.

۶- در این لحظه محتوای PC و PSW مربوط به برنامه وقفه داده شده، در پشته سیستم ذخیره است.

۷- روال وقفه را پردازش می کند.

۸- بعد از کامل شدن پردازش وقفه، مقادیر ثباتها از پشته POP شده و در ثباتها گذاشته می شود.

۹- بار کردن مجدد مقادیر PSW وPC از پشته، که موجب اجرای دستورالعمل بعدی از برنامه وقفه داده شده می شود.

**تعویض متن (Context Switch)**

هنگامیکه وقفه ای رخ می دهد، قبل از اینکه سیستم عامل کنترل را به یک روال وقفه گیر بخصوص رد کند، وضعیت پردازش جاری را در محلی حفظ می کند، تا بتواند بعدا آنرا ادامه دهد و سپس به طرف روال وقفه گیر می رود که به این جریان تعویض متن می گویند.

**فراخوانی های سیستم (System Calls)**

فراخوانیهای سیستم واسطی بین فرایند و سیستم عامل فراهم می کند که به صورت دستورات زبان اسمبلی می باشند. در بعضی از سیستم ها، فراخوانی ها مستقیما از برنامه های زبان سطح بالا ساخته شده و مانند زیر برنامه می باشند. وقتی برنامه ای اجرا می شود از فراخوانی های سیستمی به وفور استفاده می

کند و کاربر جزئیات آن را نمی بیند.

**انواع فراخوانی های سیستم عبارتند از:**

۱- کنترل فرایند

۲- دستکاری فایلها

۳- دستکاری دستگاه

۴- دستکاری اطلاعات

۵- ارتباطات

- وقتی یک وقفه، تله و یا فراخوانی سرپرست رخ می دهد، پردازنده در حالت هسته گذاشته شده و کنترل به سیستم عامل منتقل می گردد و یک تعویض حالت به یک روال سیستم عامل انجام شده، ولی چون اجرا در داخل فرایند جاری ادامه می یابد، تعویض فرایند انجام نمی شود.

**حفاظت**

مسئله حفاظت از سه دیدگاه مورد بررسی است:

**۱- حفاظت از I/O**

برای حفاظت از I/O می توان تمام دستورات I/O را به عنوان دستورات ممتاز در نظر گرفت تا کاربران فقط از طریق سیستم عامل بتوانند آن دستورات را اجرا کنند.

**۲- حفاظت از حافظه**

حفاظت حافظه را حداقل برای بردار وقفه و روال وقفه باید فراهم کرد.در واقع می خواهیم سیستم عامل را از دستیابی برنامه کاربر و همچنین برنامه های کاربر را از یکدیگر محافظت کنیم. یکی از روشهای ممکن استفاده از ثباتهای پایه و حد می باشد.

**۳- حفاظت از CPU**

باید کاری کرد که برنامه کاربر در حلقه گیر نکند و کنترل را به سیستم عامل برگرداند. برای این منظور از یک تایمر استفاده می کنیم.

**سلسله مراتب حافظه**

بین سه ویژگی کلیدی حافظه، یعنی "هزینه، ظرفیت و زمان دسترسی" باید سبک و سنگین کرد. برای این کار نمی توان بر یک حافظه یا فن آوری خاصی تکیه کرد و باید از سلسله مراتب حافظه استفاده کرد.

یک سلسله مراتب متداول حافظه در زیر آورده شده است:

۱- ثباتها ۲- حافظه پنهان ۳- حافظه اصلی

۴- حافظه پنهان دیسک ۵- دیسک مغناطیسی ۶- رسانه جا به جایی پذیر

**با حرکت به سطوح پایین تر این سلسله مراتب، شرایط زیر رخ می دهد:**

۱- کاهش هزینه در هر بیت ۲- افزایش ظرفیت

۳- افزایش زمان دسترسی ۴- کاهش تعداد دفعات دسترسی پردازنده به حافظه

مدیریت منابع هر یک از حافظه های زیر در مقابل آن نوشته شده است:

ثبات: کامپایلر

حافظه پنهان : خودکار(سخت افزاری)

حافظه اصلی و دیسک: سیستم عامل

**حافظه پنهان**

حافظه پنهان(cache)، یک حافظه کوچک و سریع بین پردازنده و حافظه اصلی است. حافظه پنهان، حاوی بخشی از حافظه اصلی است. وقتی پردازنده می خواهد کلمه ای از حافظه را بخواند، وجود آن را در حافظه پنهان بررسی می کند. اگر وجود داشته باشد، به پردازنده تحویل می شود، در غیر اینصورت یک بلوک از حافظه اصلی، شامل تعداد ثابتی از خانه های حافظه، به حافظه پنهان منتقل شده و سپس کلمه مورد نظر به پردازنده تحویل داده می شود. هنگامی که یک بلوک از داده ها به حافظه پنهان آورده می شود تا یک مراجعه به حافظه انجام شود، به دلیل پدیده محلی بودن مراجعات، احتمالا بزودی به دیگر کلمات آن بلوک نیز مراجعه خواهد شد.

**روشهای انتقال ورودی/خروجی**

برای عملیات I/O روشهای زیر وجود دارد:

### ۱- I/O برنامه سازی شده (سرکشی)

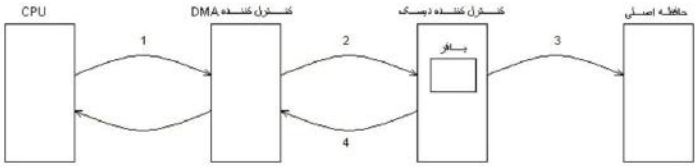
پردازنده یک فرمان I/O را از جانب فرایندی به یک مولفه I/O صادر می کند. سپس آن فرایند قبل از ادامه، تا کامل شدن عمل I/O ، به انتظار مشغولی می گذراند.

### ۲- I/O مبتنی بر وقفه

پردازنده یک فرمان I/O را از جانب فرایند صادر می کند، سپس به اجرای دستورالعملهای بعدی ادامه می دهد و با کامل شدن عمل I/O ، با وقفه مولفه I/O مواجه می شود.

### ۳- دسترسی مستقیم به حافظه(DMA)

مولفه DMA تبادل داده ها بین حافظه اصلی و مولفه I/O را کنترل می کند. پردازنده تقاضایی برای انتقال یک بلوک از داده ها را به مولفه DMA می فرستد و فقط پس از کنترل کل بلوک، مورد وقفه قرار می گیرد.




تذکر: در روش ورودی/خروجی مبتنی بر وقفه، دستورالعملهای بعدی می توانند از همان فرایند باشند، البته به شرطی که فرایند نیازی به انتظار برای تکمیل I/O نداشته باشد. در غیر اینصورت فرایند در انتظار وقفه معلق می گردد و کار دیگری انجام می گیرد.

- روش دوم کارامدتر از روش اول است.
- برای انتقال I/O چند کلمه ای، روش DMA بسیار کارآمدتر از روشهای اول و دوم است.
- در روش اول، مولفه I/O به پردازنده وقفه نمی دهد و این مسئولیت پردازنده است که وضعیت مولفه I/O را متناوبا بررسی کند تا اتمام آن عمل را دریابد. بنابراین روش اول را می توان ورودی/خروجی بر اساس سرکشی(polling) نامید.

- مشکل ورودی/خروجی برنامه سازی شده این است که پردازنده باید برای مدت طولانی منتظر بماند تا مولفهI/O برای دریافت یا ارسال آماده شود. در مدت انتظار، پردازنده به علت اینکه مکررا وضعیت را سئوال می کند، کارایی سیستم به شدت پایین می آید.
- روش I/O Programed و I/O Interrupt پردازنده اصلی را درگیر عملیات I/O می کند.

**نگاهی کلی به سیستم عامل**

سیستم عامل برنامه ای است که اجرای برنامه های کاربردی را کنترل و به صورت رابط کاربر و سخت افـزار کامپیوتر عمل می کند. برای سیستم عامل انجام سه وظیفه را می توان در نظر گرفت:

۱-سهولت

سیستم عامل استفاده از کامپیوتر را برای کاربر ساده تر می کند.

۲- کارآمدی

سیستم عامل موجب استفاده کار آمد و بهینه از منابع سیستم کامپیوتری می شود.

۳- قابلیت رشد

سیستم عامل باید به نحوی ساخته شده باشد که توسعه آن میسر باشد.

سیستم عامل چیزی جز یک برنامه کامپیوتری نیست. این برنامه پردازنـده را بـرای اسـتفاده از سـایر منابع سیستم هدایت می کند.

**جایگاه سیستم عامل**

می توان به سخت افزار و نرم افزاری که کاربردها را برای کاربر ارائه می کند به صورت لایه ای (شکل زیـر) نگاه کرد. در این نگاه، سیستم عامل یک میانجی برای تسهیل دسترسی برنامه ساز و برنامه های کاربردی از امکانات و خدمات می باشد.

| برنامه های کاربردی |
| --- |
| برنامه های سیستمی |
| سیستم عامل |
| سخت افزار |

سخت افزار، پایین ترین سطح است که شامل سه قسمت است:

۱- دستگاه های فیزیکی (مانند سیم ها، منبع تغذیه، تراشه های مدار مجتمع)

۲- ریز معماری( شامل ثبات های درون CPU و یک مسیر داده شامل واحد محاسبه و منطق)

۳- زبان ماشین

**تذکر:** برنامه های سیستمی که در بالای سیستم عامل قرار دارند( مانند کامپایلرها، ادیتورهـا و ..)، بخشـی از سیسـتم عامـل نمی باشند، اگر چه توسط بعضی از سازندگان سیستم عامل ها عرضه می شوند.

**سیستم عامل در زمینه های زیر خدمات خود را ارائه می دهد:**

۱- ایجاد برنامه
۲- اجرای برنامه
۳- دسترسی به دستگاههای ورودی/خروجی
۴- کنترل دسترسی به پرونده ها
۵- دسترسی به سیستم عامل
۶- کشف و پاسخ به خطاها
۷- حسابداری.

- سیستم عامل از طریق هسته (Kernel) ، با منابع سخت افزاری و نرم افزاری و از طریق پوسته (shell) با کاربران، ارتباط برقرار می کند.
- بخشی از سیستم عامل که در حافظه اصلی قرار دارد را هسته سیستم عامل می گویند.
- یک سیستم عامل ممکن است به دلایل زیر در طول زمان تغییر کند:

۱- ارتقاء و انواع جدید سخت افزار
۲- خدمات جدید
۳- رفع خطا

**مولفه های سیستم عامل**

سیستم بزرگی چون سیستم عامل را باید به مولفه های (components) کوچکتری تقسیم کرد. اکثر سیستم عامل ها دارای مولفه های زیر می باشند:

**۱- مدیریت فرایند**

برخورد با بن بست، ایجاد و حذف فرایندها، تعویق و از سرگیری فرایندها، هماهنگی فرایندها.

**۲- مدیریت حافظه اصلی**

تعیین بخشهای پر حافظه، تعیین فرایندی که باید لود شود، تخصیص حافظه و آزاد سازی حافظه.

**۳- مدیریت حافظه ثانویه**

مدیریت فضای آزاد، تخصیص حافظه، زمانبندی دیسک.

**۴- مدیریت فایل**

ایجاد و حذف فایلها و دایرکتوری ها، نگاشت فایلها درحافظه ثانویه و تهیه پشتیبان.

**۵- مدیریت سیستم I/O**

مدیریت بافرها، تخصیص کانالهای I/O و دستگاهها به فرایندها

**تاریخچه سیستم های عامل**

**نسل اول: سیستم های ردیفی**

در اولین کامپیوترها(حدودا ۶۰ سال قبل)، برنامه ساز مستقیما با سخت افـزار در تـراکنش بـود و سیسـتم عاملی در کار نبود. این ماشینها از طریق چراغهای نمایش، کلیدها و نوعی دستگاه ورودی و چاپگر اجرا می شدند. مساله اصلی این سیستمها، زمانبندی و زمان نصب بود. چون در ایـن عملیـات، کـاربران بـه صـورت ردیفی و یکی پس از دیگری به کامپیوتر دسترسی داشتند، آن را پردازش ردیفی می خوانیم.

**نسل دوم : سیستم های دسته‌ای ساده**

زمانی که به خاطر زمانبندی و نصب در سیستمهای ردیفی به هدر می رفت قابل قبول نبود. بـرای افـزایش استفاده از ماشین، مفهوم سیستم عامل دسته ای به وجود آمد. اولین سیستم عامل دسته ای در اوسط دهه ۱۹۵۰ به وجود آمد. ایده اصلی، استفاده از نرم افزاری به نام ناظر بود. با اسـتفاده از سیسـتم عامـل دسـته ای، دیگر کاربر دسترسی مستقیم به ماشین ندارد و کار خود را روی کارت یا نوار به متصدی کـامپیوتر مـی دهد. اپراتور کارها را به صورت ردیفی دسته کرده و همگی را روی یک دستگاه ورودی می گـذارد تـا مـورد استفاده ناظر قرار گیرد. مساله زمانبندی و زمان تنظیم شرایط اولیه کارها به عهده ناظر است.

- با هر کار، دستورالعملهایی از زبان کنترل کار(JCL) نیز وجود دارد. زبان کنترل کار، نوعی زبان برنامـه نویسی برای فرمان دادن به ناظر می باشد.

JCL : Job Control Language

**اسپولینگ**

در سیستم های اولیه،CPU گرانترین جزء کامپیوتر بود و افزایش راندمان CPU مهمترین محرک در تکامـل سیستم های عامل بوده است. برای همین منظور، تکنیک Spooling مطرح شد. اسپولینگ I/O یک کـار را با محاسبات کار دیگر به طور همزمان انجام می‌دهد. حتی می‌تواند در حین خواندن ورودی‌هـای یـک کـار، خروجی های کار دیگر را انجام دهد. اسـپولینگ موجـب مـی شـود تـا CPU و دسـتگاههایI/O بـا سـرعت بیشتری کار کنند. استفاده از spooling در سیستم عامل دسته ای باعث افزایش سرعت می شود.

**سیستم های دسته ای Offline Spooling**

در این سیستم ها، کارها توسط دستگاه کارت خوان یک کامپیوتر آهسته و ارزان خوانده می شد و به کمک یک نوار گردان بر روی یک نوار مغناطیسی ذخیره می شدند. بعد از تشکیل یک دسته از کارها، نوار توسـط

اپراتور به CPU اصلی منتقل شده تا اجرا شروع شود. نتیجه اجرا بر روی نوار دیگری نوشته می شـد. بعـد از اجرای تمام دسته کارها، اپراتور نوار خروجی را به یک پردازنده آهسته منتقل می کرد تا عمل چـاپ انجـام شود. در این روش ارتباط پردازنده اصلی محاسباتی با کارت خوان و چاپگر، غیر مستقیم بود.

**فواید سیستم های دسته ای offline Spooling نسبت به سیستم های قبلی**

۱- افزایش راندمان پردازنده گران قیمت

۲- عملیات ساده‌تر

۳- ساده شدن استفاده از سیستم از راه دور.

**معایب سیستم های دسته ای Offline Spooling**

۱- تاخیر بین زمان تحویل کار و زمان تکمیل کار

۲- نیاز به سخت افزار اضافی

۳- عدم وجود اولویت

۴- استفاده زیاد از دستگاههای جانبی

## سیستم های دسته ای Online Spooling

با ظهور دیسک های سخت، سیستم های دسته ای متحول شدند. در این سیستم ها بلافاصـله بعـد از ورود کارها، کارت ها خوانده شده و به دیسک منتقل می شدند. زمانی که اجرای یک کار تمام می شود، سیسـتم عامل با سرعت بیشتری یک کار جدید را از روی دیسک برداشته و به حافظه اصلی لود کرده و سپس آن را اجرا می کند.

در خروجی نیز از اسپولینگ استفاده می شود. اسناد خروجی ابتدا بر روی دیسک ذخیره شده و زمـانی کـه خروجی قبلی چاپگر تمام شد، این اسناد از دیسک به بافر چاپگر منتقل می شوند.

در این تکنیک که Spooling (عملیات پیوسته، مستقیم و همزمان دستگاه های جانبی) نامیـده مـی شـود، ارتباط CPU با دستگاه های جانبی مستقیم است و نیازی به پردازنده های آهسته و نوار گردان های اضافی و حمل نوارها توسط اپراتورها نمی باشد.

SPOOL : **S**imulataneonus **P**eripheral **O**peration **O**n-**L**ine

**مزایای اسپولینگ online نسبت به offline عبارتند از :**

۱- دسترسی با اولویت

۲- گردش سریع کارها

۳- بالا بودن راندمان CPU ودستگاههای I/O

۴- امکان داشتن همزمان چند مدرک ورودی/خروجی

از اسپولینگ برای پردازش داده های راه دور می‌توان استفاده کرد. CPU داده‌ها را از طریق مسیرهای ارتباطی به چاپگر راه دور می‌فرستد یا ورودی ها را از یک کارت خوان راه دور دریافت می‌کند. در پردازش راه دورCPU دخالت ندارد و فقط مواظب است که چه هنگام کار تمام شود تا دسته دیگری از کارها را جمع آوری کند.

نرم افزار spooler کارها را دسته بندی می کند. این نرم افزار اجزاء یک کار(داده، برنامه وJCL ) را از هم جدا می کند. سپس JCL ها دستور به دستور اجرا شده و توسط سیستم اسپولینگ روی دیسک اصلی قرار می گیرند و جدولی به نام (Input SpooL Table ) ISPT ساخته می شود. هر سطر این جدول در برگیرنده اطلاعات کنترلی یکJob می باشد.

**نسل سوم: سیستم های دسته‌ای چند برنامگی**

حتی با ردیف کردن خودکار کارها به وسیله سیستم عامل ساده دسته ای، به علت کند بودن دستگاههای I/O در مقایسه با پردازنده، پردازنده اکثرا بی کار است. تمام کارهایی که وارد سیستم می‌شوند، در انبار کار (job pool) نگهداری می‌شوند. این کارها شامل فرایندهایی می‌باشند که بر روی دیسک قرار دارند و منتظر تخصیص حافظه می‌باشند. اگر چند کار آماده ورود به حافظه وجود داشته باشند، سیستم یکی را انتخاب کرده (زمانبندی کار) و آنرا به حافظه می‌آورد تا اجرا کند. همچنین اگر چند کار به طور همزمان آماده اجرا باشند، سیستم یکی ازآنها را انتخاب می‌کند و CPU را به آن تخصیص می دهد. (زمانبندی CPU). ایده چند برنامه‌ای به این صورت است که سیستم عامل در هر زمان چند کار را در حافظه نگه می‌دارد و یکی از کارهای موجود در حافظه را انتخاب کرده و اجرای آن را شروع می‌کند. اگر این کار منتظر عمل دیگری مانند آماده شدن نوار یا تکمیل یک عمل I/O باشد، CPU به اجرای کار دیگری مشغول می‌شود و اگر آن کار نیز نیاز به انتظار داشته باشد، به کار دیگری می‌پردازد و این روند ادامه می‌یابد. با پایان مدت انتظار کار اول، دوباره CPU در اختیار آن قرار می‌گیرد. این عملیات باعث می‌شود که CPU بیکار نماند.

- سیستم های دسته‌ای برای اجرای برنامه‌های بزرگ که نیاز به محاوره کمی دارند، مناسب می‌باشند.

- **مشکلات سیستم دسته‌ای از نظر کاربر**

۱- کاربر برای به دست آوردن نتایج مجبور است از کارتهای کنترلی استفاده کند.

۲- مراجعه بعدی در کارهای چند مرحله‌ای ممکن است به نتایج مراحل قبلی وابسته باشد.

۳- برنامه نویس نمی‌تواند برنامه در حال اجرا را تغییر دهد.(دیباگ برنامه ها ایستا می باشد)

- ممکن است حافظه را برای نگهداری ۳،۴ یا تعداد بیشتری از برنامه ها گسترش دهیم و پردازنده به تمام آنها بپردازد. این عمل را چند برنامه ای یا چند وظیفه ای می گویند که موضوع اصلی سیستمهای عامل امروزی است.

**نسل چهارم : سیستم های اشتراک زمانی (Time sharing)**

سیستم های اشتراک زمانی(چندبرنامگی تعاملی) توسعه منطقی سیستم های چند برنامه‌ای می‌باشند. در سیستم عامل اشتراک زمانی، وقت پردازنده بین کارها به اشتراک گذاشته می شود. برای اجرای چند کار، پردازنده بین آنها سوئیچ می‌کند و کاربران می‌توانند با برنامه‌های در حال اجرا تعامل داشته باشند. در

سیستم های اشتراک زمانی ارتباط کاربر با سیستم به صورت محاوره ای (interactive یا online) است.
سیستم های اشتراک زمانی، هزینه استفاده محاوره‌ای از سیستم های کامپیوتری را معقول کرده‌اند و با استفاده از زمانبندی CPU و چند برنامه‌ای، به هر کاربر زمان محدودی را تخصیص داده و هر کاربر حداکثر یک برنامه در حافظه دارد.
در سیستمهای اشتراک زمانی، چون سیستم به سرعت از برنامه‌ای به برنامه دیگر می‌رود، هر کاربر فکر می‌کند که یک کامپیوتر اختصاصی در اختیار اوست، در حالی که یک کامپیوتر بین همه کاربران مشترک است.

خصوصیات سیستم های اشتراک زمانی عبارتند از:

۱- پیچیده‌تر از چند برنامه‌ای هستند.
۲- برای مواردی که نیاز به زمان پاسخ کوتاه است، استفاده می‌شوند.
۳- امکان چند برنامگی را فراهم می سازد.
۴- وجود یک سیستم فایل ضروری است.
۵- در main frame قابلیت چند کاربره و چند برنامگی را فراهم می کند.

چندبرنامه ای دسته ای و اشتراک زمانی هر دو از چندبرنامگی استفاده می کنند.

**تفاوتهای اصلی چندبرنامه ای دسته ای با اشتراک زمانی**

| | چندبرنامه ای دسته ای | اشتراک زمانی |
|---|---|---|
| **هدف اصلی** | حداکثر استفاده از پردازنده | حداقل زمان پاسخ |
| **منبع دستورات به سیستم عامل** | دستورالعملهای JCL که همراه کار ارائه شده است. | فرمانهایی که از پایانه وارد می شود. |

تذکر: سیستم های زیر در کتاب سیلبرشاتس آورده شده است:

۱- سیستم های موازی (یا چند پردازنده ای)

۲- سیستم های عامل توزیع شده (گسترده)

۳- سیستم های عامل بی‌درنگ (Real time)

**سیستم های موازی (یا چند پردازنده ای)**

سیستم هایی که بیش از یک پردازنده در آنها وجود دارد را چند پردازنده ای(Multi Processor) می‌نامند. پردازنده‌ها در این سیستم با یکدیگر ارتباط نزدیکی دارند و از گذرگاه آدرس، ساعت و گاهی حافظه و دستگاههای جانبی به طور اشتراکی استفاده می‌کنند. این سیستم‌ها را اتصال محکم (tightly coupled) نیز می‌نامند.

