

بسمه تعالیٰ



سیستم‌های عامل

علی چوداری خسروشاهی

Akhosroshahi@IAUT.ac.ir

دانشگاه آزاد اسلامی

مراجع

Operating Systems

نویسنده: William Stallings

ترجمه: دکتر حسین پدرام

www.akhosroshahi.ir

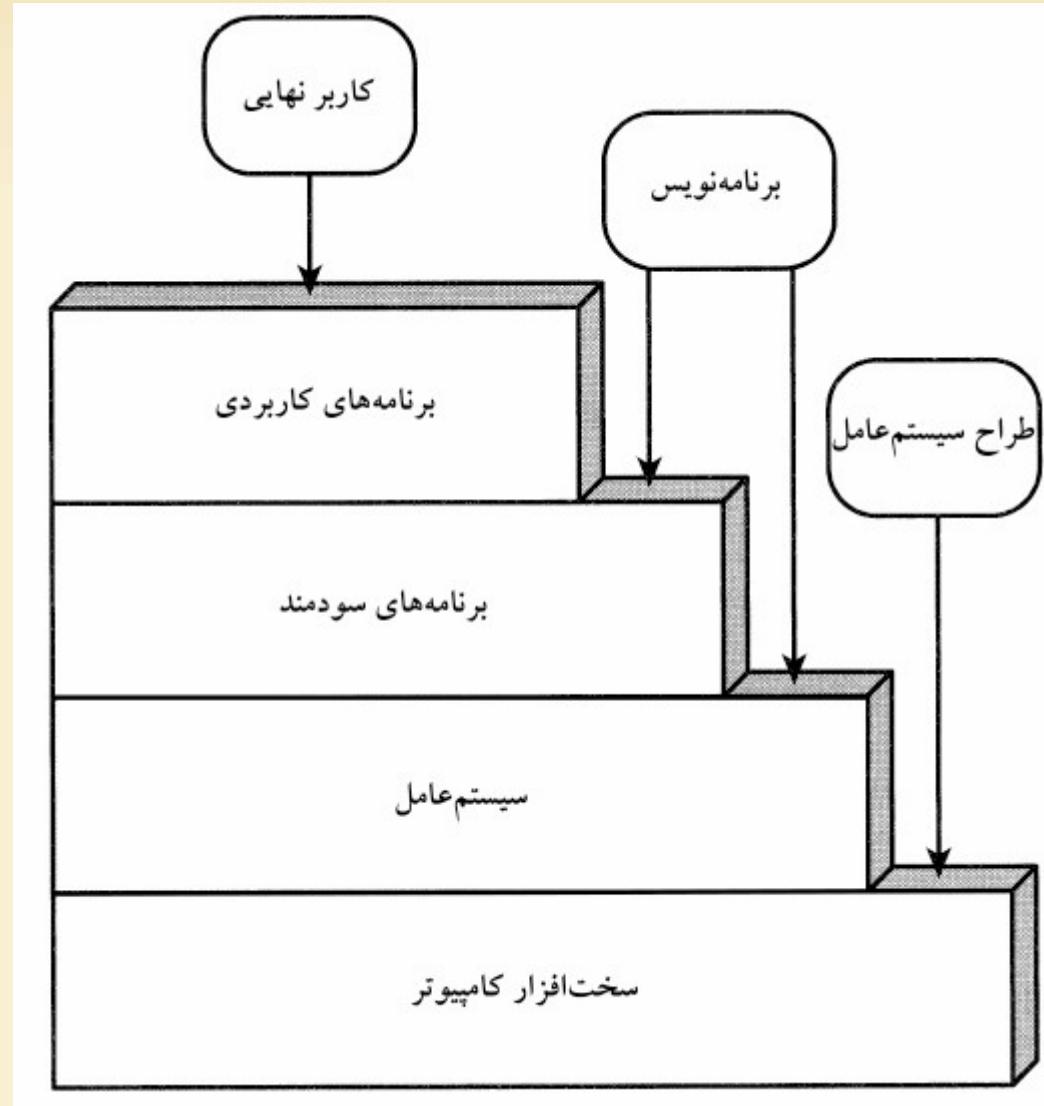
فصل اول

نگاه کلی به سیستم عامل

اهداف و وظایف سیستم عامل

- سیستم عامل برنامه‌ای است که اجرای برنامه‌های کاربردی را کنترل و بصورت رابط کاربر و سخت افزار عمل می‌کند.
- سه هدف سیستم عامل
- سهولت: سیستم عامل موجب سهولت استفاده از کامپیوتر می‌شود.
- کارآمدی: سیستم عامل موجب استفاده کارآمد از منابع سیستم کامپیوتری می‌شود.
- قابلیت رشد: سیستم عامل باید طوری ساخته شود که توسعه موثر آن، امتحان، و معرفی وظایف جدید بدون ایجاد مزاحمت در خدمات جاری امکان پذیر سازد.

لایه ها و نمای سیستم کامپیوتروی



خدمات سیستم عامل

سیستم عامل خدماتی را در زمینه های زیر ارائه می کند:

- توسعه برنامه
- اجرای برنامه
- دسترسی به دستگاه های ورودی و خروجی
- کنترل دسترسی به پرونده ها
- دسترسی به سیستم
- کشف و پاسخ به خطأ
- حسابداری

سیستم عامل به عنوان مدیر منابع

- سیستم عامل مسئول مدیریت منابع برای انتقال، ذخیره سازی و پردازش داده ها را دارد.
- سیستم عامل به عنوان راهکار کنترلی از دو جهت غیر عادی است:
 - ۱- سیستم عامل مانند نرم افزار عادی کار می کند.
 - ۲- سیستم عامل مرتبا کنترل کردن را رها می کند.

تکامل تدریجی سیستم عامل

- ۱- پردازش ردیفی (Serial Processing).
- ۲- سیستم های دسته ای ساده (Simple Batch System).
- ۳- سیستم های چند برنامه ای دسته ای (Multiprogrammed Batch System).
- ۴- سیستم های اشتراک زمانی (Time Sharing Systems).

پردازش ردیفی

- به علت دسترسی به کامپیوتر به صورت ردیفی این نام را دارند.
- در سال ۱۹۴۰-۱۹۵۰ به وجود آمد.
- کاربر با سخت افزار در ارتباط بود.
- سیتم عامل وجود نداشت.
- دارای یک میز فرمان بود.
- به زبان ماشین و به وسیله دستگاه ورودی بارگذاری می شد.
- خروجی در چاپگر ظاهر می شد.

مسئله اصلی سیستم های اولیه

■ این سیستم های اولیه، دو مشکل اساسی داشتند:

۱- زمانبندی :

توسط برگهای نوبت گیر برای ماشین وقت می گرفتند و در زمان پیش بینی شده کارش تمام نمی شد.

۲- زمان نصب :

اگر در زمان کار برای هر یک از کار ها مشکلی ایجاد شود باید از اول کارش را شروع کند.

سیستم های دسته ای

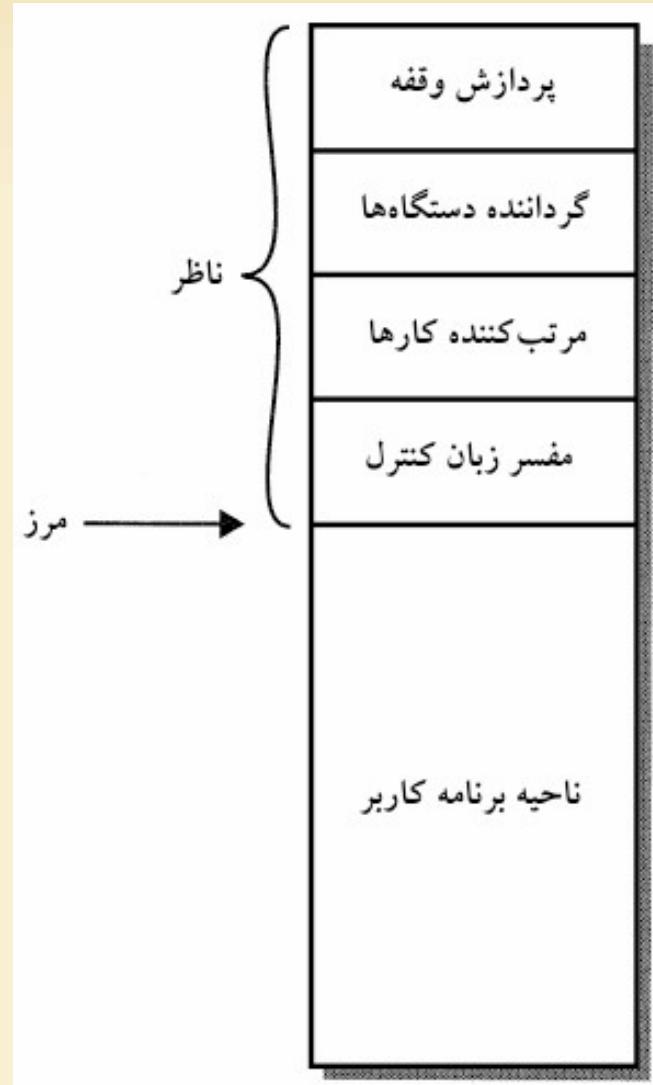
- اولین سیستم عامل دسته ای (۱۹۵۰) به وسیله General motors و برای استفاده IBM 701 به وجود آمد.
- ایده اصلی پردازش دسته ای، استفاده از نرم افزاری به نام **ناظر** بود.
- با استفاده از سیستم عامل دسته ای، دیگر کاربران دسترسی مستقیم به ماشین ندارد.
- کاربر کار را توسط نوار به متصدی کامپیوتر می دهد.
- متصدی همه کارها را روی دستگاه ورودی گذاشته و وارد کامپیوتر می شود.
- ناظر به طور خودکار بار کردن برنامه بعدی را به عهده دارد.

سیستم های دسته ای

برای آشنایی با عملکرد این طرح، آن را از دو دیدگاه بررسی می کنیم:

- از دیدگاه ناظر: ناظر است که، دنباله ای از حوادث را کنترل می کند. برای این کار، بخش عمده ناظر همیشه باید در حافظه اصلی بوده آماده اجرا باشد. این بخش را ناظر ماندگار می نامند.
- از دیدگاه پردازنده: در لحظه ای از زمان، پردازنده دستورالعمل های ناظر را که در حافظه اصلی قرار دارند، اجرا می کند. اجرای این دستورالعمل ها موجب می شوند کار بعدی در بخش دیگری از حافظه اصلی خوانده شود. وقتی کاری خوانده شد، پردازنده دستورالعمل انشعابی را در ناظر می بیند که به پردازنده دستور می دهد تا اجرا را از ابتدای برنامه کاربر ادامه دهد. سپس پردازنده دستورالعمل های موجود در برنامه کاربر را اجرا می کند تا به پایان برسد یا با شرایط خطا مواجه شود.

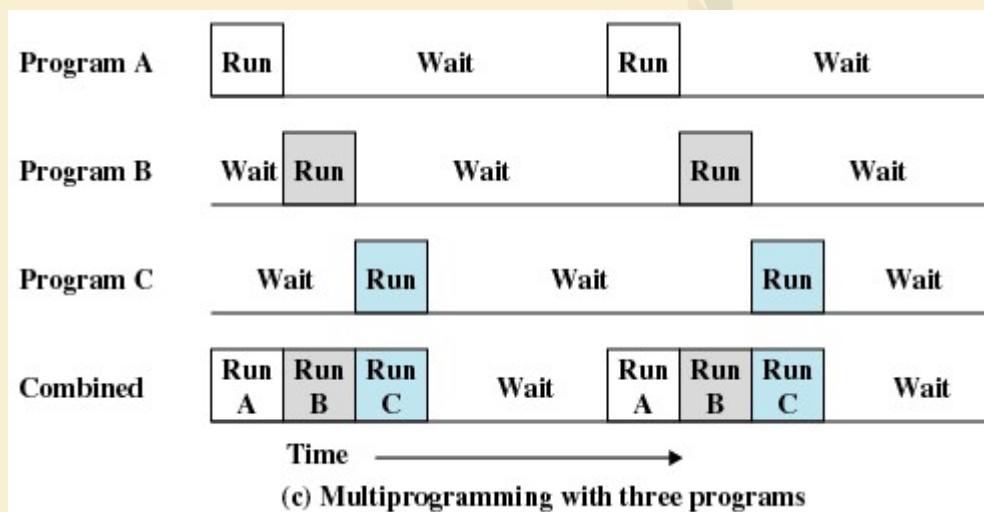
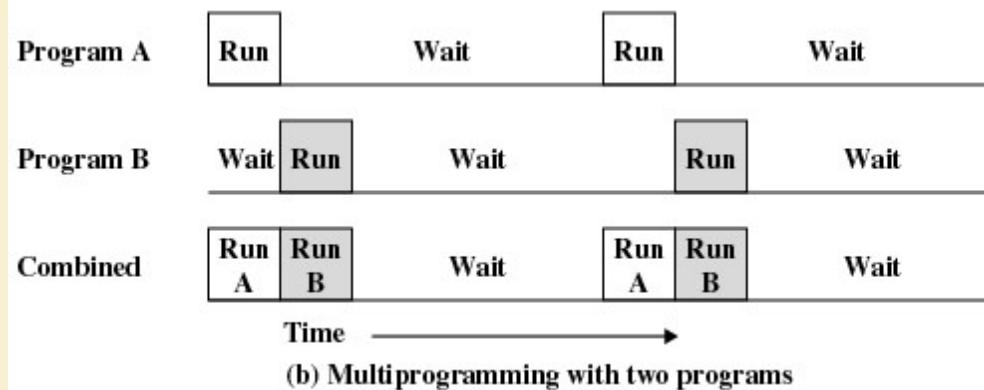
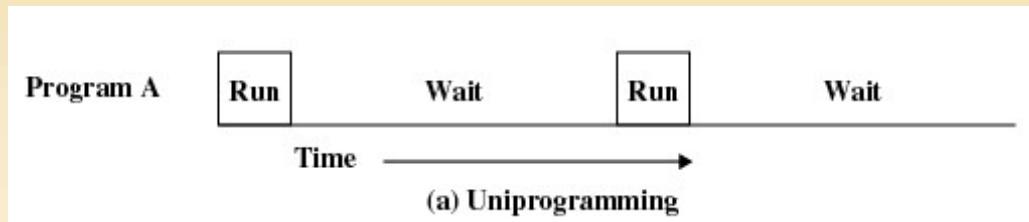
طرح حافظه برای ناظر ماندگار



سیستم های چند برنامه ای دسته ای

- حتی با مرتب سازی خودکار کارها در سیستم عامل دسته ای ساده، باز هم پردازنده غالباً بی کار است. چون دستگاه های O/I در مقایسه با پردازنده کند هستند.
- حافظه را برای سه یا چهار برنامه تخصیص داده.
- موضوع اصلی سیستم عامل های امروزی است.
- باعث افزایش سرعت کار می شود
- به خصوصیت سخت افزار تکیه دارد.
- از سیستمهای تک برنامه ای پیچیده تر است
- نیازمند نوعی مدیریت حافظه است

مثالی از چندبرنامه ای



سیستم های اشتراک زمانی

- از چند برنامگی برای رسیدن به حالت محاوره ای استفاده می کند.
- پردازنده بین کاربران به اشتراک گذاشته می شود.
- داشتن کاربران متعددی که از طریق پایانه خود به طور هم زمان از سیستم عامل استفاده می کنند.
- اگر N کاربر داشته باشد هر کاربر $N/1$ از زمان مفید پردازنده استفاده می کنند.
- عکس العمل انسان کند می باشد.

دستاوردهای اصلی

پنج دستورد توسعه سیستم عامل:

۱- فرآیندها

Process

۲- مدیریت حافظه

Memory management

۳- حفاظت اطلاعات و ایمنی

Information protection and security

۴- زمانبندی و مدیریت منابع

Scheduling and resource management

۵- ساختار سیستم

System structure

فرآیندها

- یک مفهوم بنیادی در سیستم عامل است.
- تعاریف زیادی برای واژه فرآیند ارائه شد، از جمله:
 - یک برنامه در حال اجرا است.
 - نمونه ای از برنامه در حال اجرا در کامپیوتر.
 - نهادی که می تواند به پردازنده نسبت داده شود و اجرا گردد.
- واحدی از فعالیت که توسط یک نخ ترتیبی از اجرا، حالت فعلی، و مجموعه ای از منابع وابسته سیستم مشخص می شود.

فصل دوم

شرح و کنترل فرآیند

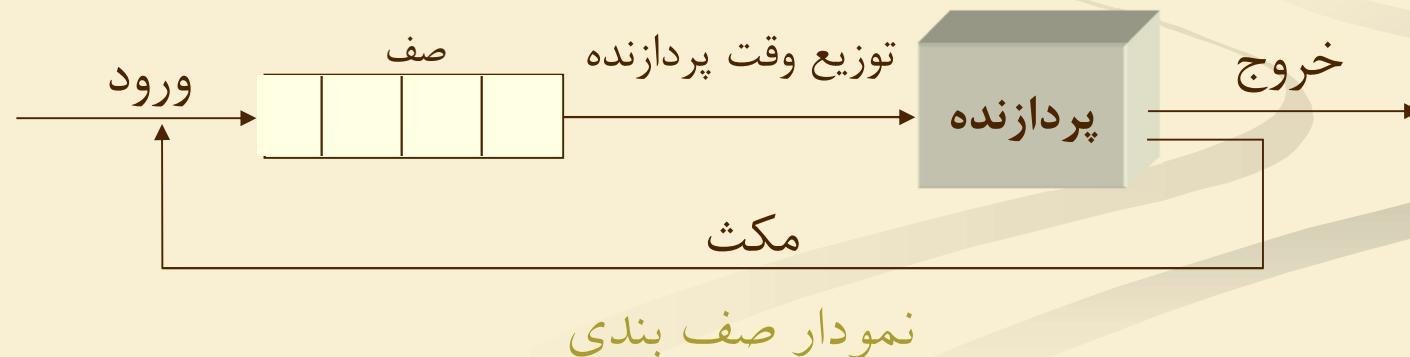
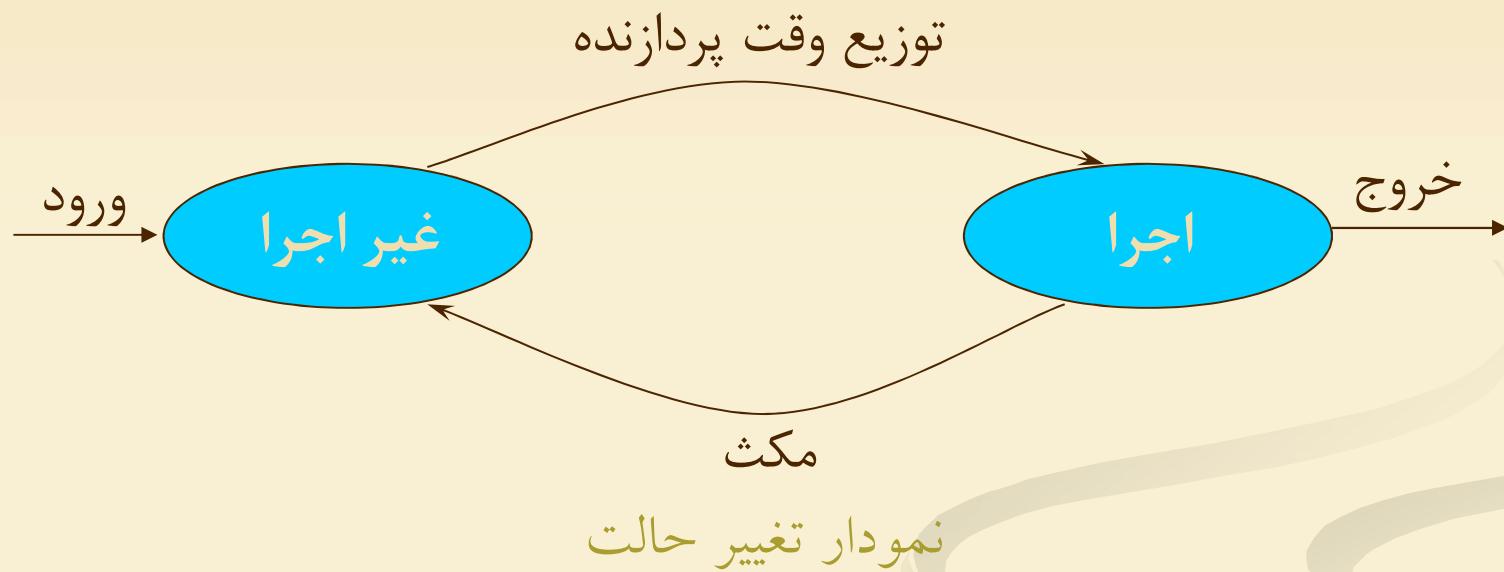
حالات فرآیند

- اساسی ترین عمل پردازندۀ اجرایی دستورالعمل های موجود در حافظه است.
- اجرا شامل دنباله ای از دستورالعمل های همان برنامه است.
- به اجرای یک برنامه خاص فرآیند یا وظیفه گویند.
- رفتار یک فرآیند به خصوصی را می توان با فهرست کردن دنباله دستورالعمل هایی که برای آن فرآیند اجرا می شود مشخص نمود که به آن رد (trace) آن فرآیند گویند.

مدل دو حالته

- مسئولیت اصلی سیستم عامل کنترل یک فرآیند است.
- یک فرآیند ممکن است در یکی از دو حالت اجرا و غیر اجرا باشد.
- وقتی یک سیستم عامل فرآیندی را دریافت می کند آن را در حالت غیر اجرا قرار می دهد.
- پس از رسیدن نوبت فرآیند فعلی را به حالت غیر اجرا برد و فرآیند را به حالت اجرا برد.
- در مورد هر فرآیندی باید اطلاعاتی را ذخیره کرد.
- از جمله وضعیت جاری و محل آن در حافظه
- فرآیندی که در حال اجرا نیست باید در صفحه انتظار نوبت قرار گیرد.
- گاهی صفحه انتظار که حاوی جداولی است که وضعیت فرآیند ها را نمایش می دهد.
- وقتی یک فرآیند در معرض وقفه قرار می گیرد به صفحه انتظار می رود.
- اگر یک فرآیند کارش تمام شود کنار گذاشته می شود.

مدل دو حالت



ایجاد فرآیند

- عمر یک فرآیند محدود به زمان ایجاد تا زمان خاتمه دادن به آن است.
- در هنگام ایجاد سیستم عامل ساختمان داده مربوط به آن فرآیند را ساخته و فضای آدرس مربوط به آن را تخصیص می دهد.
- معمولاً چهار حادثه موجب به ایجاد فرآیند می گردد

سیستم عامل با جریانی از کنترل کار دسته ای سروکار دارد که معمولاً بر روی نوار یادیسک ذخیره می شود. وقتی سیستم عامل آماده تحویل گرفتن کار جدید می شود، دنباله بعدی از فرمان های کنترل کار را می خواند. کاربر از طریق پایانه با سیستم ارتباط برقرار می کند.

سیستم عامل می تواند فرآیندی را ایجاد کند تا عملی را از طرف برنامه کاربر انجام دهد، بدون این که کاربر منتظر بماند (مثل فرآیندی برای کنترل

به منظور بهره برداری از خاصیت توازنی و مولفه ای بودن، برنامه کاربر می تواند ایجاد تعدادی از فرآیندها را دیکته کند.

کار دسته ای جدید

ارتباط محاوره ای
توسط سیستم عامل ایجاد می شود
تا سرویس ارائه کند.
عمل چاپ).

تولید مثل توسط فرآیند موجود

پایان فرآیند

- هر کار دسته ای باید حاوی دستورالعمل توقف یا فراخوانی صريح یک خدمت سیستمی برای پایان باشد.
- در بعضی از سیستم عامل ها ممکن است فرآیندی به وسیله فرآیند ایجاد کننده اش و با پایان فرآیند پدرش نیز پایان یابد.
- زایش فرزند (Process spawning):
 - فرآیندی با درخواست فرآیند دیگر به وجود می آید
 - فرزند:
 - فرآیند زاینده را پدر و فرآیند ایجاد شده را فرزند گویند

دلایل پایان فرآیند

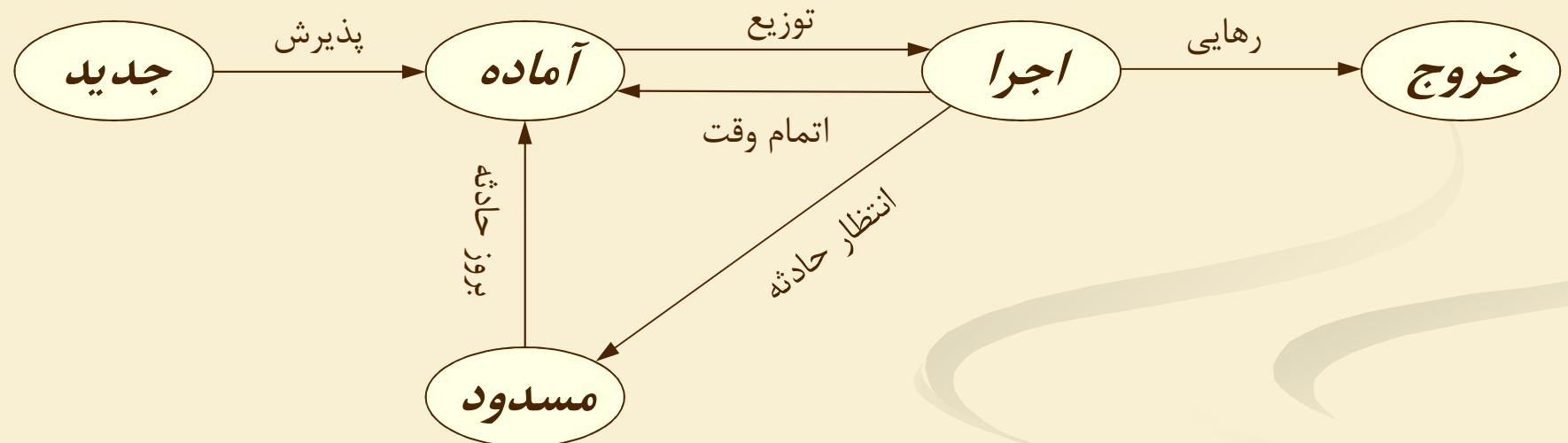
فرآیند یک فراخوانی سرویس سیستم عامل را اجرا می کند تا نشان دهد که اجرایش کامل شده است.	کامل شدن عادی
فرآیند بیش از مهلت زمانی تعیین شده است. زاه های مختلفی برای اندازه گیری مهلت زمانی وجود دارد. این راه ها شامل کل زمان مصرفی ("زمان ساعت دیواری")، مدت زمانی که صرف اجرا شده است، و در فرآیند محاوره ایی، مدت زمانی که از آخرین ورودی گذشته است.	مهلت زمانی
حافظه مورد نیاز فرآیند بیش از حافظه موجود در سیستم است.	فقدان حافظه
فرآیند سعی می کند به محلی از حافظه مراجعه کند که اجازه دستیابی ندارد.	نقض حدود
فرآیند سعی در استفاده از منبع یا فایلی دارد که مجوز آن را ندارد، یا سعی در استفاده نادرست از آن دارد، مثل نوشتمن در فایلی که فقط خواندنی است.	خطای حفاظت
فرآیند سعی در انجام محاسبات غیرمجاز، مثل تقسیم بر صفر دارد، یا سعی در ذخیره اعداد بزرگ تر از ظرفیت سخت افزار دارد.	خطای محاسباتی
فرآیند بیش از حداقل زمان لازم برای وقوع یک رویداد، منتظر مانده است.	گذشت زمان
در اثنای ورودی یا خروجی خطایی رخ می دهد، مثل نیافتن فایل، عدم موفقیت در خواندن یا نوشتمن پس از حداقل تعداد دفعات تلاش (مثل زمان روبه رو شدن با نواحی خراب نوار)، یا عملیات نامعتبر (مثل خواندن از چاپگر).	خطای I/O
فرآیند سعی در اجرای دستورالعملی دارد که موجود نیست (غالباً وقتی رخ می دهد که انشعابی به ناحیه ای از داده ها و تلاش برای اجرای داده ها صورت می گیرد).	دستورالعمل نامعتبر
فرآیند سعی در اجرای دستورالعملی دارد که برای سیستم عامل درنظر گرفته شده است.	دستورالعمل ممتاز
قطعه ای از داده با نوع غلط یا بدون مقدار اولیه.	استفاده نادرست از داده
سیستم عامل یا اپراتور به دلایلی فرآیند را خاتمه می دهند (مثل وجود بن بست).	دخالت سیستم عامل یا اپراتور
وقتی فرآیند والد خاتمه می یابد، سیستم عامل ممکن است تمام فرآیندهای فرزند آن را خاتمه دهد.	پایان والد
فرآیند والد مجوز پایان دادن به فرآیندهای فرزند را دارد.	درخواست والد

مدل پنج حالته

انواع حالت‌های ممکن

- اجرا (Running): فرآیندی هم اکنون در حالت اجرا است.
- آماده (Ready): با گرفتن فرصت به اجرا درمی‌آیند.
- مسدود (Blocked): با تمام شدن وقت و یا بروز حادثه اتفاق می‌افتد.
- جدید: فرآیندی که هم اکنون ایجاد شده.
- خروج: به علت دستور توقف یا به دلیلی قطع شده.