**انواع سیستم های چند پردازنده‌ای**

**الف- چندپردازنده‌ای متقارن**

در سیستم چندپردازنده‌ای متقارن(Symmetric)، هر پردازنده از کپی یکسانی از سیستم عامل استفاده می‌کند که این کپی‌ها در صورت لزوم با یکدیگر ارتباط برقرار می‌کنند.

**ب- چندپردازنده‌ای نامتقارن**

در سیستم چند پردازنده‌ای نامتقارن، هر پردازنده کار خاصی را انجام می‌دهد. کنترل سیستم به عهده پردازنده اصلی می‌باشد و پردازنده‌های دیگر منتظر دستور پردازنده اصلی هستند یا کار از قبل تعیین شده‌ای دارند. این طرح رئیس/ مرئوس(master/slave) را بیان می‌کند، که پردازنده اصلی(master)، کارها را برای پردازنده‌های دیگر(slave)،زمانبندی کرده و به آنها تخصیص می‌دهد.

**مزایای سیستم متقارن نسبت به نامتقارن عبارتند از :**

۱- قابل حمل بودن روی سیستم های سخت افزاری مختلف

۲- تعادل بار سیستم به علت اجرای سیستم عامل روی چند پردازنده

**دلایل ساخت سیستم های چند پردازنده‌ای**

۱- افزایش توان عملیاتی (throughput) : تعداد کارهای انجام شده در یک واحد زمانی.

۲- افزایش قابلیت اعتماد

(خرابی یک پردازنده منجر به از کار افتادن سیستم نمی شود بلکه از سرعت آن می کاهد.)

۳- مقرون به صرفه بودن از نظر اقتصادی

**سیستم های عامل توزیع شده (گسترده)**

در این سیستم ها، محاسبات بین چند پردازنده توزیع می‌شود. هر پردازنده، حافظه و ساعت مخصوص به خود را دارد و از طریق خطوط ارتباطی با یکدیگر مرتبطاند. همچنین پردازنده‌ها از نظر اندازه و عملکرد با یکدیگر فرق دارند. سیستم های توزیعی را سیستم های ارتباط ضعیف (Loselycoupled) نیز می‌گویند.

**دلایل ساخت سیستم های توزیعی**

**الف- اشتراک منابع :** کاربری دریک سایت می‌تواند از چاپگری در سایت دیگر استفاده کند.

**ب- قابلیت اعتماد:** خرابی یک کامپیوتر، دیگری را تحت تاثیر قرار نمی‌دهد.

**ج- افزایش سرعت محاسبات:** توزیع یک محاسبه در بین چند سایت

**د- ارتباطات**

**سیستم های عامل بی‌درنگ (Real time)**

سیستم عامل بی‌درنگ(بلادرنگ)، نوعی ازسیستم عامل‌های همه منظوره می‌باشد و در صورتی به کار گرفته می‌شود که برای عملکرد یک پردازنده نیاز به زمان دقیقی باشد. یک سیستم بی‌درنگ وقتی درست کار می‌کند که در محدودیت زمانی مشخص، نتایج مورد انتظار را تولید کند. یعنی پردازش باید در محدودیت زمانی خاص انجام شود وگرنه سیستم از کار می‌افتد.

سیستم های نظامی، تزریق سوخت اتومبیل، کنترل کننده‌های لوازم خانگی،کنترل صنعتی و تصویرسازی پزشکی نمونه هایی از سیستم های بی درنگ می باشند. سیستم های بی درنگ بر دو نوع نرم و سخت می باشند.

**ویژگی های سیستم های بی درنگ نرم (Soft Real Time)**

۱- دارای محدودیت زمانی نسبتا دقیقی می باشند.

۲- در پروژه های علمی پیشرفته استفاده می‌شوند.

۳- با سیستم های دیگر قابل ترکیب می باشند.

۴- مهلت زمانی را پشتیبانی نمی کنند. (از آنها در کنترل صنعتی و روبات‌ها استفاده نمی شود.)

**ویژگی های سیستم های بی درنگ سخت (Hard Real Time)**

۱- دارای محدودیت زمانی دقیقی می باشند و کارهای بحرانی به موقع انجام می‌شوند.

۲- تمام تاخیرهای موجود درسیستم باید از بین بروند.

۳- از ویژگیهای پیشرفته سیستم عامل استفاده نمی‌شود.

۴- با سیستم های اشتراک زمانی برخورد (Conflict) دارند و با یکدیگر ترکیب نمی شوند.

۵- هیچکدام از سیستم های همه منظوره موجود از عملکرد بی‌درنگ سخت پشتیبانی نمی‌کنند.

**انواع سیستم عامل از نظرساختاری**

انواع سیستم عامل از نظرساختاری عبارتند از:

۱- یکپارچه (Monolitic)

۲- لایه ای (Layered)

۳- ماشین مجازی (Virtual Machine)

۴- مشتری– خدمتگزار (Client /Server)

**ساختار یکپارچه**

ساده ترین ساختار برای سیستم عامل است که در DOS از آن استفاده می شود. در این ساختار واسط ها و سطوح عملکرد به خوبی از هم تفکیک نشده اند و برنامه های کاربردی می توانند به روالهای I/O دستیابی داشته باشند و مستقیما بر روی مونیتور یا دیسک بنویسند.

**ساختار لایه‌ای**

با اعمال خاصیت پیمانه ای بودن به سیستم عامل، سیستم عامل می تواند کنترل بیشتری بر روی کامپیوتر و برنامه های کاربردی داشته باشد. یکی از بهترین روشها برای این کار، روش لایه ای است که سیستم عامل را به تعدادی لایه تقسیم می کند و هر کدام بر روی یکدیگر قرار می گیرند. لایه پایینی(شماره صفر)، سخت افزار است و لایه بالایی واسط کاربر است. طراحی و پیاده سازی ساختار لایه ای ساده است.

**ویژگی های ساختار لایه ای**

۱- هر لایه فقط از توابع و خدمات لایه های پایین تر استفاده می کند که باعث ساده شدن خطایابی می شود.

۲- خطایابی از لایه پایین شروع می شود و سپس لایه بعدی خطا یابی می شود. اگر در حین خطایابی لایه ای، خطایی یافت شود، می دانیم که خطا در همان لایه است، زیرا لایه های پایینی قبلا خطا یابی شده اند.

۳- هر لایه می تواند عملیات، ساختمان داده ها و سخت افزار را از لایه های بالاتر مخفی کند.

**مشکلات روش لایه ای عبارتند از:**

۱- هر لایه نیاز به برنامه ریزی دقیقی دارد، چون فقط می تواند از لایه های زیرین استفاده کند.

۲- پایین بودن کارایی نسبت به روشهای دیگر.(تولید سربار زیاد)

**مزایای ساختار لایه ای نسبت به یکپارچه عبارتند از:**

۱- قابلیت گسترش بیشتر ۲- خطایابی ساده تر ۳- مدیریت ساده تر

**ساختار ماشین مجازی (Virtual Machine)**

یک سیستم کامپیوتری دارای لایه های سخت افزار، هسته و برنامه های سیستم می باشد. برنامه های سیستم که در بالای هسته قرار دارند، می توانند از فراخوانیهای سیستم و دستورات سخت افزاری استفاده کنند. بعضی سیستمها از این الگو تبعیت می کنند و حتی برنامه های سیستم می توانند از طریق برنامه های کاربردی فراخوانی شوند و برنامه های سیستم می توانند آنچه را که در زیر آنها قرار دارد را مشاهده نمایند، به طوری که گویی برنامه های کاربردی بخشی از خود ماشین هستند. این شیوه لایه ای را ماشین مجازی می نامند که یک کپی از کامپیوتر در اختیار هر فرایند قرار می دهد. اجرا شدن سیستم عامل DOS تحت محیط ویندوز با استفاده از ایده ماشین مجازی انجام گرفته است.

**مزایای ساختار ماشین مجازی**

۱- امکان اجرای هم زمان چند سیستم عامل مختلف.

۲- شبیه سازی دستورات ماشینی که کامپیوتر فاقد آن است.

۳- امکان برنامه نویسی به زبان های غیر اسمبلی.

۴- تست سیستم عامل جدید.

با تغییراتی در روش ماشین مجازی، ساختاری به نام **Exokernels** به وجود آمده است. در این ساختار نیازی به لایه نگاشت نمی باشد و همچنین برنامه ای در پایین ترین لایه در مد هسته اجرا شده که منابع را به ماشین های مجازی تخصیص داده و بررسی می کند که ماشینی از منابع ماشین دیگر استفاده نکند.

**ساختار مشتری – خدمتگزار (Client – Server)**

در این ساختار، اکثر وظایف سیستم عامل در سطح کاربر انجام می شود و هسته از طریق پیام ها بین مشتری/خدمتگزار ارتباط برقرار می سازد. ایده طراحی این ساختار، کمینه کردن هسته و انتقال کدها به لایه های بالاتر می باشد. مزایای ساختارهای مشتری– خدمتگزار عبارتند از:

۱- طراحی ساده تر سیستم عامل

۲- استفاده در سیستم های توزیعی

۳- عدم خرابی کل سیستم در صورت خرابی یک سرور

# کنکور ارشد

### ( مهندسی کامپیوتر - دولتی ۸۶)

۱- کدام یک از دستور العمل های زیر، فقط قادر به اجرا در مد کرنل (Kernel Mode) است؟

۱) خواندن ساعت سیستم ۲) خواندن PSW

۳) تنظیم زمان سیستم ۴) نوشتن در ثبات دستور العمل

پاسخ: گزینه ۳ درست است.

### ( مهندسی IT – دولتی ۸۶)

۲- مدیریت منابع حافظه ای زیر با کدام عامل است؟

۱- ثبات ها ۲- حافظه پنهان (cache) ۳- حافظه اصلی ۴- فضای دیسک

**(تذکر: منظور از خودکار، سخت افزاری است)**

۱) ۱- کاربر ۲- سیستم عامل ۳- سیستم اصلی ۴- سیستم عامل

۲) ۱- کامپایلر ۲- خودکار ۳- سیستم عامل ۴- سیستم عامل

۳) ۱- کاربر ۲- سیستم عامل ۳- کامپایلر ۴- خودکار

۴) ۱- خودکار ۲- خودکار ۳- سیستم عامل ۴- سیستم عامل یا خودکار

پاسخ: گزینه ۲ درست است.

حافظه اصلی و فضای دیسک توسط سیستم عامل مدیریت می شود.

### ( مهندسی IT – دولتی ۸۸)

۳- فرض کنید زمان محاسبات یک فرآیند 200 سیکل CPU باشد. در ضمن عملیات I/O در حال انجام برای یک فرآیند دیگر از طریق DMA در حال انجام بوده و پس از 100 سیکل CPU، پایان عملیات I/O توسط یک وقفه به سیستم اطلاع داده شود. اگر زمان اجرای ISR را 10 سیکل CPU فرض کنیم، کل عملیات مذکور چه مقدار از زمان سیستم را به خود اختصاص می دهند؟ (زمان هر سیکل cpu را معادل یک واحد زمانی فرض کنید.)

۱) 210 واحد زمانی ۲) کمتر از 210 واحد زمانی

۳) 310 واحد زمانی ۴) بیشتر از 210 واحد زمانی

پاسخ: گزینه ۴ درست است.

فرایند اول برای انجام محاسباتش به اندازه 200 سیکل به CPU نیاز دارد و فرایند دوم معادل 100 سیکل مشغول انجام عملیات I/O می باشد و چون از DMA استفاده می شود، این دو عمل همزمان انجام شده و به 200 سیکل نیاز است.

همچنین 10 سیکل نیز صرف اجرای ISR فرایند دوم می شود. البته در هنگام کار با DMA و انجام I/O ، گاهی گذرگاه در اختیار DMA قرار می گیرد و زمان اجرای فرایند اول بیشتر از 200 سیکل خواهد شد. بنابراین کل عملیات به بیش از 210 سیکل نیاز دارد.

**(مهندسی کامپوتر- آزاد ۷۹)**

۴- جدول زیر زمان های لازم برای ورود، محاسبه و خروج 3 کار را در یک سیستم دسته ای با Spooling نشان می دهد. حداقل کل زمان مصرفی برای اجرای هر 3 کار به شرط آنکه ترتیب ورود کارها تعیین کننده ترتیب پردازش و ترتیب خروجی آن ها باشد، چقدر است؟

| | زمان ورود | زمان پردازش | زمان خروج |
|---|---|---|---|
| Job1 | 5 | 4 | 1 |
| Job2 | 2 | 2 | 3 |
| Job3 | 5 | 3 | 2 |

۱) 27 ۲) 10 ۳) 20 ۴) 17

پاسخ: گزینه ۴ درست است.

در سیستم دسته ای که از تکنیک Online Spooling استفاده می کند، امکان پردازش یک کار، همزمان با I/O سایر کارها وجود دارد. طبق جدول، ورود job1 ، 5 واحد زمانی طول می کشد. در این زمان پردازشی انجام نمی شود. با پایان ورود Job1 ، پردازش آن در لحظه 5 شروع شده و تا لحظه 9 ادامه می یابد. در زمان5 تا 7 نیز job2 از ورودی دریافت می شود. اجرای job1 در لحظه 9 تمام شده و به خروجی فرستاده می شود و ....

بنابراین داریم:

| | ورود کارها | پردازش | خروج |
|---|---|---|---|
| Job1 | 0..5 | 5..9 | 9..10 |
| Job2 | 5..7 | 9..11 | 11..14 |
| Job3 | 7..12 | 12..15 | 15..17 |

در لحظه 11 تا 12 پردازشی انجام نمی شود، چون اجرای job2 تمام شده و ورود job3 کامل نشده است.

در لحظه 10 تا 11 خروجی نداریم، چون خروجی job1 تمام شده ولی پردازش job2 تمام نشده است.

در لحظه 14 تا 15 خروجی نداریم، چون خروجی job2 تمام شده ولی پردازش job3 تمام نشده است.در لحظه 17 آخرین کار خارج می شود.

**(مهندسی کامپیوتر- آزاد ۷۱)**

**۵- در یک سیستم عامل گسترده (Distributed Operating System) ، کدام یک از موارد زیر درست نیست؟**

۱) چندین CPU مستقل از نظر جغرافیایی با هم فاصله دارند و تحت یک سیستم عامل کار می کنند.

۲) در تبادل پیغام کاربران می بایست آدرس ماشین های یکدیگر را بدانند.

۳) محل استقرار فایل ها در کنترل کاربران نمی باشد.

۴) قابلیت اطمینان یک سیستم عامل گسترده از یک سیستم عامل متمرکز بیش تر است.

پاسخ: گزینه ۲ جواب است.

در سیستم عامل های توزیع شده، می توان منابع را با نام آنها (بدون نیاز به دانستن آدرس آن ها)، فراخوانی کرد. به این خاصیت "شفافیت" می گویند.

**(مهندسی IT- آزاد ۹۰)**

۶- امکان انتقال یک کلمه توسط واحد DMA در کدام نقطه از یک چرخه دستورالعمل وجود ندارد؟

۱) پس از اجرای دستورالعمل ۲) پس از کدگشایی دستورالعمل

۳) پس از واکشی عملوندها ۴) پس از ذخیره نتایج

پاسخ: گزینه ۳ جواب است.

امکان انتقال یک کلمه توسط واحد DMA پس از واکشی عملوندها وجود ندارد.

فصل ۲

# فرآیند

در این فصل مفهوم فرایند و حالات فرایند بررسی می شوند. همچنین نخ نیز توضیح داده می شود.

## فرایند و حالات آن

به برنامه در حال اجرا، فرآیند(process) می گویند. یک فرایند می تواند در یکی از حالتهای زیر باشد:

۱- **آماده(READY) :** فرایندی که وقتی به آن فرصت داده شود، برای اجرا آماده باشد.

۲- **اجرا (Running) :** فرایندی که هم اکنون در حال اجر می باشد.

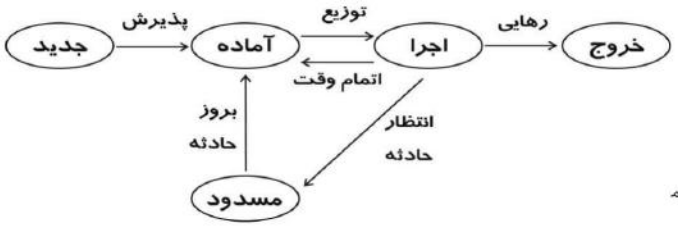
۳- **مسدود (Blocked) :** فرایندی که تا بروز حادثه‌ای (مثل اتمام یک عمل I/O)، نمی‌تواند اجرا شود.

۴- **جدید (New) :** فرایندی که هم اکنون گرفته شده است، اما هنوز جزء فرایندهای قابل اجرای سیستم عامل پذیرفته نشده باشد.

۵- **خروج (Terminated) :** فرایندی که اجرای آن پایان یافته است و یا اجرای آن قطع شده و از مجموعه فرایندهای قابل اجرای سیستم عامل خارج شده است.

تذکر : به حالت مسدود، حالت بسته یا انتظار نیز می گویند.

در شکل زیر این حالات، نشان داده شده است:



## تغییر حالات ممکن

### ۱- جدید به آماده

اگر سیستم عامل آمادگی گرفتن یک فرایند دیگر را داشته باشد، فرایند موجود در حالت جدید را به حالت آماده می‌برد.

### ۲- آماده به اجرا

سیستم عامل یکی از فرایندهای موجود در حالت آماده که وقت اجرای آن فرا رسیده است را انتخاب کرده و از حالت آماده به اجرا می‌برد. به این عمل توزیع ( Dispatch ) می گویند.

### ۳- اجرا به خروج

وقتی که فرایند جاری اعلام پایان کند، سیستم عامل آن را از حالت اجرا به خروج می‌برد.

### ۴- اجرا به مسدود

وقتی فرایندی چیزی را بخواهد که به خاطر آن باید منتظر بماند، سیستم عامل آن فرایند را از حالت اجرا به مسدود می‌برد. به این عمل بلوکه شدن می گویند. مثلاً فرایندی یک عمل I/O را شروع کند که قبل از تکمیل نیاز به بخش مشترکی از حافظه مجازی را داشته باشد که فوراً فراهم نباشد یا منتظر دریافت ورودی از یک فرایند دیگر باشد.

### ۵- مسدود به آماده

وقتی حادثه‌ای که فرایند منتظر آن بوده است رخ دهد، از حالت مسدود به آماده می‌رود. به این عمل wake up می گویند.

### ۶- اجرا به آماده

متداول ترین دلیل انتقال یک فرایند از حالت اجرا به آماده، اتمام زمان مجاز برای اجرای فرایند جاری در سیستم عامل‌های چند برنامه‌ای می باشد.

### ۷- آماده به خروج

در بعضی سیستمها، یک پدر می تواند هر لحظه که بخواهد، فرایند فرزند خود را پایان دهد و یا با پایان یافتن فرایند پدر، ممکن است همه فرایندهای فرزند آن نیز پایان یابند.

### ۸- مسدود به خروج

مانند توضیحات آماده به خروج است.

### ۹- تهی به آماده

فرایند جدیدی را برای اجرای یک برنامه ایجاد می کند.

وجود وضعیت مسدود باعث افزایش بهره وری پردازنده می شود. چون وقتی فرایند در حال اجرا نیاز بـه I/O پیـدا مـی کند، به حالت Blocking منتقل شده و فرایند آماده دیگری به قسمت اجرا منتقل مـی شـود، تـا در حـد امکـان CPU بیکار نماند.

**دسته بندی فرایندها**

**بطور کلی فرایندها به دو دسته تقسیم می‌شوند:**

**۱- محدود به CPU (CPU Limited)**

بیشتر زمان کامپیوتر صرف محاسبات CPU می‌شود.

**۲- محدود به ورودی/ خروجی (I/O Limited)**

بیشتر زمان کامپیوتر صرف ورود داده‌ها و خروج اطلاعات می‌شود.

**دلایل ایجاد یک فرایند جدید**

**۱- برقراری ارتباط محاوره ای**

کاربر از طریق پایانه با سیستم ارتباط برقرار می کند.

**۲- ارائه سرویس به وسیله سیستم عامل**

سیستم عامل می تواند فرایندی را برای ارائه خدمتی از طرف برنامه کاربر ایجاد نماید، بدون اینکه کاربر ناچار به انتظـار باشد.

**۳- زایش توسط یک فرایند موجود**

به منظور بهره گیری از توازی با تفکیک، برنامه کاربر می تواند ایجاد فرایندهای جدیدی را دیکته کند.

**۴-کار دسته ای جدید**

سیستم عامل با جریانی از کارهای دسته ای روبه رو می باشد. وقتی برای گرفتن یک کار جدید آماده است، دنباله بعدی از فرمانهای کنترل کار را می خواند.

**عملیات ایجاد فرایند**

۱- تخصیص یک شناسه یکتا به فرایند جدید ۲- تخصیص فضا برای فرایند

۳- مقدار دهی اولیه PCB ۴- برقراری پیوندهای لازم

۵- ایجاد و گسترش ساختمان داده های ممکن دیگر.

## دلایل پایان یک فرایند

| | |
|---|---|
| **پایان طبیعی** | فراخوانی یک سرویس سیستم عامل برای بیان تکمیل اجرایش |
| **سقف زمانی** | در فرایند محاوره ای، مقدار زمانی که از آخرین ورودی کاربر گذشته است. |
| **گذشت زمان** | انتظار زیادتر از حد برای بروز یک حادثه مشخص |
| **نبود حافظه** | نیاز به حافظه ای بیش از آنچه که سیستم می تواند فراهم کند. |
| **تجاوز از حدود** | مراجعه به محلهای غیر مجاز در حافظه |
| **خطای حفاظت** | تلاش برای دسترسی به منبعی که مجاز به استفاده از آن نیست. |
| **خطای محاسباتی** | مانند تقسیم بر صفر یا تلاش برای ذخیره عددی بزرگتر از ظرفیت سخت افزاری |
| **خطای ورودی/خروجی** | مانند پیدا نکردن یک فایل |
| **دستورالعمل نامعتبر** | تلاش برای اجرای دستورالعملی که وجود ندارد |
| **دستورالعمل ممتاز** | تلاش برای اجرای دستورالعملی که مخصوص سیستم عامل است |
| **استفاده نامناسب از داده** | داده با نوع نامناسب یا بدون مقدار اولیه |
| **دخالت سیستم عامل** | به دلایلی مانند بن بست |
| **پایان یافتن پدر** | با پایان یک فرایند، فرایندهای فرزند آن نیز پایان داده شوند |
| **درخواست پدر** | فرایند حق پایان دادن به هر یک از فرایندهای فرزند خود را دارد. |

## بلوک کنترل فرایند (PCB)

هر فرایند در سیستم بوسیله یک ساختاری به نام بلوک کنترل فرایند (Process Control Bolock ) مشخص می‌شود. این بلوک مهمترین و محوریترین ساختمان داده در سیستم عامل است که تمام اطلاعات مورد نیاز سیستم عامل در مورد یک فرایند را در بردارد.