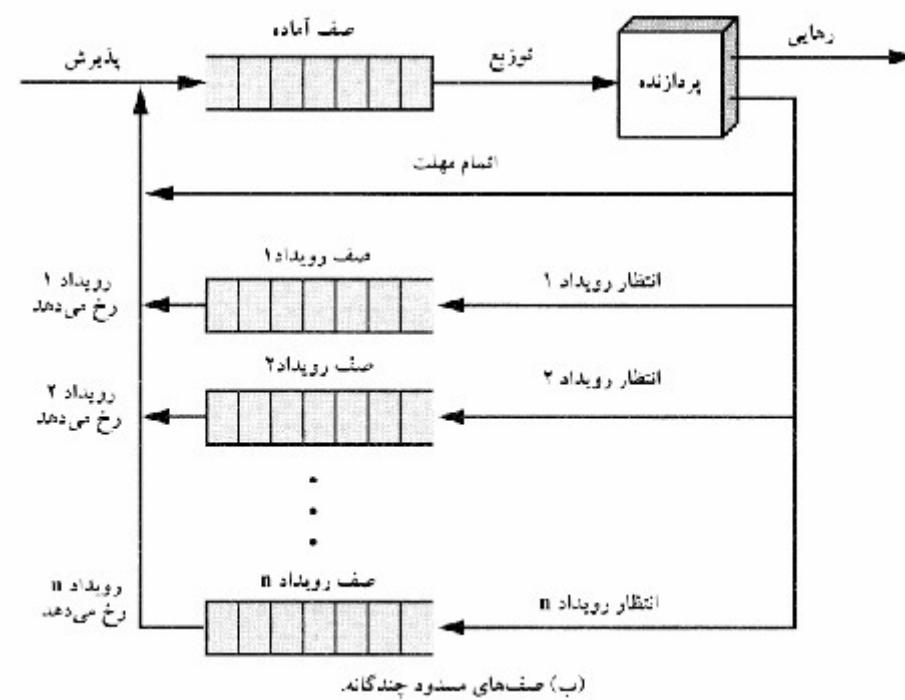
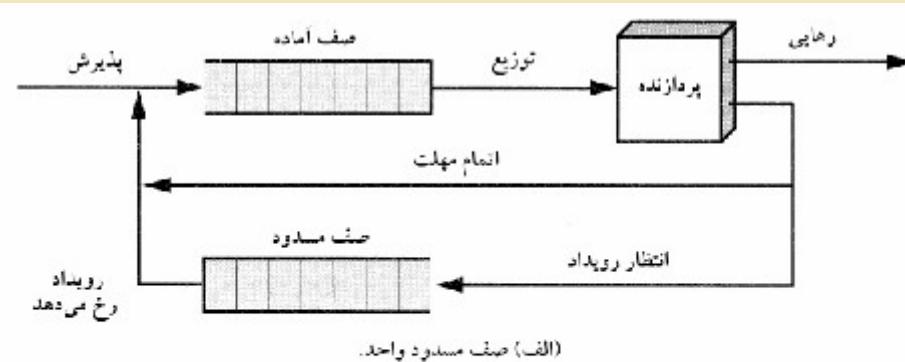
مدل پنج حالته برای فرایند



انواع حوادثی که منجر به تغییر حالت شده

- تهی ← آماده: فرایند جدیدی را برای اجرای یک برنامه ایجاد می کند.
- جدید ← آماده: آمادگی برای گرفتن یک برنامه.
- آماده ← اجرا: زمان انتخاب یک فرآیند رسیده.
- اجرا ← خروج: فرآیند جاری اعلام می کند که تمام شده.
- اجرا ← آماده: اتمام زمان مجاز برای یک فرآیند.
- اجرا ← مسدود: درخواست یک منبع با انتظار.
- مسدود ← آماده: حادثه مورد نظر اتفاق افتاده.
- آماده ← خروج: با تصمیم پدر.
- مسدود ← خروج: توضیح تغییر حالت قبل در اینجا نیز صادق است.

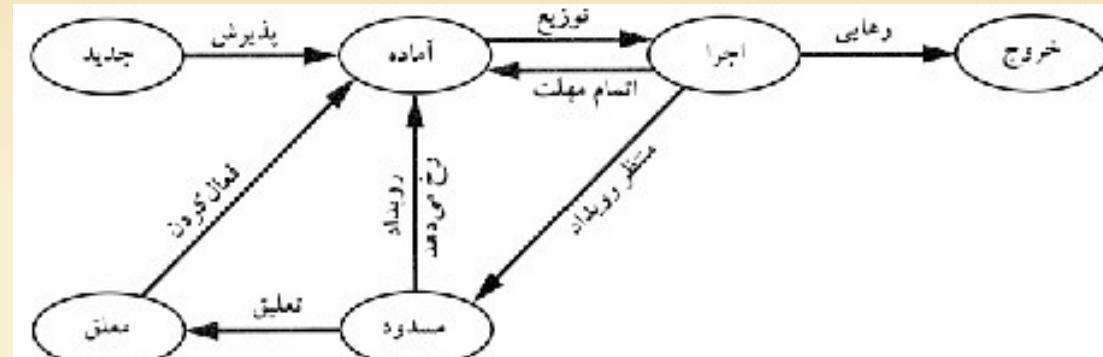
مدل صفحه بندی



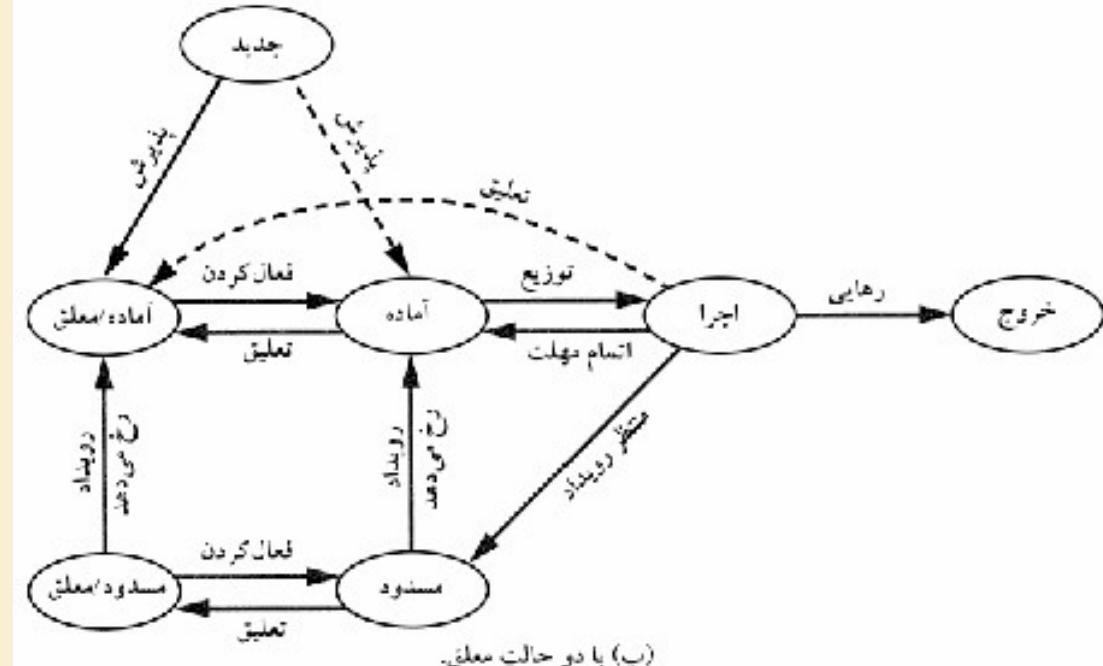
فرآیند معلق (Suspend)

- اگر هیچ کدام از فرآیندهای موجود در حافظه در حالت آماده نباشند، سیستم عامل یکی از فرآیندهای مسدود را به صفت معلق بر روی دیسک انتقال می‌دهد (مبادله به خارج). این صفت، حاوی فرآیندهایی است که موقتاً از حافظه اصلی خارج یا معلق شدند. سپس سیستم عامل فرآیند دیگری را از صفت معلق وارد حافظه می‌کند یا درخواست فرآیند جدید را می‌پذیرد (مبادله به داخل). فرآیند تازه وارد اجرا می‌شود.
- وقتی تمام فرآیندهای موجود در حافظه در حالت مسدود باشند، سیستم عامل می‌تواند فرآیندی را به **تعليق** درآورد. برای این منظور، آن را در حالت معلق قرار می‌دهد و به دیسک منتقل می‌کند. با خروج فرآیند از حافظه اصلی، می‌تواند فرآیند جدیدی را در حافظه قرار داد.
- وقتی سیستم عامل یک عمل مبادله به خارج را انجام داد، برای وارد کردن فرآیندی به حافظه، دو انتخاب در پیش دارد.
 - پذیرش فرآیندی است که جدیداً ایجاد شده.
 - پذیرش فرآیندی است که قبلاً معلق شده است. به نظر می‌رسد که بهتر است فرآیند معلق به حافظه اورده شود، زیرا وارد کردن فرآیند جدید به سیستم، بار آن را افزایش می‌دهد.

نمودار تغییر حالت فرآیند، با حالت معلق



(الف) با یک حالت معلق.



(ب) با دو حالت معلق.

کاربردهای دیگر تعلیق

به طور کلی فرآیندی را متعلق می گوییم که ویژگی های زیر را داشته باشد:

1. فرآیند فوراً آماده اجرا نیست.
2. فرآیند ممکن است منتظر رویدادی باشد یا نباشد. اگر منتظر باشد، این شرط مسدود بودن مستقل از شرایط تعلیق است و وقوع رویداد مسدود کننده نمی تواند فرآیند را قادر به اجرا کند.
3. فرآیند توسط عاملی مثل خودش، فرآیند والد، یا سیستم عامل به حالت متعلق برود تا از اجرای آن جلوگیری شود.
4. تا زمانی که عامل تعلیق صراحتاً دستور ندارد، فرآیند نمی تواند از حالت متعلق خارج شود.

دلایل تعلیق فرآیند

- مبادله: سیستم عامل نیاز به حافظه کافی دارد.
- دلایل دیگر سیستم عامل: فرآیندی را که سودمند یا لازم و... است را معلق کند.
- درخواست کاربر محاوره‌های: کاربر به منظور استفاده از منبع برنامه را معلق کند.
- ترتیب زمانی: فرآیند به طور دوره‌ای اجرا شود.
- درخواست فرآیند پدر: فرآیند فرزند زا به تعلیق بیندازد.

ساختارهای کنترل فرآیند

- اگر قرار باشد سیستم عامل فرآیندی را مدیریت و کنترل کند، باید اولاً محل قرار گرفتن آن و ثانیاً صفاتی که در مدیریت آن موثر است بداند.
- هر فرآیند دارای یک صفات است که معمولاً همراه آن به سیستم عامل می‌آید که به مجموعه این صفات بلوک کنترل فرآیند (Process) می‌گویند.
- به مجموعه برنامه، داده‌ها، پشته و صفات، تصویر فرآیند (Image) می‌گویند و محل آن به طرح مدیریت حافظه بستگی دارد و در حافظه ثانویه نگهداری می‌شود و برای اجرا باید به حافظه اصلی بار شد.

اجزای متداول تصویر یک فرآیند

داده های کاربر

بخش قابل تغییر فضای کاربر. ممکن است شامل داده های برنامه، ناحیه پشته کاربر، و برنامه هایی باشد که باید تغییر کنند.

برنامه کاربر

برنامه ای که باید اجرا شود.

پشته سیستم

همراه هر فرآیند یک یا چند پشته سیستم وجود دارد. پشته برای ذخیره پارامترهای و آدرس های فراغوانی مربوط به رویه ها و سیستم ها استفاده می شود.

بلوک کنترل فرآیند

داده های مورد نیاز سیستم عامل برای کنترل فرآیند.

عناصر بلوک کنترل فرآیند

شناسه فرآیند

شناسه ها

شناسه های عدی که می توانند با بلوک کنترل فرآیند ذخیره شوند عبارت اند از:

- شناسه این فرآیند
- شناسه فرآیندی که این فرآیند را ایجاد کرد (فرآیند والد)
- شناسه کاربر

اطلاعات حالت فرآیند

ثبتات های کاربر

ثبتات های وضعیت و کنترل

این ها ثبات هایی هستند که برای کنترل عمل پردازنده به کار می روند:

- شمارنده برنامه حاوی دستور العمل بعدی است که باید واکشی شود.
- کدهای وضعیت نتیجه آخرین عمل محاسباتی یا منطقی (علامت، صفر، رقم نقلی، مساوی، سرریز).
- اطلاعات وضعیت شامل پرچم های فعال سازی و غیرفعال سازی وقفه، و حالت اجرا است.

اشارة گرهای پشته

هر فرآیند حاوی یک یا چند پشته سیستم است. پشته برای ذخیره پارامترها و آدرس فراخوانی رویه و سیستم است. اشاره گر پشته بر بالای پشته اشاره می کند.

اطلاعات کنترل فرآیند

اطلاعات حالت و زمان بندی

سیستم عامل برای زمان بندی به این اطلاعات نیاز دارد. این اقلام اطلاعاتی عبارت اند از:

- حالت فرآیند آمادگی فرآیند برای زمان بندی و اجرا را تعریف می کند (اجرا، آماده، منتظر، متوقف).
- اولویت یک یا جند فیلد ممکن است برای توصیف اولویت زمان بندی فرآیند استفاده شوند.

• اطلاعات مربوط به زمان بندی به الگوریتم زمان بندی بستگی دارد. مثال ها عبارت اند از مدت انتظار فرآیند، مدت اجرای فرآیند در آخرین باری که در حال اجرا بود.

- رویداد هویت رویدادی که فرآیند قبل از ازسرگیری منتظر آن است.

ساختمنان داده ها

هر فرآیندی ممکن است به فرآیند دیگری در یک صفحه، حلقه یا ساختمنان دیگری پیوند داده شود.

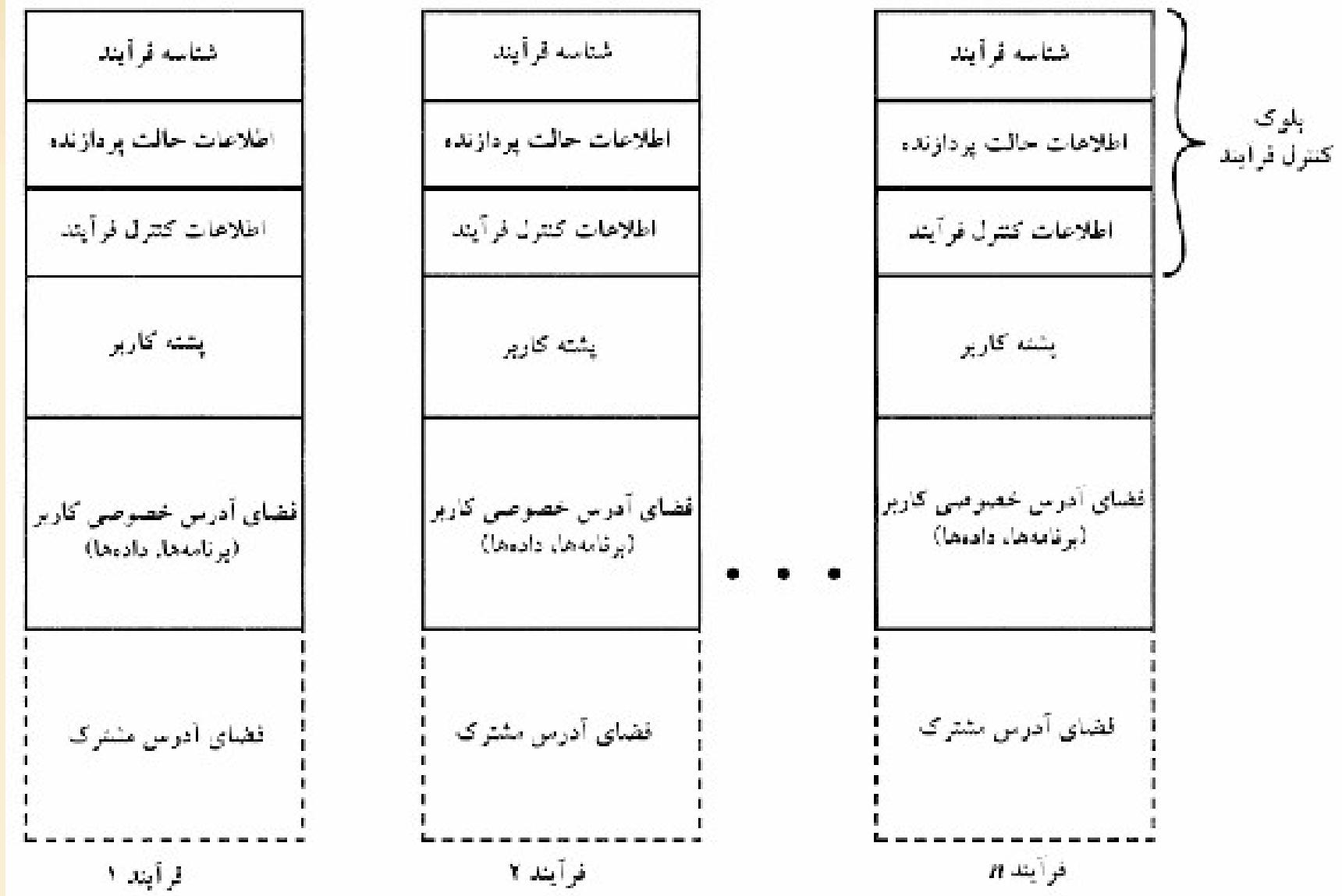
ارتباط بین فرآیندها

امتیازات فرآیند

مدیریت حافظه

مالکیت و برهه وری از منبع

فرآیند کاربر در حافظه مجازی



کلمه وضعیت برنامه

Program Statue Word (PSW)

- طراحی تمام پردازندۀ ها شامل یک ثبات یا مجموعه‌ای از ثبات‌ها است که به نام کلمه وضعیت برنامه (PSW) شناخته می‌شود و حاوی اطلاعات وضعیت است.

فصل سوم

نَخْ

Thread

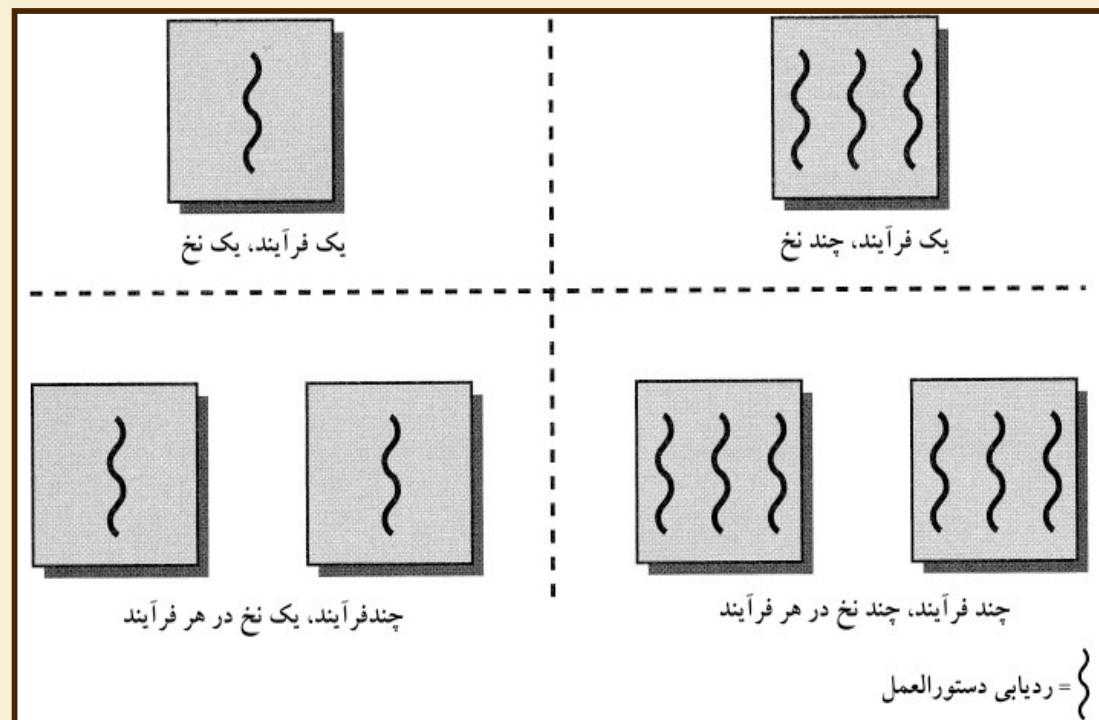
فرایندها و نخها

بحثی که تاکنون ارائه شد، دو ویژگی برای فرآیند مطرح کرده است:

- **تملک منبع:** هر فرآیند حاوی یک فضای آدرس مجازی برای نگهداری تصویر فرآیند است
- **توزیع وقت پردازنده/اجرا:** هر فرآیند دارای یک حالت اجرا (در حال اجرا، آماده و غیره) و اولویت توزیع وقت پردازنده است و نهادی است که توسط سیستم عامل زمان بندی و توزیع وقت می شود.
- برای تمایز این دو ویژگی، واحد توزیع وقت پردازنده به نام **نخ** (thread) یا **فراینده سبک وزن** (light weight process) خوانده می شود، در حالی که واحد مالکیت منبع به نام **فرآیند** یا **وظیفه** خوانده می شود.

چند نخ

- رویکرد سنتی یک نخ اجرا در هر فرایند می باشد (که در آن مفهوم نخ منظور نبوده است) رویکرد تک نخی می گویند.
- منظور از چندنخی، توانایی سیستم عامل در پشتیبانی از اجرای چندین نخ در داخل یک فرآیند است.

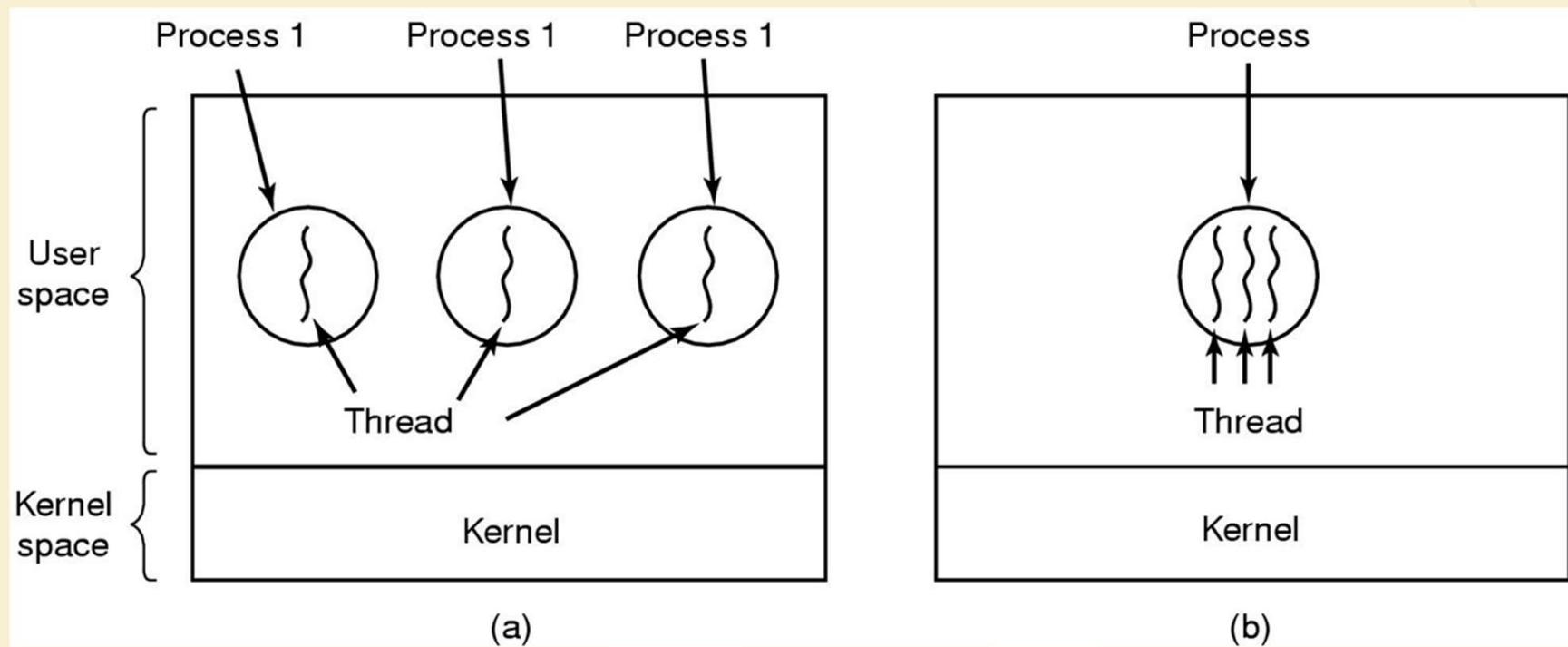


چند نخی (ادامه)

- در محیط چندنخی، هر فرآیند به عنوان یک واحد تخصیص حافظه و یک واحد حفاظت تعریف می شود.
- موارد زیر نیز به همراه فرآیندها وجود دارند:
 - فضای آدرس مجازی که تصویر فرآیند را نگهداری می کند.
 - دستیابی حفاظت شده به پردازنده ها، فرآیندهای دیگر (برای ارتباط بین فرآیندها)، فایل ها و منابع I/O (دستگاه ها و کانال ها).