**PCB شامل موارد زیر است:**

۱- **حالت فرایند :** یک فرایند می تواند در یکی از حالات جدید، آماده، اجرا، انتظار و غیره باشد.

۲- **شمارنده برنامه :** شامل آدرس دستور بعدی که باید اجرا شود.

۳- **اطلاعات زمانبندی CPU :** شامل اولویت فرایند و اشاره گر به صف زمانبندی.

۴- **اطلاعات مدیریت حافظه :** شامل مقدار ثباتهای پایه و حد، جدولهای صفحه یا قطعه.

۵- **اطلاعات حسابرسی** : میزان استفاده از پردازنده، محدودیتهای زمانی، شماره فرایند.

۶- **اطلاعات وضعیت I/O :** شامل لیست دستگاههای I/O تخصیص داده شده به فرایند

### تفاوت تعویض حالت با تعویض فرایند

تعویض حالت و تعویض فرایند(تعویض متن) دو مفهوم مجزا هستند. ممکن است تعویض حالت بدون تغییر حالت فرایندی که در حال اجراست، صورت گیرد. تعویض فرایند که متضمن تغییر حالت است، در مقایسه با تعویض حالت، به تلاش بیشتری نیاز دارد.

### گامهای که در یک تعویض فرایند کامل وجود دارد:

۱- ذخیره سازی متن پردازنده

۲- بهنگام سازی بلوکهای کنترل فرایندی که هم اکنون در حالت اجراست.

۳- انتقال بلوک کنترل فرایند مربوط به این فرایند به صف مناسب

۴- انتخاب فرایند دیگری برای اجرا

۵- بهنگام کردن بلوک کنترل فرایند مربوط به فرایند انتخاب شده

۶- بهنگام سازی ساختمان داده های مدیریت حافظه

۷- بارگذاری مجدد متن پردازنده

### فرایند معلق

فرایند معلق، فرایندی است که فوراً آماده اجرا نیست. فرایند معلق توسط عاملی مانند سیستم عامل، خودش یا فرایند پدر، در حالت معلق قرار گرفته. تا وقتی عامل تعلیق فرمان ندهد، نمی توان فرایند را از حالت معلق خارج کرد.

### دلایل معلق کردن یک فرایند

**۱- مبادله :**برای آوردن فرایندی که آماده اجراست، سیستم عامل نیاز به آزاد کردن حافظه کافی دارد.

۲- **ترتیب زمانی:** اجراء به طور دوره ای و به تعلیق رفتن آن هنگام انتظار برای اجرای بعدی

**۳- درخواست کاربر محاوره ای :** به منظور اشکالزدایی یا استفاده از منابع

**۴- دلایل دیگر سیستم عامل :** معلق کردن یک فرایند مضنون

**۵- درخواست فرایند پدر :** به طور نمونه فرایند A ، فرایند B را تولید کرده تا عمل خواندن از یک فایل را انجام دهد و فرایند B در هنگام خواندن با خطایی مواجه شود و آنرا به فرایند A گزارش دهد، فرایند A در این حالت برای رسیدگی به علت خطا، فرایند B را به حالت معلق می‌برد.

**نمودار تغییرحالت فرایند با حالات معلق**

وقتی هیچ یک از فرایندهای موجود در حافظه اصلی در حالت آماده نیستند، سیستم عامل یکی از فرایندهای مسدود را از حافظه اصلی خارج و به صف فرایندهای معلق در روی دیسک می برد (مبادله) و فرایند دیگری را از صف معلق به حافظه اصلی می آورد یا درخواست فرایند جدیدی را می پذیرد و اجرا با این فرایند جدید ادامه می یابد.

وجود وضعیت معلق، موجب اجرای فرایند جدید حتی در صورت پر بودن حافظه اصلی می شود.

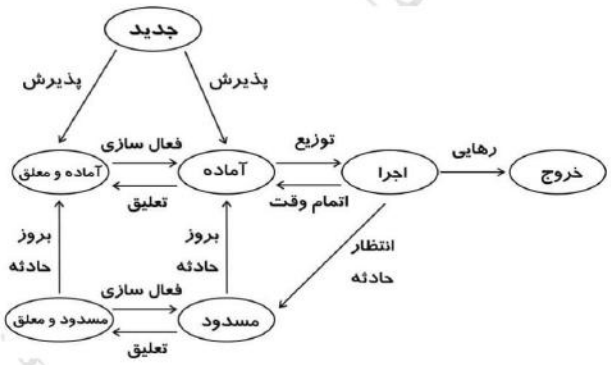
**بنابراین حالت معلق را به حالتهای گفته شده، اضافه می کنیم:**

**۱- مسدود و معلق(Suspend-Wait)**

فرایند مورد نظر در حافظه ثانوی و منتظر حادثه ای است.

**۲- آماده و معلق (Suspend-Ready)**

فرایند مورد نظر در حافظه ثانوی و به محض لود شدن در حافظه اصلی آماده اجراست.



## حالات ممکن در این نمودار

**۱- مسدود به مسدود و معلق**

وقتی هیچ فرایند آماده ای وجود نداشته باشد، فرایند مسدودی به خارج برده می شود تا جا برای فرایند دیگری که مسدود نیست فراهم شود. این تغییر حالت حتی در مواردی که فرایندهای آماده هم وجود دارد می تواند جهت حفظ کارایی انجام شود.

**۲- مسدود و معلق به آماده و معلق**

وقتی حادثه ای که یک فرایند مسدود و معلق منتظر آن بوده است رخ دهد به حالت آماده و معلق می‌رود.

**۳- آماده و معلق به آماده**

وقتی که هیچ فرایند آماده ای در حافظه اصلی نباشد، سیستم عامل یک فرایند آماده و معلق را به حالت آماده می آورد. همچنین ممکن است فرایند موجود در حالت آماده و معلق دارای اولویت بیشتری نسبت به همه فرایندهای آماده باشد که در این حالت فرایند به حالت آماده، آورده می شود.

**۴- آماده به آماده و معلق**

به طور معمول سیستم عامل ترجیح می دهد یک فرایند مسدود را به جای فرایند آماده، به حال تعلیق در آورد. ولی در صورتی که راهی برای خالی کردن حافظه اصلی نباشد، یک فرایند آماده را به تعلیق در می آورد. سرانجام ممکن است سیستم عامل یک فرایند آماده ولی با اولویت کم را به جای فرایند مسدودی که اولویتش بیشتر است و گمان می کند بزودی آماده می شود به حالت معلق می برد.

**۵- مسدود و معلق به مسدود**

اگر اولویت فرایندی که در صف مسدود و معلق است از اولویت همه فرایندهای موجود در صف آماده و معلق بیشتر باشد و سیستم عامل گمان کند که حادثه ای که این فرایند مسدود، منتظر آن بوده است به زودی رخ دهد، آن را به حالت مسدود می آورد. البته باید مقداری از حافظه اصلی خالی باشد تا آوردن یک فرایند مسدود به حافظه نسبت به یک فرایند آماده معقول به نظر برسد.

**۶- اجرا به آماده و معلق**

معمولا با پایان زمان منظور شده برای فرایند جاری، فرایند به حالت آماده منتقل می شود. در اینحالت اگر این فرایند به خاطر فرایند با اولویت بیشتری قبضه شود، سیستم عامل می تواند فرایند جاری را مستقیما به صف آماده و معلق منتقل

کند تا بخشی از حافظه اصلی آزاد شود.

**۷- مختلف به خروج**

ممکن است یک فرایند از هر حالتی به حالت خروج منتقل شود.

انواع زمانبندها

کلید چند برنامگی زمانبندی است. زمانبندی بر روی کارایی سیستم اثر می‌گذارد، زیرا مشخص می‌کند کدام فرایندها منتظر مانده و کدام فرایندها به جلو بروند. انواع زمانبندی برای پردازنده عبارت است از:

۱- **زمانبند بلند مدت** (Long Term Scheduler)

تصمیم گیری در مورد افزودن به مجموعه فرایندها برای اجرا.

۲- **زمانبند میان مدت**(Middle Term Scheduler)

تصمیم گیری در مورد افزودن به تعداد فرایندهایی که بخشی یا تمام آنها در حافظه اصلی است.

۳- **زمانبند کوتاه مدت** ( Short Term Scheduler)

تصمیم گیری در مورد اینکه کدام یک از فرایندهای موجود در حافظه اصلی، برای اجرا توسط پردازنده انتخاب شود.

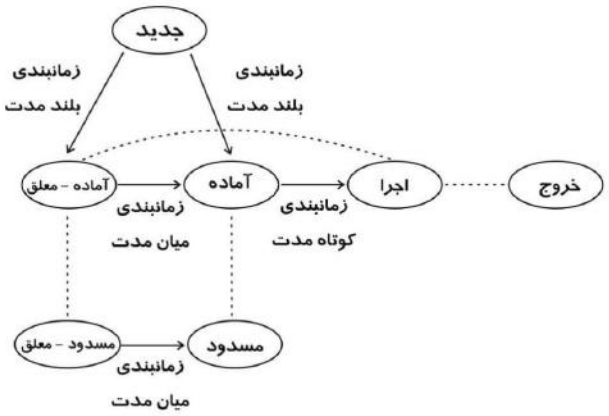
۴- **زمانبندی ورودی/خروجی**

تصمیم می گیرد که کدام درخواست I/O فرایندها به وسیله یک دستگاه I/O موجود انجام بگیرد.

- وظیفه فعال سازی و تعلیق فرآیندها بر عهده زمانبند میان مدت(Medium-term scheduler) می باشد.
- زمانبند میان مدت، فرایندی را از حافظه اصلی حذف و به حافظه جانبی می برد. این فرایند بعدا می تواند به حافظه اصلی لود شود. این الگو را مبادله (swapping) می گویند.
- ایده اصلی زمانبندی میان مدت، این است که می تواند فرایندی را از حافظه حذف کند و درجه چند برنامگی را کاهش دهد.
- زمانبند بلند مدت، ترکیب خوبی از فرایندهای I/O limited و CPU Limited ، انتخاب می کند.
- نام دیگر زمانبند بلند مدت، زمانبند کار است.
- نام دیگر زمانبند کوتاه مدت، زمانبند پردازنده است.
- زمانبند بلند مدت نسبتاً دفعات کمی اجرا می‌شود، زمانبند میان مدت نسبتاً دفعات بیشتری به اجرا در می‌آید و زمانبند کوتاه مدت، بیشترین دفعات اجرا را دارد.

**تذکر**: اکثر سیستم های اشتراک زمانی فاقد زمانبند بلند مدت می باشند.

**نمودار تغییر حالت فرایند همراه با زمانبندی**




**مثال**

اگر ۱۰ میلی ثانیه طول بکشد تا فرایندی انتخاب گردد که ۱۰۰ میلی ثانیه اجرا شود، آنگاه چند درصد از زمان CPU ، صرف زمانبندی کار می شود (به هدر می رود)؟

$$\frac{10}{10+100}\times 100 = \%9$$

■

## توزیع کننده (Dispatcher)

توزیع کننده، پیمانه ای است که کنترل را به پردازنده ای می دهد که توسط زمانبند کوتاه مدت انتخاب شده است. این عمل شامل موارد زیر است:

۱- تعویض بستر(Context Switch)

۲- تغییر به حالت کاربر

۳- پرش به محل مناسبی در برنامه کاربر و آغاز مجدد آن برنامه.

## نخ (Thread)

تا به حال، مفهوم فرایند را به صورت واحدی معرفی کرده ایم که دارای دو خصوصیت تملک منبع و توزیع وقت پردازنده است. این دو خصوصیت می توانند مستقلا توسط سیستم عامل رسیدگی شوند. برای تمایز این دو خصوصیت، به توزیع وقت پردازنده، نخ و به تملک منبع، فرایند می گویند.

مهم ترین مزیت استفاده از سیستم های چند نخی این است که در فرایند تک نخی، هر گاه فراخوان

سیستمی مسدود کننده ای اجرا شود، یا یک نقص صفحه رخ دهد، کل فرایند مسدود می شود. بـه عنـوان مثال یک برنامه صفحه گسترده را در نظر بگیرید و فرض کنید کاربری می خواهد به طور دائمـی و تعـاملی مقادیر را تغییر دهد. می دانیم که در برنامه صفحه گسترده، وابستگی تابعی بین سلول هـای مختلـف بایـد حفظ شود. بنابراین اگر سلولی تغییر کند، محاسبات زیادی انجام می گیرد. اگر فقط یک نـخ وجـود داشـته باشد، وقتی برنامه منتظر دریافت ورودی است، محاسبات نمی تواند ادامه یابـد، چـون فراخـوان سیسـتمی دریافت ورودی، کل فرایند را مسدود کرده است. راه حل این است که از یک نخ برای خواندن ورودی کـاربر و از نخ دیگری برای بهنگام سازی صفحه گسترده استفاده شود.

**مثال**

بسیاری از صفحات وب، شامل چند تصویر کوچک هستند که مرورگـر بایـد یـک اتصـال جداگانـه را بـرای دریافت هر تصویر برقرار کند که زمان زیادی لازم دارد. اگر در مرورگر از روش چند نخی استفاده شود، هـر کدام از نخ ها می تواند به طور همزمان با نخ های دیگر تصویر مربوط به خود را درخواست کند.

به عنوان مثالی دیگر از کاربرد نخ ها، جهت حفاظت در مقابل قطع برق می توان یک کلمه پـرداز را طـوری طراحی کرد که هر یک دقیقه بافر RAM را روی دیسک بنویسد. یک نـخ مـی توانـد تنهـا بـرای گـرفتن پشتیبان دوره ای ایجاد شود و خودش را مستقیما با سیستم عامل زمانبندی کند.

**نخها و فرایندها می توانند ۴ حالت را بوجود آورند:**

۱- یک فرایند- یک نخ

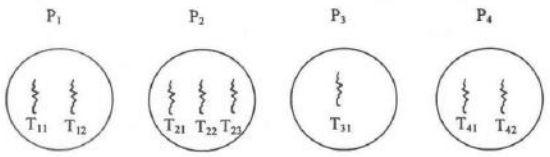
۲- یک فرایند – چند نخ

۳- چند فرایند- یک نخ در هر فرایند

۴- چند فرایند– چند نخ در هر فرایند

**مثال**

شکل زیر سیستمی با چهار فرآیند را نشان می دهد که فرآیند اول شامل دو نخ $T_{11}$ و $T_{12}$ می باشد. فرایند دوم شامل سه نخ است و ...




- حالات اصلی نخ عبارت است از: اجرا، آماده و مسدود. (نخ حالت معلق ندارد)
- تمام نخ های یک فرایند، در حالت و منابع آن فرایند شریک هستند.
- اگر نخ یک فرآیند در حال اجرا باشد و آن فرآیند به حالت خروج برود، نخ نمی تواند به اجرا ادامه دهد.
- تعویض متن میان دو نخ متعلق به دو فرایند جداگانه، مثل این است که تعویض متن فرایند رخ داده است. یعنی عمل سوئیچینگ مابین دو نخ متعلق به دو فرآیند جداگانه از نوع تعویض متن فرآیندی است.
- نخ در برنامه نویسی، بخشی از یک فرایند یا برنامه بزرگتر می باشد.
- ساختار نخ گاهی در محیط تک پردازنده ای، برای ساده کردن ساختار برنامه ای که منطقا اعمال متعددی را انجام می دهد، نیز مفید است.
- با تقسیم یک کار به چند نخ، برنامه ساز می تواند کنترل زیادی روی مولفه ای بودن آن کاربرد و تنظیم وقت حوادث مربوط به آن داشته باشد.

**پیاده سازی نخ**

سه روش برای پیاده سازی و مدیریت نخ وجود دارد:

۱- نخ های سطح کاربر

۲- نخ های سطح هسته

۳- روش های ترکیبی

**نخ های سطح کاربر**

در این روش تمام عملیات راهبری در فضای آدرس کاربر انجام می شود. در این روش هزینه ایجاد نخ با هزینه تخصیص حافظه برای بر پا سازی پشته (stack) نخ تعیین می شود. تعویض متن نخ اغلب با تعداد کمی از دستورات العمل ها انجام می شود و فقط کافی است محتوای ثبات های پردازنده برای نخ فعلی، ذخیره شود و سپس مقادیر قبلا ذخیره شده مربوط به نخی که به آن سوئیچ می شود، بار گذاری شوند. در تعویض متن نخ ها نیازی به فلاش کردن TLB ، تغییر نگاشت های حافظه و حسابداری پردازنده نمی باشد.

**مزایای کتابخانه نخ سطح کاربر**

۱- ایجاد و حذف سریع نخ ها

۲- همگام سازی سریع نخ ها

۳- تعویض متن کم هزینه و سریع

**نقاط ضعف نخ های سطح کاربر**

۱- اگر یک نقص صفحه (page fault) برای یک نخ رخ دهد، کل فرایند و در نتیجه همه نخ های درون آن به اشتباه مسدود می شود.

۲- اگر نخی یک فراخوان سیستمی بلوکه کننده را صدا بزند، کل فرایند و همه نخ های درون آن نیز به اشتباه مسدود می شوند.

۳- چون سیستم عامل از وجود نخ ها آگاه نمی باشد، نخ ها را بین چندین پردازنده به خوبی پخش و زمان بندی نمی کند.

**نخ های سطح هسته**

اگر نخ ها در هسته سیستم عامل پیاده سازی شوند، مشکلات روش قبل رخ نمی دهند. اما چون هر عملیات نخ باید توسط هسته انجام شود، نیاز به یک فراخوانی سیستمی دارد و هزینه بالا می رود. تعویض متن نخ در این روش ممکن است به اندازه تعویض متن فرایند پر هزینه باشد.

تذکر: مزایا و معایب نخ های سطح کاربر و نخ های سطح هسته، بر عکس یکدیگرند.

**روش ترکیبی**

برای غلبه بر مشکلات دو روش قبل، نخ های سطح کاربر و سطح هسته را با تلفیق می کنیم. برای این کار فرایندهای سبک وزن(LWP) ایجاد می کنیم. LWP دورن متن یک فرایند اجرا شده و به ازای هر فرایند ممکن است چند LWP وجود داشته باشد. سیستم علاوه بر داشتن LWP ها، یک بسته نخ سطح کاربر برای عملیات ایجاد و حذف نخ ها به برنامه های کاربردی ارائه می دهد. بسته نخ کاملا در فضای کاربر پیاده سازی می شود و تمام عملیات روی نخ ها بدون دخالت هسته انجام می شود. برنامه های کاربردی چند نخی با ایجاد نخ ها و سپس انتساب هر نخ به یک LWP ساخته می شوند.

در این روش اگر نخی فراخوان سیستمی مسدود کننده ای را اجرا کند، اجرا از مد کاربر به مد هسته تغییر می کند ولی همچنان در متن LWP جاری ادامه می یابد. در جایی که LWP جاری نتواند ادامه دهد، متن به LWP دیگری تعویض می شود که باعث یک تعویض متن برای بازگشت به مد کاربر نیز می شود.

- در روش ترکیبی، فراخوانی های سیستمی از نوع مسدود با تامین هم روندی حمایت می شوند.

- تغییر متن (context switch) در نخ های یک فرایند، اشاره گر پشته (SP) را تغییر می دهد ولی ثبات ها و جداول مدیریت حافظه را تغییر نمی دهد و ارتباطی با TLB ندارد.

- TLB یک سخت افزار جستجوی سریع، کوچک و پنهان است. در صفحه بندی تعداد اندکی از ورودیهای جدول صفحه در TLB قرار می گیرد.

(TLB : Translation Lookaside Buffers)

# کنکور ارشد

**( مهندسی کامپیوتر – دولتی ۸۶)**

۱- در مدل انتقال حالت (transition state) یک سیستم عامل که اجازه داده می شود یک فرآیند از حافظه اصلی بیرون کشیده شود و در زمان مناسب دوباره به حافظه اصلی بازگردانده شود (مانند unix)، کدام یک از انتقال ها نمی تواند مجاز باشد؟

(sleep swapped: جابجا شده، به خواب رفته)

(ready to run in memory : آماده برای اجرا درحافظه)

۱) از sleep swapped به ready to run in memory

۲) از ready to run in memory به ready to run swapped

۳) از ready to run swapped به ready to run in memory

۴) از sleep in memory به sleep swapped

پاسخ: گزینه ۱ جواب است.

با توجه به نمودار حالات فرایند، گزینه یک، یعنی انتقال از "مسدود و معلق" به "آماده" ، مجاز نمی باشد.به تعاریف زیر توجه کنید:

sleep swapped : مسدود و معلق

sleep in memory : مسدود

ready to run in memory : آماده

ready to run swapped : آماده و معلق

طبق این تعاریف گزینه ها را می توان به صورت زیر نوشت:

۱) از "مسدود و معلق" به "آماده"

۲) از "آماده" به "آماده و معلق"

۳) از "آماده و معلق" به "آماده"

۴) از "مسدود" به "مسدود و معلق"

**(مهندسی کامپیوتر- دولتی ۹۲)**

۲- کدم گزینه درباره مدل های چند نخی درست نیست؟

۱) مدل های یک به یک و چند به چند، توانایی بکارگیری بهتر از پردازنده ها/هسته ها را دارند.

۲) مدل چند به یک نسبت به مدل یک به یک، از کارایی کمتری برخوردار است.

۳) در مدل های یک به یک و چند به یک، امکان همزمانی کامل بین نخ ها وجود دارد.

۴) مدل یک به یک نسبت به مدل چند به یک، از همزمانی بیشتری برخوردار است.

پاسخ: گزینه ۳ نادرست است.

چون در مدل چند به یک، امکان همزمانی کامل بین نخ ها وجود ندارد.

توضیح بیشتر: بین نخ ها و فرایندها می تواند ۴ رابطه زیر برقرار باشد:

| | | |
|---|---|---|
| یک به یک | UNIX | هر نخ اجرا، یک فرایند یکتا با فضای آدرسها و منابع خویش است. |
| چند به یک | NT , Solris,OS/2 | یک فرایند، یک فضای آدرس و مالکیت پویای منابع را تعریف می کند. ممکن است نخهای متعدد در داخل آن فرایند ایجاد و اجرا گردد. |
| یک به چند | Clouds | ممکن است یک نخ از محیط یک فرایند به محیط فرایند دیگر مهاجرت کند. این موضوع حرکت یک نخ در بین سیستمهای مجزا را میسر می سازد. |
| چند به چند | TRIX | ترکیب خصوصیات موارد چند به یک و یک به چند |

در رابطه با این مدل ها می توان گفت که:

الف- توانایی بکارگیری پردازنده ها در مدل های یک به یک و چند به چند، بیشتر است.

ب- کارایی مدل یک به یک از مدل چند به یک، بیشتر است.

ج- مدل یک به یک همزمانی بیشتری نسبت به مدل چند به یک دارد.

## (مهندسی IT- دولتی ۹۱)

۳- در رابطه با مدیریت نخ (thread) کدام یک از جملات زیر صحیح است؟

(توجه : LWP مخفف Light Weight Process است و محیط اجرای نخ می باشد.)

۱) تغییر متن(Context switch) مابین نخ ها شامل : (۱) ذخیره ثبات های پردازنده مربوط به نخ بیرون رونده و بار کردن ثبات های پردازنده مربوط به نخ داخل شونده و (۲) ذخیره لیست فایل های باز شده توسط نخ است.

۲) یک نخ عادی در طول حیات خود ممکن است در LWP های متفاوتی، بخش هایی از اجرای خود را بگذراند.

۳) به ازای هر نخ، سیستم عامل یک LWP ایجاد می کند و نخ تا پایان حیات خود به آن LWP منتسب است. زمان بندی نخ می تواند توسط سیستم عامل یا کاربر انجام پذیرد.

۴) نخ مستقیما زیر نظر سیستم عامل اجرا می شود و مدیریت آن نمی تواند در سطح کاربر باشد.

پاسخ: گزینه ۲ درست است.

فرایند سبک وزن(LWP) ، نگاشتی بین نخ سطح کاربر و نخ سطح هسته می باشد. هر LWP از یک نخ سطح کاربر یا بیشتر حمایت می کند و به یک نخ سطح هسته می نگارد. LWP ها مستقلا توسط هسته زمانبندی شده و ممکن است به صورت موازی روی پردازنده های متعدد اجرا شوند.

یک نخ عادی در طول حیات خود ممکن است در LWP های متفاوتی، بخش هایی از اجرای خود را بگذراند. بنابراین گزینه ۲ درست است. گزینه ۳ نادرست است، چون تعداد LWP ها به اندازه تعداد نخ هایی است که هم اکنون با هسته درگیر هستند و سیستم عامل به ازای هر نخ، یک LWP ایجاد نمی کند. به طور نمونه اگر دو LWP موجود باشد و نخ سوم فرایند، درخواستی از هسته داشته باشد، باید منتظر بماند تا یکی از LWP های قبلی کارش با هسته پایان یابد.

## منتخبی از عناوین آموزشی منتشر شده بر روی فرادرس

| برنامه‌نویسی | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| مبانی برنامه نویسی - کلیک کنید (+) | ۳ ساعت |
| برنامه نویسی - C کلیک کنید (+) | ۱۳ ساعت |
| آموزش برنامه نویسی++C | ۲۰ ساعت |
| برنامه نویسی کاربردی سی شارپ - کلیک کنید (+) | ۱۴ ساعت |
| آموزش جامع شی گرایی در سی شارپ - کلیک کنید (+) | ۱۴ ساعت |
| برنامه نویسی جاوا - کلیک کنید (+) | ۲۳ ساعت |
| آموزش برنامه نویسی - PHP کلیک کنید (+) | ۲۸ ساعت |
| آموزش فریمورک PHP کدایگنایتر - (CodeIgniter) کلیک کنید (+) | ۷ ساعت |
| آموزش اسکریپت برنامه نویسی - jQuery کلیک کنید (+) | ۷ ساعت |
| آموزش ویژوال بیسیک دات نت - (Visual Basic.NET) کلیک کنید (+) | ۱۳ ساعت |
| آموزش تکمیلی ویژوال بیسیک دات نت - (Visual Basic.NET) کلیک کنید (+) | ۱۶ ساعت |
| آموزش برنامه نویسی با روش سه لایه به زبان - VB.Net کلیک کنید (+) | ۴ ساعت |
| برنامه نویسی اسمال بیسیک یا - Small Basic کلیک کنید (+) | ۱۶ ساعت |
| آموزش ساخت بازی ساده در ویژوال بیسیک - کلیک کنید (+) | ۲ ساعت |
| آموزش کاربردی - SQL Server کلیک کنید (+) | ۱۱ ساعت |
| آموزش آشنایی با LINQ to SQL در - C# کلیک کنید (+) | ۲ ساعت |
| آموزش برنامه نویسی با روش سه لایه به زبان سی شارپ - کلیک کنید (+) | ۴ ساعت |
| آموزش برنامه نویسی تحت شبکه با سی شارپ در قالب پروژه - کلیک کنید (+) | ۱ ساعت |
| آموزش Cryptography در دات نت - کلیک کنید (+) | ۳ ساعت |

| برنامه‌نویسی (ادامه از صفحه قبل) | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| آموزش قفل نرم افزاری در سی شارپ از طریق رجیستری | ۴ ساعت |
| آموزش ساخت اپلیکیشن کتاب و کار با داده‌ها در اندروید - کلیک کنید (+) | ۱۳ ساعت |
| آموزش ارتباط با دیتابیس سمت سرور در اندروید - کلیک کنید (+) | ۱۴ ساعت |
| آموزش ساخت روبات و کنترل آن با اندروید - کلیک کنید (+) | ۱۶ ساعت |
| آموزش ساخت اپلیکیشن دیکشنری صوتی دو زبانه با قابلیت تشخیص صوت کاربر - کلیک کنید (+) | ۷ ساعت |
| آموزش مدیریت بانک اطلاعاتی اوراکل - کلیک کنید (+) | ۹ ساعت |
| آموزش مدیریت بانک اطلاعاتی اوراکل پیشرفته - کلیک کنید (+) | ۷ ساعت |
| آموزش راه اندازی اوراکل ۱۲ cدر لینوکس | ۱ ساعت |
| آموزش دیتاگارد در اوراکل - کلیک کنید (+) | ۳ ساعت |
| برنامه نویسی متلب - کلیک کنید (+) | ۹ ساعت |
| متلب برای علوم و مهندسی - کلیک کنید (+) | ۱۴ ساعت |
| برنامه نویسی متلب پیشرفته - کلیک کنید (+) | ۷ ساعت |
| طراحی رابط های گرافیکی (GUI) در متلب - کلیک کنید (+) | ۸ ساعت |
| آموزش برنامه نویسی R و نرم‌افزار - RStudioکلیک کنید (+) | ۷ ساعت |
| آموزش تکمیلی برنامه نویسی R و نرم‌افزار - RStudio کلیک کنید(+) | ۵ ساعت |
| آموزش برنامه نویسی پایتون ۱ - کلیک کنید (+) | ۲۰ ساعت |
| آموزش برنامه نویسی پایتون ۲ - کلیک کنید (+) | ۵ ساعت |
| آموزش گرافیک کامپیوتری با - OpenGL کلیک کنید(+) | ۱۶ ساعت |

| راه اندازی و مدیریت وبسایت‌ها و سرورها | |
|---|---|
| **عنوان فرادرس** | **مدت زمان تقریبی** |
| آموزش برنامه نویسی - PHP کلیک کنید (+) | ۲۸ ساعت |
| آموزش فریمورک PHP کدایگنایتر - (CodeIgniter) کلیک کنید (+) | ۷ ساعت |
| آموزش طراحی وب با - HTML کلیک کنید (+) | ۳ ساعت |
| آموزش طراحی وب با - CSS کلیک کنید (+) | ۵ ساعت |
| آموزش پروژه محور HTML و - CSS کلیک کنید - کلیک کنید (+) | ۴ ساعت |
| آموزش جاوا اسکریپت - (JavaScript) کلیک کنید (+) | ۹ ساعت |
| آموزش کار با - cPanel کلیک کنید (+) | ۱ ساعت |
| آموزش مدیریت هاست با - DirectAdmin کلیک کنید - کلیک کنید (+) | ۱ ساعت |
| راه اندازی سایت و کار با وردپرس - کلیک کنید (+) | ۷ ساعت |
| راه اندازی فروشگاه دیجیتال با وردپرس و - Easy Digital Downloads کلیک کنید (+) | ۱ ساعت |
| آموزش راه اندازی سایت شخصی با وردپرس - کلیک کنید (+) | ۱ ساعت |
| آموزش ترجمه قالب وردپرس - کلیک کنید (+) | ۲ ساعت |
| آموزش راه اندازی سایت خبری با وردپرس - کلیک کنید (+) | ۲ ساعت |

| علوم کامپیوتر | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| ساختمان داده‌ها -کلیک کنید (+) | ۱۰ ساعت |
| آموزش ساختمان داده‌ها (مرور – تست کنکور ارشد) - کلیک کنید (+) | ۲۰ ساعت |
| آموزش نظریه زبان‌ها و ماشین‌ها -کلیک کنید (+) | ۹ ساعت |
| آموزش نظریه زبان‌ها و ماشین (مرور – تست کنکور ارشد) - کلیک کنید (+) | ۸ ساعت |
| آموزش سیستم های عامل - کلیک کنید (+) | ۱۱ ساعت |
| آموزش سیستم عامل (مرور اجمالی و تست کنکور) - کلیک کنید (+) | ۱۲ ساعت |
| آموزش پایگاه داده‌ها -کلیک کنید (+) | ۸ ساعت |
| آموزش پایگاه داده‌ها (مرور – تست کنکور ارشد) - کلیک کنید (+) | ۵ ساعت |
| آموزش طراحی و پیاده سازی زبان های برنامه سازی - کلیک کنید (+) | ۱۰ ساعت |
| آموزش طراحی و پیاده سازی زبان های برنامه سازی (مرور – تست کنکور ارشد) - کلیک کنید (+) | ۱۲ ساعت |
| آموزش روش های حل روابط بازگشتی - کلیک کنید (+) | ۴ ساعت |
| آموزش روش تقسیم و حل در طراحی الگوریتم - کلیک کنید (+) | ۲ ساعت |
| آموزش ذخیره و بازیابی اطلاعات - کلیک کنید (+) | ۸ ساعت |
| آموزش ساختمان گسسته با رویکرد حل مساله - کلیک کنید (+) | ۱۶ ساعت |
| آموزش جامع مدارهای منطقی - کلیک کنید (+) | ۱۰ ساعت |
| آموزش معماری کامپیوتر با رویکرد حل مسأله - کلیک کنید (+) | ۲۰ ساعت |
| آموزش ساختمان گسسته (مرور و حل تست های کنکور کارشناسی ارشد) - کلیک کنید (+) | ۱۲ ساعت |
| آموزش طراحی الگوریتم - کلیک کنید (+) | ۸ ساعت |
| آموزش شبکه های کامپیوتری ۱ - کلیک کنید (+) | ۱۹ ساعت |

| علوم کامپیوتر (ادامه از صفحه قبل) | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| آموزش نظریه گراف و کاربردها - کلیک کنید (+) | ۱۴ ساعت |
| آموزش نتورک پلاس - (Network+) کلیک کنید (+) | ۱۰ ساعت |
| آموزش مدل سازی UML با نرم‌افزار - Rational Roseکلیک کنید (+) | ۳ ساعت |
| آموزش پردازش ویدئو - کلیک کنید (+) | ۳ ساعت |
| پردازش تصویر در متلب - کلیک کنید (+) | ۱۶ ساعت |
| آموزش پردازش تصویر با - OpenCV کلیک کنید (+) | ۱۰ ساعت |

| هوش مصنوعی | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| الگوریتم ژنتیک در متلب - کلیک کنید (+) | ۱۴ ساعت |
| الگوریتم PSO در متلب - کلیک کنید (+) | ۱۰ ساعت |
| الگوریتم ازدحام ذرات (PSO) گسسته باینری - کلیک کنید (+) | ۲ ساعت |
| ترکیب الگوریتم ژنتیک و PSO در متلب - کلیک کنید (+) | ۱ ساعت |
| حل مسأله فروشنده دوره گرد با استفاده از الگوریتم ژنتیک - کلیک کنید (+) | ۲ ساعت |
| الگوریتم مورچگان در متلب - کلیک کنید (+) | ۶ ساعت |
| الگوریتم رقابت استعماری در متلب - کلیک کنید (+) | ۱۳ ساعت |
| طراحی سیستم های فازی عصبی یا ANFIS با استفاده از الگوریتم های فرا ابتکاری و تکاملی - کلیک کنید (+) | ۲ ساعت |
| الگوریتم فرهنگی یا Cultural Algorithm در متلب - کلیک کنید (+) | ۲ ساعت |

| هوش مصنوعی (ادامه از صفحه قبل) | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| شبیه سازی تبرید یا Simulated Annealing در متلب - کلیک کنید (+) | ۴ ساعت |
| جستجوی ممنوع یا Tabu Search در متلب - کلیک کنید (+) | ۲ ساعت |
| الگوریتم کرم شب تاب یا Firefly Algorithm در متلب - کلیک کنید (+) | ۱ ساعت |
| بهینه سازی مبتنی بر جغرافیای زیستی یا BBO در متلب - کلیک کنید (+) | ۲ ساعت |
| جستجوی هارمونی یا Harmony Search در متلب - کلیک کنید (+) | ۲ ساعت |
| کلونی زنبور مصنوعی یا Artificial Bee Colony در متلب - کلیک کنید (+) | ۳ ساعت |
| الگوریتم زنبورها یا Bees Algorithm در متلب - کلیک کنید (+) | ۲ ساعت |
| الگوریتم تکامل تفاضلی - کلیک کنید (+) | ۱ ساعت |
| الگوریتم بهینه سازی علف هرز مهاجم یا IWO در متلب - کلیک کنید (+) | ۲ ساعت |
| الگوریتم بهینه سازی مبتنی بر و یادگیری یا - TLBO کلیک کنید (+) | ۱ ساعت |
| الگوریتم بهینه سازی جهش قورباغه یا SFLA در متلب - کلیک کنید (+) | ۴ ساعت |
| بهینه سازی چند هدفه در متلب - کلیک کنید (+) | ۱۹ ساعت |
| بهینه سازی مقید در متلب - کلیک کنید (+) | ۹ ساعت |
| شبکه های عصبی مصنوعی در متلب - کلیک کنید (+) | ۲۸ ساعت |
| آموزش کاربردی شبکه های عصبی مصنوعی - کلیک کنید (+) | ۹ ساعت |
| آموزش استفاده از شبکه عصبی مصنوعی با نروسولوشن - کلیک کنید (+) | ۳ ساعت |
| شبکه عصبی GMDH در متلب - کلیک کنید (+) | ۴ ساعت |
| شبکه های عصبی گازی به همراه پیاده سازی عملی در متلب - کلیک کنید (+) | ۳ ساعت |
| طبقه بندی و بازشناسی الگو با شبکه های عصبی LVQ در متلب - کلیک کنید (+) | ۳ ساعت |
| آموزش پیاده سازی الگوریتم های تکاملی و فرابتکاری در سی شارپ - کلیک کنید (+) | ۸ ساعت |

| آمار و داده کاوی | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| گنجینه فرادرس های یادگیری ماشین و داده کاوی - کلیک کنید (+) | ۸۸ ساعت |
| گنجینه فرادرس های محاسبات هوشمند - کلیک کنید (+) | ۷۱ ساعت |
| آموزش یادگیری ماشین - کلیک کنید (+) | ۲۴ ساعت |
| داده کاوی یا Data Mining در متلب - کلیک کنید (+) | ۲۴ ساعت |
| آموزش داده کاوی در - RapidMiner کلیک کنید (+) | ۲ ساعت |
| آموزش وب کاوی - کلیک کنید (+) | ۱۷ ساعت |
| شبکه های عصبی مصنوعی در متلب - کلیک کنید (+) | ۲۸ ساعت |
| آموزش کاربردی شبکه های عصبی مصنوعی - کلیک کنید (+) | ۹ ساعت |
| شبکه عصبی GMDH در متلب - کلیک کنید (+) | ۴ ساعت |
| شبکه های عصبی گازی به همراه پیاده سازی عملی در متلب - کلیک کنید (+) | ۳ ساعت |
| طبقه بندی و بازشناسی الگو با شبکه های عصبی LVQ در متلب - کلیک کنید (+) | ۳ ساعت |
| خوشه بندی با استفاده از الگوریتم های تکاملی و فراابتکاری - کلیک کنید (+) | ۳ ساعت |
| تخمین خطای کلاسیفایر یا - Classifier کلیک کنید (+) | ۲ ساعت |
| انتخاب ویژگی یا - Feature Selection کلیک کنید (+) | ۲ ساعت |
| انتخاب ویژگی با استفاده از الگوریتم های فرا ابتکاری و تکاملی - کلیک کنید (+) | ۴ ساعت |
| کاهش تعداد رنگ تصاویر با استفاده از روش های خوشه بندی هوشمند - کلیک کنید (+) | ۱ ساعت |
| آموزش پردازش سیگنال های واقعی در متلب - کلیک کنید (+) | ۴ ساعت |
| مبانی و کاربردهای راهبرد تلفیق داده یا - Data Fusion کلیک کنید (+) | ۹ ساعت |
| آمار و احتمال مهندسی - کلیک کنید (+) | ۱۳ ساعت |
| آزمون‌های فرض مربوط به میانگین جامعه نرمال در - SPSS کلیک کنید (+) | ۳ ساعت |

| آمار و داده کاوی (ادامه از صفحه قبل) | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| آموزش محاسبات آماری در اکسل - کلیک کنید (+) | ۲ ساعت |
| آموزش کنترل کیفیت آماری - کلیک کنید (+) | ۵ ساعت |
| آموزش کنترل کیفیت آماری با - SPSS کلیک کنید (+) | ۲ ساعت |
| آموزش مدل سازی معادلات ساختاری با - Amos کلیک کنید (+) | ۷ ساعت |
| تجزیه و تحلیل اطلاعات با نرم‌افزار - SASکلیک کنید (+) | ۴ ساعت |

| مهندسی برق | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| طراحی دیجیتال با استفاده از وریلوگ یا - Verilog کلیک کنید (+) | ۷ ساعت |
| آموزش جامع مدارهای منطقی - کلیک کنید (+) | ۱۰ ساعت |
| آموزش مروری طراحی و پیاده سازی مدارات منطقی - کلیک کنید (+) | ۴ ساعت |
| آموزش میکروکنترلر AVR و نرم‌افزار - CodevisionAVRکلیک کنید (+) | ۴ ساعت |
| آموزش تکمیلی میکروکنترلر AVR و نرم‌افزار - CodevisionAVRکلیک کنید (+) | ۴ ساعت |
| آشنایی باPLC های ساخت شرکت های Omron و - Keyence کلیک کنید (+) | ۶ ساعت |
| میکروکنترلر PIC با کامپایلر - CCS کلیک کنید (+) | ۹ ساعت |
| آموزش تحلیل و طراحی مدارات الکترونیکی با - Proteus کلیک کنید (+) | ۳ ساعت |
| آموزش شبیه سازی و تحلیل مدارهای الکتریکی و الکترونیکی با پی اسپایس (PSpice) -کلیک کنید (+) | ۳ ساعت |
| آموزش مقدماتی - ADS کلیک کنید (+) | ۳ ساعت |
| آموزش تکمیلی آنالیز مدار با نرم‌افزار - ADSکلیک کنید (+) | ۲ ساعت |

| مهندسی برق (ادامه از صفحه قبل) | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| آموزش تحلیل ریاضی مدارات الکتریکی با - OrCAD کلیک کنید (+) | ۲ ساعت |
| آموزش شبیه سازی مدارات الکترونیکی با - Orcad Capture کلیک کنید (+) | ۲ ساعت |
| آموزش برنامه نویسی آردوینو ( - (Arduinoکلیک کنید (+) | ۸ ساعت |
| آموزش تکمیلی برنامه نویسی آردوینو - (Arduino) کلیک کنید (+) | ۷ ساعت |
| آموزش طراحی برد مدار چاپی به کمک نرم‌افزار - Altium Designer کلیک کنید (+) | ۷ ساعت |
| آموزش مبانی ربات های برنامه پذیر - کلیک کنید (+) | ۵ ساعت |
| آموزش ساخت روبات و کنترل آن با اندروید - کلیک کنید (+) | ۱۶ ساعت |
| آموزش مدارهای الکتریکی ۱ - کلیک کنید (+) | ۹ ساعت |
| آموزش مدارهای الکتریکی ۲ - کلیک کنید (+) | ۱۱ ساعت |
| آموزش سیستم های کنترل خطی - کلیک کنید (+) | ۱۰ ساعت |
| آموزش مکاترونیک کاربردی ۱ - کلیک کنید (+) | ۱۳ ساعت |
| آموزش کامسول (مباحث منتخب) - کلیک کنید (+) | ۳ ساعت |
| آموزش سینماتیک مستقیم و معکوس ربات‌ها -کلیک کنید (+) | ۳ ساعت |
| آموزش تجزیه و تحلیل سیگنال‌ها و سیستم ها - کلیک کنید (+) | ۲۷ ساعت |
| آموزش متلب با نگرش تحلیل آماری، تحلیل سری های زمانی و داده های مکانی - کلیک کنید (+) | ۸ ساعت |
| پردازش سیگنال های دیجیتال با استفاده از نرم‌افزار متلب - کلیک کنید (+) | ۴ ساعت |
| شبیه سازی سیستم با سیمیولینک - کلیک کنید (+) | ۴ ساعت |
| آموزش سیستم های قدرت در سیمولینک و متلب - کلیک کنید (+) | ۱۱ ساعت |
| آنالیز پایداری و کنترل سیستم های قدرت با استفاده از جعبه ابزارهای نرم‌افزار متلب - کلیک کنید (+) | ۲ ساعت |
| آشنایی با SimPowerSystems در شبیه سازی سیستم های قدرت - کلیک کنید (+) | ۳ ساعت |

| مهندسی برق (ادامه از صفحه قبل) | |
|---|---|
| عنوان آموزش | مدت زمان تقریبی |
| شبیه سازی ماشین های الکتریکی در تولباکس های Simulink و SimPowerSystem در نرم‌افزار متلب - کلیک کنید (+) | ۴ ساعت |
| آموزش الکترونیک قدرت – شبیه سازی در متلب و سیمولینک - کلیک کنید (+) | ۸ ساعت |
| آموزش شبیه سازی عملکرد انواع ماشین های الکتریکی در سیمولینک متلب - کلیک کنید (+) | ۱۰ ساعت |
| برنامه های پاسخگویی بار - کلیک کنید (+) | ۴ ساعت |
| آموزش نرم‌افزار ETAP برای تحلیل سیستم های قدرت - کلیک کنید (+) | ۲۱ ساعت |
| آموزش مقدماتی نرم‌افزار GAMSبرای حل مسائل بازار برق - کلیک کنید (+) | ۵ ساعت |
| آموزش پخش بار اقتصادی (دیسپاچینگ اقتصادی) در - GAMS کلیک کنید (+) | ۲ ساعت |
| کاربرد فازی در سیستم های قدرت - کلیک کنید (+) | ۲ ساعت |
| آموزش نرم‌افزار - HFSSکلیک کنید (+) | ۵ ساعت |
| طراحی آنتن مایکرواستریپ به کمک نرم‌افزار - HFSSکلیک کنید (+) | ۱ ساعت |
| آموزش طراحی و شبیه سازی آنتن های SIW با - HFSS کلیک کنید (+) | ۱ ساعت |
| آموزش بررسی کامل آنتن های مایکرواستریپ و طراحی آن توسط - CST کلیک کنید (+) | ۳ ساعت |
| آموزش تجزیه سیگنال به مولفه های مود ذاتی یا Empirical Mode Decomposition -کلیک کنید (+) | ۴۰ دقیقه |
| نمونه برداری و بازسازی اطلاعات در سیستم های کنترل دیجیتال - کلیک کنید (+) | ۳ ساعت |
| بررسی پاسخ ورودی پله در شناسایی فرآیندهای صنعتی - کلیک کنید (+) | ۱ ساعت |
| مدل سازی و شناسایی سیستم های دینامیکی با استفاده از مدل ARX و شبکه فازی عصبی - ANFIS کلیک کنید (+) | ۱ ساعت |
| طراحی و تنظیم ضرایب کنترل کننده PID با منطق فازی - کلیک کنید (+) | ۲ ساعت |
| آموزش کنترل سیستم چهار تانک - کلیک کنید (+) | ۲ ساعت |

فصل ۳

# زمان بندی پردازنده

هدف از زمان بندی پردازنده، تخصیص فرایندها به پردازنده در طول زمان است به گونه ای که هدف های سیستم از قبیل زمان پاسخ، توان عملیاتی و کارایی پردازنده را برآورده سازد. زمان بندی پردازنده، اساس سیستمهای عامل چند برنامه ای است. با حرکت پردازنده بین فرایندها، سیستم عامل می تواند بهره وری کامپیوتر را افزایش دهد.