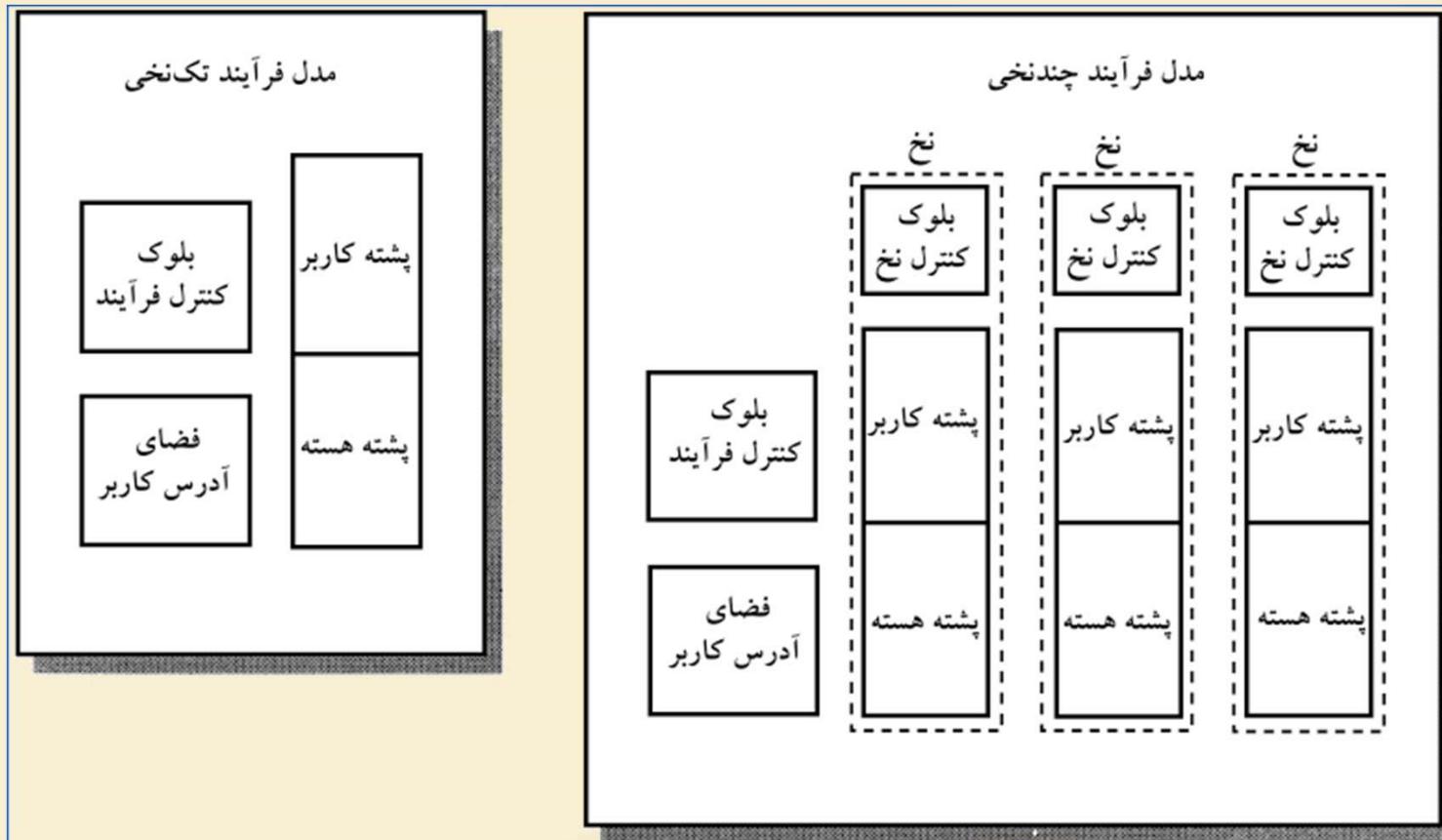
چند نخی (ادامه)

- (a) Three processes each with one thread
- (b) One process with three threads



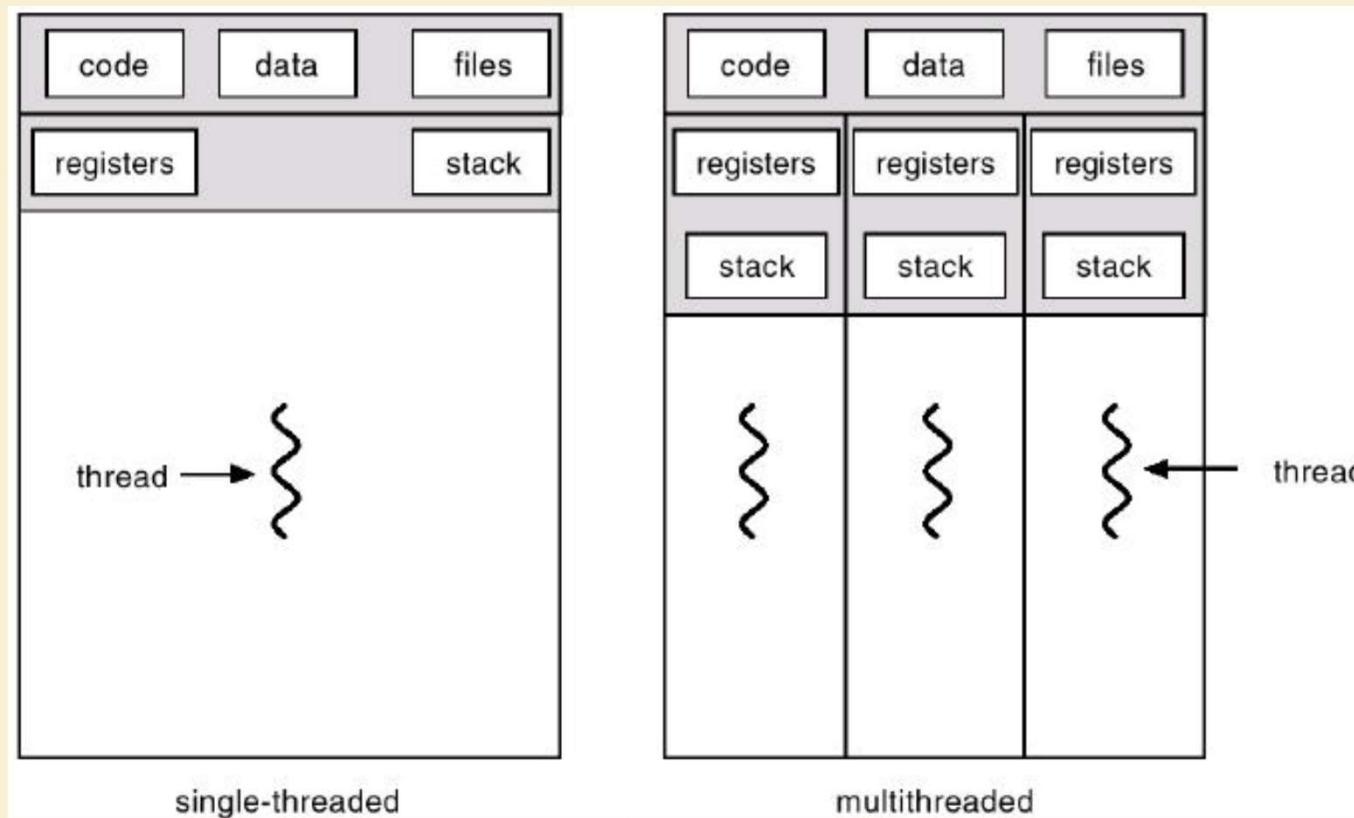
چند نخی (ادامه)

■ مدل های تک نخی و چندنخی



چند نخی (ادامه)

مدل های تک نخی و چندنخی ■



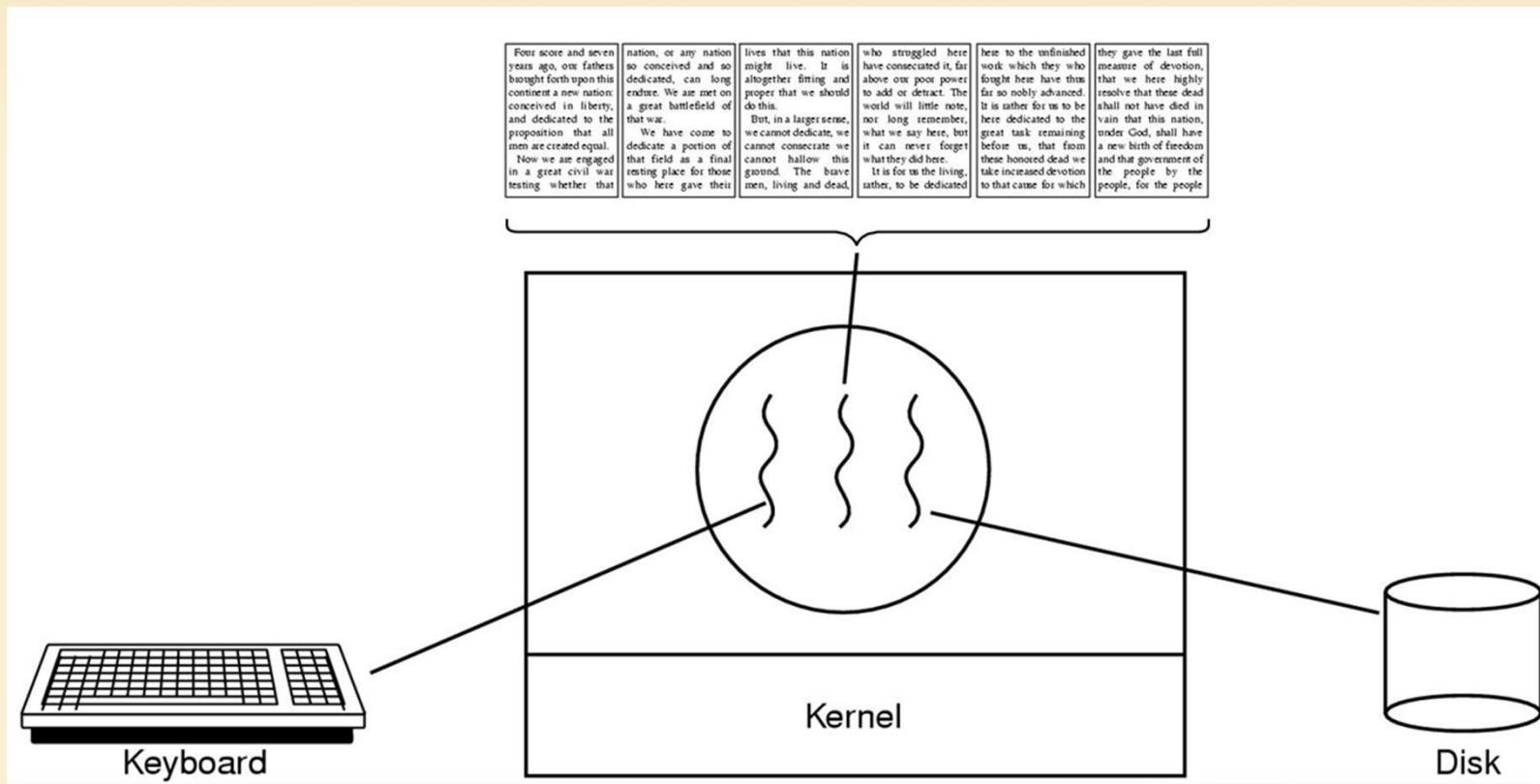
چند نخی (ادامه)

فایده نخ ها، نقش مهم آن ها در کارایی است:

1. ایجاد نخ در فرآیند موجود، خیلی کمتر از ایجاد فرآیند وقت می گیرد.
2. خاتمه دادن به نخ سریع تر از خاتمه دادن به فرآیند است.
3. تعویض نخ در داخل یک فرآیند، زمان کمتری می برد.
4. ارتباط بین فرایندها مستلزم دخالت هسته برای حفاظت و تهییه راهکارهای مورد نیاز ارتباط است. ولی برای نخ دخالت هسته نیاز نیست.

اگر یک خدمتگزار چند پردازنده ای باشد در این صورت نخهای چندگانه داخل آن فرایند می توانند به صورت همزمان روی پردازنده های مختلف اجرا شوند. ■

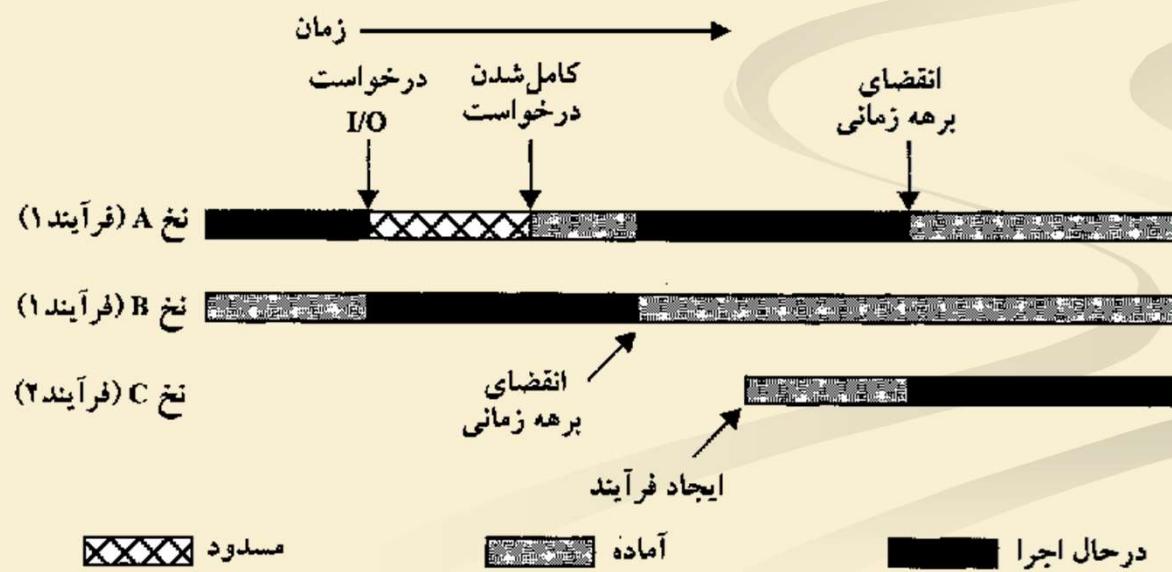
چند نکی (ادامہ)



A word processor with three threads

عملکرد نخ

- مهم ترین حالت های نخ ها عبارت اند از در حال **اجرا**، **آماده** و **مسدود**. معمولاً نخ ها قادر حالت **تعليق** هستند، زیرا این حالت در سطح فرآیندها مطرح است.
- در محیط چند پردازنده‌ای، روش چند برنامه‌ای این امکان را فراهم می‌کند که چندین نخ در داخل چندین فرآیند در بین یکدیگر قرار گیرند.
- در مثال زیر، سه نخ در دو فرآیند بر روی یک پردازنده بین یکدیگر قرار گرفته‌اند. وقتی نخ در حال اجرا مسدود می‌شود یا برهه زمانی آن خاتمه می‌یابد، اجرا به نخ دیگری منتقل می‌شود.



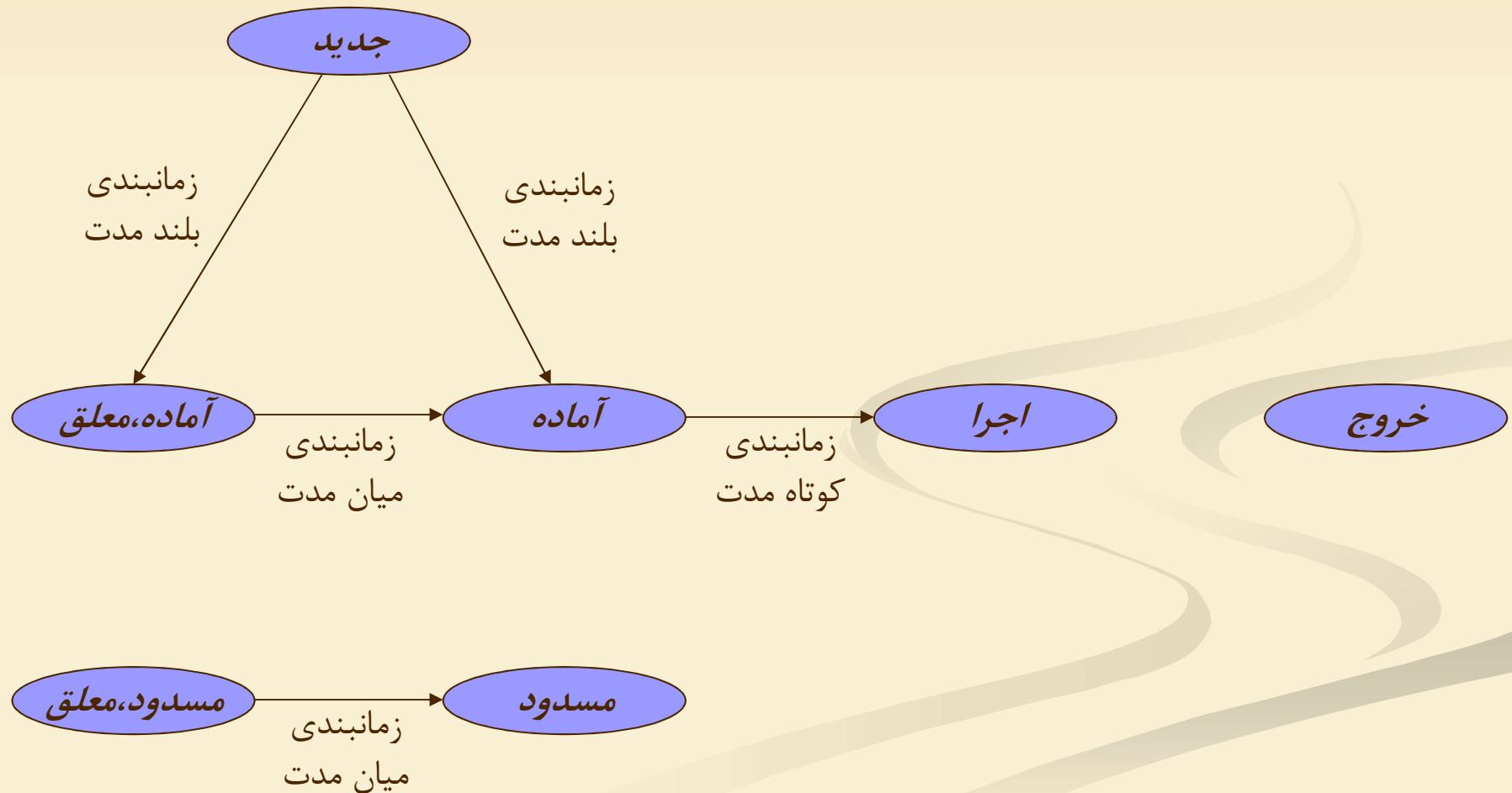
فصل چهارم

زمانبندی تک پردازنده‌ای

زمانبندی

- هدف از زمان بندی، تخصیص فرآیندها به پردازنده ها در طول زمان است به گونه ای که به هدف های سیستم از قبیل زمان پاسخ، توان عملیاتی و کارآیی پردازنده برسد.
- انواع زمانبندی ها
 - زمانبندی بلند مدت
 - تصمیم گیری در مورد افزودن به مجموعه فرایندها برای اجرا.
 - زمانبندی میان مدت
 - تصمیم گیری در مورد افزودن به تعداد فرایندها.
 - زمانبندی کوتاه مدت
 - کدام فرایند برای اجرا فرستاده شود.
 - زمانبندی ورودی / خروجی
 - کدام درخواست ورودی و خروجی فرایند توسط کدام دستگاه اجرا شود.

زمانبندی و تغییر حالت‌های فرآیند



معیارهای زمانبندی

ملاکهایی که یک الگوریتم زمانبندی کوتاه مدت خوب باید دارا باشد :

- ۱- عدالت (Fairness) : هر پروسس سهم عادلانه ای از CPU را دریافت نماید.
- ۲- کارایی (Efficiency) : CPU بیکار نماند و وقتی پروسس امکان جلو رفتن ندارد CPU به پروسس دیگری داده شود .
- ۳- زمان پاسخ (Response Time) : یعنی به حداقل رساندن فاصله زمانی بین ارائه یک تقاضا تا شروع ظهور پاسخ آن.
- ۴- حداقل بودن زمان بازگشت (Turnaround Time) : زمان بازگشت برای یک کار Batch طول زمان از لحظه ورود آن به سیستم تا لحظه پایان یافتن (کامل شدن) آن می باشد. ولی زمان پاسخ زمانی است که از صدور یک تقاضا تا تولید اولین پاسخ آن طول می کشد(نه زمان خروجی کل برنامه).
- ۵- حداقل شدن Throughput: تعداد کارهایی است که در واحد زمان انجام می شود.
- ۶- زمان انتظار (Waiting time): مجموع پریودهای زمانی صرف شده در صف آماده می باشد.

سیاست های زمانبندی

تابع انتخاب فرایند بعدی را از میان فرایند های آماده اجرا انتخاب می کند.

■ سه کمیت زیر مهم است:

■ 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. این کمیت باید تخمین زده شده یا توسط کاربر مشخص شود.

حالت تصمیم‌گیری

زمانی که تابع انتخاب به اجرا در می‌آید را مشخص می‌کند . دو گروه بندی زیر را دارد:

1. بدون قبضه کردن (Nonpreemptive) ■
تا تکمیل شدن برود یا خودش برای I/O گرفتن مسدود شود.

2. با قبضه کردن (Preemptive) ■
تصمیم به قبضه کردن می‌تواند هنگام ورود و... ایجاد شود.

زمانبندی خدمت به ترتیب ورود FCFS

First Come First Service (FCFS)

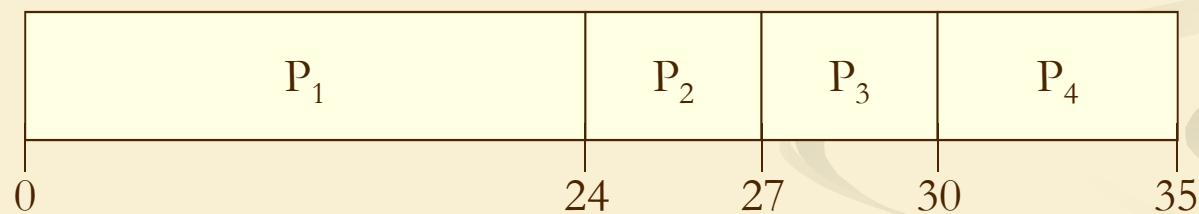
- ساده ترین الگوریتم
- هر فرآیند با آماده شدن به صف آماده ملحق شده وقتی فرایند جاری از اجرا باز ماند قدیمیترین فرایند در صف در صف آماده به اجرا قرار می گیرد.

زمانبندی خدمت به ترتیب ورود FCFS

- تابع انتخاب : $\max[w]$
- حالت تصمیم گیری : بدون قبضه
- توان عملیاتی : تاکید نشده است
- زمان پاسخ : می تواند زیاد باشد
- سربار: حداقل
- تاثیر بر روی فرآیند ها: به فرایند کوتاه صدمه می زند
- گرسنگی : خیر

زمانبندی خدمت به ترتیب ورود FCFS

Process	Arrival Time	Burst Time
P ₁	0	24
P ₂	0	3
P ₃	0	3
P ₄	0	5



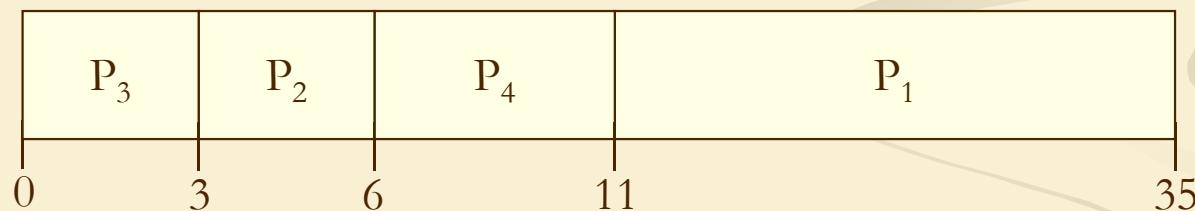
زمان انتظار P₁=0; P₂=24; P₃=27; P₄=30

(0+24+27+30)/4=20.25 متوسط زمان انتظار

زمانبندی خدمت به ترتیب ورود FCFS

فرض کنید ترتیب ورود بصورت زیر باشد.

Process	Arrival Time	Burst Time
P ₃	0	3
P ₂	0	3
P ₄	0	5
P ₁	0	24



$P_1=11$; $P_2=3$; $P_3=0$; $P_4=6$: زمان انتظار

($11+3+0+6$)/4=5: متوسط زمان انتظار

زمانبندی نوبت گردشی RR

Round Robin (RR)

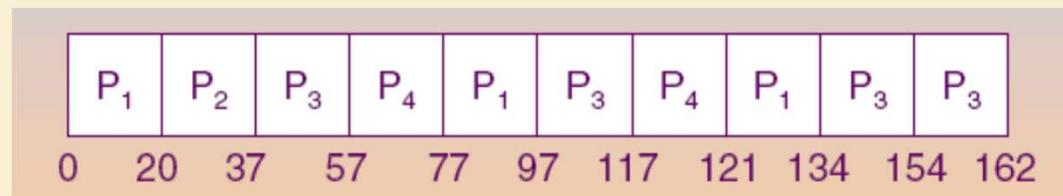
- یکی از رایج ترین و ساده ترین الگوریتمهای زمانبندی است . پیاده سازی آن بسیار ساده است . کافی است یک لیستی از پروسسهای آماده اجرا نگهداری شود .
- به هر پروسس یک Quantum (کوانتم) یا Time-slice (برش زمانی) CPU داده می شود . اگر پروسس در پایان کوانتم هنوز خاتمه نیافته باشد ، CPU از آن گرفته می شود و به پروسس بعدی در صف داده می شود .
- یک وقفه ساعت در فواصل زمانی دوره ای تولید می گردد با بروز وقفه فاریندی که در حال اجراست در صف آماده قرار می گیرد و کار آماده بعدی براساس FCFS انتخاب می شود

زمانبندی RR

- تابع انتخاب: ثابت
 - حالت تصمیم گیری: با قبضه کردن
 - توان عملیاتی: اگر برهه زمانی خیلی کوچک باشد کم
 - زمان پاسخ: برای فرایند های کوتاه زمان بسیار خوب
 - سربار: کم
 - تاثیر بر روی فرآیند ها: عملکرد عادلانه
 - گرسنگی: خیر
- غیر انحصاری، به هر پردازش مقدار مشخصی ϕ_{II} می دهند. به عبارت بهتر زمان استفاده از آن به کوانتوم هایی تقسیم می شود.