**معیارهای زمان بندی**

معیارهای زمان بندی به دو دسته تقسیم می‌شوند. معیارهای از دید کاربر مانند زمان پاسخ و معیارهای از دید سیستم : مانند توان عملیاتی، بهره وری پردازنده.

**توان عملیاتی**(throughput) : تعداد فرایندهایی که در واحد زمان تکمیل می شوند.

**زمان انتظار** = " زمان خروج – زمان اجرا – زمان ورود "

**زمان برگشت** = " زمان خروج – زمان ورود "

**زمان برگشت** = " زمان انتظار + زمان اجرا "

تذکر: نام های دیگر زمان برگشت (turnaround time) عبارتنداز :

زمان اجرای کامل، زمان کل، زمان تکمیل و زمان پاسخ(response time).

- استفاده از پردازنده (بهره وری CPU) در سیستم های اشتراکی معیار مهمی است.
- به زمان مشخصی از پردازنده که برای اجرای مجموعه دستور العملهای مربوطه به CPU در نظر گرفته می شود،CBT ( CPU Brust Time) می گویند.
- از دید کاربر زمان پاسخ و از دید سیستم بهره وری پردازنده، مهم است.
- معیارهای زمان بندی گفته شده به یکدیگر وابسته هستند و بهینه سازی همه آن ممکن نمی‌باشد.

**دسته بندی سیاست های زمان بندی**

**۱- بدون قبضه کردن (non preemptive)**

در حالت بدون قبضه کردن (انحصاری)، همین که یک فرایند درحالت اجرا قرار گرفت، آنقدر به اجرا ادامه می‌دهد تا خاتمه یابد یا اینکه خودش (داوطلبانه)، برای انتظار I/O مسدود شود.

**۲- با قبضه کردن (preemptive)**

در حالت با قبضه کردن (غیر انحصاری)، فرایند در حال اجرا می‌تواند توسط سیستم عامل متوقف شود و به حالت آماده منتقل شود.

- تصمیم به قبضه کردن می‌تواند در یکی از حالات زیر انجام گیرد:
  - ۱- به صورت دوره ای براساس وقفه ساعت
  - ۲- در زمان ورود یک فرایند جدید
  - ۳- در زمانی که فرایند مسدودی به حالت آماده برود.
- سیاست هایی که با قبضه کردن همراه است سربار بیشتری را به همراه دارند ولی خدمت بهتری را ارائه می‌دهند.

**الگوریتم های زمان بندی**

الگوریتم های زمان بندی پردازنده عبارتند از:

۱- سرویس به ترتیب ورود (FCFS)

۲- نوبت گردشی (RR)

۳- کوتاهترین فرایند (SPN یا SJF)

۴- کوتاهترین زمان باقی مانده (SRT)

۵- بالاترین نسبت پاسخ (HRRN)

۶- فیدبک (FB)

و الگوریتم های زیر که در کتاب سیلبرشاتز آورده شده است.

۱ - MLFQ ۲- MLQ ۳- Priority

FCFS : First–Come First-Served
RR : Round Robin
SPN : Shortest Process Next
SRT : Shortest Remaining Time
HRRN : Highest Response Ratio Next
FB : Feed Back
MLFQ : Multi Level Feedback Queue
MLQ : Multi Level Queue

### سرویس به ترتیب ورود (FCFS)

در این الگوریتم فرایندی انتخاب می‌شود که بیشتر منتظر بوده است، یعنی زودتر CPU را درخواست کرده است. پیاده سازی این الگوریتم با یک صف (FIFO) انجام می‌شود. وقتی فرایندی وارد صف آماده می‌شود، PCB آن در انتهای صف قرار می‌گیرد و با آزاد شدن CPU ، فرایند موجود در اول صف، CPU را در اختیار خواهد گرفت و از صف حذف می‌شود.

### ویژگی های زمان بندی FCFS

۱- گرسنگی ندارد.

۲- برای فرایندهای طولانی بسیار بهتر از فرایندهای کوتاه عمل می‌کند.

۳- سربار حداقل است، چون نیازی به اطلاعات قبلی در مورد فرایندها نمی‌باشد.

۴- تابع انتخاب برابر Max (wait) است.

۵- میانگین زمان انتظار بسیار بالا می‌باشد.

۶- در یک سیستم تک پردازنده ای، روش خوبی نیست.

۷- انحصاری است.

۸- به فرایندهای I/O bound ضربه می‌زند.(چون وقتی فرایند CPU bound در حال اجراست، فرایندهای I/O bound باید منتظر باشند.)

۹- می‌تواند باعث استفاده ناکارآمد از CPU و دستگاههای I/O شود. وقتی فرایند جاری حالت اجرا را ترک می‌کند، فرایندهای آماده I/O bound به سرعت حالت اجرا را گذرانده و برای رویدادهای I/O مسدود می‌شوند. در این لحظه، اگر فرایند CPU bound نیز مسدود باشد، CPU بیکار می‌شود.

تذکر: برای بدست آوردن زمان پاسخ و زمان انتظار فرایندها از روشی به نام گانت استفاده می شود.

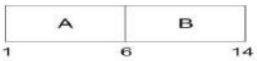
### مثال

با توجه الگوریتم FCFS ، گانت را برای فرایندهای زیر رسم نمایید.

| | زمان ورود | زمان اجرا |
|---|---|---|
| A | 1 | 5 |

| B | 4 | 8 |
|---|---|---|

حل:

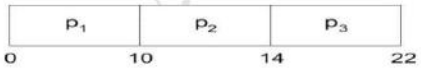



این نمودار مشخص می کند که در لحظه یک اجرای فرایند A ، بدون هیچ انتظاری شروع شده و بعد از 5 میلی ثانیه در لحظه 6 ، پایان یافته است. سپس اجرای فرایند B با 2 میلی ثانیه تاخیر شروع شده و بعد از 8 ثانیه دیگر در لحظه 14 پایان یافته است. از گانت می توان زمان پاسخ را نیز بدست آورد. کافی است که زمان ورود (که در جدول اولیه داده شده) را از زمان خروج (که در گانت مشخص است) کم کرد. بنابراین زمان پاسخ فرایند A برابر 5 میلی ثانیه (1-6) و زمان پاسخ فرایند B برابر 10 میلی ثانیه (4-14) می باشد.

**مثال**

در صورت استفاده از الگوریتم FCFS ، میانگین زمان انتظار برای سه فرایند P1 و P2 و P3 با زمانهای اجرای 10 و 4 و 8 میلی ثانیه را بدست آورید.(فرایندها در زمان صفر وارد شده اند.)

حل: ابتدا فرایند P1 اجرا شده که 10 میلی ثانیه طول می کشد، سپس فرایند P2 که 10 میلی ثانیه منتظر بوده است اجرا شده و بعد از 4 میلی ثانیه (در زمان 14) اجرای آن تمام می شود. در نهایت فرایند P3 بعد از 14 میلی ثانیه انتظار اجرا می شود. گانت آن به صورت زیر است:




زمان انتظار را به کمک رابطه "زمان خروج – زمان اجرا" محاسبه می کنیم:

$$\frac{(10-10)+(14-4)+(22-8)}{3}=\frac{0+10+14}{3}=8$$

**مثال**

در الگوریتم FCFS با توجه به فرایندهای داده شده، مطلوب است:

الف- میانگین زمان اجرا

ب- میانگین زمان انتظار

ج- میانگین زمان برگشت(پاسخ)

| نام فرایند | زمان ورود | زمان پردازش |
|---|---|---|
| A | 0 | 3 |
| B | 1 | 3 |
| C | 4 | 3 |
| D | 6 | 2 |

حل:

گانت فرایندها به صورت زیر می باشد:


الف- میانگین زمان اجرا:

$$\frac{3+3+3+2}{4}=\frac{11}{4}$$

ب- میانگین زمان انتظار:

$$\frac{(3-3-0)+(6-3-1)+(9-3-4)+(11-2-6)}{4}=\frac{0+2+2+3}{4}=\frac{7}{4}$$

ج- میانگین زمان برگشت:

$$\frac{(3-0)+(6-1)+(9-4)+(11-6)}{4}=\frac{3+5+5+5}{4}=\frac{18}{4}$$

تذکر: البته می توانستیم با توجه به رابطه زیر نیز ، میانگین زمان برگشت را محاسبه کنیم:

میانگین زمان برگشت = میانگین زمان اجرا + میانگین زمان انتظار

**نوبت گردشی (RR)**

این الگوریتم شبیه به FCFS است، با این تفاوت که زمانبند پردازنده بین فرایندها در یک صف چرخشی حرکت کرده وCPU حداکثر به مدت یک کوانتوم زمانی به هر فرایند تخصیص داده می‌شود. اگر در یک

کوآنتوم زمانی، اجرای پروسسی تمام نشود به ته صف می‌رود.

**ویژگی های روش RR**

۱- در سیستمهای اشتراک زمانی یا در سیستم پردازش تراکنش بسیار مؤثر می‌باشد.

۲- اگر برهه زمانی از زمان اجرای بلندترین فرایند بیشتر باشد، سیاست RR به FCFS تنزل می‌یابد.

۳- زمان بندی RR، غیر انحصاری (قبضه شدنی) می‌باشد.

۴- گرسنگی ندارد.

۵- عملکرد آن عادلانه است.

۶- اگر برهه زمانی خیلی کوچک باشد، توان عملیاتی ان کم است.

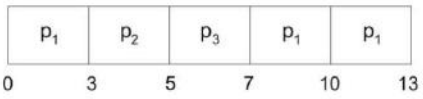
۷- برای فرایندهای کوتاه، زمان برگشت خوبی ارائه می کند.

۸- از مشکلات آن، رفتار با فرایندهای در تنگنای پردازنده در مقایسه با فرایندهای در تنگنای I/O می باشد.

**مثال**

میانگین زمان انتظار سه پردازش زیر با استفاده از سیاست زمان بندی RR با کوانتوم زمانی 3 میلی ثانیـه را بدست آورید؟ (زمان ورود= صفر ) (زمان پردازش : P1=9,P2=2,P3=2)

حل: گانت بصورت زیر می باشد:



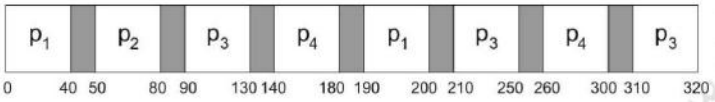

اجرای پروسس P1 در اولین برش زمانی تمام نمی‌شود و به انتهای صف مـی‌رود و پروسـس P2 و P3 قبـل از پایان برش زمانی، به پایان می‌رسند. میانگین زمان انتظار برابر است با:

$$\frac{(13-9)+(5-2)+(7-2)}{3}=4$$

**مثال**

اگر زمان اجرای پروسسهای P4,P3,P2,P1 به ترتیب 80,90,30,50 باشد و روش RR بـا کوانتـوم زمـانی 40 و زمان تعویض متن 10 استفاده شود، میانگین زمان برگشت کدام است؟ (زمان ورود= 0)

حل: گانت فرایندها به صورت زیر است:




میانگین زمان برگشت برابر است با:

$$\frac{200+80+320+300}{4}=225$$ ■

**مثال**

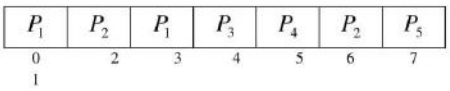
پنج فرایند با مشخصات زیر به یک سیستم با زمان بندی RR با برش زمانی q=1 وارد شوند. نمودار گانت را رسم کنید.

| | زمان ورود | زمان اجرا |
|---|---|---|
| P1 | 0 | 2 |
| P2 | 0 | 2 |
| P3 | 1 | 1 |
| P4 | 1 | 1 |
| P5 | 2 | 1 |

فرض: همیشه بین فرایندی که در لحظه t برش زمانی خود را به پایان می رساند و فرایند ورودی در لحظـه t، اولویت با فرایند قبلی موجود در سیستم است و در شرایط کاملا یکسان بین دو فرایند، اولویت با فراینـد با شماره کوچکتر است.

حل:

گانت به صورت زیر می باشد:




در لحظه 0 ، P1 و P2 وارد می شوند. پردازنده به P1 داده می شود و P2 در صف قرار می گیرد. در لحظه 1 ، پردازنده از P1 گرفته شده و بعد از P2 در صف قرار می گیرد. همچنین P3 و P4 که در همین لحظه وارد

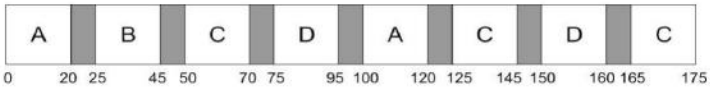
شده اند، در صف پشت سر P1 قرار می گیرند. پردازنده به P2 داده می شود. در لحظه 2 ، پردازنده از P2 گرفته شده و به انتهای صف بعد از P4 رفته و P5 که در همین لحظه وارد شده در انتهای صف بعد از P2 قرار می گیرد. پردازنده هم به P1 (فرایند ابتدای صف)داده می شود. در لحظه 3 ، اجرای P1 تمام شده و پردازنده به P3 که اول صف است داده می شود. در لحظه 4 ، اجرای P3 تمام شده و پردازنده به فرایند اول صف یعنی P4 داده می شود. در لحظه 5 ، اجرای P4 تمام شده و پردازنده به P2 داده می شود. در لحظه 6 ، اجرای P2 تمام شده و پردازنده به P5 داده می شود.

**مثال**

به کمک روش زمان بندی نوبه‌ای با کوانتوم زمانی 20 میلی ثانیه، متوسط زمان انتظار پردازه‌ها را محاسبه کنید؟ (زمان Context Switch برابر 5 میلی ثانیه است.)(زمان ورود همه فرایندها = 0 )

| فرایند | زمان اجرا |
|---|---|
| A | 40 |
| B | 20 |
| C | 50 |
| D | 30 |

حل: گانت آن به صورت زیر است:




و میانگین زمان انتظار برابر است با :

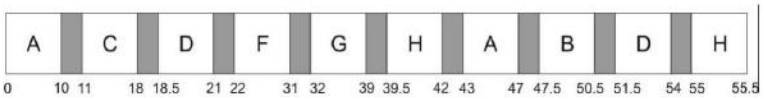
$$\frac{(120-40)+(45-20)+(175-50)+(160-30)}{4}=\frac{80+25+125+130}{4}=90$$

**مثال**

سیستمی شامل چهار فرایند است که داخل هر فرایند می تواند بیش از یک نخ اجرایی وجود داشته باشد مطابق جدول زیر مفروض است. برای فرایندها از الگوریتم Round-Robin با برش زمانی q=10 استفاده کنید. داخل هر فرایند از روش FIFO برای تعویض نخ ها استفاده می شود و تا زمانی که اجرای یک نخ تمام نشده، نوبت به نخ بعدی نمی رسد. برای تعویض فرایند 1ms و برای تعویض نخ در داخل فرایند 0.5ms زمان لازم است. نمودار گانت را رسم کنید.

| فرایند | $P_1$ | | $P_2$ | | $P_3$ | $P_4$ | |
|---|---|---|---|---|---|---|---|
| نخ | A | B | C | D | F | G | H |
| زمان اجرا | 14 | 3 | 7 | 5 | 9 | 7 | 3 |

حل: نمودار گانت به صورت زیر می باشد:



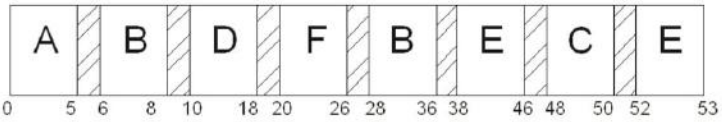

■

**مثال**

سیستمی شامل سه فرایند مطابق جدول زیر مفروض است. برای فرایندها از الگوریتم Round-Robin با برش زمانی 8 میلی ثانیه استفاده کنید. داخل هر فرایند از روش FIFO برای تعویض نخ ها استفاده می شود و تا زمانی که اجرای یک نخ تمام نشده، نوبت به نخ بعدی نمی رسد. برای تعویض فرایند 2ms و برای تعویض نخ در داخل فرایند 1ms زمان لازم است. میانگین زمان پاسخ را برای نخ های هر فرایند محاسبه کنید.

| فرایند | $P_1$ | | | $P_2$ | | $P_3$ |
|---|---|---|---|---|---|---|
| نخ | A | B | C | D | E | F |
| زمان اجرا | 5 | 10 | 2 | 8 | 9 | 6 |

حل: نمودار گانت به صورت زیر می باشد:




میانگین زمان پاسخ برای نخ های هر فرایند برابر است با:

$$A=5, B=36, C=50 \Rightarrow \frac{91}{3}$$

$$D=18, E=53 \Rightarrow \frac{71}{2}$$

$$F=26$$

■

**کوتاهترین فرایند ( SPN یا SJF)**

در این سیاست فرایندی برای اجرا انتخاب می‌شود که به کوتاهترین زمان پردازش نیاز دارد . یعنی فرایند کوتاه از روی فرایندهای بلند می‌گذرد و به ابتدای صف می‌آید.

**ویژگی های الگوریتم SPN**

۱- از معایب آن، نیاز به دانستن زمان پردازش هر فرایند است.

۲- انحصاری است.

۳- امکان گرسنگی فرایندهای طولانی وجود دارد.

۴- برای محیط های اشتراک زمانی، مناسب نیست.

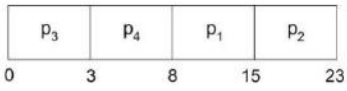
۵- میانگین زمان انتظار، کمینه است.

**مثال**

میانگین زمان انتظار را برای فرایندهای زیر به روش SJF بدست آورید؟(ورود همه در لحظه صفر)

(P1=7 , P2=8 , P3=3 , P4=5 )

حل: ترتیب اجرا برابر است با: (از کوتاهترین کار شروع می‌شود)




میانگین زمان انتظار برابر است با:

$$\frac{(15-7)+(23-8)+(3-3)+(8-5)}{4}=6.5$$

■

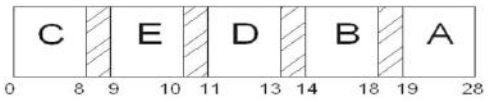
**مثال**

با توجه به جدول زیر متوسط زمان انتظار را در هر یک از روشهای SJF محاسبه کنید.

(زمان تعویض متن= ۱ میلی ثانیه)

| پروسس | زمان ورود به سیستم | زمان موردنیاز پردازش |
|---|---|---|
| A | 0 | 9 |
| B | 2 | 4 |
| C | 0 | 8 |
| D | 3 | 2 |
| E | 5 | 1 |

پاسخ: نمودار گانت به صورت زیر است:




زمان انتظار هر یک از فرایندها برابر است با:

$$A=19, B=12, C=0, D=8, E=4 \quad \Rightarrow \frac{19+12+0+8+4}{5}=\frac{43}{5}$$ ■

### پیش بینی زمان انفجار محاسباتی بعدی

در الگوریتم SJF راهی وجود ندارد که از انفجار محاسباتی بعدی آگاهی پیدا کنیم. یک روش این است که زمانبندی SJF تخمین زده شود. ممکن است طول بعدی را ندانیم، اما می توانیم اندازه اش را پیش بینی کنیم. با تخمین طول انفجار محاسباتی بعدی، می توانیم فرآیندی را انتخاب کنیم که طول انفجار محاسباتی بعدی آن کوتاهتر است. برای تخمین از رابطه $S_n = \alpha S_{n-1} + (1-\alpha)T_{n-1}$ استفاده می شود. $(0 \le \alpha \le 1)$

### مثال

سه وظیفه A، B، C را در نظر بگیرید که تاکنون n-1 بار در سیکل آماده-اجرا-مسدود طی مسیر کرده اند. زمان اجرای واقعی سیکل n-1 ام این وظایف بترتیب، 2، 4 و 6 میلی ثانیه و زمان برآورد شده برای اجرای n-1 ام آنها نیز بترتیب 4، 6 و 6 میلی ثانیه می باشد.زمان اجرای واقعی در سیکلn ام به ترتیب 5، 4 و 7 است. با فرض $\alpha = 0.5$ نحوه زمانبندی این وظایف با استفاده از الگوریتم SJF را مشخص کنید.

حل: توسط الگوریتم سالمندی(aging) ، می توان زمان اجرای کارها را به کمک رابطه $S_n = \alpha S_{n-1} + (1-\alpha)T_{n-1}$ تخمین زد. که $S_{n-1}$ زمان برآورد مرحله قبل و $T_{n-1}$ زمان واقعی مرحله قبل می باشد. با فرض $\alpha = 0.5$ داریم:

$$S_n = 0.5 \times S_{n-1} + (1-0.5) \times T_{n-1} = 0.5 \times (S_{n-1} + T_{n-1})$$

بنابراین می توان $S_n$ را به صورت زیر محاسبه کرد:

| | $S_{n-1}$ | $T_{n-1}$ | $S_n$ |
|---|---|---|---|
| A | 4 | 2 | $\frac{1}{2} \times (4+2) = 3$ |
| B | 6 | 4 | $\frac{1}{2}(6+4) = 5$ |
| C | 6 | 6 | $\frac{1}{2}(6+6) = 6$ |

با توجه به $S_n$ به دست آمده، مشخص است که ترتیب اجرا برابر است با: $A \to B \to C$

(طبق الگوریتم SJF ، کارها از کوچک به بزرگ اجرا می شوند)

حال با توجه به زمان اجرای واقعی در سیکل n ام که برابر 5، 4 و 7 می باشد، نمودار گانت را رسم می کنیم:

| A | B | C |
|---|---|---|
| 0 5 | 9 | 16 |

## کوتاهترین زمان باقیمانده ( SRT)

زمان بندی SRT یک نوع SPN با قبضه کردن است و فرایندی برای اجرا انتخاب می‌شود که انتظار می‌رود کوتاهترین زمان پردازش باقیمانده را داشته باشد. اگر فرایند جدیدی وارد صف آماده شود و زمان باقیمانده کمتری نسبت به فرایندی که درحال اجراست داشته باشد، فرایند درحال اجرا قبضه می‌شود و فرایند جدید اجرا می‌شود.