زمانبندی RR

Process	Arrival Time	Burst Time
P ₁	0.0	53
P ₂	0.0	17
P ₃	0.0	68
P ₄	0.0	24



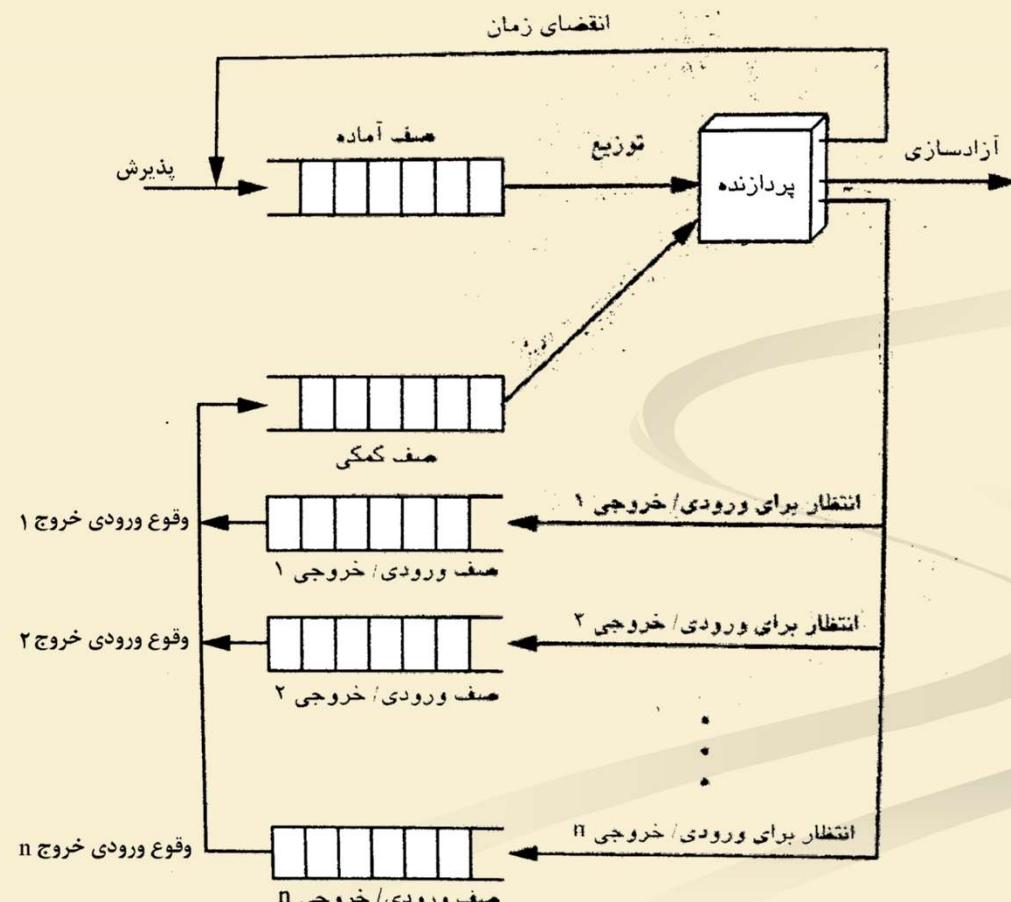
زمان انتظار P₁=81; P₂=20; P₃=94; P₄=97

متوسط زمان انتظار (81+20+94+97)/4=73

زمانبندی نوبت گردشی مجازی VRR

Virtual Round Robin

RR به نفع فرایندهای در تنگنای پردازنده عمل می کند. ■



شکل ۷-۹: نمودار صفت‌بندی برای زمانبندی نوبت گردشی مجازی

زمانبندی کوتاه ترین فرایند SPN

Shortest Process Next (SPN)

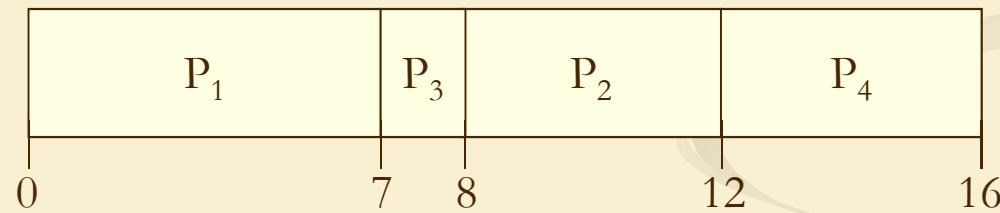
- در این سیاست بدون قبضه کردن فرایند هایی که زمان پاسخ کوتاهی دارند اول اجرا می شوند. و فرآیند کوتاه از روی کارهای طولانی می گذرد و در ابتدای صف قرار میگیرد.
- امکان دارد فرایندهای بزرگ هیچگاه انجام نشود.
- یک مشکل SPN نیاز به دانستن زمان پردازش هر فرایند و یا لااقل تخمین این زمان است.

زمانبندی SPN

- تابع انتخاب: $\min[s]$
- حالت تصمیم گیری: بدون قبضه کردن
- توان عملیاتی: زیاد
- زمان پاسخ: برای فرایند های کوتاه خوب
- سربار: می تواند زیاد باشد
- تاثیر بر روی فرآیند ها: به فرایند های طولانی صدمه زده
- گرسنگی (starvation): امکان دارد فرایند هایی که زمان پاسخ کوتاهی دارند اول اجرا می شوند.

زمانبندی SPN

Process	Arrival Time	Burst Time
P ₁	0.0	7
P ₂	2.0	4
P ₃	4.0	1
P ₄	5.0	4



زمان انتظار P₁=0; P₂=6; P₃=3; P₄=7

متوسط زمان انتظار $(0+6+3+7)/4=4$

زمانبندی کوتاه‌ترین زمان باقیمانده

SRT

Shortest Remaining Time

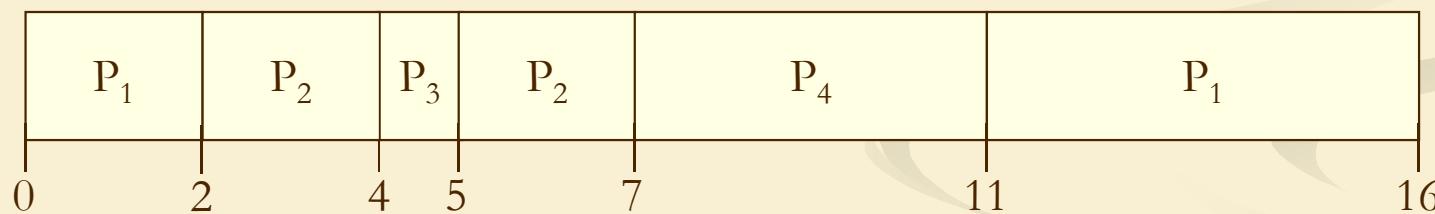
- فرایندی اجرا می‌شود که کمترین باقی‌مانده زمان پاسخ را داشته باشد.
- وقتی فرایندی به صفت آماده وارد می‌شود امکان دارد زمان باقی‌مانده کمتری نسبت به فرایند در حال اجرا داشته باشد که با عمل قبضه کردن آن فرایند به اجرا در می‌آید.
- نوعی SPN با قبضه کردن است.
- مانند SPN باید تخمینی از زمان پردازش داشته باشیم.
- خطر گرسنگی فرایندهای طولانی‌تر وجود دارد.

SRT زمانبندی

- تابع انتخاب: $\min[s-e]$
 - حالت تصمیم گیری: با قبض کردن
 - توان عملیاتی: زیاد
 - زمان پاسخ: خوب
 - سربار: می تواند زیاد باشد
 - تاثیر بر روی فرآیند ها: به فرایند های طولانی صدمه زده
 - گرسنگی: امکان دارد
- فرایندی اجرا می شود که کمتری باقی مانده زمان پاسخ را داشته باشد.

SRT زمانبندی

Process	Arrival Time	Burst Time
P ₁	0.0	7
P ₂	2.0	4
P ₃	4.0	1
P ₄	5.0	4



زمان انتظار: P₁=9; P₂=1; P₃=0; P₄=2

متوسط زمان انتظار: $(9+1+0+2)/4=3$

زمانبندی بالاترین نسبت پاسخ HRRN

Highest Response Ratio Next

- اولویت داینامیک بوده و کارهای کوتاه تر اولویت بیشتر داشته و زودتر اجرا می شوند.

$$\text{فرمول تشخیص بالاترین پاسخ : } \max(w+s/s) \quad ■$$

- W = زمان انتظار برای پردازنده
- S = زمان خدمت مورد انتظار

- سن فرایند نیز دخالت داده می شود.

همانند SRT و SPN زمان خدمت مورد انتظار باقیستی قبل از به کارگیری HRRN تخمین زده شود.

زمانبندی HRRN

- تابع انتخاب: $\max(w+s/s)$
- حالت تصمیم گیری: بدون قبضه کردن
- توان عملیاتی: زیاد
- زمان پاسخ: خوب
- سربار: می تواند زیاد باشد
- تاثیر بر روی فرآیند ها: توازن مناسب
- گرسنگی: خیر
- اولویت داینامیک بوده و کارهای کوتاه تر اولویت بیشتر داشته و زودتر اجرا می شوند.

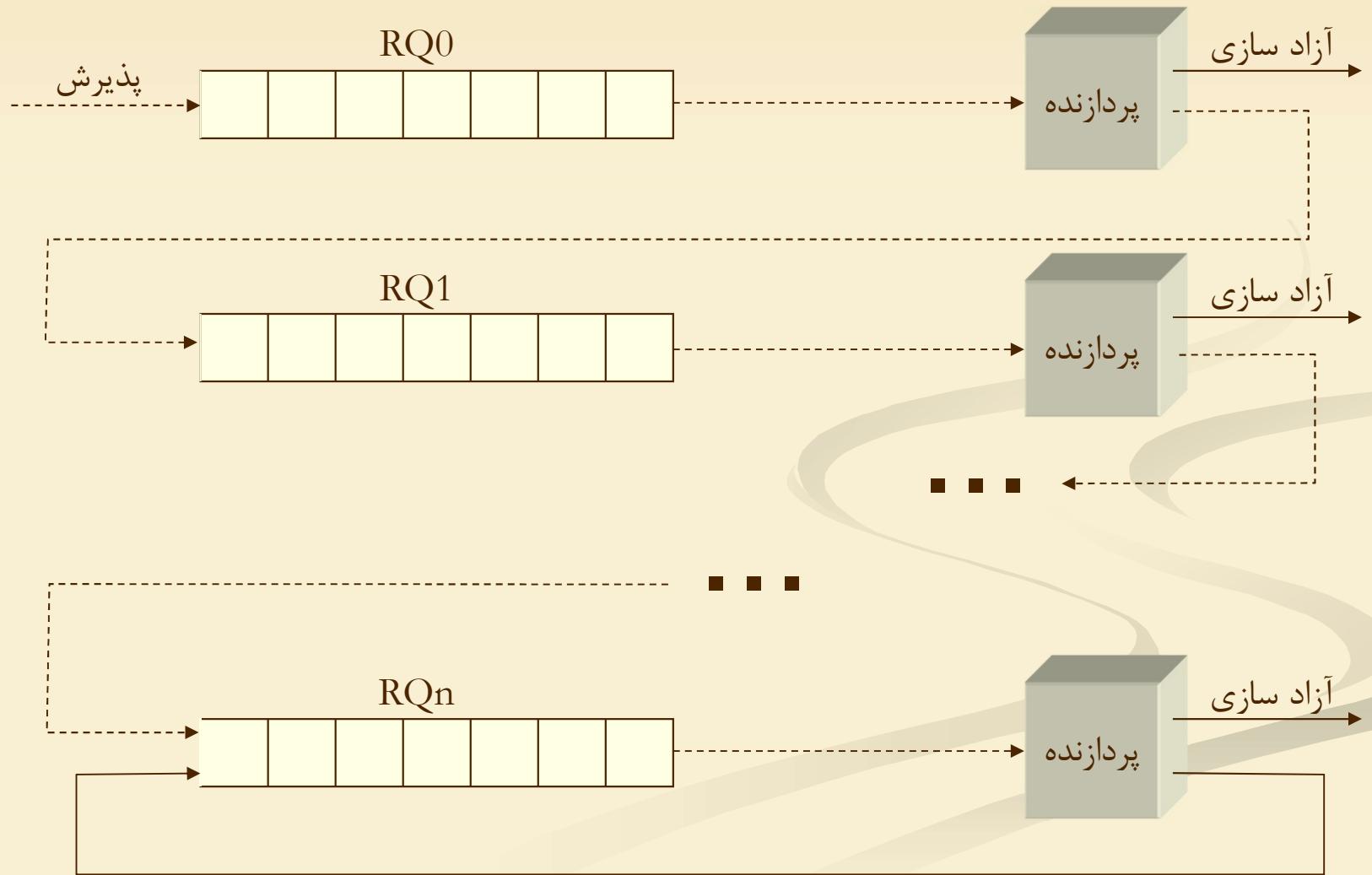
زمانبندی فید بک FB

Feedback

اگر هیچ نشانه ای از طول نسبی فرایندها متفاوت نداشته باشیم هیچ یک از روش‌های HRRN ، SRT ، SPN را نمی توان بکار برد. راه دیگر:

- از طریق جریمه کردن کارهایی است که زمان طولانی تری به اجرا در آمده اند.
- بهتر است روی زمان سپری شده تمرکز کنیم.
- از قبضه کردن در زمان کوانتم صورت می گیرد.
- از اولویت پویا استفاده می شود.
- هر فراید پس از اجرا به صف کم اولویت تر بعدی منتقل می شود.
- در هر صف یک راهکار ساده FCFS استفاده می شود.
- وقتی به در صف کمترین اولویت قرار گرفت نمی تواند پایین تر برود و مکرراً به همان صف باز گردانده می شود.

زمانبندی فید بک FB



زمانبندی FB

- تابع انتخاب:
 - حالت تصمیم گیری: با قبضه کردن
 - توان عملیاتی: تاکید نشده است
 - زمان پاسخ: تاکید نشده است
 - سربار: می تواند زیاد باشد
 - تاثیر بر روی فرآیند ها: به نفع فرایند های در تنگنای I/O
 - گرسنگی: امکان دارد
- از طریق جریمه کردن کارهایی که زمان طولانی تری برای اجرا دارند.

مقایسه کارآمدی

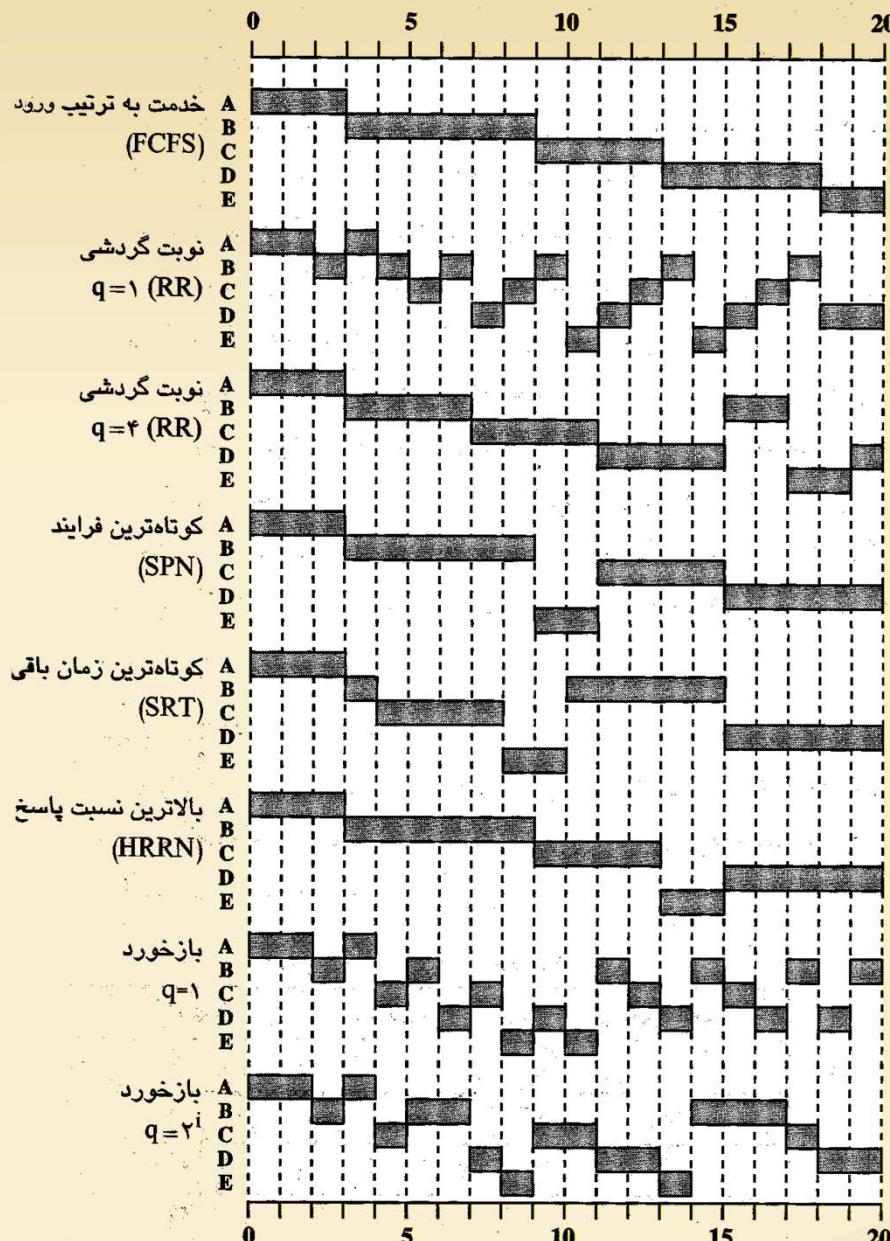
زمان کل (TAT) turnaround time (TAT)

- عبارت است از زمان اقامت (T_r) یا کل زمانی که فرایнд در سیستم می‌گذراند.
- زمان انتظار به علاوه زمان خدمت.

■ مثال: زمانبندی فرایند

فرایند	زمان ورود	زمان خدمت
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

مثال: زمانبندی فرایند



شکل ۹-۵: مقایسه سیاستهای زمانبندی

فرایند	زمان ورود	زمان خدمت
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

مثال: زمانبندی فرایند

میانگین	E	D	C	B	A	فرایند زمان ورود زمان خدمت (T_s)	
۸/۶	۲۰	۱۸	۱۳	۹	۳	زمان پایان زمان کل (T_r) T_r/T_s	FCFS
۲/۵۶	۶	۲/۴۰	۲/۲۵	۱/۱۷	۱		
۱۰/۸۰	۱۵	۲۰	۱۷	۱۸	۴	زمان پایان زمان کل (T_r) T_r/T_s	RR (q=1)
۲/۷۱	۳/۵۰	۲/۸۰	۳/۲۵	۲/۶۷	۱/۳۳		
۱۰	۱۹	۲۰	۱۱	۱۷	۳	زمان پایان زمان کل (T_r) T_r/T_s	RR (q=4)
۲/۷۱	۱۱	۱۴	۷	۱۵	۳		
۰/۵۰	۰/۵۰	۲/۸۰	۱/۷۵	۲/۵	۱		
۷/۶۰	۱۱	۲۰	۱۵	۹	۳	زمان پایان زمان کل (T_r) T_r/T_s	SPN
۱/۸۴	۳	۱۴	۱۱	۷	۳		
۱/۵۰	۱/۵۰	۲/۸۰	۲/۷۵	۱/۱۷	۱		
۷/۲۰	۱۰	۲۰	۸	۱۵	۳	زمان پایان زمان کل (T_r) T_r/T_s	SRT
۱/۰۹	۲	۱۴	۴	۱۳	۳		
۱	۱	۲/۸۰	۱	۲/۱۷	۱		
۸	۱۵	۲۰	۱۳	۹	۳	زمان پایان زمان کل (T_r) T_r/T_s	HRRN
۲/۱۴	۷/۵۰	۲/۸۰	۲/۲۵	۱/۱۷	۱		
۱۰	۱۱	۱۹	۱۶	۲۰	۴	زمان پایان زمان کل (T_r) T_r/T_s	FB (q=1)
۲/۲۹	۳	۱۳	۱۲	۱۸	۴		
۱/۵۰	۱/۵۰	۲/۶۰	۳	۳	۱/۳۳		
۱۰/۶۰	۱۴	۲۰	۱۸	۱۷	۴	زمان پایان زمان کل (T_r) T_r/T_s	FB (q=2)
۲/۶۳	۶	۱۴	۱۴	۱۵	۴		
۳	۲/۸۰	۳/۵۰	۲/۵۰	۱/۳۳			

جدول ۵-۹: مقایسه سیاستهای زمانبندی

فصل پنجم

همزمانی:
انحصار متقابل و همگام سازی

مقدمه

- همه موضعات محوری در طراحی سیستم عامل به مدیریت فرایند ها و نخ ها مربوط است.
- چند برنامه ای : مدیریت فرایند فرایندهای متعدد در داخل یک کامپیوتر تک پردازنده ای.
- چند پردازشی: مدیریت فرایند فرایندهای متعدد در داخل یک کامپیوتر چند پردازنده ای.
- پرداش توزیعی: مدیریت فرایند فرایندهای متعدد در روی سیستم های کامپیوتري متعدد. برای هر سه زمینه فوق مسئله هم زمانی است.
- نیاز اصلی برای حمایت از فرآیند های همزمان، توان اعمال انحصار متقابل (mutual exclusion) است.
- یعنی وقتی به یک فرآیند قدرت انجام عملی داده شد. بتوان تمام فرایندها دیگر را از این قدرت بازداشت.

اصول همزمانی

- در یک سیستم تک پردازندۀ ای و چند برنامه ای، فرایندها در طول زمان در بین یکدیگر اجرا می شوند تا اجرای همزمان را نشان دهند.
- در این روش مشکلات زیر پیش می آید:
 - ۱- اشتراک منابع سراسری پر مخاطره است.
 - ۲- مدیریت تخصیص بهینه منابع برای سیستم عامل است.
 - ۳- تعیین محل خطای برنامه سازی مشکل می شود.

اصول همزمانی

یک مثال ساده ■

```
void echo ()  
{  
    chin = getch ();  
    chout = chin;  
    putch ( chout );  
}
```

رویه زیر را در نظر بگیرید ■

دنباله زیر را در نظر بگیرید ■

Process P1

```
    chin = getch ();  
    .  
    .  
    .  
    chout = chin;  
    putch ( chout );
```

Process P2

```
    chin = getch ();  
    chout = chin;  
    putch ( chout );  
    .
```

شرط مسابقه (Race Condition)

- شرط مسابقه وقتی به وجود می آید که چندین فرآیند یا نخ اقلام داده ای را بطور همزمان می خوانند و می نویسند. نتیجه نهايی داده های اشتراكی بر اين اساس مشخص می شود که چه فرآيندي آخر پايان می پذيرد.
- برای جلوگیری از شرایط مسابقه، فرآيندهای همزمان باید هماهنگ شوند.

ملاحظات سیستم عامل در همزمانی

- سیستم عامل باید بتواند فرایندهای فعال مختلف را دنبال کند که این کار توسط بلوک های کنترل فرایند انجام می شود.
- سیستم عامل باید منابع را به هر یک از فرایندها تخصیص دهد و یا باز پس بگیرد از جمله:
(وقت پردازنده، حافظه، پرونده ها، دستگاههای ورودی و خروجی)
- سیستم عامل باید داده ها و منابع فیزیکی هر فرایند را در مقابل دخالت ناخواسته فرایند دیگر محافظت کند.
- نتایج یک فرایند باید مستقل از سرعت پیشرفت اجرای فرایندهای همزمان دیگر باشد (موضوع این فصل).

محاوره فرآینده ها

- **محاوره فرآیندها** : ارتباطات بین فرآیندها و نخ ها
- می توان محاوره ها را بر اساس اطلاع فرآیندها از وجود یکدیگر دسته بندی کرد:
 - **بی اطلاعی فرآیند ها از یکدیگر**: فرآیند ها مستقل هستند.
 - **رقابت**
- **اطلاع غیر مستقیم فرآیندها از یکدیگر** : دسترسی به بعضی از اشیاء یکدیگر.
- **همکاری بوسیله اشتراک**
- **اطلاع از یکدیگر** : با نام ، با یکدیگر در ارتباط هستند.
- **همکاری توسط ارتباط**

محاوره فرآینده‌ها

مسائل بالقوه کنترل	تأثیر فرآیندها بر یکدیگر	رابطه	درجه آگاهی
<ul style="list-style-type: none"> ■ انحصار متقابل ■ بن بست(منابع تجدیدشدنی) ■ گرسنگی 	<ul style="list-style-type: none"> ■ نتایج یک فرآیند مستقل از فعالیت سایر فرآیندها است. ■ امکان تأثیر در زمان بندی فرآیند. 	رقابت	فرآیندها از یکدیگر بی خبر هستند
<ul style="list-style-type: none"> ■ انحصار متقابل ■ بن بست(منابع تجدیدشدنی) ■ گرسنگی ■ وابستگی داده ها 	<ul style="list-style-type: none"> ■ نتایج فرآیند ممکن است به اطلاعات ناشی از فرآیندهای دیگر بستگی داشته باشد. ■ امکان تأثیر در زمان بندی فرآیند. 	همکاری از طریق اشتراک	فرآیندها به طور غیرمستقیم از یکدیگر خبر دارند (مثلًاً از طریق شیء مشترک)
<ul style="list-style-type: none"> ■ بن بست(منابع مصرفی) ■ گرسنگی 	<ul style="list-style-type: none"> ■ نتایج یک فرآیند ممکن است به اطلاعات ناشی از فرآیندهای دیگر بستگی داشته باشد. ■ امکان تأثیر در زمان بندی فرآیند. 	همکاری از طریق ارتباط	فرآیندها به طور مستقیم از یکدیگر خبر دارند (فراهم بودن عوامل اولیه ارتباط برای آنها)

جدول ۲_۵ محاوره فرآیندها.

رقابت فرایندها برای منابع

- هنگامی که چند فرایند برای استفاده از منابع یکسان رقابت می کنند، به درگیری می رسد.
- در مورد فرایندهای رقیب، با سه مسئله باید برخورد شود.
- ضرورت انحصار متقابل (mutual exclusion) در اجرای بخش بحرانی (critical section)
 - تعریف : بخشی از برنامه که طی آن Shared Memory (حافظه مشترک) مورد دستیابی قرار می گیرد Critical Section (ناحیه بحرانی) یا Critical region (بخش بحرانی) نامیده می شود .
 - فعالیت های محدوده زمانی در هر بخش.
 - بن بست (deadlock)
- تعداد مرتبه های رفتن به ناحیه بحرانی در حضور منتظران محدود است.
- گرسنگی (starversion)

ملزومات انحصار متقابل

هر امکانات یا قابلیتی که برای حمایت از انحصار متقابل تهیه شود باید ملزومات زیر را پاسخگو باشد:

.1 Mutual Exclusion: انحصار متقابل باید اعمال گردد.

.2 Progress: فرایندی که در بخش غیر بحرانی خود متوقف می شود ، باید طوری عمل کند که هیچ دخالتی در فرایند های دیگر نداشته باشد .

.3 Bounded Waiting: برای فرایندی که نیاز به دسترسی یک بخش بحرانی دارد، نباید به تاخیر انداختن نامحدود آن وجود داشته باشد، بن بست یا گرسنگی نمی تواند مجاز باشد.

ملزومات انحصار متقابل

هر امکانات یا قابلیتی که برای حمایت از انحصار متقابل تهیه شود باید
ملزومات زیر را پاسخگو باشد:

- .4 هنگامی که هیچ فرایندی در ناحیه بحرانی نیست ورود یک فرایند
بدون تأخیر است.
- .5 هیچ فرضی درباره تعداد فرایند ها و یا سرعت آنها نمی توان داشت.
- .6 هر فرایندی تنها برای زمان محدودی در ناحیه بحرانی قرار می گیرد.

انحصار متقابل: رویکرد نرم افزاری

■ رویکرد نرم افزاری را میتوان برای فرایند های همزمانی که روی ماشینهای تک پردازنده ای یا چند پردازنده ای که از حافظه مشترک استفاده می کنند، پیاده سازی کرد.

انحصار متقابل: رویکرد نرم افزاری

اولین تلاش

```
do {  
    entry section  
    critical section  
    exit section  
    reminder section  
} while (1);
```

- فقط دو فرایند P_1 و P_2 وجود دارد.
- حالت کلی فرایند P_i (فرایند دیگر P_j)

- فرایندها ممکن است برخی متغیر های عمومی خود را برای هماهنگ سازی فعالیتهای خود به اشتراک بگذارند.

انحصار متقابل: رویکرد نرم افزاری

اولین تلاش

■ متغیرهای اشتراکی

■ int turn

■ مقدار اولیه برابر صفر (int turn=0;)

■ رویه انتظار مشغول (Busy waiting) برای فرایند P_i

```
do {  
    while (turn != i) ;  
    critical section  
    turn = j;  
    reminder section  
} while (1);
```

انحصار متقابل: رویکرد نرم افزاری

اولین تلاش

- فرایندی که منتظر فرصت خود است مشغول است و وقت پردازنه را مصرف می کند.
- دو اشکال این روش:
 - فرایندها باید برای استفاده از ناحیه بحرانی یک در میان عمل کنند، بنابر این سرعت اجرا بوسیله فرایند کنتر هدایت می شود.
 - اگر فرایند با شکست مواجه شود حال چه در بخش بحرانی باشد یا خارج از آن، در هر حالت فرایند دیگر تا ابد مسدود خواهد ماند.
 - شرط اول (mutual exclusion) برآورده می شود، اما شرط دوم (progress) نه.

انحصار متقابل: رویکرد نرم افزاری

دومین تلاش

■ متغیر های اشتراکی

boolean flag[2]; ■

flag[0]=flag[1]=false; ■

برای فرایند P_i ■

```
do {  
    while (flag[j]) ;  
    flag[i] = true;  
    critical section  
    flag [i] = false;  
    remainder section  
} while (1);
```

انحصار متقابل: رویکرد نرم افزاری

دومین تلاش

- اگر فرایندی در داخل بخش بحرانی خود، یا بعد از مقدار گذاری متغیر flag (مربوط به خود) با true درست قبل از ورود به بخش بحرانی، شکست بخورد، در این صورت فرایند دیگر تا ابد مسدود خواهد بود.
- در حقیقت این روش بدتر از روش اولین تلاش می باشد است
 - شرط اول (mutual exclusion) را تضمین نمی کند.
 - شرط دوم (progress) را برآورد نمی کند.
- راه حل پیشنهادی مستقل از سرعت نسبی اجرای فرایندها نیست.

انحصار متقابل: رویکرد نرم افزاری

سومین تلاش

■ متغیر های اشتراکی

boolean flag[2]; ■

■ مقدار اولیه; flag[0]=flag[1]=false;

■ برای فرایند P_i

```
do {  
    flag[i] = true;  
    while (flag[j]) ;  
        critical section  
        flag[i] = false;  
        remainder section  
    } while (1);
```

■ شرط اول (mutual exclusion) برآورده می شود ولی شرط دوم (progress) نه.

انحصار متقابل: رویکرد نرم افزاری

چهارمین تلاش

متغیر های اشتراکی ■

boolean flag[2]; ■

مقدار اولیه flag[0]=flag[1]=false; ■

برای پردازندگی P_i ■

```
do {  
    flag [i] = true;  
    while (flag [j])  
    {  
        flag [i] = false;  
        delay  
        flag [i] = true;  
    }  
    critical section  
    flag [i] = false;  
    remainder section  
} while (1);
```

ممکن است دنباله ای از اجرا وجود داشته باشد که باعث بروز بن بست شود. ■

انحصار متقابل: رویکرد نرم افزاری

الگوریتم DEKKER

- برای انحصار متقابل دو فرایند، الگوریتمی را منتشر کرد که توسط Dekker ریاضیدان آلمانی طراحی شده بود.
- امتیاز این رویکرد آن است که بسیاری از خطاهای متداول در ایجاد برنامه های همزمان، تشریح می شود.

```

boolean flag[2];
int turn;
void P0()
{
    while(true)
    {
        flag[0] = true;
        while(flag[1])
            if(turn == 1)
            {
                flag[0] = false;
                while(turn == 1)
                    /*do nothing*/
                flag[0] = true;
            }
        /*critical section*/
        turn = 1;
        flag[0] = false;
        /*remainder section*/
    }
}

```

```

void P1()
{
    while(true)
    {
        flag[1] = true;
        while(flag[0])
            if(turn == 0)
            {
                flag[1] = false;
                while(turn == 0)
                    /*do nothing*/
                flag[1] = true;
            }
        /*critical section*/
        turn = 0;
        flag[1] = false;
        /*remainder section*/
    }
}
void main()
{
    flag[0] = false;
    flag[1] = false;
    turn = 1;
    parbegin( P0, P1);
}

```

انحصار متقابل: رویکرد نرم افزاری

تمرین

الگوریتم Dekker را اثبات کنید؟

مسئله ۵-۶ کتاب

انحصار متقابل: رویکرد نرم افزاری

الگوریتم Peterson ■

متغیرهای اشتراکی همان متغیرهای اولین و دومین تلاش می باشد. ■

برای فرایند P_i ■

```
do {  
    flag [i]:= true;  
    turn = j;  
    while (flag [j] && turn == j) ;  
        critical section  
        flag [i] = false;  
        remainder section  
    } while (1);
```

آرایه سراسری flag نمایانگر وضع هر فرایند نسبت به انحصار متقابل است و متغیر سراسری turn در گیریهای همزمانی را حل می کند. ■

تمام نیازها را برآورد می کند و ناحیه بحرانی را برای دو فرایند حل می کند. ■

اثبات کنید? ■

انحصار متقابل: حمایت سخت افزار

از کار انداختن وقفه

- در یک ماشین تک پردازنده، هم زمان نمی شود فرایнд ها وارد ناحیه بحرانی شوند.
- برای ضمانت انحصار متقابل، کافی است از مواجهه فرایند با وقفه جلوگیری شود.

```
while ( true )  
{  
    /* disable interrupts */  
    /* critical section */  
    /* enable interrupts */  
    /* remainder section */  
}
```

انحصار متقابل: حمایت سخت افزار

- از کار انداختن وقفه (ادامه)
- چون بخش بحرانی نمی تواند وقفه داده شود، لذا انحصار متقابل ضمانت شده است.
- کارایی اجرایی به طور متقابل کم می شود.
- این روش در معماری چند پردازنده‌ای کارایی ندارد.

انحصار متقابل: حمایت سخت افزار

دستور العملهای ویژه ماشین

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

■ دستور العمل آزمون و مقدار گذاری

در این روش هر فرایند در هنگام وارد شدن به ناحیه بحرانی مقدار متغیر را یک می کند و هنگام خارج شدن آن را صفر می کند، هنگامی که مقدار یک می باشد هیچ فرایندی اجازه ورود به این ناحیه را ندارد.

انحصار متقابل: حمایت سخت افزار

دستور العملهای ویژه ماشین

- انحصار متقابل با دستور العمل آزمون و مقدار گذاری
 - داده مشترک

boolean lock = false;

فرايند P_i

```
do {  
    while (!testset (lock)) ;  
        critical section  
    lock = false;  
        remainder section  
}while(1);
```

انحصار متقابل: حمایت سخت افزار

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

دستور العملهای ویژه ماشین

■ انحصار متقابل با دستور العمل
آزمون و مقدار گذاری

انحصار متقابل: حمایت سخت افزار

دستور العملهای ویژه ماشین

■ دستور العمل معاوضه

■ دستور العمل معاوضه را می‌توان بصورت زیر نوشت:

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

انحصار متقابل: حمایت سخت افزار

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

دستور العملهای ویژه ماشین

■ انحصار متقابل با دستور العمل معاوضه

انحصار متقابل: حمایت سخت افزار

دستور العملهای ویژه ماشین

■ مزایای این روش

- برای هر تعداد از فرایندها، روی یک پردازنده و یا چند پردازنده که از حافظه مشترک استفاده می کنند، قابل به کارگیری است.
- ساده است و لذا وارسی آن آسان می باشد.
- از آن برای حمایت از بخش‌های بحرانی متعدد می توان استفاده کرد.

■ معایب این روش

- انتظار مشغول جود دارد.
- امکان گرسنگی وجود دارد.
- امکان بنبست وجود دارد.

راهنماها (semaphore)

- برای علامت دهی بین فرایند ها و اعمال یک نظام انحصار متقابل استفاده کرد.
- اصل اساسی این است که دو یا چند فرآیند می توانند به وسیله سیگنال های ساده همکاری کنند، به طوری که فرآیند می تواند در نقطه خاصی مجبور به توقف شود و منتظر سیگنالی بماند.
- برای علامت دادن از متغیر ویژه ای به نام راهنما استفاده می کند.

راهنماها (semaphore)

- برای فرستادن علامت توسط راهنمای s ، فرایند اولیه $signal(s)$ را اجرا می کند.
- برای دریافت علامت توسط یک راهنمای s ، فرایند اولیه $wait(s)$ را اجرا می کند.
- اگر علامت مربوط، هنوز فرستاده نشده باشد، این فرایند تا زمان آن، معلق می ماند.

راهنماها (semaphore)

- برای ایجاد این اثر، سمافور را می توانیم به عنوان متغیری درنظر بگیریم که مقدار آن از نوع صحیح است و سه عمل بر روی آن تعریف می شود:
- 1. سمافور می تواند یک مقدار غیر منفی را بپذیرد.
 - 2. عملیات Wait(s) یک واحد از مقدار سمافور می کاهد. اگر این مقدار منفی شود، فرآیند اجرا کننده Wait مسدود می شود.
 - 3. عملیات Signal(s) یک واحد به مقدار سمافور اضافه می کند. اگر مقدار مثبت نباشد، فرآیند مسدود شده توسط Wait، از حالت مسدود خارج می شود.

(semaphore) راهنماها

```
struct semaphore {
    int count;
    queueType queue;
}

void semWait(semaphore s)
{
    s.count--;
    if (s.count < 0)
    {
        place this process in s.queue;
        block this process
    }
}
void semSignal(semaphore s)
{
    s.count++;
    if (s.count <= 0)
    {
        remove a process P from s.queue;
        place process P on ready list;
    }
}
```

راهنماها (semaphore)

```
struct binary_semaphore {
    enum{zero, one} value;
    queueType queue;
};

void semWaitB(binary_semaphore s)
{
    if (s.value == 1)
        s.value = 0;
    else
    {
        place this process in s.queue;
        block this process;
    }
}

void semSignalB(semaphore s)
{
    if (s.queue is empty())
        s.value = 1;
    else
    {
        remove a process P from s.queue;
        place process P on ready list;
    }
}
```

راهنماها (semaphore)

انحصار متقابل با Semaphore

داده مشترک ■

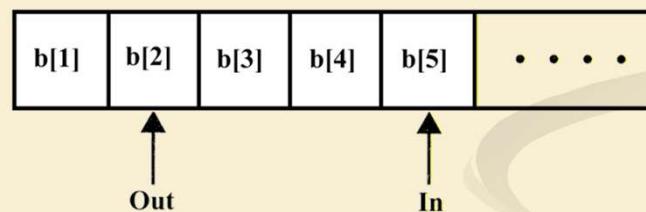
semaphore mutex; //initially mutex = 1

P_i فرایند ■

```
do {  
    wait(mutex);  
    critical section  
    signal(mutex);  
    remainder section  
} while (1);
```

مسئله تولید کننده و مصرف کننده

- یک یا چند تولید کننده، داده هایی را (از نوع رکورد، کارکترها) تولید و در یک میانگیر قرار می دهند.
- یک مصرف کننده، در هر زمان یک قلم از داده ها را از میانگیر برمی دارد.
- فرض می کنیم میانگیر نامتناهی و شامل آرایه ای از عناصر است. به طور انتزاعی می توانیم توابع مصرف کننده و تولید کننده را به صورت زیر تعریف کنیم:



producer:

```
while (true)
{
    /* produce item v */;
    b[in] = v;
    in++;
}
```

consumer:

```
while (true)
{
    while (in <= out)
        /* do nothing */;
    w = b[out];
    out++;
    /* consume item w */
}
```

مسئله تولید کننده و مصرف کننده

- سعی می کنیم این سیستم را با استفاده از سمافورهای دودویی پیاده سازی کنیم.
- به جای استفاده از اندیس های in و out ، از متغیر صحیح n ($n = in - out$) برای نگهداری تعداد اقلام موجود در میانگیر استفاده می کنیم.

مسئله تولید کننده و مصرف کننده

```
int n;
binary_semaphore s = 1;
binary_semaphore delay = 0;
void producer()
{
    while (true)
    {
        produce();
        waitB(s);
        append();
        n++;
        if(n==1)
            signalB(delay);
        signalB(s);
    }
}
```

یک راه حل نادرست برای مسئله تولید کننده و
مصرف کننده با استفاده از سمافورها.

```
void consumer()
{
    waitB(delay);
    while(true)
    {
        waitB(s);
        take();
        n--;
        signalB(s);
        consume();
        if(n==0)
            wateB(delay);
    }
}
void main()
{
    n = 0;
    parbegin(producer, costumer);
}
```

مسئله تولید کننده و مصرف کننده

```
int n;
binary_semaphore s = 1;
binary_semaphore delay = 0;
void producer()
{
    while (true)
    {
        produce();
        waitB(s);
        append();
        n++;
        if(n==1) signalB(delay);
        signalB(s);
    }
}
```

یک راه حل درست برای میانگیر نامتناهی مسئله تولید کننده و مصرف کننده با استفاده از سمافورها دودویی.

```
void consumer()
{
    int m; /*a local variable*/
    waitB(delay);
    while(true)
    {
        waitB(s);
        take();
        n--;
        m = n;
        signalB(s);
        consume();
        if(m==0) wateB(delay);
    }
}
void main()
{
    n = 0;
    parbegin(producer, consumer);
}
```

مسئله تولید کننده و مصرف کننده

```
semaphore n = 0;  
semaphore s = 1;  
void producer()  
{  
    while (true)  
    {  
        produce();  
        wait(s);  
        append();  
        signal(s);  
        signal(n);  
    }  
}
```

```
void consumer()  
{  
    while(true)  
    {  
        wait(n);  
        wait(s);  
        take();  
        signal(s);  
        consume();  
    }  
}  
void main()  
{  
    parbegin(producer, consumer);  
}
```

راه حلی برای مسئله تولید کننده و مصرف کننده با استفاده از سمافورها.

مسئله تولید کننده و مصرف کننده

```
semaphore n = 0;
semaphore s = 1;
semaphore e = sizeofbuffer;
void producer()
{
    while (true)
    {
        produce();
        wait(e);
        wait(s);
        append();
        signal(s);
        signal(n);
    }
}
```

```
void consumer()
{
    while(true)
    {
        wait(n);
        wait(s);
        take();
        signal(s);
        signal(e);
        consume();
    }
}
void main()
{
    parbegin(producer, consumer);
}
```

راه حلی برای مسئله تولید کننده و مصرف کننده با استفاده از سمافورها.

مسئله تولید کننده و مصرف کننده

```

semWait(s)
{
    while (!testset(s.flag))
        /* do nothing */;
    s.count--;
    if (s.count < 0)
    {
        place this process in s.queue;
        block this process (must also set s.flag to 0)
    }
    else
        s.flag = 0;
}

semSignal(s)
{
    while (!testset(s.flag))
        /* do nothing */;
    s.count++;
    if (s.count <= 0)
    {
        remove a process P from s.queue;
        place process P on ready list
    }
    s.flag = 0;
}

```

(الف) دستور العمل تست و مقداردادن.

```

semWait(s)
{
    inhibit interrupts;
    s.count--;
    if (s.count < 0)
    {
        place this process in s.queue;
        block this process and allow interrupts
    }
    else
        allow interrupts;
}

semSignal(s)
{
    inhibit interrupts;
    s.count++;
    if (s.count <= 0)
    {
        remove a process P from s.queue;
        place process P on ready list
    }
    allow interrupts;
}

```

(ب) وقتهای.

دو پیاده سازی ممکن برای سمافورها.

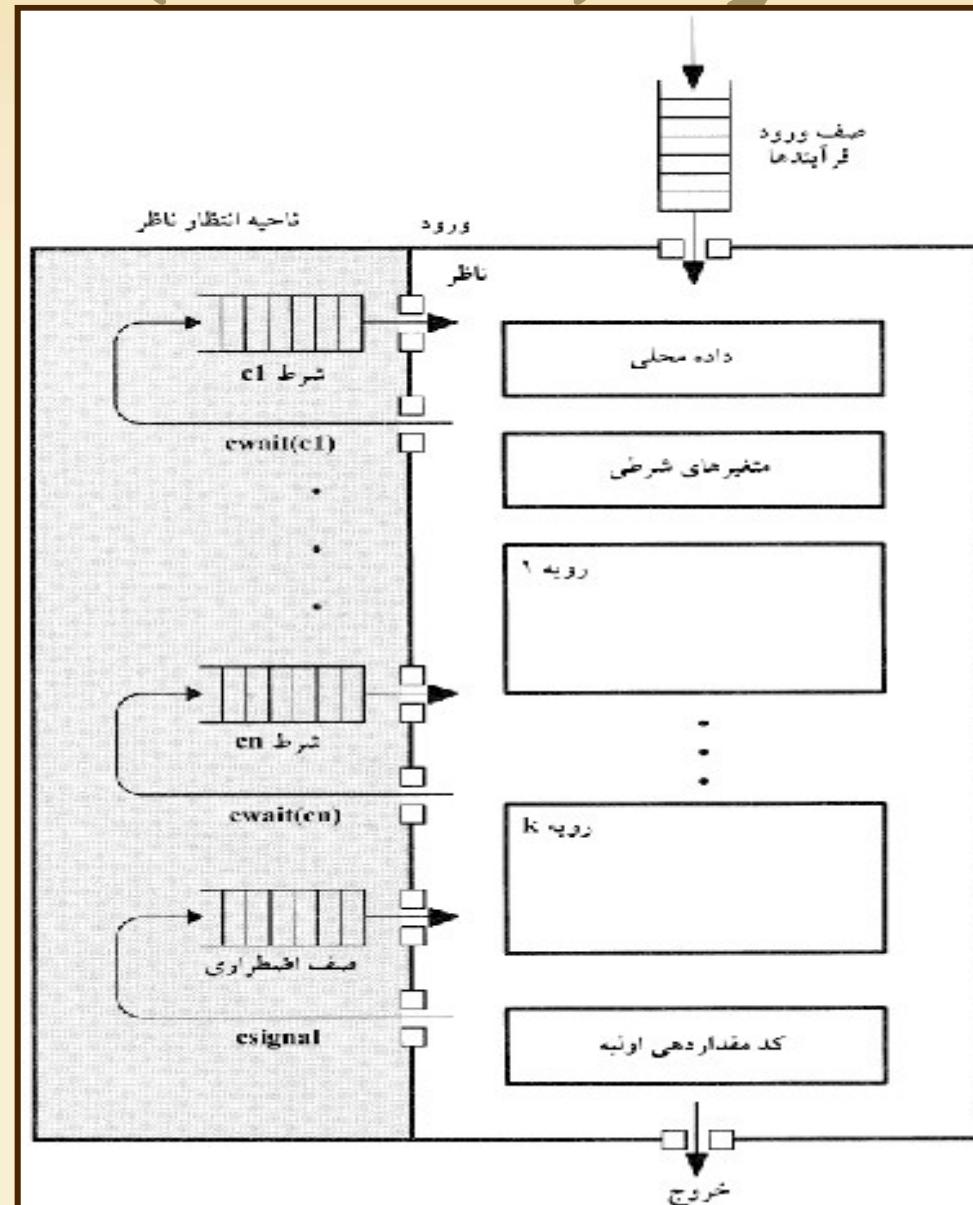
ناظرها (monitor)

- ایجاد یک برنامهٔ درست از سمافورها، دشوار است.
- ناظر یک مولفهٔ نرم افزاری است که از یک یا چند رویه، دنبالهٔ مقداردهی اولیه، و داده‌های محلی تشکیل شده است.
- ویژگی‌های مهم ناظر عبارت اند از:
 - متغیرهای محلی فقط توسط رویه‌های ناظر قابل دستیابی اند و رویه‌های خارجی نمی‌توانند به آن‌ها دستیابی داشته باشند.
 - فرآیند با فراخوانی یکی از رویه‌های ناظر وارد آن می‌شود.
 - در هر زمان فقط یک فرآیند می‌تواند در ناظر اجرا شود. سایر فرآیندهایی که ناظر را فراخوانی می‌کنند معلق می‌شوند و منتظر می‌مانند تا ناظر مهیا شوند.

ناظرها (monitor)

- ناظر، به وسیله متغیرهای شرطی موجود در خود که فقط در داخل آن ناظر قابل دستیابی اند، همگام سازی را پشتیبانی می کند.
- دو عمل برای متغیرهای شرطی انجام می شوند:
 - اجرای فرآیند فراخوان را در شرایط c معلق می کند.
 - اکنون فرآیند دیگر می تواند از ناظر استفاده کند.
- اجرای فرآیندی را که پس از عمل cwait معلق شد، بر روی همان شرط c از سر می گیرد. اگر چندین فرآیند وجود داشته باشند، یکی از آن ها را انتخاب می کند. اگر هیچ فرآیندی نباشد، کاری انجام نمی دهد.

ناظرها (monitor)



```

monitor boundedbuffer;
char buffer[N];
int nextin, nextout;
int count;
int notfull, notempty;

void append(char x)
{
    if (count == N)
        cwaite(notfull);
    buffer[nextin] = x;
    nextin = (nextin+1)%N
    count++;
    csignal( notempty );
}

void take( char x )
{
    if (count == 0)
        cwaite(notempty);
    x = buffer[nextout];
    nextout = (nextout+1)%N
    count--;
    csignal( notfull );
}

```

```

void producer()
{
    char x;
    while( true)
    {
        produce( x );
        append( x );
    }
}

void consumer()
{
    char x;
    while( true )
    {
        take(x);
        consume( x );
    }
}

void main()
{
    nextin=0; nextout=0; count=0;
    Parbegin(producer, consumer);
}

```

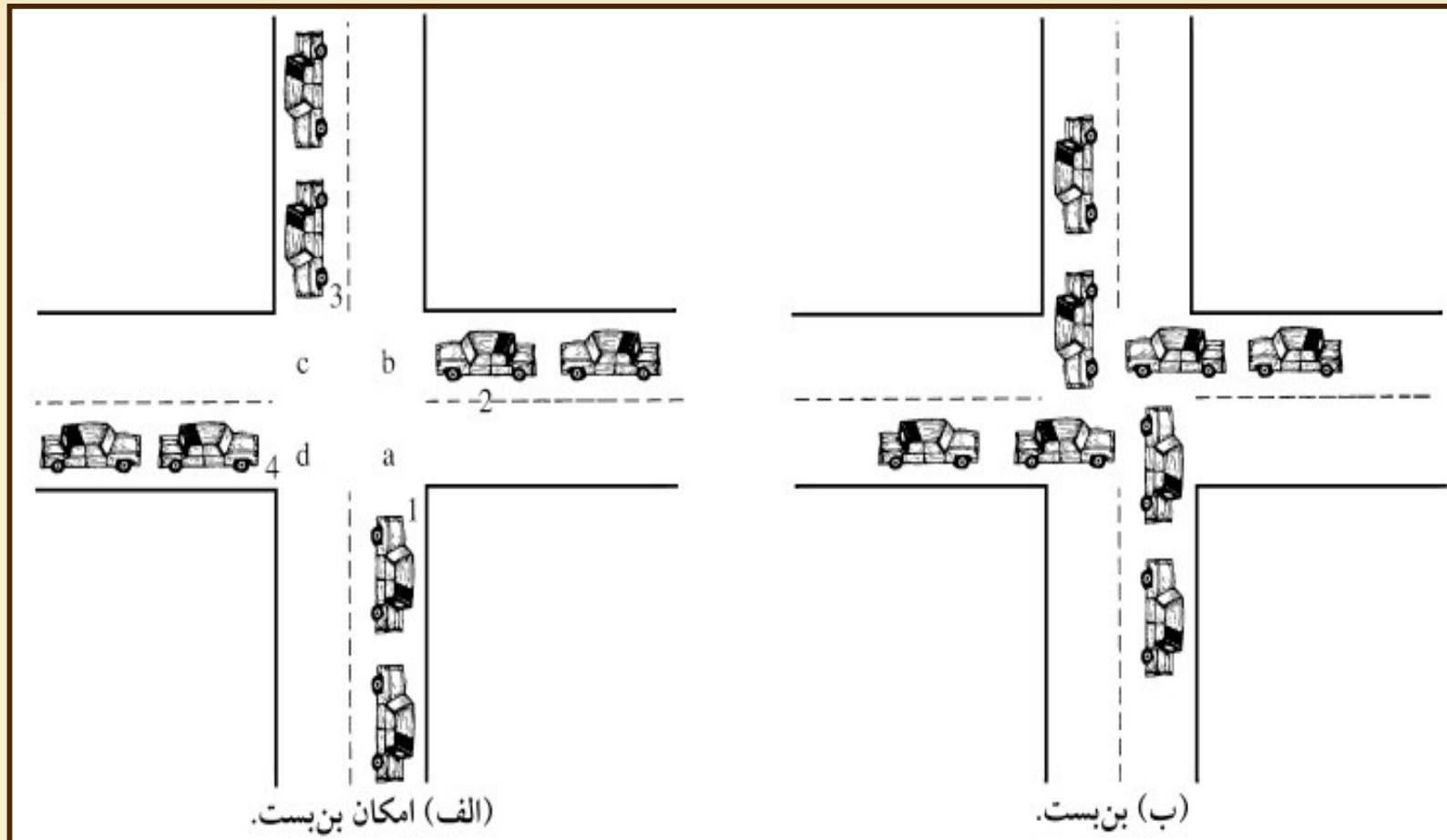
فصل ششم

همزمانی:
بنبست و گرسنگی

اصول بن‌بست

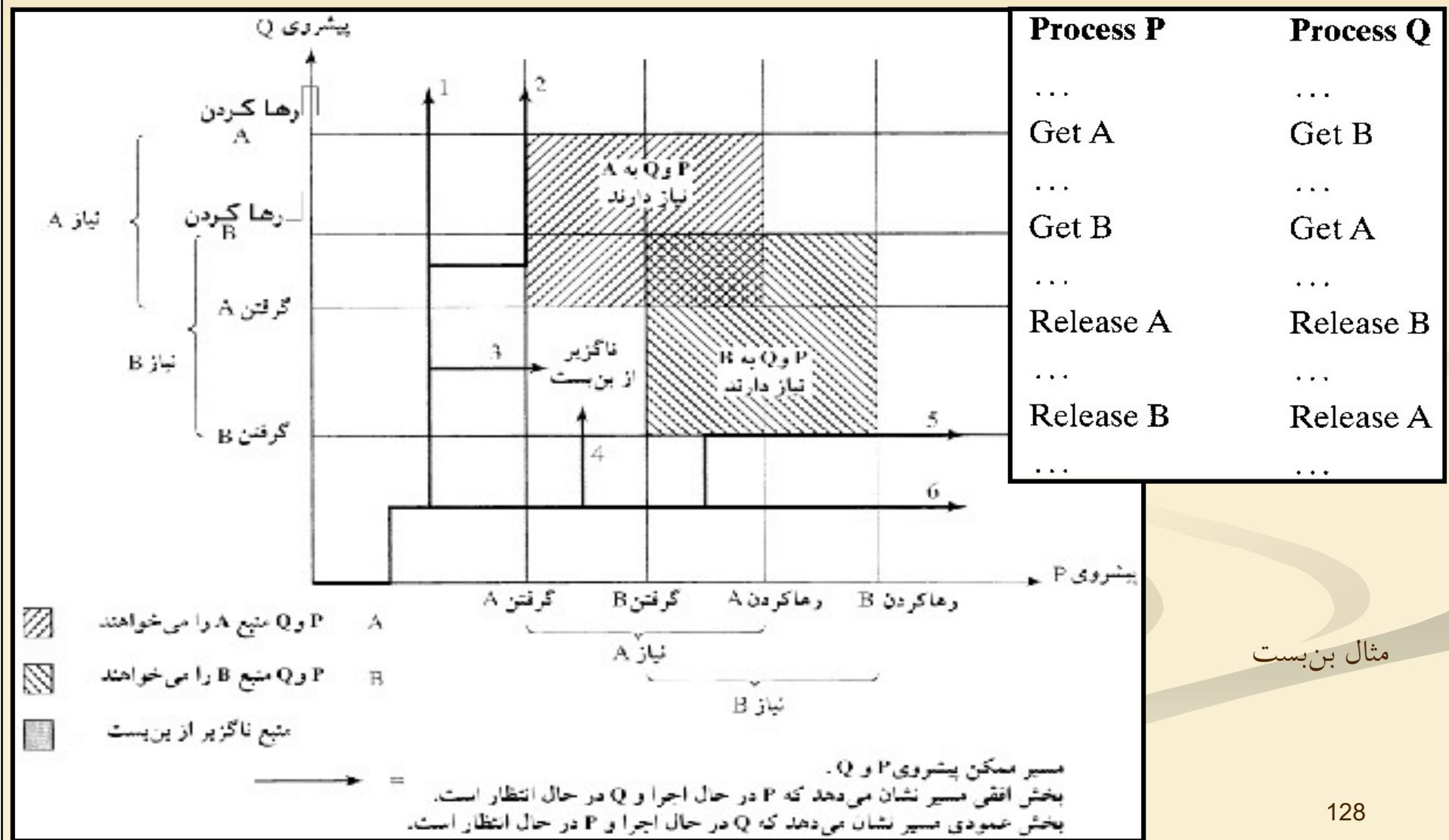
- بن‌بست را به صورت مسدود بودن دائمی مجموعه‌ای از فرآیندها که برای منابع سیستم رقابت می‌کند یا با یکدیگر در ارتباط هستند.
- راه حل کارامدی برای بن‌بست وجود ندارد.
- تمام بن‌بست‌ها در اثر نیازهای متضاد دو یا چند فرآیند به منابع رخ می‌دهند.