### ویژگی های الگوریتم SRT

۱- غیر انحصاری است.

۲- امکان گرسنگی برای کارهای طولانی زیاد است.

۳- بر خلاف RR، وقفه های اضافی بوجود نمی‌آید، بنابراین سربار کاهش می یابد ولی از طرف دیگر، زمان خدمت سپری شده باید ثبت شود، که ایجاد سربار می کند.

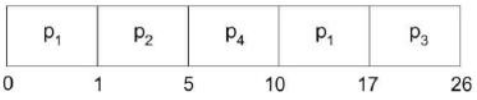
۴- زمان کل SRT نسبت به SPN بهتر است، چون کار کوتاه اولویت بیشتری نسبت به کار بلند در حال اجرا دارد.

### مثال

در صورت استفاده از الگوریتم SRT برای فرایندهای زیر، میانگین زمان انتظار را بدست آورید.

| فرایند | زمان ورود | زمان اجرا |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |

حل: اجرای فرایند P1 در لحظه صفر شروع شده و با ورود P2 در زمان 1، اجرای P1 قطع شده (چون P2 زمان اجرای کمتری نسبت به زمان باقیمانده برای P1 یعنی 7 میلی ثانیه دارد) و اجرای P2 شروع خواهد شد و بعد از اجرای P2، اجرای P4 شروع می‌شود و سپس اجرای P1 ادامه می‌یابد و در نهایت P3 اجرا خواهد شد. بنابراین گانت آن به صورت زیر است:




میانگین زمان انتظار برابر است با:

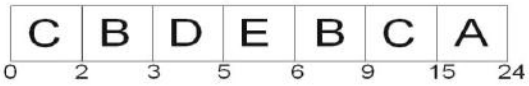
$$\frac{(17-8-0)+(5-4-1)+(26-9-2)+(10-5-3)}{4}=6.5$$

**مثال**

با توجه به جدول زیر متوسط زمان انتظار را در هر یک از روشهای SRT محاسبه کنید.

| پروسس | زمان ورود به سیستم | زمان موردنیاز پردازش |
|---|---|---|
| A | 0 | 9 |
| B | 2 | 4 |
| C | 0 | 8 |
| D | 3 | 2 |
| E | 5 | 1 |

پاسخ: نمودار گانت به صورت زیر است:




زمان انتظار هر یک از فرایندها برابر است با:

$$A=24, B=7, C=15, D=2, E=1 \Rightarrow \frac{24+7+15+2+1}{5}=\frac{49}{5}$$

■

### بالاترین نسبت پاسخ (HRRN)

در این روش برای هر فرایند نسبت پاسخی از رابطه $\frac{W+S}{S}$، بدست می‌آید، سپس فرایندی اجرا می‌شود که بالاترین نسبت پاسخ را دارد.

( W = زمان انتظاربرای CPU ) و ( S = زمان اجرا) و ( W+S = زمان پاسخ (P) )

### ویژگی های الگوریتم HRRN

۱- گرسنگی ندارد.

۲- از معایب این روش ، نیاز به تخمین زمان خدمت مورد نیاز قبل از به کارگیری می‌باشد.

۳- تابع انتخاب آن برابر $MAX(\frac{W+S}{S})$ است.

۴- سربار می‌تواند زیاد باشد.

۵- توان عملیاتی زیاد است.

۶- زمان بندی انحصاری (بدون قبضه کردن) است.

### فیدبک (FB)

اگر نتوانیم روی زمان باقیمانده برای اجرا تمرکز کنیم، بهتر است روی زمان اجرای سپری شده تمرکز کنیم. زمان بندی FB براساس قبضه کردن صورت می‌گیرد و از یک روش اولویت پویا استفاده می‌شود.فرایندی که ابتدا وارد شود به صف 0 می‌رود و هنگامی که بعد از اولین اجرا به حالت آماده می‌رود در صف 1 با اولویت کمتر قرار می‌گیرد پس از هر اجرا به صف کم اولویت تر بعدی می‌رود. فرایند کوتاه بدون انتقال به صفهای پایین تر به سرعت اجرا می‌شود، اما فرایندهای طولانی به صف‌های پایین می‌روند و فرایندهای جدیدتر و کوتاهتر به فرایندهای قدیمی‌تر و بلند تر ارجحیت دارند. در صف با کمترین اولویت از سیاست RR و درصفهای دیگر از سیاست FCFS استفاده می‌شود. (چون فرایند درصف کمترین اولویت، نمی‌تواند به صف‌های پایین تر برود.)

زمان بندی FB غیر انحصاری می باشد، چون فرایندی از صفی به صف دیگر می تواند منتقل شود.

زمان بندی های زیر در کتاب سیلبرشاتس آورده شده است.

### زمان بندی صف بازخوردی چند سطحی (MLFQ)

این زمانبند غیر انحصاری، به فرایندها اجازه می‌دهد تا از صفی به صف دیگر منتقل شوند. فرایندی که زیاد CPU را در اختیار داشته به صف کم اولویت‌تر می‌رود و فرایند I/O bound و محاوره‌ای به صف با اولویت بالاتر می‌رود و فرایندی که زیاد در صف با اولویت پایین بوده به صف با اولویت بالاتر می‌رود، یعنی کهنگی (سالمندی) از ایجاد گرسنگی جلوگیری می‌کند.

**مثال**

زمانبند صف بازخوردی چند سطحی(MLFQ) با 3 صف (0 تا 2) را در نظر بگیرید. صف اول از روش RR با کوانتوم 8 میلی ثانیه و صف دوم نیز از روش RR با کوانتوم 16 میلی ثانیه و صف اخر از روش FCFS استفاده می کند. زمانبند ابتدا تمامی فرایندهای موجود درصف 0 را اجرا می‌کند، اگر صف 0 خالی بود فرایندهای صف 1 اجرا می‌شود و اگر صف 0 و1 خالی بود فرایندهای صف 2 اجرا می‌شود.فرایند تازه وارد در صف 0 قرار می‌گیرد اگر در مدت 8 میلی ثانیه اجرای آن به پایان نرسد به انتهای صف 1 می‌رود. به فرایند موجود در ابتدای صف 1، کوانتوم زمانی 16 میلی ثانیه داده می‌شود که اگر اجرای آن در این مدت تمام نشود به صف 2 می‌رود. اگر هر دو صف 0 و 1 خالی باشد، فرایندهای صف 2 براساس FCFS اجرا می‌شوند. بنابراین فرایندهای طولانی به صف 2 می‌روند و به ترتیب FCFS اجرا می‌شوند.

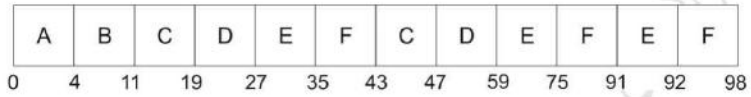
**مثال**

یک سیستم تک پردازنده ای با صف بازخورد چند سطحی (MLFQ)را در نظر بگیرید. به صف اول تکه زمانی 8 میکرو ثانیه، به صف دوم، تکه زمانی 16 میکرو ثانیه داده می شود. همچنین صف سوم با روش FCFS زمان بندی می شود. میانگین زمان پاسخ و میانگین زمان انتظار چقدر خواهد بود؟

| فرایند | زمان اجرا |
|---|---|
| A | 4 |
| B | 7 |
| C | 12 |
| D | 20 |
| E | 25 |
| F | 30 |

حل:

ابتدا برنامه ها وارد صف اول می شوند و در صورت نیـاز بـه بـیش از 8 میکروثانیـه بـه صـف دوم وارد مـی شوند(فرایندهای F,E,D,C) و در صف دوم در صورت نیاز به بیش از 16 میکروثانیه به صف سوم منتقل مـی شوند (فرایندهای F,E) و در صف سوم به روش FCFS به آنها رسیدگی می شود:



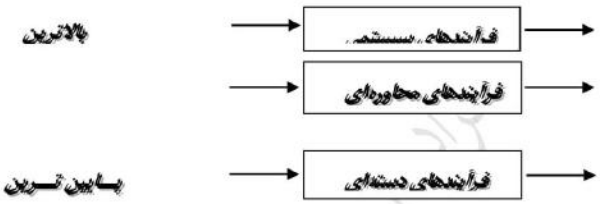

میانگین زمان انتظار برابر است با :

$$\frac{(4-4)+(11-7)+(47-12)+(59-20)+(92-25)+(98-30)}{6}=\frac{213}{6}=35.5$$

### زمان بندی صف چند سطحی (MLQ)

این الگوریتم صف آماده را به چند بخش تقسیم می‌کند. هر فرایند براساس صفات خود (اولویت،نوع) در صفی قرار می‌گیرد، که در هر صف الگوریتم زمان بندی خاصی وجود دارد. به طور معمول فرایندها به دو نوع ForeGround و BackGround تقسیم می شوند. اولویت فرایندهای پیش‌زمینه ممکن است از پس‌زمینه بالاتر باشد. در صف پیش‌زمینه از الگوریتم RR و در صف پس زمینه از الگوریتم FCFS استفاده می‌شود. همچنین بین صفها نیز باید زمان بندی وجود داشته باشد. که بر اساس الگوریتم غیرانحصاری و با اولویت ثابت پیاده‌سازی می‌شود.

شکل زیر الگوریتم زمان بندی صف چند سطحی با ۳ صف را نشان می دهد:




در این مثال، هیچ فرایندی در صف محاوره‌ای نمی‌تواند اجرا شود، مگر اینکه صف مربوط به فرایندهای سیستمی خالی باشد. همچنین اگر در حین اجرای فرایند دسته‌ای، یک فرایند محاوره‌ای وارد صف شود، فرایند دسته‌ای قبضه می‌شود، چون اولویت فرایند محاوره‌ای بالاتراست.

### زمان بندی اولویت (Priority)

در این الگوریتم به هر فرایند اولویتی داده می‌شود و CPU به فرایندی داده می‌شود که بالاترین اولویت را دارد و فرایندهایی که اولویت آنها یکسان باشد به ترتیب FCFS زمان بندی می‌شوند.

**مثال**

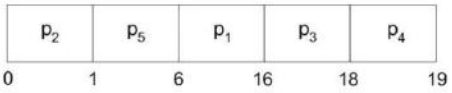
میانگین زمان انتظار برای پنج فرایند زیر را در روش زمان بندی اولویت بدست آورید.

(اعداد کوچک، اولویت بالا را نشان می‌دهد.)

| فرایند | زمان اجرا | اولویت |
|---|---|---|

| | | |
|---|---|---|
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

حل: گانت فرایند ها به صورت زیر است:




میانگین زمان انتظار :

$$\frac{(16-10)+(1-1)+(18-2)+(19-1)+(6-5)}{4}=8.2$$

■

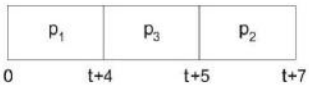
**مثال**

برای سه پردازه زیر، زمان متوسط پاسخگویی در روش اولویت کدام است؟

(عدد کمتر در ستون اولویت، نشان دهنده اولویت بیشتر است.)

| پردازه | اولویت | زمان ورود | زمان اجرا |
|---|---|---|---|
| $P_1$ | 1 | t | 4 |
| $P_2$ | 3 | t | 2 |
| $P_3$ | 2 | t +3 | 1 |

حل: ابتدا فرایند P1 اجرا می شود (چون اولویت آن بیشتر است) و سپس فرایند P3 و در نهایت فرایند P2 اجرا می شود. گانت به صورت زیر می باشد:




بنابراین میانگین زمان پاسخگویی برابر است با :

$$=\frac{(t+4-t)+(t+7-t)+(t+5-t-3)}{3}=\frac{13}{3}$$

■

## زمانبندی LCFS

در این روش، آخرین ورودی ابتدا سرویس می گیرد. این روش می تواند قبضه شدنی یا قبضه نشدنی باشد.

LCFS : Last-Come First-Served

### مثال

پنج فرایند زیر به سیستمی وارد شده است. نمودار گانت را در صورت استفاده از الگوریتم LCFS انحصاری (NP-LCFS) را محاسبه کنید.

| فرایند | زمان ورود | زمان پردازش |
|---|---|---|
| A | 0 | 1.5 |
| B | 1 | 1.5 |
| C | 2 | 1.5 |
| D | 3 | 1.5 |
| E | 4 | 1.5 |

حل:

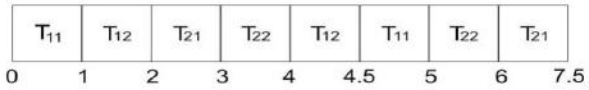

پردازنده در لحظه 0 به تنها فرایند موجود یعنی A داده می شود و چون قبضه شدنی نمی باشد، تا پایان اجـرای فراینـد A ، پردازنده را در اختیار دارد. در زمان 1.5 فقط فرایند B رسیده و پردازنده را می گیرد. در زمان 3 کـه C و D حاضـرند، ابتـدا پردازنده به D داده می شود، چون آخرین ورودی است. بعد از اجرای D ، چون E نیز رسیده است، پردازنـده بـه E داده مـی شود و درنهایت به C داده می شود.

### مثال

با توجه به جدول زیر، نمودار گانت را رسم کنید. (برای فرایندها از الگوریتم Round-Robin با برش زمانی q=2 و بـرای نـخ های درون هر فرآیند، از الگوریتم LCFS استفاده کنید.)

| فرایند | نخ | زمان پردازش | زمان ورود |
|---|---|---|---|
| P1 | T11 | 1.5 | 0 |
| | T12 | 1.5 | 1 |
| P2 | T21 | 2.5 | 2 |
| | T22 | 2 | 3 |

حل: نمودار گانت به صورت زیر می باشد:




متوسط زمان بازگشت نخ های فرایند P1 برابر است با:

$$P1: \frac{(5-0)+(4.5-1)}{2} = 4.25$$

■

### زمانبندی در سیستم چند پردازنده ای

الگوریتم های متداول برای زمانبندی سیستم های چند پردازنده ای عبارتند از:

۱- LPT ۲- RPT ۳- SPT

### الگوریتم LPT

این الگوریتم از بین کارهای باقیمانده، طولانی ترین کار را برای اجرا انتخاب می کند. این الگوریتم بهینه نیست ولی معمولا منجر به زمانبندی هایی با طول معقول می گردد.

**مثال**

طول زمانبندی را در سیستم تکالیف {8,4,1,7,2,13,2,6}در حالت دو پردازنده محاسبه کنید.

حل: ابتدا کارها را به ترتیب نزولی مرتب می کنیم:

$$\{\tau_1\} = \{13,8,7,6,4,2,2,1\}$$

سپس کارها را به ترتیب به پردازنده ها داده و هر پردازنده ای که کارش را انجام داد، کار بعدی را اجرا خواهد کرد:

| | | | | |
|---|---|---|---|---|
| P1 | 13 | 6 | 2 | 1 |
| P2 | 8 | 7 | 4 | 2 |

بنابراین طول زمانبندی برابر ۲۲ خواهد بود. (زمان مشغول بودن پردازنده اول که بیشترین زمان است.)■

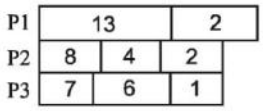
**مثال**

طول زمانبندی را در سیستم تکالیف {8,4,1,7,2,13,2,6}در حالت سه پردازنده محاسبه کنید.

حل: ابتدا کارها را به ترتیب نزولی مرتب می کنیم:

$$\{\tau_1\} = \{13,8,7,6,4,2,2,1\}$$

سپس کارها را به ترتیب به پردازنده ها داده و هر پردازنده ای که کـارش را انجـام داد، کـار بعـدی را اجـرا خواهد کرد:




بنابراین طول زمانبندی برابر ۱۵ خواهد بود. (زمان مشغول بودن پردازنده اول که بیشترین زمان است.)

■

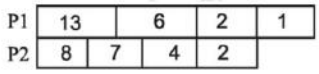
### الگوریتم RPT

این الگوریتم از نظر طول زمانبندی مشابه LPT عمل کرده ولی زمان پاسخ بهتری را می دهد. در واقع RPT مانند LPT ای است که ترتیب تکالیف هر پردازنده معکوس شده است.

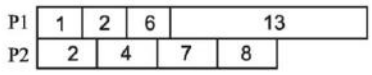
**مثال**

نتیجه استفاده از RPT برای $\{\tau_i\}=\{13,8,7,6,4,2,2,1\}$ در شرایط دو پردازنده را مشخص نمایید؟

حل: به روش LPT داریم:




که با معکوس کردن ترتیب تکالیف، خواهیم داشت




■

### الگوریتم SPT

این الگوریتم چندپردازنده ای، ابتدا کارها را بر اساس زمان اجرای افزایشی مرتب کرده و سـپس m تـا کـار اول را برای اجرا در m پردازنده موجود زمانبندی می کند (یک کار برای هر یک از پردازنده هـا)، سـپس m تا کار بعدی زمانبندی می گردند و بهمین ترتیب تا آخر.

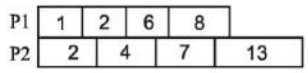
**مثال**

طول زمانبندی در سیستم تکالیف {8,4,1,7,2,13,2,6} با دو پردازنده به روشSPT را تعیین نمایید.

حل: ابتدا کارها را به ترتیب صعودی مرتب می کنیم:

{1,2,2,4,6,7,8,13}

سپس کارها را به ترتیب به پردازنده ها داده و هر پردازنده ای که کـارش را انجـام داد، کـار بعـدی را اجـرا خواهد کرد:




بنابراین طول زمانبندی برابر ۲۶ خواهد بود. (زمان مشغول بودن پردازنده دوم که بیشترین زمان است.)

■

**مشخصات سیاست های زمان بندی**

| | FCFS | RR | SPN | SRT | HRRN | FB |
|---|---|---|---|---|---|---|
| تابع انتخاب | MAX[W] | ثابت | Min[s] | Min[s-e] | $\max(\frac{w+s}{s})$ | |
| حالت تصمیم گیری | بدون قبضه کردن | با قبضه کردن (در برهه زمانی) | بدون قبضه کردن | با قبضه کردن در ورود | بدون قبضه کردن | با قبضه کردن (در برهه زمانی) |
| توان عملیاتی | تاکید نشده است | اگر برهه زمانی خیلی کوچک باشد، کم می شود. | زیاد | زیاد | زیاد | تاکید نشده است. |
| زمان پاسخ | می تواند زیاد باشد،به خصوص اگر واریانس زمانهای اجرا خیلی بزرگ باشد. | برای فرایندهای کوتاه زمان پاسخ خوبی را ارائه می دهد. | برای فرایندهای کوتاه زمان پاسخ خوبی را ارائه می دهد. | زمان پاسخ خوبی را ارائه می کند. | زمان پاسخ خوبی را ارائه می کند. | تاکید نشده است |
| سربار | حداقل | کم | می تواند زیاد باشد | می تواند زیاد باشد | می تواند زیاد باشد | می تواند زیاد باشد |
| تاثیر بر روی فرایندها | به فرایندهای کوتاه و فرایندهای در تنگنای I/O صدمه می زند. | عملکرد عادلانه | به فرایندهای طولانی صدمه می زند. | به فرایندهای طولانی صدمه می زند. | توازن مناسب | می تواند به نفع فرایندهای در تنگنای I/O باشد. |
| گرسنگی | خیر | خیر | امکان دارد | امکان دارد | خیر | امکان دارد |

w = زمان سپری شده در سیستم برای انتظار و اجرا تا به حال

e= زمان سپری شده، برای اجرا تا به حال

S= کل زمان مورد نیاز فرایند، که شامل e نیز هست.

شرط اینکه یک سیستم بلادرنگ قابل زمان بندی باشد این است که : $\sum_{i=1}^{m} \frac{c_i}{p_i} \leq 1$

$c_i$ : زمان اجرای رخداد i  $p_i$ : تناوب رخداد i  m : تعداد رخدادها

# کنکور ارشد

**(مهندسی کامپیوتر- دولتی ۸۷)**

۱- فرض کنید در سیستمی که از زمان بندی Round-Robin استفاده می کند، s زمان مورد نیاز برای سوئیچ کردن، q زمان برش و r میانگین زمان اجرای پردازش ها قبل از I/O را نشان می دهد. کارایی CPU توسط کدام یک از گزینه های زیر بیان می شود؟ (با فرض به اینکه رابطه s=q<r برقرار باشد.)

(از زمان موردنیاز برای سوئیچ کردن بین پردازش ها به دلیل I/O صرفنظر می‌شود)

۱) به سمت صد در صد میل می کند.
۲) کمتر از 50 درصد می باشد.
۳) به سمت صفر میل می کند.
۴) 50 درصد

پاسخ: جواب گزینه ۴ است.

$$\frac{q}{q+s}\times 100=\frac{q}{q+q}\times 100=\frac{1}{2}\times 100=50\%$$

■

**(مهندسی کامپیوتر- آزاد ۸۷)**

۲- در سیستمی 5 فرآیند موجود هستند. اگر الگوریتم زمان بندی فرآیندها، RR با مقدار کوانتوم 10 میلی ثانیه و زمان تعویض متن 1 میلی ثانیه باشد، آن گاه حداکثر زمانی که یک فرآیند منتظر می ماند تا نوبت به اجرای کوانتوم زمانی بعدی اش برسد کدام است؟

۱) 40 ۲) 50 ۳) 55 ۴) 44

حل: جواب گزینه ۴ است. حداکثر زمان انتظار برای دریافت کوانتوم بعدی برابر است با:

$$(n-1)(s+q)=(5-1)(10+1)=44$$

**(مهندسی IT- آزاد ۸۹)**

۳- سیستمی از روش زمانبندی نوبتی چرخشی استفاده می کند. اگر c زمان مورد نیاز برای تعویض متن ، q برش زمانی(کوانتوم) ، r میانگین زمان اجرای فرایندها قبل از I/O و q>r باشد، کارایی CPU برابر است با:

۱) $\frac{r}{q+c}$ ۲) $\frac{r}{r+c}$ ۳) $\frac{q}{q+c}$ ۴) $\frac{q}{r+c}$

حل: جواب گزینه ۲ است. چون میانگین زمان اجرای فرایندها قبل از ورودی/خروجی کمتر از زمان برش زمانی است، اجـرای فرایندها قبل از پایان کوانتوم زمانی، به اتمام می رسد. بنابراین کارایی پردازنده، یعنی نسبت زمـان مفیـد (r) بـه کـل زمـان (r+c) برابر است با: $\frac{r}{r+c}$

■

## (مهندسی کامپیوتر- دولتی ۷۴)

۴- پنج کار در وضعیت آماده ، در انتظار اجرا شدن هستند. زمان تخمین زده شده برای اجرای آنها برابر است با
x , 8 , 6 , 5, 10 میکرو ثانیه. (x مجهول است.) از کدام روش زمان بندی استفاده شود تا متوسط زمان پاسخگوئی حداقل شود؟

۱) FCFS ۲) SJF ۳) SRT ۴) RR

حل: جواب گزینه ۲ است.