أصول بنبست

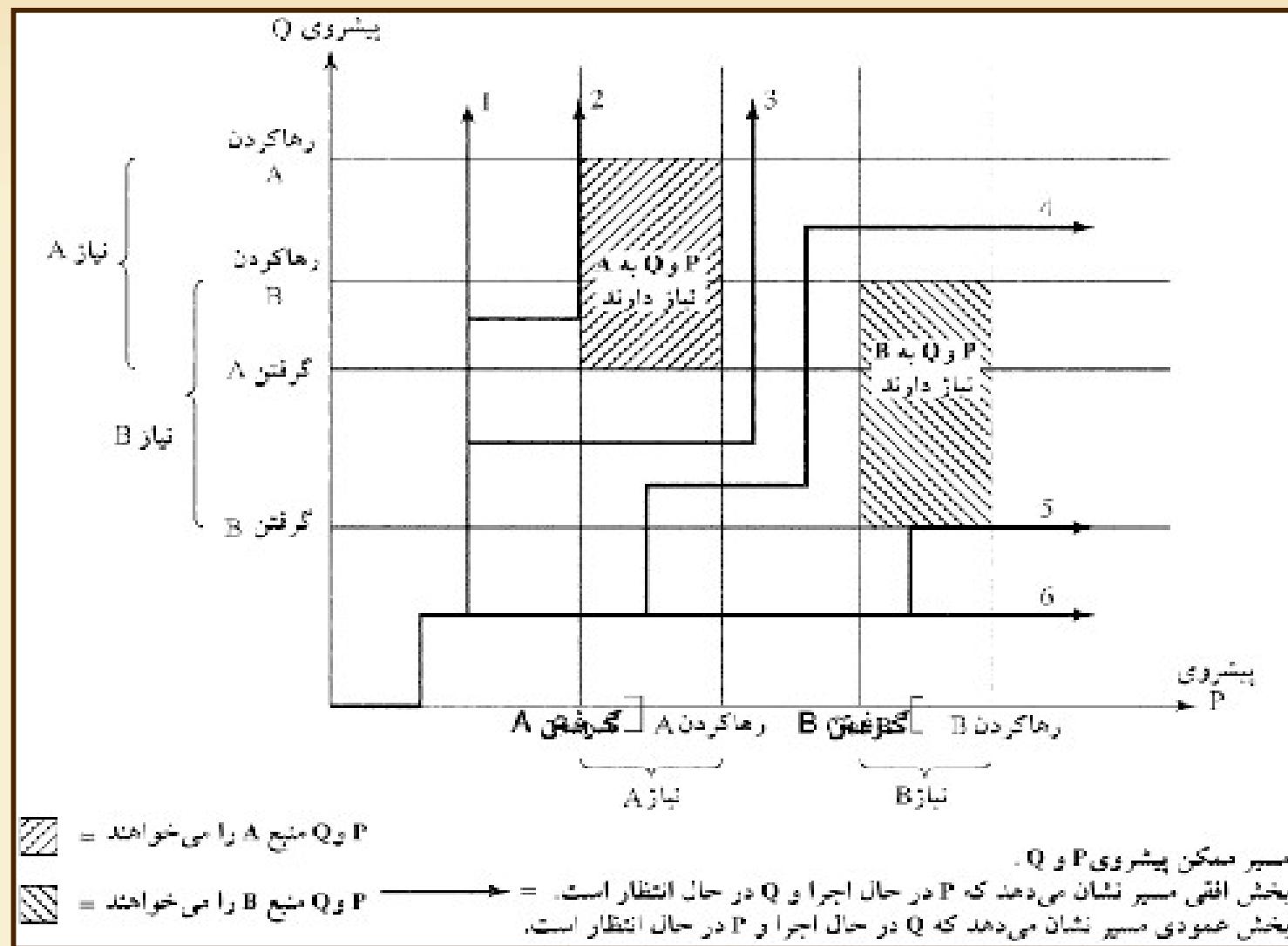


نمایش بنبست

اصول بنبست



اصول بنبست



مثال عدم بنبست

انواع منابع

■ **منابع قابل استفاده مجدد:** بدون صدمه توسط یک فرآیند می توان در هر زمان مورد استفاده قرار می گیرد و تمام نمی شود.

■ مثل :پردازنده ها، حافظه اصلی و ثانویه و ...

■ **منابع مصرف شدنی:** منبعی هستند که می تواند ایجاد و نابود گردد.

■ نوعا روی تعداد منابع مصرف شدنی از یک نوع به خصوص، حدی وجود ندارد.

■ نمونه هایی از منابع مصرف شدنی وقفه ها، سیگنال ها، پیام ها، و اطلاعات موجود در میانگیرهای I/O هستند.

انواع منابع

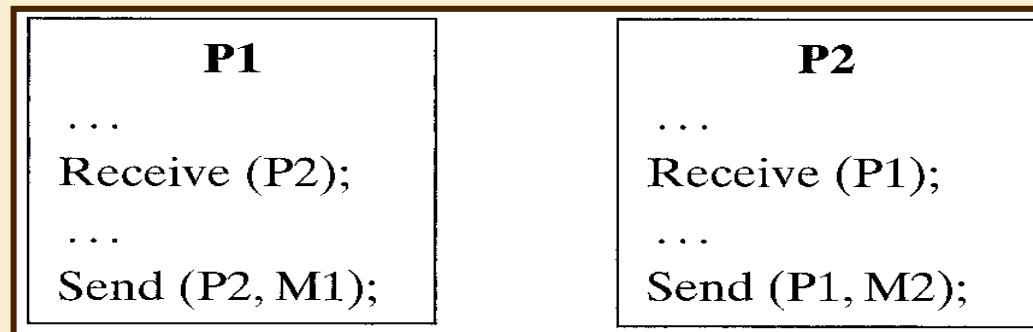
- به عنوان مثالی از بن بستِ شامل منابع قابل استفاده مجدد، دو فرآیند را در نظر بگیرید که برای دستیابی انحصاری به فایل دیسک D و گرداننده نوار T با هم رقابت می کنند.
- بن بست وقتی رخ می دهد که هر فرآیند یک منبع را در اختیار داشته باشد و دیگری را درخواست کند.

فرآیند Q		فرآیند P	
مرحله	عمل	مرحله	عمل
	درخواست T		درخواست D
	مسوده کردن D		مسوده کردن T
	درخواست D		درخواست T
	مسوده کردن D		مسوده کردن T
	انجام وظیفه		انجام وظیفه
	T آزاد کردن		D آزاد کردن
	D آزاد کردن		T آزاد کردن
	q_0		p_0
	q_1		p_1
	q_2		p_2
	q_3		p_3
	q_4		p_4
	q_5		p_5
	q_6		p_6

دو فرآیندی که برای منبع قابل استفاده مجدد رقابت می کنند.

انواع منابع

■ به عنوان مثالی از بن بست ناشی از منابع مصرف شدنی، دو فرآیند زیر را درنظر بگیرید که هر فرآیند سعی می کند پیامی را از فرآیند دیگر دریافت کند و سپس پیامی را به فرآیند دیگر بفرستد:



■ اگر `Receive` مسدود شونده باشد (یعنی فرآیند گیرنده مسدود باشد تا پیام دریافت شود)، بن بست رخ می دهد.
■ راهبرد موثر منحصر به فردی برای برخورد با انواع مختلف بن بست وجود ندارد.

شایط بنبست

■ شایط زیر باید برقرار باشد تا بروز بنبست امکان‌پذیر گردد:

■ **انحصار متقابل**: (شرط لازم)

■ استفاده یک فرآیند از یک منبع.

■ **نگهداشتن و انتظار**: (شرط لازم)

■ با داشتن منبع دیگری در خواست منبع جدید می‌کند.

■ در این صورت یک منبع به مدت طولانی در اختیار یک فرآیند خواهد بود.

■ **قبضه نکردن**: (شرط لازم)

■ منبع را نمی‌توان به زور پس گرفت.

■ تنها وقتی از بنبست جلوگیری می‌کند: که منابع دارای اولویت باشند.

■ **انتظار مدور**: (شرط لازم و کافی)

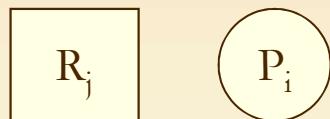
■ چند فرآیند درخواست منبع‌هایی می‌کنند که در اختیار منبع دیگر است

■ و به همین سبب در انتظار چرخشی قرار می‌گیرند.

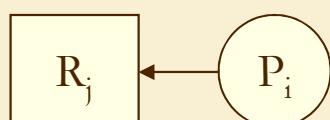
■ موجب کند کردن فرآیند‌ها و رد کردن غیر ضروری دسترسی منابع می‌شود

گراف تخصیص منابع

یک گراف جهت دار است که حالت سیستم، منابع و فرآیندها را نشان می دهد.



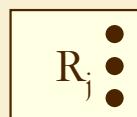
هر فرآیند و منبع با یک گره نشان داده می شوند.



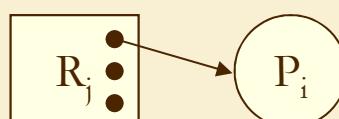
یال گراف از یک فرآیند به یک منبع، نشان می دهد که فرآیند آن منبع را

درخواست کرد ولی هنوز آن را به دست نیاورده است.

در داخل یک گره منبع، برای هر نمونه از آن منبع یک نقطه قرار داده شده است.

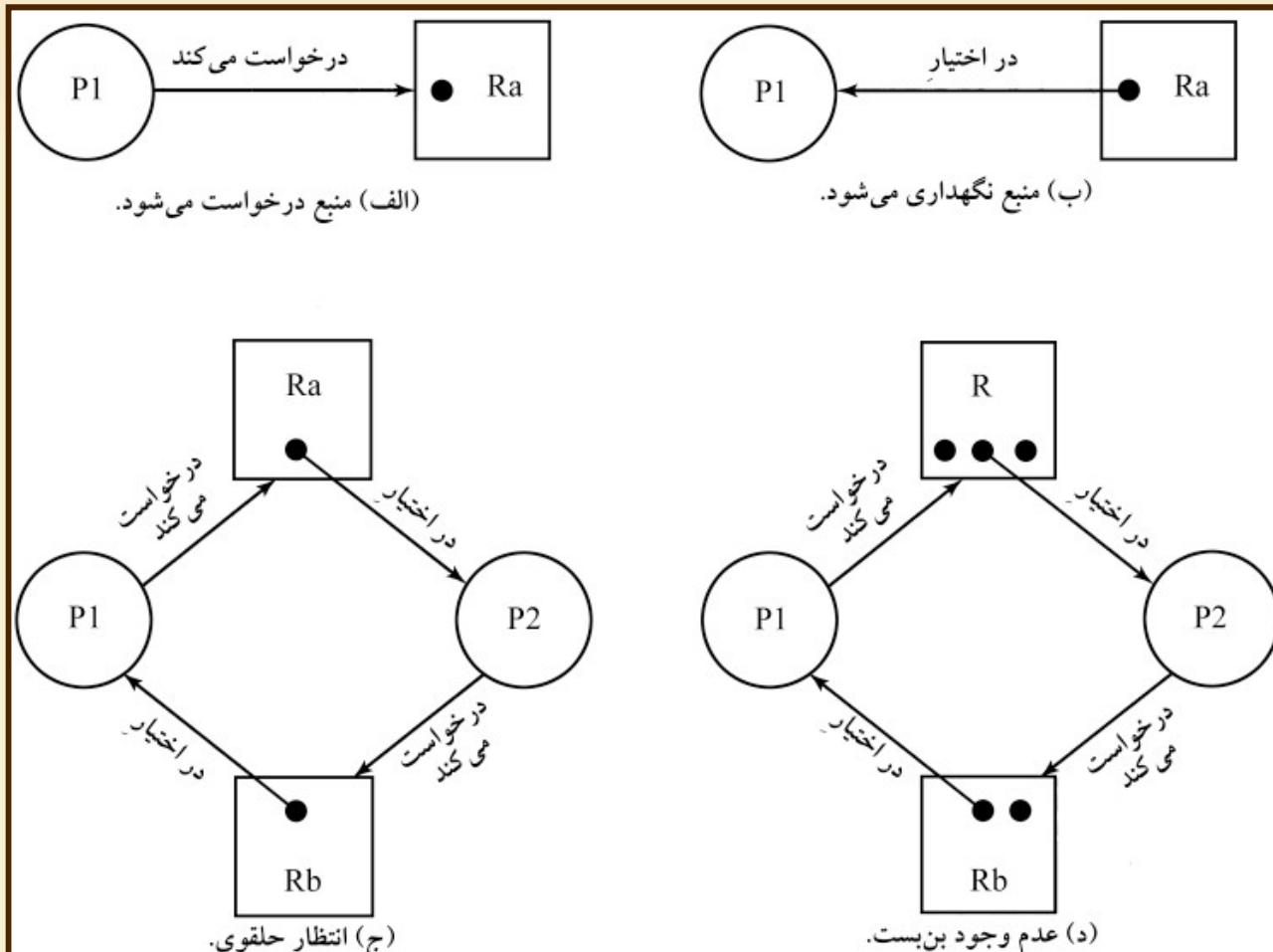


یال گراف از گره منبع به یک فرآیند، تقاضایی را نشان می دهد که برآورده شده



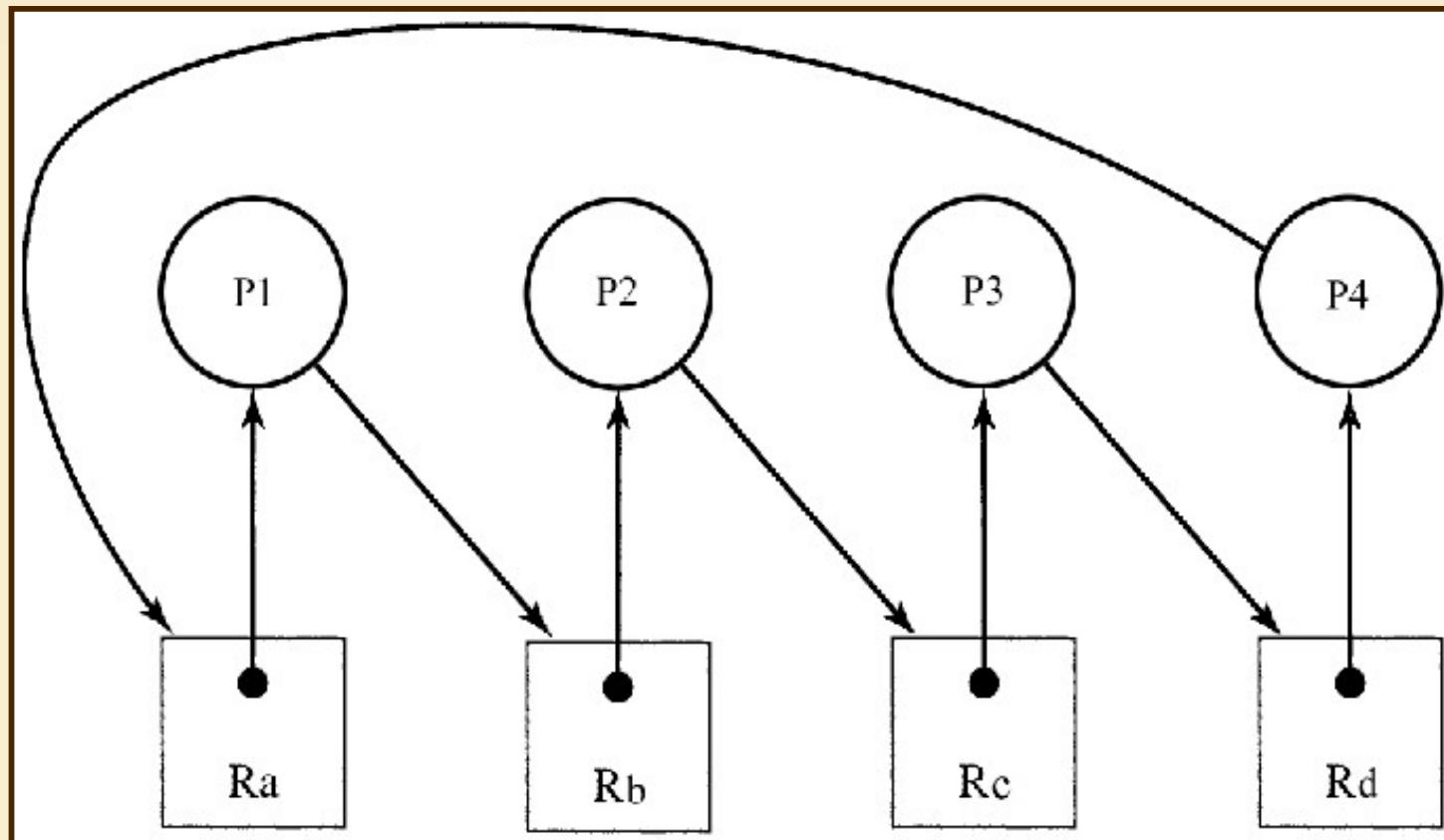
است. (منبع به فرآیند اختصاص یافته).

گراف تخصیص منابع



مثال هایی از گراف های تخصیص منابع.

گراف تخصیص منابع



مثالی از گراف های تخصیص منابع.

رویکردهای برخورد با بن بست

■ سه رویکرد در برخورد با بن بست

پیشگیری

■ محافظه کارانه ، درگیر کردن منابع کمتر

اجتناب

■ راهکاری بین پیشگیری و کشف

کشف

■ بسیار آزاد، منابع مورد درخواست شده در صورت امکان تخصیص داده می شود.

پیشگیری از بن بست

- روش غیر مستقیم
 - پیشگیری از حالت‌های ۱ و ۲ و ۳ (انحصار متقابل، نگهداشتن و انتظار، قبضه نکردن)
- روش مستقیم
 - پیشگیری از بروز انتظار مدور

پیشگیری از بن بست

انحصار متقابل

■ به طور کلی، نمی توان مانع اولین شرط از چهار شرط مذکور شد. اگر دستیابی به منبع مستلزم انحصار متقابل باشد، باید توسط سیستم عامل پشتیبانی شود.

نگهداشتن و انتظار

■ برای جلوگیری از شرط نگهداشتن و انتظار، باید فرآیند را ملزم کنیم که تمام منابع مورد نیازش را درخواست کند و آنقدر مسدود شود تا تمام منابع مورد نیاز در اختیارش قرار گیرند.

■ این روش به دو دلیل ناکارآمد است.

■ اولاً ممکن است فرآیند به مدت زیادی منتظر بماند، در حالی که می تواند با تعدادی از منابع به کارش ادامه دهد.

■ ثانیاً، منابع تخصیص یافته به فرآیند ممکن است مدت زیادی مورد استفاده قرار نگیرند.

■ مسئله دیگر این است که ممکن است فرآیند نداند در آینده به چه منابع نیاز دارد.

پیشگیری از بن بست

قبضه نکردن

- به چند روش می توان از این شرط جلوگیری کرد.
- اولاً، اگر فرآیندی که منبعی را در اختیار دارد، اجازه نداشته باشد منبع دیگری را درخواست کند، باید منابع اصلی خود را رها نماید، و در صورت لزوم، آن ها را دوباره به همراه منابع دیگر درخواست کند.
- ثانیاً، اگر فرآیندی، منبعی را درخواست کند که فعلاً در اختیار فرآیند دیگری است، سیستم عامل ممکن است فرآیند دوم را قبضه کند و آن را ملزم به رها کردن منابع نماید. این طرح وقتی منجر به بن بست می شود که هیچ دو فرآیندی اولویت یکسان نداشته باشند.
- این روش وقتی عملی است که به منابعی مثل پردازنده اعمال گردد که حالت آنها به آسانی قابل ذخیره و بازیابی است.

انتظار حلقوی

- با تعریف ترتیب خطی انواع منابع، می توان از وقوع این شرط جلوگیری کرد. اگر منابع نوع R به فرآیندی تخصیص داده شوند، این فرآیند بعداً فقط می تواند منابعی را درخواست کند که در ترتیب خطی بعد از R قرار دارند.

اجتناب از بن بست

- در این روش درخواست های منابع را محدود می کنیم تا حداقل یکی از چهار شرط بن بست رخ ندهد.
- سه شرط ضروری را اجازه می دهد، اما در انتخاب ها طوری عمل می کند که نقطه وقوع بن بست پیش نیاید.
- این تصمیم گیری بصورت پویا انجام می شود که اگر منبع اختصاص پیدا کرد آیا می تواند بالقوه منجر به بن بست گردد یا نه.
- در این بخش، دو روش اجتناب از بن بست را بررسی می کنیم:
 - اگر درخواست فرآیندی منجر به بن بست می شود، آن فرآیند شروع نشود.
 - اگر درخواست منبع اضافی موجب بن بست می شود، به آن درخواست فرآیند پاسخ داده نشود.

عدم شروع فرآیند

سیستمی با n فرآیند و m نوع منبع مختلف را درنظر بگیرید. بردارها و ماتریس های زیر را تعریف می کنیم:

$$\text{Resource } R = (R_1, R_2, \dots, R_m)$$

کل منابع موجود در سیستم

$$\text{Available } V = (V_1, V_2, \dots, V_m)$$

تعداد کل هر منبعی که به فرآیندی تخصیص نیافت و موجود است

$$\text{Claim } C = \begin{vmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ C_{n1} & C_{n2} & \dots & C_{nm} \end{vmatrix}$$

= نیاز فرآیند i به منبع j

$$\text{Allocation } A = \begin{vmatrix} A_{11} & A_{12} & \dots & A_{1m} \\ A_{21} & A_{22} & \dots & A_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nm} \end{vmatrix}$$

= تخصیص فعلی فرآیند i به منبع j

ماتریس Claim حداکثر نیاز هر فرآیند به هر منبع را نشان می دهد و هر سطر آن مربوط به یک فرآیند است.
برای این که اجتناب از بن بست امکان پذیر باشد، فرآیند باید این اطلاعات را از قبل اعلان کند.

عدم شروع فرآیند

روابط زیر برقرار است:

■ برای تمام i ها داریم:

$$R_i = V_i + \sum_{k=1}^n A_{ki}$$

■ معنایش این است که تمام منابع، تخصیص داده شده اند یا مهیا هستند.

■ برای تمام k ها و i ها داریم:

$$C_{ki} \leq R_i$$

■ معنایش این است که هیچ فرآیندی نمی تواند منابعی بیش از منابع موجود در سیستم را درخواست کند.

■ برای تمام k ها و i ها داریم:

$$A_{ki} \leq C_{ki}$$

■ معنایش این است که به هیچ فرآیندی بیش از تعداد منابعی از هر نوع که در ابتدا اعلان کرده است تخصیص داده نمی شود.

عدم شروع فرآیند

■ فرآیند را وقتی شروع کنید که برای تمام آها داشته باشیم:

$$R_i \geq C_{(n+1)i} + \sum_{k=1}^n C_{ki}$$

■ یعنی، فرآیند جدید در صورتی شروع می شود که حداکثر درخواست تمام فرآیندهای فعلی و درخواست فرآیند جدید بتواند برآورده شود.

■ بعید به نظر می رسد که این راهبرد بهینه باشد، زیرا فرض می کند تمام فرآیندها همزمان حداکثر نیاز خود را درخواست می کنند.

عدم تخصیص منبع

- این راهبرد به عنوان الگوریتم بانکداران (Banker) شناخته می‌شود.
- حالت سیستم، تخصیص فعلی منابع به فرآیندها است.
- حالت شامل دو بردار Available و Resource و دو ماتریس Claim و Allocation است.
- دو حالت وجود دارد:
 - حالت امن
 - حالت نامن
- حالتی است که در آن حداقل یک دنباله از فرآیندها وجود دارند که دچار بن بست نمی‌شوند.
- حالت نامن حالتی است که امن نیست.

عدم تخصیص منبع

- در این روش باید منابع با دقت به درخواست کننده‌ها ارائه شود که دچار کمبود منابع نشویم.
- آیا هیچ کدام از فرآیندها می‌توانند با منابع موجود به طور کامل اجرا شوند؟
- با توجه به ماتریس‌ها و بردارهایی که معرفی شدند، شرطی که باید برای فرآیند i برقرار باشد، به صورت زیراست:

$$C_{ij} - A_{ij} \leq V_i \quad \text{برای تمام } j \text{‌ها}$$

عدم تخصيص منبع

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim ماتریس

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation ماتریس

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C — A

	R1	R2	R3
Available	0	1	1
بردار			
Resource	9	3	6

بردار

(الف) حالت اولیه.

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim ماتریس

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation ماتریس

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C — A

	R1	R2	R3
Available	6	2	3
بردار			
Resource	9	3	6

بردار

(ب) P2 به طور کامل اجرا می شود.

عدم تخصيص منبع

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim ماتریس

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation ماتریس

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	0	3
P4	4	2	0

C — A

	R1	R2	R3
Available بردار	7	2	3
Resource بردار	9	3	6

(ج) P1 به طور کامل اجرا می شود.

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim ماتریس

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation ماتریس

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C — A

	R1	R2	R3
Available بردار	9	3	4
Resource بردار	9	3	6

(د) P3 به طور کامل اجرا می شود.

عدم تخصیص منبع

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

ماتریس Claim

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

ماتریس Allocation

	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

C — A

	R1	R2	R3
	9	3	6

بردار Resource

	R1	R2	R3
	1	1	2

بردار Available

(الف) حالت اولیه.

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

ماتریس Claim

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

ماتریس Allocation

	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

C — A

	R1	R2	R3
	9	3	6

بردار Resource

	R1	R2	R3
	0	1	1

بردار Available

(ب) یک واحد از R1 و R3 در خواست می کند.

عدم تخصیص منبع

- امتیاز اجتناب از بن بست این است که قبضه کردن و به عقب برگشتن لازم نیست و کمتر از پیشگیری محدود کننده است.
- محدودیتهای زیر را دارد:
 - حداکثر منابع مورد نیاز هر فرآیند باید پیش بینی شود.
 - فرآیندهای مورد نیاز باید مستقل باشند.
 - تعداد منابع تخصص باید ثابت باشند.
 - فرآیندی که منبعی را در اختیار دارد نمی تواند خارج شود.

کشف بن‌بست

- راهبرد های کشف بن‌بست، دستیابی به منابع یا فعالیت های فرآیند را محدود نمی‌کند.
- در کشف بن‌بست، منابع درخواستی در صورت امکان به فرآیندها تحويل می‌شوند.
- سیستم عامل متناوباً الگوریتمی را اجرا می‌کند که انتظار حلقوی (شرط ۴ از شرایط مذکور) را تشخیص می‌دهد.

کشـف بنـبـست

الگوریتم کشـف بنـبـست

- از ماتریس Allocation و بردار Available استفاده می شود.
- علاوه براین، ماتریس درخواست Q طوری تعریف می شود که Q_{ij} مبین مقدار درخواست فرایند i از منبع j می باشد.
- این الگوریتم، فرآیندهایی را که در بن بست نیستند علامت گذاری می کند. در آغاز، تمام فرآیندها بدون علامت هستند. سپس مراحل زیر انجام می شود:
 1. فرآیندی را که تمام عناصر سطر متناظر آن در ماتریس Allocation صفر است، علامت بزن.
 2. بردار موقت W را مساوی بردار Available ایجاد کن.
 3. اندیس i را طوری پیدا کن که فرآیند i فعلًا علامت دار نیست و سطر i م Q کمتر یا مساوی W است. یعنی، برای $1 \leq k \leq m$ ، داشته باشیم $Q_{ik} \leq W_k$. اگر چنین سطري پیدا نشد، الگوریتم را خاتمه بده.
 4. اگر چنین سطري پیدا شد، فرآیند i را علامت بزن و سطر متناظر از Allocation به W اضافه کن. یعنی $W_k = W_k + A_{ik}$ برای $1 \leq k \leq m$. به گام سوم برگرد.

کشـف بنـبـست

- بن بست فقط وقتی وجود دارد که در پایان الگوریتم، فرآیند بدون علامتی وجود داشته باشد.
- فرآیندهای بدون علامت در بن بست هستند.

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

ماتریس درخواست Q

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

ماتریس Allocation

	R1	R2	R3	R4	R5
	2	1	1	2	1
بردار Resource					
	0	0	0	0	1
	0	0	0	0	1

بردار Available

کشف بن‌بست

ترمیم

- .1 قطع تمام فرآیندهای موجود در بن بست.
- .2 برگشت فرآیندهای بن بست به نقاط کنترلی از قبل تعریف شده، و شروع مجدد آن‌ها.
- .3 قطع پی در پی فرآیندهای بن بست تا حدی که دیگر بن بست وجود نداشته باشد.
- .4 قبضه کردن پی در پی منابع تا زمانی که دیگر بن بست وجود نداشته باشد.

برای موارد (۳) و (۴)، معیار انتخاب باید به صورت زیر باشد. انتخاب فرآیندی با ویژگی‌های زیر:

- تاکنون کمترین وقت پردازنده را مصرف کرده باشد.
- تاکنون کمترین مقدار خروجی را تولید کرده باشد.
- بیشترین زمان باقیمانده تخمینی.
- تاکنون کمترین منبع به آن تخصیص داده شده باشد.
- کمترین اولویت را دارد.

بن‌بست

جدول ۱-۶ خلاصه‌ای از روش‌های کشف، پیشگیری، و اجتناب از بن‌بست برای سیستم‌عامل.

روش	خط مشی تخصیص منبع	طرح‌های مختلف	مزایای عمدده	معایب عمدده
بیشگیری منبع	درخواست یکباره منابع	برای فرآیندهای مناسب است که	» ناکارآمدی	» تأخیر شروع به کار فرآیند.
	فیضه کردن	فعالیت انبوهی را انجام می‌دهند.	» اطلاع از منابع غروری بعدی	» بیاز به قبضه کردن نیست.
محققه کری؛ درگیر کردن کمتر منبع	منابع کردن منبع	سهوت نه کارگیری در منابعی که ذخیره و بازیابی حالت آن‌ها آسن است.	» فضه کردن بیش از حد لازم	» فضه کردن نه چندان موثر
	دستکاری برای منبع	در زمان توجهه قابل بروزی است.	» مجاز نیاز به محاسبه رمان اجراء زیرا در طراحی سبتو حل شده است.	» مجاز ندانستن درخواست افزایشی منبع
اجتناب	حداقل یک مسیر امن	عدم لیز به قبضه کردن	» اطلاع از منابع غروری بعدی	» امکان مسدودشدن طولانی فرآیندها
کشف	بن‌بست	بررسی مذالم وجود	» ضرورهای قبضه کردن	» عدم تأخیر در شروع به کار فرآیند.
		» تسهیل احراز کردن online.		

فصل هفتم

مدیریت حافظه

نیازهای مدیریت حافظه

دارای پنج نیاز زیر است:

1. جابجایی

سخت افزار پردازنده و نرم افزار سیستم عامل باید قادر باشند مراجعات به حافظه در کد برنامه را به آدرس ها حافظه فیزیکی واقعی ترجمه کنند.

2. حفاظت

هر فرآیند باید در مقابل مراجعه ناخواسته فرآیندهای دیگر چه به صورت تصادفی و چه عمدی) حفاظت شود. لذا، برنامه های فرآیندهای دیگر نباید قادر باشند بدون مجوز به محل های حافظه فرآیندهای دیگر مراجعه کنند.

3. اشتراک

هر راهکار حفاظتی باید این انعطاف لازم را داشته باشد که اجازه دهد چندین فرآیند به بخش یکسانی از حافظه اصلی دستیابی داشته باشند.

نیازهای مدیریت حافظه

4. سازمان منطقی

حافظه به صورت فضای آدرس خطی یا یک بعدی سازمان یافته است و شامل دنباله ای از بایتها و کلمه ها است.

5. سازمان فیزیکی

حافظه کامپیوتر حداقل در دو سطح حافظه اصلی و حافظه ثانویه سازمان دهی می شود. در این طرح دو سطحی، سازمان دهی جریان اطلاعات بین حافظه اصلی و ثانویه موضوع مهم سیستم است و یک مسئولیت سیستمی است.

بخش بندی حافظه

بخش بندی ثابت

- در اغلب طرح های مدیریت حافظه، می توانیم فرض کنیم که سیستم عامل بخش ثابتی از حافظه اصلی را اشغال می کند و بقیه حافظه اصلی برای استفاده فرآیندها مهیا است.
- ساده ترین طرح مدیریت بر این حافظه موجود، بخش بندی آن به ناحیه هایی با مرزهای ثابت است.

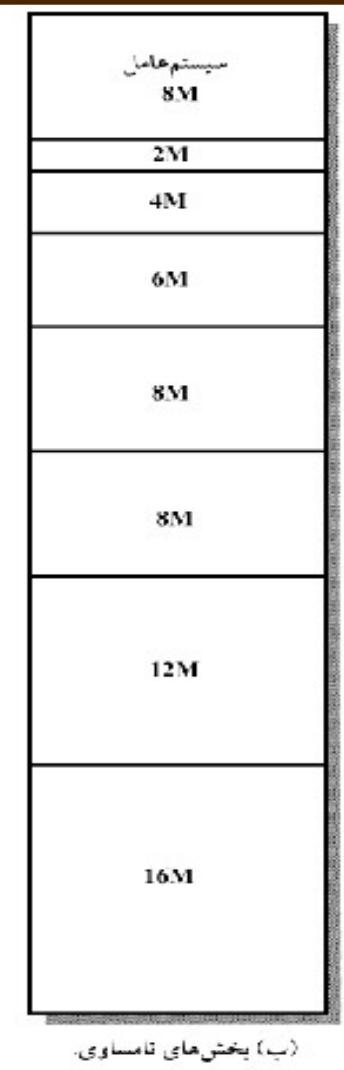
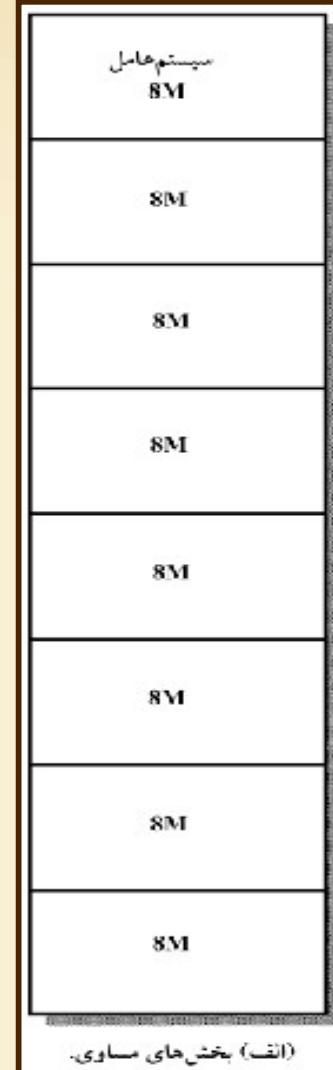
اندازه بخشهای

بخش‌های مساوی

- در این حالت، هر فرآیندی که اندازه اش کمتر یا مساوی اندازه بخش باشد، می تواند در بخش موجود بار شود.
- دو مشکل وجود دارد
- ممکن است برنامه خیلی بزرگ باشد و در یک بخش جا نشود.
- بهره وری از حافظه اصلی کاملاً نا کارآمد است.

بخش‌های نامساوی

بخش بندی حافظه



مثالی از بخش بندی ثابت در حافظه 64M

بخش بندی حافظه

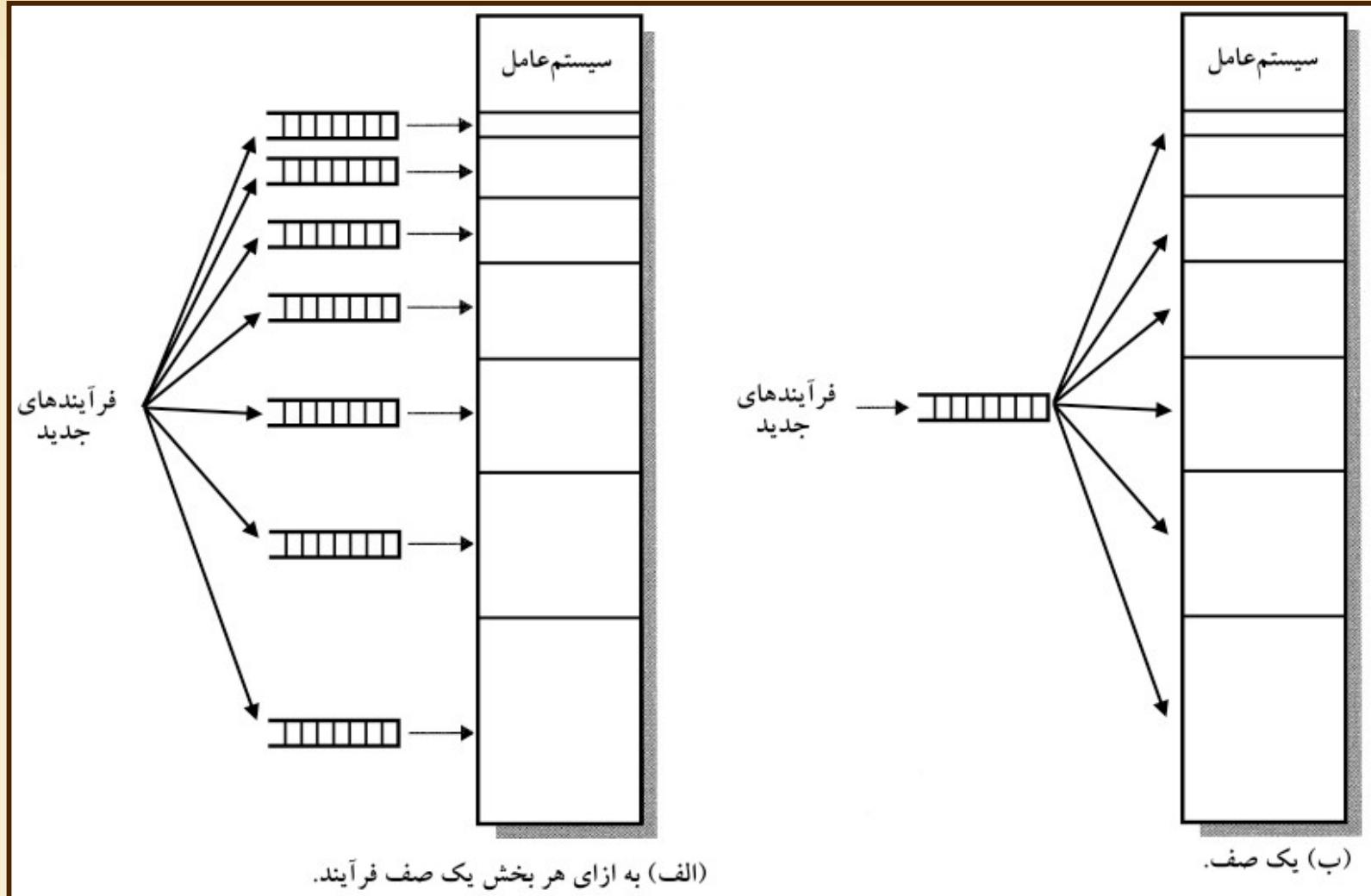
- تکه تکه شدن داخلی (internal fragmentation)
- اینکه قسمتی از داخل یک بخش به هدر رود پدیده تکه تکه شدن داخلی می گویند.

بخش بندی حافظه

الگوریتم جاگذاری

- در بخش ثابت، جاگذاری فرآیندها در حافظه آسان است.
- در بخش های نامساوی، دو روش برای تخصیص بخش ها به فرآیندها وجود دارد.
- ساده ترین راه این است که کوچکترین بخش ممکن که فرآیند در آن جا می شود تخصیص یابد.
- در این مورد، برای هر بخش نیاز به صفت زمان بندی است تا فرآیندهایی که از آن بخش به خارج مبادله شدند در آن نگهداری شوند.
- امتیاز این روش این است که فرآیندها طوری تخصیص می یابند که تکه تکه شدن داخلی به حداقل برسد.
- گرچه این تکنیک از دیدگاه هر بخش بهینه به نظر می رسد، در کل از دیدگاه سیستم بهینه نیست. به عنوان مثال، حالتی را در نظر بگیرید که در یک دوره زمانی، فرآیندی با اندازه بین ۱۲ و ۱۶ مگابایت وجود ندارد. در این حالت، از بخش ۱۶ مگابایتی استفاده نمی شود، گرچه فرآیند کوچک تری می توانست به آن نخصیص یابد.
- روش مناسب تر این است که برای تمام فرآیندها فقط یک صفت در نظر گرفته شود.

بخش بندی حافظه



تخصیص حافظه برای بخش بندی ثابت.

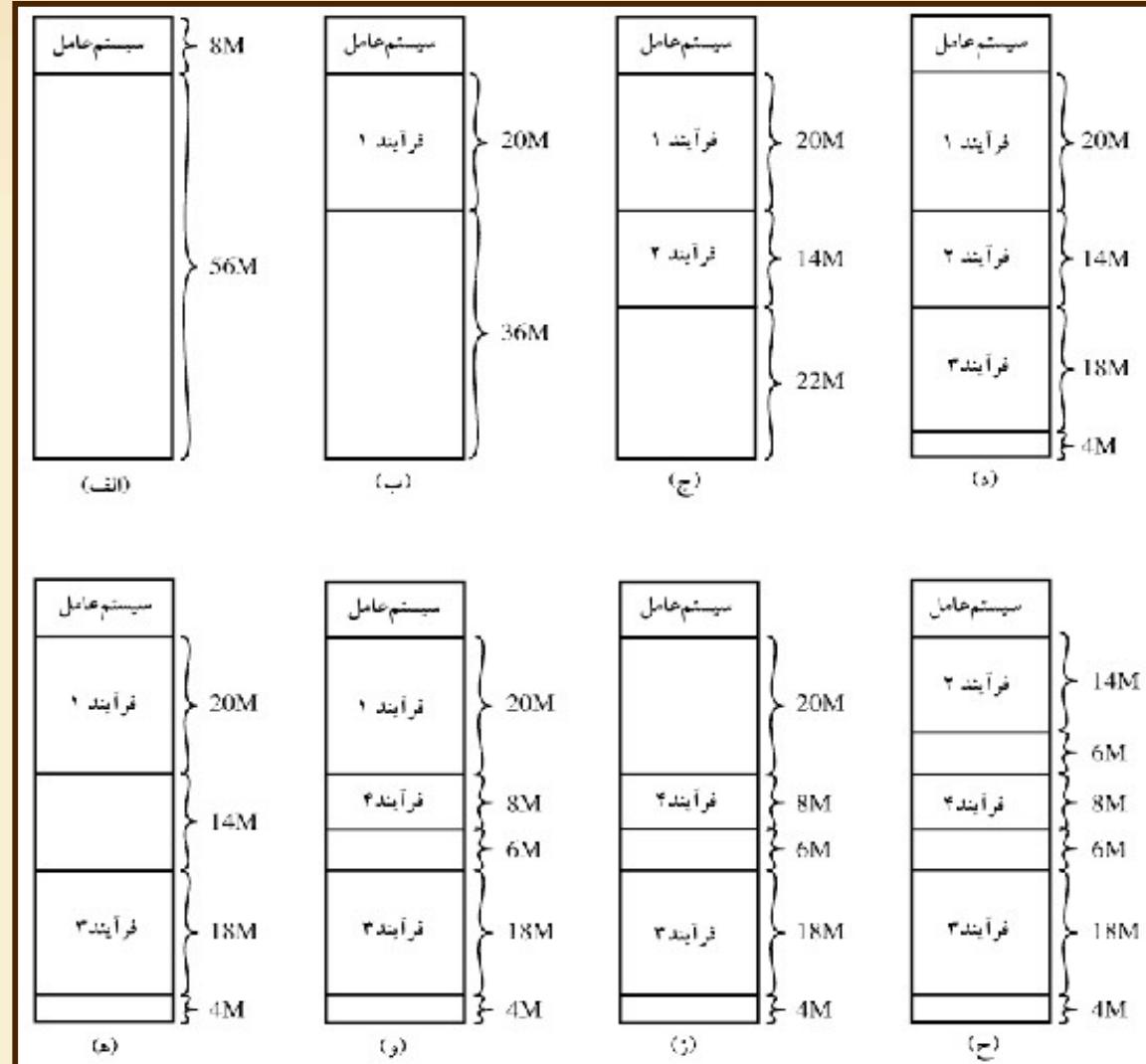
بخش بندی حافظه

- مزایای بخش بندی ایستا
 - به حداقل نرم افزار سیستم عامل و سربرار پردازشی نیاز دارد.
- معایب بخش بندی ایستا
 - تعداد بخشهای تعریف شده در زمان ایجاد سیستم، محدود کننده تعداد فرایندهای فعال (و نه معلق) در سیستم است.
 - از آنجا که اندازه بخشهای در زمان ایجاد سیستم انتخاب شده اند، کارهای کوچک نمی توانند با کارایی از فضای بخش استفاده کنند.

بخش بندی پویا

- در بخش بندی پویا، طول و تعداد بخش ها متغیر است.
- وقتی فرآیندی به حافظه آورده می شود، حافظه ای به اندازه مورد نیازش برای آن تخصیص می یابد.
- شروع این روش خوب است اما سرانجام وضعیتی به وجود می آید که در آن حفره های زیادی ایجاد می شود. با گذشت زمان، حافظه تکه تکه می شود و بهره وری از حافظه کاهش می یابد.
- این پدیده را تکه تکه شدن خارجی (External fragmentation) گویند.
- یک تکنیک غلبه بر تکه تکه شدن خارجی، فشرده سازی است.

بخش بندی پویا



اثر بخش بندی پویا.

بخش بندی پویا

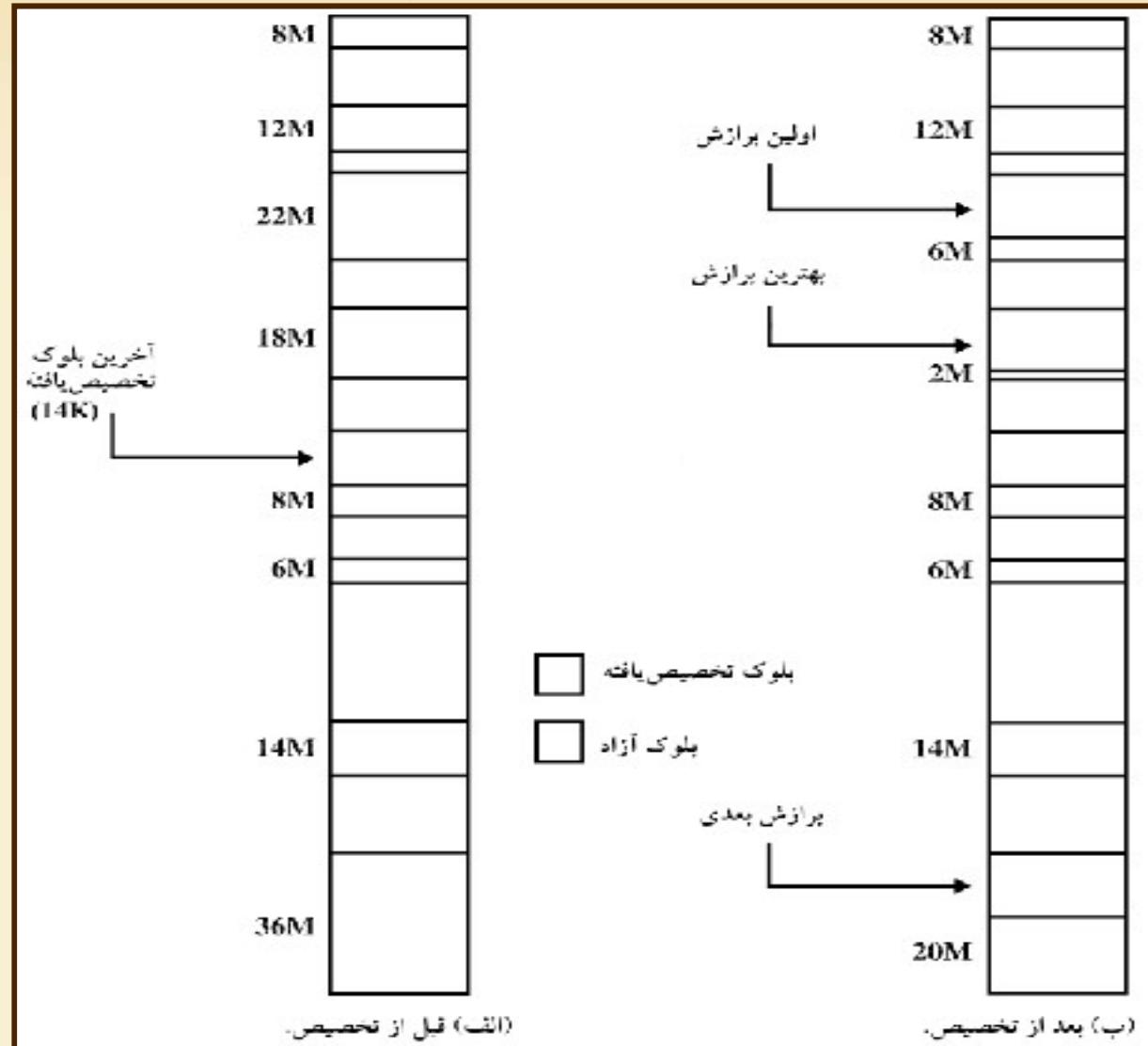
الگوریتم جاگذاری

- چون فشرده سازی حافظه وقت گیر است. طراح سیستم عامل باید در تخصیص حافظه به فرآیندها (چگونگی پر کردن حفره ها) هوشمندانه عمل کند.
- اگر در هنگام بار کردن یا مبادله فرآیند به حافظه، بیش از یک بلوک حافظه وجود داشته باشد که برای آن فرآیند کافی باشد، سیستم عامل باید تصمیم بگیرد کدام بلوک را تخصیص دهد.
- سه الگوریتم جاگذاری عبارت اند از:
 - بهترین برازش (best fit)
 - اولین برازش (first fit)
 - در پی برازش (next fit)

بخش بندی پویا

- بهترین برازش، فهرست تمام بلوک های موجود را جستجو می کند و بهترین بلوک را انتخاب می کند.
- اولین برازش، کل لیست بلوک های موجود را مرور می کند و اولین بلوک با اندازه کافی را برمی گزیند.
- در پی برازش، حافظه را از محل آخرین جایابی به بعد مرور می کند و اولین بلوک با اندازه کافی را انتخاب می کند.
- الگوریتم اولین برازش ساده بوده و همچنین بهترین و سریع ترین الگوریتم می باشد.
- الگوریتم بهترین برازش بر خلاف اسمش بدترین کارایی را دارد.

بخش بندی پویا



نمونه ای از پیکربندی حافظه، قبل و بعد از تخصیص بلوک ۱۶ مگابایتی.

بخش بندی پویا

الگوریتم جایگزینی

- در سیستم چند برنامه‌ای که از بخش بندی پویا استفاده می‌شود، زمانی فرا می‌رسد که تمام فرآیندهای موجود در حافظه در حالت مسدود هستند و حتی پس از فشرده سازی نیز فضای کافی برای فرآیند دیگر وجود ندارد.
- برای اجتناب از اتلاف وقت پردازنده در زمان انتظار برای خارج شدن یک فرآیند فعال از حالت مسدود، سیستم عامل یکی از فرآیندها را از حافظه خارج می‌کند تا فضای کافی برای فرآیند جدید یا فرآیندی با حالت آماده_معلق ایجاد شود. لذا، سیستم عامل باید تصمیم بگیرد کدام فرآیند را جایگزین کند.

سیستم رفاقتی (buddy system)

- هر دو طرح بخش بندی ثابت و پویا معاایبی دارند.
- یک روش متعادل، سیستم رفاقتی است.
- در سیستم رفاقتی اندازه بلوک های حافظه 2^K است، که در آن $L \leq K \leq U$
- کوچک ترین اندازه بلوکی که تخصیص یافت. $=2^L$
- بزرگ ترین اندازه بلوکی که تخصیص یافت. معمولاً 2^U برابر با کل حافظه موجود برای تخصیص است.

سیستم رفاقتی

- در شروع کار، کل فضای موجود برای تخصیص، به عنوان بلوکی از اندازه 2^U درنظر گرفته می شود.
- اگر حافظه ای به اندازه S درخواست شود که $2^{U-1} < S \leq 2^U$ ، کل بلوک تخصیص می یابد. و گرنه، بلوک به دو رفیق به اندازه های مساوی 2^{U-1} تقسیم می شود.
- اگر $2^{U-2} < S \leq 2^{U-1}$ باشد، درخواست به یکی از دو رفیق تخصیص می یابد. در غیر این صورت، یکی از رفیق ها دوباره تقسیم می شود.
- این روند ادامه می یابد تا کوچک ترین بلوک بزرگ تر یا مساوی S تولید شود و به درخواست تخصیص یابد.
- در هر زمان، سیستم رفاقتی لیستی از حفره های به اندازه (بلوک های تخصیص نیافته) را نگهداری می کند. یک حفره می تواند از لیست $1 + i$ حذف شود و به صورت دو رفیق به اندازه 2^i تخصیص نیافته درآیند، از آن لیست حذف می شوند و به صورت یک بلوک در لیست $(1 + i)$ ترکیب می شوند.