الگوریتم SJF همواره کمترین زمان پاسخ را نتیجه می‌دهد و به زمان اجراهای داده شده بستگی ندارد.

■

## (مهندسی کامپیوتر- آزاد ۸۴)

۵- کدام گزینه در مورد الگوریتم زمان بندی SJF (Shortest Job First) درست نیست؟

۱) این الگوریتم زمان برگشت را کاهش می‌دهد. ۲) توان عملیاتی (throughput) را بالا می‌برد.

۳) این الگوریتم بر اساس اولویت عمل می‌کند. ۴) این الگوریتم بهره‌وری CPU را بالا می‌برد.

حل: جواب گزینه ۴ است.

گزینه ۱ درست است. چون الگوریتم SJF ، نسبت به سایر الگوریتم های زمان بندی انحصاری، دارای میانگین زمان برگشت کمتری است.

گزینه ۲ درست است ، چون در این الگوریتم کارهای کوتاه تر، زودتر اجرا می شوند، بنابراین تعـداد کارهـای انجـام شـده در واحد زمان (توان عملیاتی) بیشتر است.

گزینه ۳ درست است، چون الگوریتم SJF اولویت را به کارهای کوتاه تر می دهد، یک الگوریتم اولویت است.

گزینه ۴ نادرست است، چون SJF سعی به کم کردن تعداد تعویض متن ها ندارد، بنابراین بهره وری CPU را بالا نمی برد.

■

## (مهندسی کامپیوتر- آزاد ۸۳)

۶- یک سیستم تک پردازنده‌ای از الگوریتم زمان بندی کوتاه‌ترین زمان باقی مانده (SRT) استفاده می‌نماید. چهار فرایند با زمان اجرای تخمینی 3,2,3,6 میلی ثانیه به ترتیب در زمانهای 7,3,1,0 میلی ثانیه وارد سیستم می‌شوند. اگر زمان تعویض متن ناچیز باشد و تمامی فرایندها فقط کار پردازشی داشته باشند، آنگاه میانگین زمان انتظار برای اجرای کامل فرایندها چند میلی ثانیه است؟

۱) 1.5 ۲) 0.25 ۳) 2.5 ۴) 2.25

حل: جواب گزینه ۴ است.

نمودار گانت به صورت زیر می باشد:


بنابراین میانگین زمان انتظار برابر است با:

$$\frac{(14-6-0)+(4-3-1)+(6-2-3)+(10-3-7)}{4}=\frac{8+0+1+0}{4}=2.25$$

■

## (مهندسی کامپیوتر-آزاد ۸۳)

۷- کدامیک از ویژگیهای زیر به عنوان ملاک الگوریتم زمان بندی صف‌های چندگانه (Multiple Queues) نیست؟

۱) افزایش گذردهی ۲) کاهش تعداد تعویض متن

۳) افزایش بهره‌وری از پردازنده ۴) اعمال اولویت (ابتدا کوتاه‌ترین فرایند)

حل: جواب گزینه ۴ است.

در الگوریتم زمان بندی صف های چند گانه، اولویت لزوما به معنای کوتاهترین فرایند نمی باشد. ■

## ( مهندسی IT – آزاد ۸۹)

۸- چهار فرایند بر اساس جدول زیر وارد سیستم می شوند. در این سیستم از زمان بندی HRRN استفاده می شود. زمان تعویض متن یک میلی ثانیه است. میانگین زمان برگشت(TURNAROUND TIME) برابر است با:

| فرایند | زمان ورود (میلی ثانیه) | زمان اجرا (میلی ثانیه) |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 3 |
| P3 | 3 | 6 |
| P4 | 3 | 5 |

۱) 12.75 ۲) 13.75 ۳) 14 ۴) 13

پاسخ: جواب گزینه ۱ است.

در زمان صفر فقط فرایند P1 در سیستم وجود دارد و پردازنده به آن داده می شود تا اجرایش به پایان برسد. در این زمان (یعنی 7 ) ، اولویت فرایندها طبق فرمول الگوریتم HRRN یعنی $\frac{w+s}{s}$ محاسبه می شود:

$$P2: \frac{(7-2)+3}{3} = 2.6$$

$$P3: \frac{(7-3)+6}{6} = 1.6$$

$$P4: \frac{(7-3)+5}{5} = 1.8$$

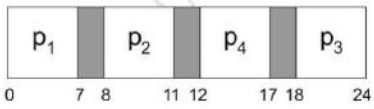
حال چون اولویت P2 از فرایندهای P3 و P4 بیشتر است ، پردازنده در زمان 8 (بعد از یکی میلی ثانیه از پایان اجرای P1 به علت تویض متن) به آن داده می شود. اجرای P2 در لحظه 11 به پایان می رسد. در زمان 11 ، مجددا اولویت فرایندهای باقی مانده را حساب می کنیم:

$$P3: \frac{(11-3)+6}{6} = 2.3$$

$$P4: \frac{(11-3)+5}{5} = 2.6$$

بنابراین پردازنده به P4 داده می شود. و در نهایت بعد از اجرای P4 به P3 داده می شود.

نمودار گانت به صورت زیر است:




بنابراین میانگین زمان برگشت برابر است با:

$$\frac{(7-0)+(11-2)+(17-3)+(24-3)}{4} = 12.75$$

■

# فصل ۴

## هم روندی: انحصار متقابل و همگام سازی

عملکرد چند برنامه ای به خاطر این ابداع شد که زمان پردازش کامپیوتر به صورت پویا بین تعدادی کار بتواند تقسیم گردد. برای زمینه های چند برنامه ای، چند پردازشی و پردازش توزیعی و همچنین طراحی سیستم عامل، موضوع همزمانی اساسی است. فرایندهای هم روند (Concurrent) ، به دفعات نیاز به برقراری ارتباط با یکدیگر دارند. این فرایندها نیاز به هماهنگی، تبادل داده و استفاده از منابع مشترک دارند.

**مباحث مطرح در ارتباط بین فرایندها**

در طراحی سیستم عامل، سه موضوع زیر در رابطه با ارتباط بین فرایندها مطرح است:

**۱- همگام سازی (Synchronization)**

اگر بین فرایندها وابستگی وجود داشته باشد، ترتیب درست انجام کارها باید رعایت شود.

**۲- تبادل اطلاعات (Communication)**

فرایندها می توانند با مکانیسم هایی چون "حافظه مشترک، تبادل پیام، فایل مشترک و لوله" با یکدیگر تبادل اطلاعات کنند.

**۳- رقابت فرایندها**

فرایندها در فعالیت های بحرانی یکدیگر مداخله نکنند و شرایط رقابتی(Race Condition) برای آنها رخ ندهد.

**حالت های ممکن ارتباط بین فرایندها (IPC)**

۱- فرایندهایی که به طور مستقیم با یکدیگر تبادل داده و همکاری دارند.

۲- فرایندهایی که غیر مستقیم با یکدیگر تبادل داده و همکاری دارند.

۳- فرایندهایی که هیچ اطلاعی از یکدیگر ندارند.

**سه مسئله کنترلی**

در مورد فرایندهای رقیب، با سه مسئله کنترلی زیر باید برخورد شود:

۱-انحصار متقابل

۲- بن بست

۳- گرسنگی(قحطی)

**انحصار متقابل**

فرض کنید چند فرایند برای دسترسی به یک منبع غیر اشتراکی مانند چاپگر رقابت می کنند. در طی اجراء، هر یک از فرایندها ، فرمان هایی را به دستگاه ورودی/ خروجی ارسال می کنند. چنین منبعی را منبع بحرانی و بخشی از برنامه که از آن استفاده می کند را بخش بحرانی آن برنامه می گوییم. مهم این است که در یک زمان، تنها یک برنامه مجاز است تا در بخش بحرانی خود باشد.

بخش هایی از برنامه که رفتار آنها با عوامل مشترک، ایجاد رقابت می کنند، را ناحیه بحرانی (Critical Region) می گویند.

**بن بست**

اعمال انحصار متقابل دو مسئله کنترلی، بن بست و گرسنگی را به وجود می آورد. دو فرایند P1 و P2 و دو منبع بحرانی R1 و R2 مفروض است که هر یک از فرایندها برای انجام عمل خود به هر دو منبع نیاز دارند. اگر منبع R1 به P2 و منبع R2 به P1 داده شود، هر یک منتظر منبع دیگر می باشند و هیچ یک از فرایندها، منبعی را که در اختیار دارد را رها نمی کند تا فرایند دیگر آن را دریافت کرده و بخش بحرانی خود را انجام دهد. بنابراین هر دو فرایند در بن بست قرار می گیرند.

**گرسنگی**

فرض کنید هر یک از سه فرایند P1 و P2 و P3 متناوباً نیازمند دسترسی به منبع R هستند. وقتی P1 این منبع را در اختیار گیرد، P2 و P3 در انتظار آن منبع، به تاخیر انداخته می شوند. با خروج P1 از ناحیه بحرانی، یکی از P2 یا P3 باید R را در اختیار گیرند. اگرP3 منبع R را بگیرد و قبل از پایان بخش بحرانی

مجدداً P1 درخواست R را بکند و بعد از پایان P3 ، اجازه به P1 داده شود و مکرراً ایـن عمـل بـین P1 و P3 ادامه یابد، P2 به صورت نامحدود از دسترسی به منبع R محروم می ماند و گرسنگی می کشد.

**مثال**

در صورت اجرای هم روند و موازی دو پروسس زیر، چه خروجی هایی ممکن می باشد؟

| p1 | p2 |
|---|---|
| cout<< 1; cout<< 2; | cout<< 3; cout<< 4; |

پاسخ:

امکان تولید خروجی های زیر می باشد:

۱- **1234** : ابتدا P1 به طور کامل اجرا شده و سپس P2 اجرا می شود.

۲- **3412** : ابتدا P2 به طور کامل اجرا شده و سپس P1 اجرا می شود.

۳- **1342** : ابتدا دستور اول در P1 اجرا شده و سپس P2 به طور کامل اجرا شده و در نهایت دستور دوم P1 اجرا می شود.

۴- **3124** : ابتدا دستور اول از P2 ، بعد اجرای کامل P1 ، مجددا دستور دوم از P2 اجرا شود.

۵- **1324** : ابتدا دستور اول از P1، بعد دستور اول از P2، مجددا دستور دوم از P1 و در نهایت دستور دوم از P2 اجرا شود.

۶- **3142** : ابتدا دستور اول از P2 ، بعد دستور اول از P1 ، مجددا دستور دوم از P2 و در نهایت دستور دوم از P1 اجرا شود.

■

**مثال**

با فرض اینکه دو پردازه P1 و P2 به صورت هم روند وجود دارند. نحوه ایجاد رشته $A(CD)^*B$ چگونه می باشد؟

| P1 | P2 |
|---|---|
| while(TRUE){<br>cout<< "A";<br>cout<< "B";<br>} | while(TRUE){<br>cout<<"C";<br>cout<<"D";<br>} |

پاسخ:

ابتدا اجرای P1 شروع شده و دستور cout<<"A" اجرا می شود. در این لحظه وقفه رخ داده و به P2 سوئیچ می شود. فرایند P2 چند مرتبه اجرا می شود. در نهایت با رخ دادن وقفه و سوئیچ به P1 ، دستور چاپ B اجرا می شود.

■

## مثال

در صورتی که دو پروسس P1 و P2 به صورت هم روند اجرا شوند، نحوه چاپ BADC چگونه است؟ (مقدار اولیه متغیرهای x و y برابر صفر است.)

| P1 | P2 |
|---|---|
| **while** (x=0);<br>cout<< "A";<br>cout<< "D";<br>y=1; | cout<<" B";<br>x=1;<br>**while** (y=0);<br>cout<<"C"; |

پاسخ:

نحوه چاپ رشته BADC به این صورت است که ابتدا فرایند P2 اجرا شده و کاراکتر B را چاپ کرده و بعد از یک شدن x ، به فرایند P1 سوئیچ شده و از حلقه عبور کرده و کاراکتر A و سپس D چاپ شده و در نهایت بعد از یک شدن y به P2 سوئیچ شده و از حلقه عبور کرده و کاراکتر C چاپ می شود. ■

**رویکردهای انحصار متقابل**

شرایطی که باید رعایت شود تا یک همکاری درست و کارا بین فرایندهای هم روند برقرار باشد، عبارتند از:

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

از بین فرایندهایی که برای یک منبع یکسان دارای ناحیه بحرانی هستند، در هر لحظه فقط یک فرایند مجاز است که در ناحیه بحرانی خود باشد.

**۲- پیشرفت (Progress)**

فرایندی که فعلا تصمیم به ورود به ناحیه بحرانی را ندارد و در ناحیه غیر بحرانی می باشد،و دستورالعمل های عادیی برنامه خود را اجرا می کند، نباید در تصمیم گیری برای ورود فرایندهای دیگر به ناحیه بحرانی شرکت کند.(امکان ممانعت نداشته باشد)

**۳- انتظار محدود (Bounded Waiting)**

باید مدت انتظار فرایندهایی که نیاز به ورود به ناحیه بحرانی دارند، محدود باشد. یعنی نباید دچار گرسنگی و بن بست شوند.

گرسنگی: به مدت نامعلوم و بدون حد بالای مشخص، منتظر فرایندهای دیگر بودن.

بن بست: تا ابد منتظر ورود به ناحیه بحرانی خود بودن.

البته علاوه بر رعایت ۳ شرط بالا، مسئله را باید در حالت کلی حل کرد و فرضی برای ساده سازی راه حل به کار نبرد. همچنین الگوریتم حالت قطعی و غیر تصادفی داشته باشد.

اگر فرایند P2 بخواهد وارد ناحیه بحرانی شود در حالی که P1 در ناحیه بحرانی قرار دارد، P2 باید منتظر بماند تا P1 خارج شود. P2 برای انتظار کشیدن دو راه " انتظار مشغول و مسدود شدن" پیش رو دارد. روش انتظار مشغول دارای مشکل اتلاف پردازنده است و از این روش وقتی استفاده می کنیم که زمان انتظار کوتاه باشد.

برای تحقق انحصار متقابل، پیشنهادهای مختلفی وجود دارد. این راه حل ها را به صورت چهار رویکرد زیر، دسته بندی می کنیم:

۱- نرم افزاری

۲- با حمایت سخت افزار (با کمک دستورالعمل های خاص CPU)

۳- با حمایت سیستم عامل (با کمک فراخوان های سیستمی خاص)

۴- با حمایت زبان برنامه سازی (با کمک کامپایلر)

**رویکردهای نرم افزاری انحصار متقابل**

راه حل های نرم افزاری مستقیما توسط برنامه ها استفاده می شوند و وجود حافظه اشتراکی ضروری می باشد. در این راه حل ها از دستورالعمل های خاص توسط سخت افزار استفاده نمی شود و حمایتی از سیستم عامل و زبان های برنامه سازی نداریم.

**الگوریتم Decker**

آقای Decker اولین شخصی بود که یک راه حل نرم افزاری دو فرایندی برای مسئله انحصار متقابل ارائه داد. Decker با پنج مرحله تلاش به راه حل درست رسید. این تلاش ها در زیر آورده شده است.

**تلاش اول (تناوب قطعی)**

در این روش از یک متغیر سراسری مشترک به نام turn استفاده شده که دو مقدار 0 یا 1 را می تواند بگیرد. مقدار اولیه turn برابر 0 است. بنابراین ابتدا P0 می تواند وارد ناحیه بحرانی شود. در این حالت P1 در حلقه while منتظر می ماند تا P0 از ناحیه بحرانی خارج شده و turn را 1 کند. در این صورت P1 می تواند وارد ناحیه بحرانی شود. برنامه فرایندها به صورت زیر است:

```
P0(void)
{
  while(TRUE)
  {
     while( turn != 0) ;  /*wait*/
     critical-section( );
     turn = 1;
     non-critical- section( );
  }
```

```
P1(void)
{
  while(TRUE)
  {
     while( turn != 1) ;  /*wait*/
     critical-section( );
     turn = 0;
     non-critical- section( );
  }
```

| | |
|---|---|
| } | } |

**بررسی شرط ها در تلاش اول:**

**۱- انحصار متقابل**

راه حل گفته شده انحصار متقابل را تضمین می کند و امکان ندارد که P0 و P1 با هم وارد ناحیه بحرانی شوند. چون P0 در صورتی وارد ناحیه بحرانی می شود که مقدار turn برابر 0 باشد و P1 بـا مقـدار 1 وارد می شود. بنابراین امکان ندارد که هر دو به turn نگاه کنند و هر دو با هم مقدارهای 0 و1 را ببینند.

**۲- پیشرفت**

این روش شرط پیشرفت را رعایت نمی کند. چون اگر P1 وارد ناحیه بحرانی شود و کارش تمـام شـده و به بخش غیر بحرانی برود، در این حالت turn برابر 0 می باشد. حال نوبت P0 است که وارد ناحیه بحرانی شود، ولی می خواهد به مدت طولانی در ناحیه غیر بحرانی بماند. P1 به سـرعت کـارش در ناحیـه غیـر بحرانی تمام شده و قصد ورود مجدد به ناحیه بحرانی را دارد. اما چون turn برابر 0 است در حلقه انتظـار می ماند تا بالاخره P0 وارد ناحیه بحرانی شده و بعد از خروج، turn را 1 کـرده تـا P1 بتوانـد وارد ناحیـه بحرانی شود. در این سناریو، P1 توسط فرایندی منتظر مانده بود کـه در ناحیـه بحرانـی نبـود و جلـوی پیشرفت اش را گرفته بود.

**۳- انتظار محدود**

در این روش قحطی نداریم، چون فرایندها به صورت یک در میان و نوبتی وارد ناحیه بحرانی می شـوند. همچنین بن بست نیز نداریم. بنابراین شرط انتظار محدود رعایت می شود.

- یکی از معایب این روش این است که سرعت عملیات توسط فراینـد کنـدتر تعیـین مـی شـود، چـون فرایندها برای دسترسی به ناحیه بحرانی باید به صورت یک در میان عمل کنند.

- یکی از معایب این روش این است که اگر فرایندی در ناحیه بحرانی از کار بیفتد، فرایند دیگـر تـا ابـد منتظر خواهد ماند.

**تلاش دوم**

در این روش از دو متغیر پرچم مشترک به نام های flag[0] و flag[1] با مقدار اولیـه FALSE اسـتفاده مـی شود که هر کدام متعلق به یک فرایند است. هر فرایندی که قصد ورود به ناحیه بحرانی خود را دارد، پـرچم خود را TRUE می کند.

در این روش هر فرایند دارای کلید مجزا برای ورود به ناحیه بحرانی است تا اگر فرایندی قصد استفاده از ناحیه بحرانی را نداشت، فرایند دیگر بتواند به ناحیه بحرانی خود دسترسی داشته باشد. برنامه فرایندها به صورت زیر است:

boolean flag[2] = {FALSE,FALSE};

```
P0(void){
  while(TRUE)  {
     while( flag[1] );
     flag[0] = TRUE;
     critical-section( );
     flag[0] = FALSE;
     non-critical- section ( );
   }
}
```

```
P1(void){
  while(TRUE)  {
     while( flag[0] );
     flag[1] = TRUE;
     critical-section( );
     flag[1] = FALSE;
     non-critical- section ( );
   }
}
```

**بررسی شرط ها در تلاش دوم**

**۱- انحصار متقابل**

این روش انحصار متقابل رعایت نمی شود. یعنی حتی از روش اول هم بدتر است.

سناریو: فرض کنید که P0 ، flag[1] را خوانده و آن را FALSE می بیند، اما قبل از TRUE کردن flag[0]، P1 اجرا شود و flag[0] را خوانده و آن را FALSE می بیند و برای ورود به ناحیه بحرانی flag[1] را TRUE کرده و وارد می شود. حال در این زمان به P0 سوئیچ شده و flag[0] را TRUE کرده و این فرایند هم وارد ناحیه بحرانی می شود! بنابراین چون هر دو فرایند در یک زمان وارد ناحیه بحرانی خود شده اند، شرط انحصار متقابل برقرار نمی باشد.

**۲- پیشرفت**

این روش شرط پیشرفت را رعایت می کند. اگر P0 در ناحیه غیر بحرانی خود باشد، flag[0] را FALSE نگه می دارد تا P1 بتواند وارد ناحیه بحرانی خودش شود. اگر P0 برای مدت طولانی تصمیم به ورود به ناحیه بحرانی را نداشته باشد، P1 به دفعات می تواند وارد ناحیه بحرانی شود و سپس خارج شود. یعنی P0 جلوی پیشرفت P1 را نمی گیرد.

**۳- انتظار محدود**

این روش شرط انتظار محدود را رعایت نمی کند، چون امکان قحطی دارد. در این تلاش امکان ورود پی در پی یک فرایند به ناحیه بحرانی و عدم دستیابی فرایند دیگر به ناحیه بحرانی وجود دارد.

سناریو: فرض کنید P0 در ناحیه بحرانی است و به P1 سوئیچ می شود. چون flag[0] برابر TRUE است، P1 نمی تواند وارد ناحیه بحرانی شود. بعد از پایان کوانتوم، پردازنده به P0 داده شده و این فرایند سریعا ناحیه بحرانی اش را اجرا کرده و سعی به ورود مجدد به ناحیه بحرانی را دارد و این اجازه به او داده می شود. P0 مجددا flag[0] را TRUE کرده و اگر به P1 سوئیچ شود، باز هم نمی تواند اجازه ورود بگیرد. بنابراین تلاش برای دستیابی به ناحیه بحرانی تصادفی است و امکان قحطی وجود دارد.

یکی از معایب این روش این است که اگر فرایندی در ناحیه بحرانی از کار بیفتد، فرایند دیگر تا ابد منتظر خواهد ماند.

**تلاش سوم**

در تلاش دوم هر فرایند ابتدا وضعیت فرایند مقابل را چک کرده و سپس flag خود را TRUE می کند. بنابراین اگر هر دو به طور همزمان قصد ورود به ناحیه بحرانی را داشته باشند، flag یکدیگر را FALSE می بینند و با هم وارد می شوند. برای حل این مشکل دو سطر مسئله را عوض می کنیم.

```
boolean   flag[2] = {FALSE,FALSE};
```

| | |
|---|---|
| **P0**(void){<br>**while**(TRUE) {<br>flag[0] = TRUE ;<br>**while**( flag[1] );<br>**critical-section( );**<br>flag[0]= FALSE;<br>non-critical- section( );<br>}<br>} | **P1**(void){<br>**while**(TRUE) {<br>flag[1] = TRUE;<br>**while**( flag[0] );<br>**critical-section( );**<br>flag[1]= FALSE;<br>non-critical- section( );<br>}<br>} |

**بررسی شرط ها:**

**۱- انحصار متقابل**

اگر یکی از فرایندها بتواند وارد ناحیه بحرانی شود، چون قبل از بررسی flag مقابل، flag خود را TRUE کرده ، جلوی ورود فرایند مقابل را می گیرد. بنابراین شرط انحصار متقابل برقرار است.

**۲- پیشرفت**

این روش شرط پیشرفت را رعایت می کند. (با همان استدلال تلاش دوم)

**۳- انتظار محدود**

در این روش شرط انتظار محدود رعایت نمی شود، چون امکان بن بست وجود دارد.

سناریو: فرض کنید که P0، falg[0] را TRUE کند ولی قبل از بررسی flag[1] در برنامه P0 ، به P1 سوئیچ شود و P1 ، flag[1] را TRUE کند. در این صورت هر دو فرایند تا ابد در حلقه انتظار گرفتار می شوند و بن بست رخ می دهد.

**تلاش چهارم (ادب و تعارف)**

در تلاش قبلی هر فرایند می تواند روی حق خود برای ورود به بخش بحرانی اش پافشاری کند. در این روش هر فرایند متغیر flag خود را TRUE کرده تا خواست خود برای ورود به بخش بحرانی را نشان دهد، اما آماده است flag را تغییر دهد تا به فرایند دیگر احترام گذارد. یعنی فرایندی که قصد ورود به ناحیه بحرانی را دارد، اگر ببیند که فرایند مقابل هم می خواهد به ناحیه بحرانی وارد شود، falg خود را برای مدت کوتاهی FALSE کرده تا فرایند مقابل بتواند وارد شود.

برنامه فرایندها به صورت زیر است:

```
boolean   flag[2] = {FALSE,FALSE};
```

```
P0(void)
{
   while(TRUE)
   {
     flag[0] = TRUE;
     while( flag[1])
       {
         flag[0] = FALSE;
         delay_for_a_short_time( );
         flag[0] = TRUE;
       }
     critical-section( );
     flag[0] = FALSE;
     non-critical- section( );
   }
}
```

```
P1(void)
{
  while(TRUE)
  {
    flag[1] = TRUE;
    while( flag [0])
      {
        flag[1] = FALSE;
        delay_for_a_short_time( );
        flag[1] = TRUE;
      }
    critical-section( );
    flag[1] = FALSE;
    non-critical- section( );
  }
}
```

**بررسی شرط ها:**

**۱- انحصار متقابل:**

راه حل گفته شده انحصار متقابل را تضمین می کند. (با استدلال تلاش سوم)

**۲- پیشرفت:**

این روش شرط پیشرفت را رعایت می کند. (با همان استدلال تلاش دوم و سوم)

**۳- انتظار محدود:**

در این روش شرط انتظار محدود رعایت نمی شود، چون ممکن است یک فرایند به مدت نامعلوم و بدون حد بالای مشخص، گرفتار قسمت تاخیر (delay) شود و فرایند مقابل به دفعات وارد ناحیه بحرانی شود. بنابراین به دلیل امکان گرسنگی، شرط انتظار محدود رعایت نمی شود. همچنین به علت امکان وجود Livelock ، نیز شرط انتظار محدود رعایت نمی شود.

### مشکل Livelock

در تلاش چهارم مشکل بن بست وجود ندارد، اما مشکل جدیدی به نام Livelock وجود دارد. با دنبال کردن اجرای زیر، این مشکل را توضیح می دهیم:

۱) P0 ، flag[0] را TRUE کند.

۲) P1 ، flag[1] را TRUE کند.

۳) P0 ، flag[1] را بررسی کند.

۴) P1 ، flag[0] را بررسی کند.

۵) P0 ، flag[0] را FALSE کند.

۶) P1 ، flag[1] را FALSE کند.

هر دو فرایند در یک زمان به مدت کوتاه یکسان عقب نشینی می کنند. سپس با هم بر می گردند و مراحل بالا را تکرار می کنند. اگر این دنباله به طور نامحدود تکرار شود، ممکن است هیچ کدام از فرایندها نتوانند وارد ناحیه بحرانی شوند. البته این تکرار بن بست نمی باشد، چون با تغییر در سرعت نسبی فرایندها، این چرخه شکسته می شود.

**تلاش پنجم (راه حل صحیح)**

Decker بعد از چهار تلاش ناموفق، یک راه حل درست که شرایط انحصار متقابل، انتظار محدود و پیشـرفت را با هم رعایت می کند، ارائه داد. در این روش متغیر نوبت را با متغیرهای پرچم ترکیب کرد. این الگـوریتم در زیر آورده شده است.

```
boolean   flag[2] = {FALSE,FALSE};
turn=0;
```

```
P0(void){
  while(TRUE){
     flag[0] = TRUE;
     while( flag[1] )
      if (turn == 1){
         flag[0] = FALSE;
         while( turn==1) do;
         flag[0] = TRUE;
      }
    critical-section( );
     turn = 1;
     flag[0] = FALSE;
     non-critical-section ( );
  }
}
```

```
P1(void){
  while(TRUE){
    flag[1] = TRUE;
    while( flag[0])
      if (turn == 0){
         flag[1] = FALSE;
         while( turn==0) do;
         flag[1] = TRUE;
      }
   critical-section( );
     turn = 0;
     flag[1] = FALSE;
     non-critical-section( );
  }
}
```

هنگامی که P0 بخواهد وارد بخش بحرانی خود شود، در flag مربوط به خود مقدار TRUE می گذارد. سپس flag مربوط به P1 را بررسی می کند که دو حالت رخ می دهد:

**الف- flag[1] برابر TRUE باشد:**

اگر turn برابر یک باشد، P0 به P1 احترام گذاشته و با FALSE کردن پرچم اش منتظـر مـی مانـد. در ایـن هنگام P0 کاری انجام نمی دهد تا turn برابر صفر شود و سپس flag خودش را TRUE می کند.

**ب- flag[1] برابر FALSE باشد:**

P0 وارد بخش بحرانی شده و بعد از خروج از بخش بحرانی، در flag خود مقدار FALSE می گذارد تا بخش بحرانی را آزاد کند و در turn مقدار 1 را قرار می دهد تا حق پافشاری را به P1 واگذارد.

**خلاصه ای از وضعیت تلاش های Decker**

در جدول زیر، هر جا که شرط رعایت می شود با علامت ✓ مشخص شده است:

| تلاش های Decker | انحصار متقابل | پیشرفت | انتظار محدود |
|---|---|---|---|
| تلاش اول | ✓ | - | ✓ |
| تلاش دوم | - | ✓ | - |
| تلاش سوم | ✓ | ✓ | - |
| تلاش چهارم | ✓ | ✓ | - |
| تلاش پنجم | ✓ | ✓ | ✓ |

### الگوریتم Peterson

چندین سال بعد، peterson راه حل ساده و زیبایی را برای حل مسئله انحصار متقابل ارائه کرد. این الگوریتم به سادگی برای n فرایند نیز قابل تعمیم است.

```
boolean   flag[2] = {FALSE,FALSE};
turn=0;
```

| | |
|---|---|
| **P0**(void){<br>{<br>  **while** (TRUE){<br>  flag[0] = TRUE;<br>  turn = 0;<br>  **while** (turn==0 **&&** flag[1]<br>**);**<br>  **critical-section( );**<br>  flag[0] = FALSE;<br>  non-critical-section( );<br>  }<br>} | **P1**(void){<br>{<br>  **while** (TRUE){<br>  flag[1] = TRUE;<br>  turn = 1;<br>  **while**(turn==1 **&&** flag[0]<br>**);**<br>  **critical-section( );**<br>  flag[1] = FALSE;<br>  non-critical-section( );<br>  }<br>} |

در کتاب استالینگز، مقدار turn برعکس مقدار دهی و تست می شود که تفاوتی با روش بالا ندارد. به طور مثال برای P0 ، داریم:

```
turn = 1;
while(turn==1 && flag[1]);
```

**نحوه کار کردن این الگوریتم:**

فرض کنید P0 وP1 به طور تقریبا همزمان(P1 کمی دیرتر)، قصد ورود به ناحیه بحرانی را دارند. هر دو فرایند، شماره خود را در turn ذخیره کرده ولی P1 که دیرتر شماره اش را ذخیره کرده، شماره اش در turn می ماند و turn برابر1 می شود. حال زمانی که P0 وP1 به دستور while می رسند، P0 از حلقه عبور کرده و وارد ناحیه بحرانی می شود ولی P1 در حلقه می چرخد(انتظار مشغول) و وارد ناحیه بحرانی نمی شود.

الگوریتم Peterson، شرایط انحصار متقابل، انتظار محدود و پیشرفت را با هم رعایت می کند. و مشکل بن بست و Livelock و قحطی ندارد. این الگوریتم ساده ترین و کوتاهترین راه حل نرم افزاری است.

معایبی که در هر یک از تلاش های Decker و همچنین روش Peterson وجود دارند، عبارتنداز:

۱- اگر یکی از فرایندها در داخل ناحیه بحرانی از کار بیفتد، فرایند دیگر تا ابد منتظر می ماند.

۲- مبتنی بر انتظار مشغول می باشند.

### رویکردهای انحصار متقابل با حمایت سخت افزار

می توان به کمک راه حل هایی که از دستورالعمل های ویژه ماشین استفاده می کنند، و نیاز به حمایت از طرف پردازنده دارند، مسئله انحصار متقابل را حل کرد. این راه حل ها عبارتند از:

۱- دستورالعمل از کار انداختن وقفه ها

۲- دستور العمل TSL

۳- دستورالعمل SWAP

### دستورالعمل از کار انداختن وقفه ها

در این راه حل، هر فرایند باید به محض ورود به ناحیه بحرانی، تمام وقفه ها را از کار بیندازد و دقیقا قبل از خروج از ناحیه بحرانی، همه وقفه ها را مجددا فعال سازد. در این صورت پردازنده نمی تواند از فرایندی به فرایند دیگر سوئچ کند، چون وقفه ساعت غیر فعال است. بنابراین وقتی که فرایندی وقفه ها را غیر فعال می کند، می تواند بدون ترس از دخالت فرایندهای دیگر، به خواندن و نوشتن در حافظه مشترک بپردازد. رویکرد از کار انداختن وقفه ها برای انحصار متقابل به صورت زیر است:

```
P(int i){
  while(TRUE)
  {
    disable_interrupts( );
    critical_section( );
    enable_interrupts( );
    non_critical_section( );
  }
}
```

- امکان دارد فرایندی وقفه ها را بعد از غیر فعال کردن، مجددا فعال نکند.(از معایب این روش)
- در سیستم های چندپردازنده ای، غیرفعال کردن وقفه ها، فقط در پردازنده ای که این دستورالعمل را اجرا کرده تاثیر گذار است و پردازنده های دیگر می توانند به ناحیه بحرانی دسترسی داشته باشند.

### دستور العمل TSL

بسیاری از کامپیوترها دارای دستورالعمل TSL(Test and Set Lock) می باشند. این دستور محتویات یک کلمه از حافظه (lock) را خوانده و در ثبات قرار می دهد. سپس مقدار 1 را در همان آدرس از حافظه

ذخیره می کند. این عملیات غیر قابل تقسیم انجام می شوند و تا اینکه اجرای دستورالعمل تمام نشود، هیچ فرایند و یا پردازنده دیگری نمی تواند به این کلمه از حافظه دسترسی پیدا کند.

در زیر تابع ورود به ناحیه بحرانی(enter_region) با استفاده از دستوالعمل TSL ، به زبان اسمبلی آورده شده است:

```
enter_region:
  tsl   reg , lock
  cmp  reg , #0
  jne   enter_region
  ret
```

توسط اولین دستور، مقدار قبلی lock در رجیستر ذخیره شده و سپس در lock مقدار 1 ذخیره می شود. توسط دستور دوم مقدار قبلی lock با 0 مقایسه می شود. اگر 0 نبود، این عملیات تکرار شده (حلقه انتظار مشغول) و اگر 0 بود، زیربرنامه بازگشت کرده، در حالی که lock را 1 کرده است.

تذکر: برای پاک کردن lock در هنگام خروج از ناحیه بحرانی، کافی است در آن 0 را ذخیره کرد:

```
move  lock , #0
ret
```

راه حل داده شده، انحصار متقابل و پیشرفت را رعایت می کند ولی انتظار محدود به دلیل گرسنگی را رعایت نمی کند.

### دستور swap

می توان از دستورالعملی به نام swap ، برای نوشتن یک روال ورود به ناحیه بحرانی استفاده کرد. این دستور می تواند در یک عمل واحد غیر قابل تقسیم، محتویات یک رجیستر پردازنده را با محتویات یک کلمه حافظه، جابجا کند.

```
enter_region:
  move  reg , #1
  swap  reg, lock
  cmp   reg, #0
  jne    enter_region
  ret
```

تذکر: برای پاک کردن lock در هنگام خروج از ناحیه بحرانی، کافی است در آن 0 را ذخیره کرد.

تذکر: xchg نام دیگر swap می باشد.

راه حل داده شده، انحصار متقابل و پیشرفت را رعایت می کند ولی انتظار محدود به دلیل گرسنگی را رعایت نمی کند.

**راهکارهای سیستم عامل و زبان برنامه سازی برای تدارک همزمانی**

راهکارهای سیستم عامل و زبان برنامه سازی برای تدارک همزمانی عبارتند از:

۱- سمافور(راهنما)

۲- ناظر (مانیتور)

۳- تبادل پیام.

**سمافور**

راه حل های نرم افزاری و سخت افزاری که بررسی کردیم ، دارای نقاط ضعف زیادی بودند. حال ابـزاری بـه نام سمافور را معرفی می کنیم که دارای قدرت زیادی در برقراری انحصار متقابل مـی باشـد و همچنـین از عهده انواع مختلف مسایل همگام سازی بر می آید.

سمافور یک **ساختار** شامل فیلدهای زیر است:

**۱- شمارنده صحیح (count)**

از شمارنده برای شمارش تعداد wakeup هایی که می خواهند هدر بروند، استفاده می شود. با ایـن کـار این سیگنال ها برای استفاده های بعدی ذخیره می شوند.

**۲- صف (queue)**

از صف برای نگهداری فرایندهای بلوکه شده بر روی سمافور استفاده می شود.

توسط Dijkstra ، دو تابع wait و signal مطرح شدند که به ترتیب تعمیم یافته sleep و wakeup هستند.

توابع wait وsignal به صورت زیر است:

```
void wait(semaphore s)
{
  s.count = s.count –1;
  if (s.count < 0)
  {
   place this process in
s.queue;
   block this process;
  }
}
```

```
void signal(semaphore s)
{
  s.count = s.count +1;
  if (s.count <= 0 )
  {
   remove a process from s.queue;
   place this process in ready queue;
  }
}
```

| | |
|---|---|

تابع wait ، یک واحد از شمارنده کم کرده و اگر منفی شود، فرایند مربوطه در صف، مسدود می شود.
تابع signal، یک واحد به شمارنده اضافه می کند، اگر مقدار شمارنده بیشتر از صفر نشود، یک فرایند از قبل مسدود شده در صف، آزاد می شود.

- در بعضی از متون از حرف P به جای wait و از حرف V به جای signal استفاده می شود.
- در بعضی از متون از down به جای wait و از up به جای signal استفاده می شود.

سمافور بر دو نوع عمومی و دودویی می باشد. در سمافور عمومی که آن را بررسی کردیم، شمارنده می تواند مثبت، صفر و یا منفی باشد. اما شمارنده در سمافور دودویی، فقط مقادیر 0 و1 را دریافت می کند.
توابع wait وsignal برای سمافور باینری به صورت زیر است:

```
void wait(semaphore s)
{
  if (s.count =1)
     s.count = 0;
  else{
    place this process in
s.queue;
    block this process;
  }
}
```

```
void signal(semaphore s)
{
  if (s.queue is empty )
     s.count =1;
  else{
    remove a process from s.queue;
    place this process in ready queue;
  }
}
```

- قدرت سمافور دودویی معادل با سمافور عمومی است.

در سمافورها صفی برای نگهداری فرایندهای بلوکه شده استفاده می شود. اگر خروج از ایـن صـف بـه ترتیب ورود باشد(FIFO) ،به آن سمافور قوی می گویند و اگر این چنین نباشد، به آن سمافور ضعیف می گویند. در سمافور ضعیف، امکان گرسنگی وجود دارد. در این کتاب سمافورها از نوع قوی فرض می شوند.

**انحصار متقابل با استفاده از سمافورها**

می خواهیم انحصار متقابل را به کمک سمافورها برقرار کنیم. برای حل یک مسئله با سمافور، باید بدانیم که:

۱- به چند سمافور نیاز است.

۲- مقدار اولیه شمارنده هر سمافور چند باید باشد.

۳- عمل signal و wait در کجای برنامه باید روی سمافور انجام شود.

مسئله انحصار متقابل برای دو فرایند p1 و p2 توسط برنامه زیر پیاده سازی شده است. در این برنامه عمل wait قبل از ورود به ناحیه بحرانی و عمل signal بعد از خروج از ناحیه بحرانی روی سمافور دودویی mutex با مقدار اولیه 1 انجام شده است:

```
semaphore mutex =1;
void p(int i)
{
  while(TRUE)
  {
    wait(mutex);
    critical-section( );
    signal(mutex);
    non-critical-section( );
  }
}
```

فرض کنیم P1 اول اجرا شود. با اجرای تابع wait، چون مقدار سمافور یک است،آن را صفر می کند و وارد ناحیه بحرانی می شود. در زمانی که P1 در ناحیه بحرانی است، اگر P2 سعی به ورود به ناحیه بحرانی داشته باشد، چون سمافور برابر 0 است، این فرایند بلوکه شده و در صف قرار می گیرد. بعد از خروج P1 از ناحیه بحرانی، تابع signal را اجرا کرده و چون صف خالی نیست، P2 که در صف قرار دارد را آزاد می کند و P2 می تواند وارد ناحیه بحرانی شود.

- می توان با استفاده از سمافور عمومی، انحصار متقابل را برای بیش از دو فرایند نیز پیاده سازی کرد.
- سمافوری که ترتیب خروج در آن مشخص نباشد، سمافور ضعیف نامیده می شود. در این نوع سمافور، امکان گرسنگی وجود دارد.(در سمافور قوی، منطق صف FIFO است.)

پیاده سازی انحصار متقابل به کمک سمافور دارای مزایایی است از جمله: ارضای شرط های انحصار متقابل،پیشرفت و انتظار محدود . این روش عادلانه است،CPU را تلف نمی کند و مشکل اولویت معکوس ندارد.

**همگام سازی با استفاده از سمافورها**

سمافور دارای توانایی زیادی در حل مسائل همگام سازی دارد. دو فرایند را در نظر بگیرید که باید ابتدا دستور S1 در P1 و سپس دستور S2 در P2 اجرا شود. برای پیاده سازی این همگام سازی از سمافوری به نام S با مقدار اولیه صفر به صورت زیر استفاده می کنیم:

| P1 | P2 |
|---|---|
| .<br>S1;<br>signal(S);<br>. | .<br>wait(S);<br>S2;<br>. |

مشخص است که تا زمانی که S1 از P1 اجرا نشود، اجرای S2 ممکن نیست.

تذکر: معمولا مقدار اولیه سمافور در همگام سازی برابر صفر و در انحصار متقابل برابر 1 است.

**مثال**

آیا ترتیب اجرای P1,P2,P3 ممکن است؟ (مقدار اولیه سمافورها S و Q برابر صفر است)

| P1 | P2 | P3 |
|---|---|---|
| .<br>.<br>signal(S); | wait(Q);<br>.<br>. | wait(S);<br>.<br>signal(Q); |

پاسخ: خیر.

بعد از اجرای P1 ، فرایند P2 نمی تواند اجرا شود، چون در ابتدای P2 دستور wait(Q) قرار دارد که به علت یک بودن سمافور Q ، باعث بلوکه شدن P2 می شود. ولی اگر بعد از P1 ، فرایند P3 اجرا شود، به علت وجود دستور signal(Q) در انتهای آن، می توان بعد از P3 فرایند P2 را اجرا کرد. پس یک ترتیب اجرای ممکن عبارت است از : P1,P3,P2 .

**مثال**

با فرض اینکه مقدار اولیه دو سمافور S و Q برابر صفر باشد، نحوه چاپ ABCDE را مشخص کنید.

| P0 | P1 |
|---|---|
| signal(Q)<br>wait(S);<br>cout<<"C";<br>cout<<"D"; | wait(Q);<br>cout<<"A";<br>signal(S);<br>cout<<"B"; |

| | |
|---|---|
| | cout<<"E"; |

پاسخ:

در زیر ترتیب اجرا با شماره مشخص شده است:

| P0 | P1 |
|---|---|
| (1) signal(Q);<br>(6) wait(S);<br>(7) cout<<"C";<br>(8) cout<<"D"; | (2) wait(Q);<br>(3)<br>cout<<"A";<br>(4)<br>cout<<"B";<br>(5) signal(S);<br>(9) cout<<"E"; |

**مثال**

نحوه چاپ 1324 را مشخص کنید؟ (مقدار اولیه دو سمافور S و Q برابر صفر است)

| P1 | P2 |
|---|---|
| cout<<"1";<br>**signal**(S);<br>**wait**(Q);<br>cout<<"2";<br>**signal**(S); | **wait**(S);<br>cout<<"3";<br>**signal**(Q);<br>**wait**(S);<br>cout<<"4"; |

پاسخ: به علت استفاده از دستور wait(S) در ابتدای P2 ، و صفر بودن S ، اجرا نمی تواند با P2 شروع شود. بنابراین ابتدا P1 اجرا شده و عدد 1 چاپ می شود. سپس توسط signal(S) مقدار سمافور S برابر یک می شود. به علت رسیدن به wait(Q) ، و صفر بودن Q ، نمی توان ادامه داد . با تعویض متن به p2 ، چون S برابر یک شده از wait(S) رد شده و S صفر می شود. سپس 3 چاپ می شود. حال دستور signal(Q) مقدار Q را برابر یک کرده و با رسیدن به wait(S) ، چون S ، صفر است، نمی توان ادامه داد و به ادامه P1 رفته و چون Q برابر یک است از wait(Q) رد شده و Q برابر صفر شده و سپس عدد 2 چاپ می شود. در نهایت توسط signal(S) مقدار S برابر یک شده و به P2 پرش کرده و از wait(S) عبور کرده و مقدار 4 چاپ می شود. در زیر ترتیب اجرا با شماره مشخص شده است:

| P1 | P2 |
|---|---|