سیستم رفاقتی

■ با تقاضایی به اندازه $K \leq 2^i < 2^{i-1}$ است، الگوریتم بازگشتی زیر برای یافتن حفره ای به اندازه 2^i به کار می رود:

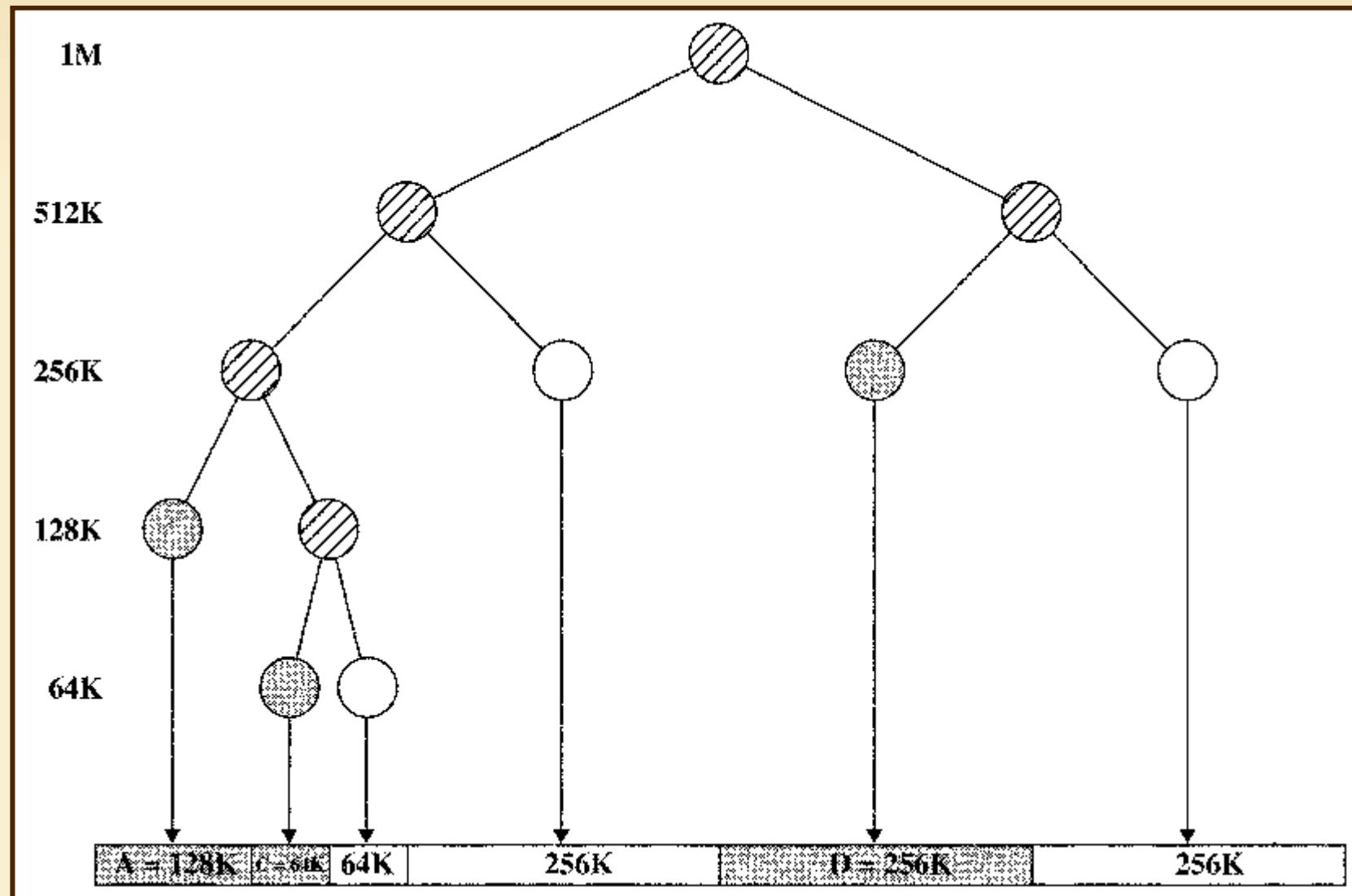
```
void get_hole(int i)
{
    if (i == (U + 1))
        <failure>;
    if (<i_list empty>)
    {
        get_hole(i + 1);
        <split hole into buddies>;
        <put buddies on i_list>;
    }
    <take first hole on i_list>;
}
```

سیستم رفاقتی

بلوک یک مگابایتی				1M
100K درخواست	A = 128K	128K	256K	512K
240K درخواست	A = 128K	128K	B = 256K	512K
64K درخواست	A = 128K	C = 64K	64K	B = 256K
256K درخواست	A = 128K	C = 64K	64K	B = 256K
B آزادسازی	A = 128K	C = 64K	64K	256K
A آزادسازی	128K	C = 64K	64K	256K
75K درخواست	E = 128K	C = 64K	64K	256K
C آزادسازی	E = 128K	128K	256K	D = 256K
E آزادسازی		512K	D = 256K	256K
D آزادسازی			1M	

نمونه ای از سیستم رفاقتی.

سیستم رفاقتی

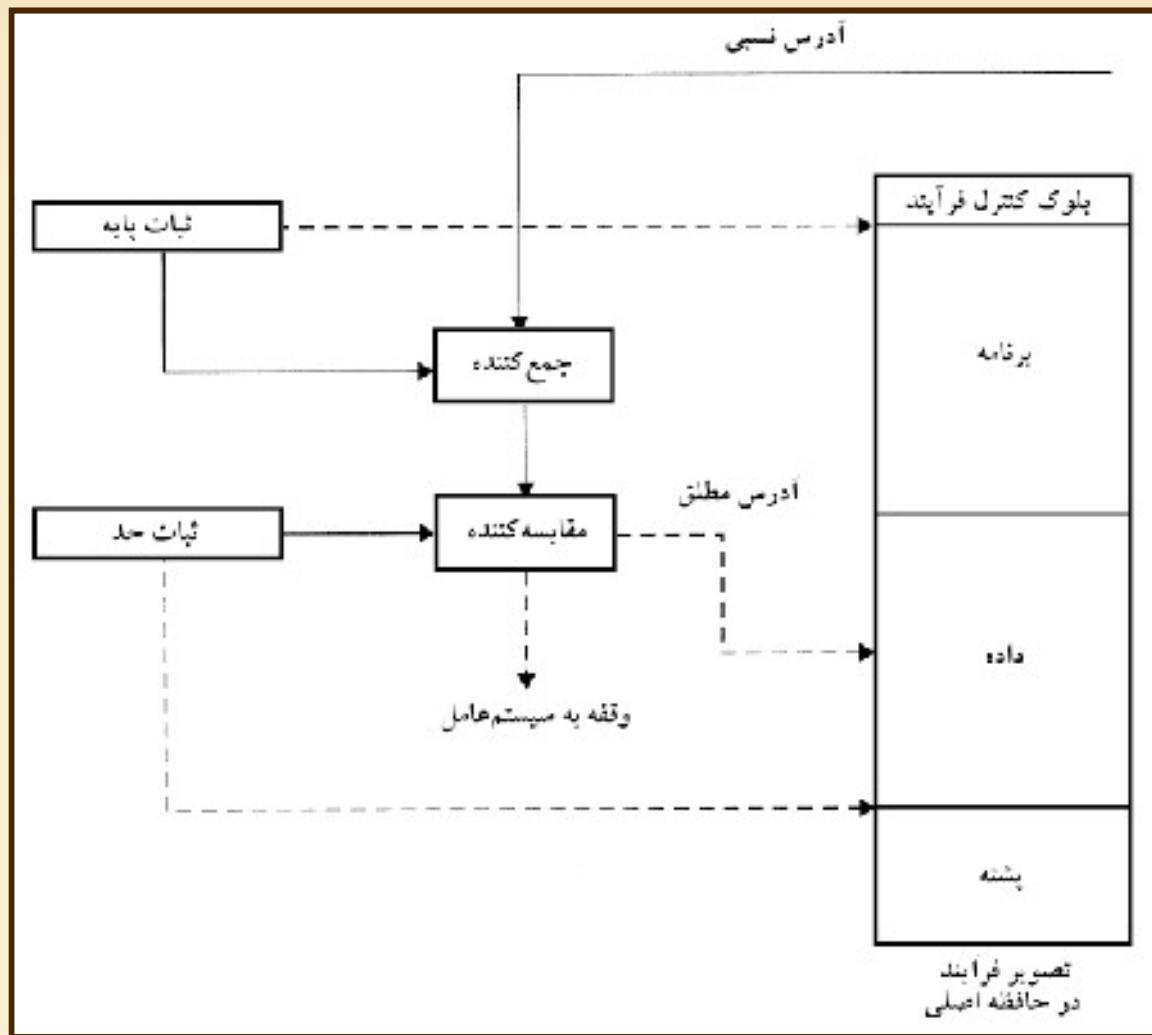


نمایش درختی سیستم رفاقتی.

جا به جایی

- وقتی فرآیندی در حالت اجرا قرار می گیرد، آدرس شروع فرآیند در حافظه اصلی، در ثبات خاصی از پردازنده به نام ثبات پایه قرار می گیرد.
- یک ثبات "حد" وجود دارد که محل پایانی برنامه را مشخص می کند.
- هنگام بارشدن برنامه به حافظه یا مبادله شدن تصویر فرآیند به حافظه، مقادیر این ثبات ها باید تعیین شوند.
- در حین اجرای فرآیند با آدرس های نسبی سروکار داریم که شامل محتويات ثبات دستورالعمل، آدرس های ناشی از دستورالعمل های انشعاب و فراخوانی، و آدرس داده ها در دستورالعمل های ذخیره و بارگردان است.

جا به جایی



پشتیبانی سخت افزار برای جا به جایی.

صفحه بندی (Paging)

- حافظه اصلی به بخش‌های نسبتاً کوچک هم اندازه تقسیم می‌شود.
- هر فرآیند به تکه‌های هم اندازه با آنها تقسیم می‌شود.
- به تکه‌های هر فرآیند صفحه (Page) می‌گویند.
- به تکه‌های موجود در حافظه قاب (Frame) می‌گویند.
- اتلاف حافظه برای هر فرآیند به خاطر تکه‌شدن داخلی است که فقط در آخرین صفحه آن فرآیند به وجود می‌آید.
- تکه‌شدن خارجی وجود ندارد.
- یک ثبات پایه کافی نیست.
- سیستم عامل یک جدول صفحه (Page Table) برای هر فرآیند ایجاد می‌کند.
- تبدیل آدرس منطقی به فیزیکی توسط سخت افزار انجام می‌شود.

صفحه بندی

حافظه اصلی	شماره قاب
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(الف) پنج قاب آزاد.

حافظه اصلی	
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(ب) بارگردان فرآیند A.

حافظه اصلی	
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	
8	
9	
10	
11	
12	
13	
14	

(ج) بارگردان فرآیند B.

حافظه اصلی	
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(د) بارگردان فرآیند C.

حافظه اصلی	
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(ه) مبادله فرآیند B به خارج.

حافظه اصلی	
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

(و) بارگردان فرآیند D.

تخصیص صفحات فرآیند به قاب های آزاد.

0	0
1	1
2	2
3	3

جدول صفحه
فرآیند A

0	—
1	—
2	—

جدول صفحه
فرآیند B

0	7
1	8
2	9
3	10

جدول صفحه
فرآیند C

0	4
1	5
2	6
3	11
4	12

جدول صفحه
فرآیند D

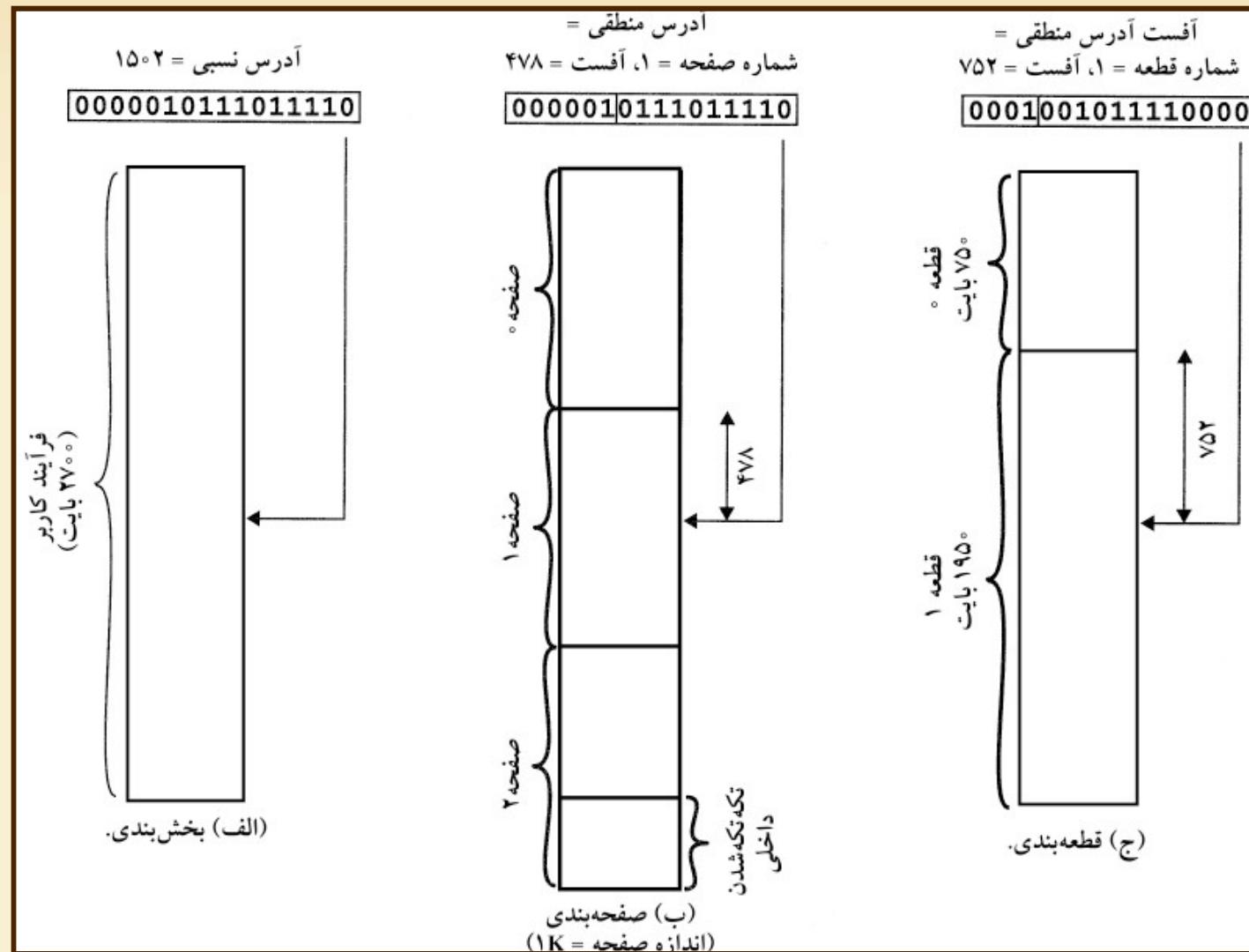
13
14

لیست قاب آزاد

صفحه بندی

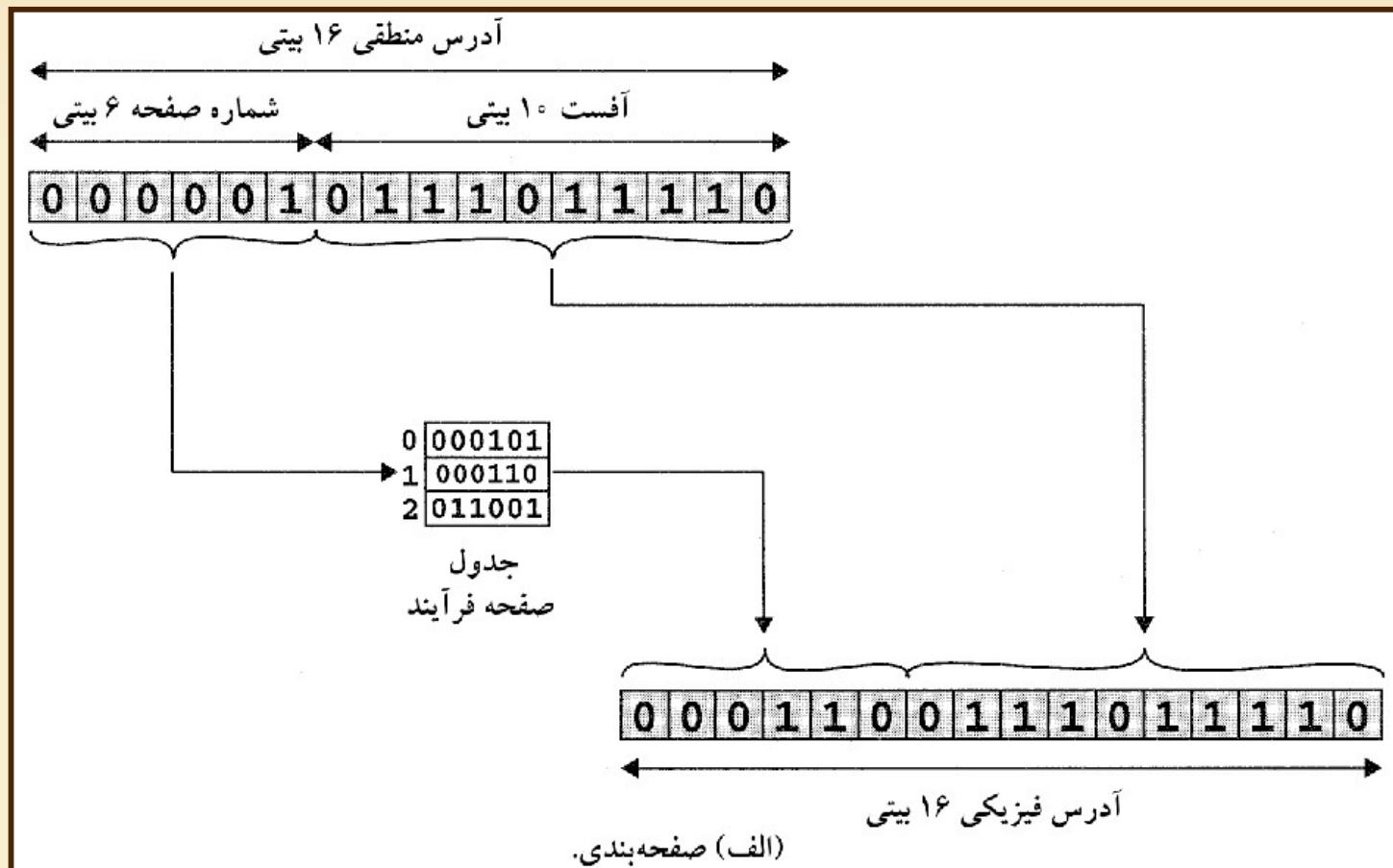
- یک آدرس $n+m$ بیتی را در نظر بگیرید.
 - n شماره صفحه
 - m انحراف (آفست)
- برای ترجمه آدرس مراحل زیر را انجام دهید:
 - استخراج n بیت سمت چپ آدرس منطقی به عنوان شماره صفحه.
 - استفاده از شماره صفحه به عنوان شاخصی در جدول صفحه فرآیند برای یافتن شماره قاب (K).
 - شروع آدرس فیزیکی برابر با $k \times 2^m$ و آدرس فیزیکی بایت مورد مراجعه برابر با مجموع آن عدد و آفست است.
 - نیاز به محاسبه این آدرس فیزیکی نیست، بلکه با الحاق کردن شماره قاب به آفست به دست می آید.

صفحه بندی



آدرسهای منطقی.

صفحه بندی



مثال هایی از ترجمه آدرس منطقی به فیزیکی.

قطعه بندی (Segmentation)

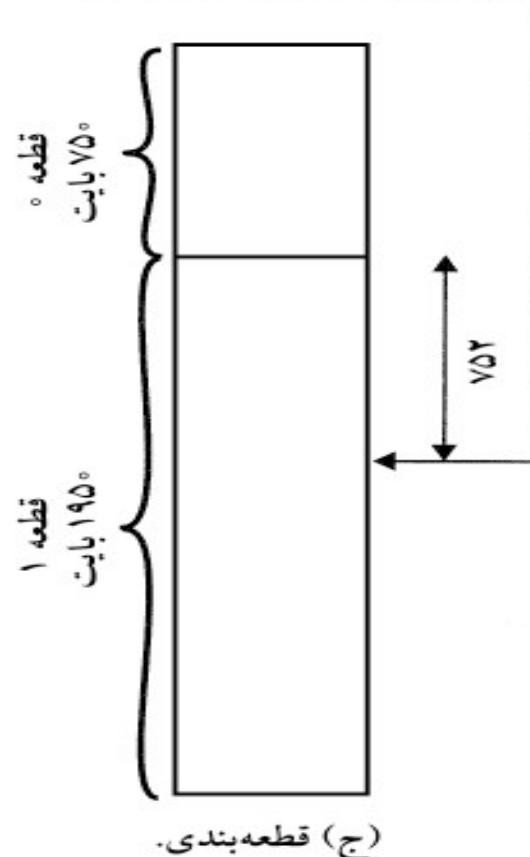
- روش جایگزین برای تقسیم برنامه کاربر، قطه بندی است.
- برنامه و داده ها به تعدادی قطعه (Segment) تقسیم می شود.
- لزومی ندارد قطعه ها دارای اندازه یکسان باشند.
- گرچه حداکثری برای طول قطعه وجود دارد.
- مشابه بخش بندی پویا است، ولی تفاوتش این است که یک برنامه بیش از یک بخش را اشغال کند و لزومی ندارد این بخشها پیوسته باشند.
- همانند صفحه بندی، آدرس منطقی در قطعه بندی شامل دو بخش شماره قطعه و آفست است.
- برخلاف صفحه بندی که توسط برنامه نویس قابل رویت نیست، قطعه بندی معمولاً قابل رویت است و موجب تسهیل سازمان دهی برنامه ها و داده ها می شود.

قطعه بندی

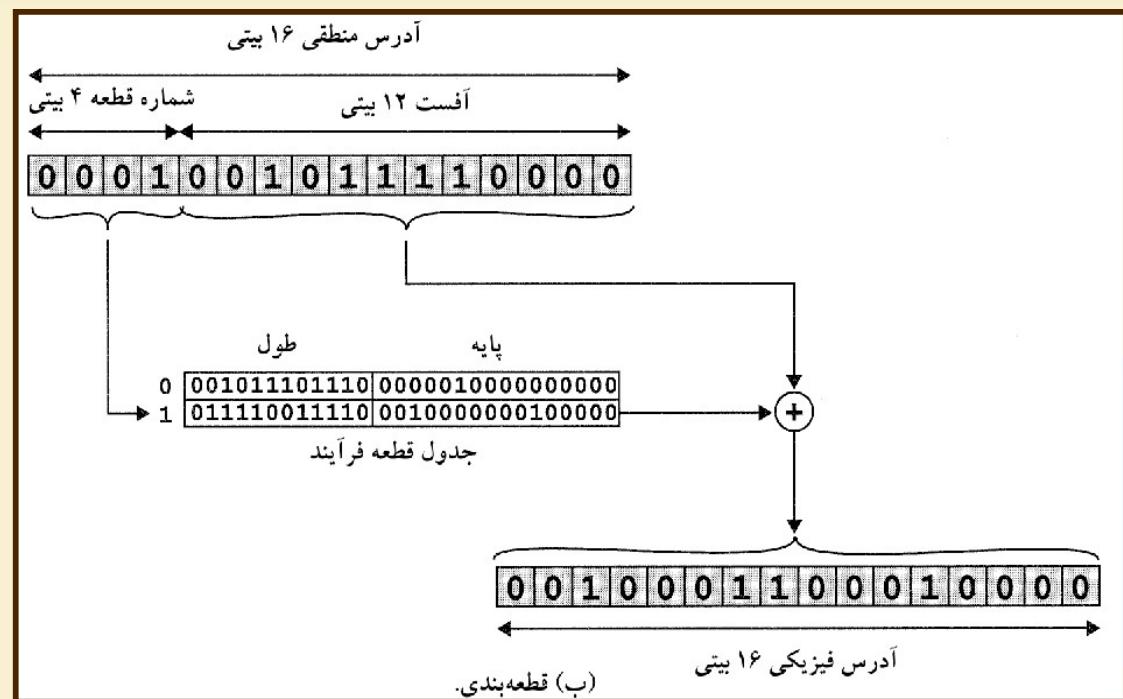
- آدرس $m + n$ بیتی را در نظر بگیرید که n بیت سمت چپ شماره قطعه و m بیت سمت راست آفست است.
- برای ترجمه آدرس مراحل زیر باید انجام گیرند:
 - استخراج n بیتی سمت چپ آدرس منطقی به عنوان شماره قطعه.
 - استفاده از شماره قطعه به عنوان شاخصی در جدول قطعه برای یافتن آدرس شروع قطعه.
- مقایسه آفست (m بیت سمت راست آدرس منطقی) با طول قطعه. اگر آفست بزرگ تر از طول قطعه باشد، آدرس نامعتبر است.
- آدرس فیزیکی موردنظر برابر با مجموع شروع آدرس فیزیکی قطعه آفست است.

قطعه بندی

آفست آدرس منطقی =
شماره قطعه = ۱، آفست = ۷۵۲
0001001011110000



آدرس منطقی.



ترجمه آدرس منطقی به فیزیکی.

فصل هشتم

حافظه مجازی

(Virtual Memory)

حافظه مجازی

مشخصه صفحه بندی و قطعه بندی:

- کلیه مراجع یک فرآیند به حافظه، آدرس منطقی هستند که به آدرس فیزیکی تبدیل می شوند.
- ممکن است یک برنامه به تکه های مختلفی شکسته شود و لازم نیست این تکه ها در حین اجرا به طور پیوسته در حافظه قرار گیرد.

تکه (Piece) به صفحه یا قطعه اطلاق می شود.

- به بخشی از فرآیند که مقیم در حافظه است را **مجموعه مقیم (resident set)** گویند.
- برای استفاده از این مجموعه از جدول صفحه استفاده می کند.
- اگر فرآیند به آدرس منطقی دسترسی نداشته باشد یک وقفه صادر می کند.
- این وقفه به معنی خطای دسترسی به حافظه وجود دارد و فرآیند مسدود می شود و برای اجرا ان فرآیند سیستم عامل ان را به حافظه اصلی می برد.

حافظه مجازی

- دو پیامد این امر
 - فرآیند بیشتری را می توان در داخل حافظه اصلی قرار دارد و استفاده بهتر از پردازنده می شود.
 - امکان دارد یک فرایند بزرگتر از حافظه اصلی باشد.
 - لذا برنامه نویس باید از اندازه فرآیند ها آگاه باشد.
 - با وجود صفحه بندی و قطعه بندی این وظیفه به عهده سخت افزار و سیستم عامل است.
- به حافظه اصلی **حافظه حقیقی** (*real memory*) می گویند.
- به حافظه بزرگتر و کارآمد تر که کاربر از آن استفاده می کند **حافظه مجازی** (*virtual memory*) می گویند.

حافظه مجازی

- کوبیدگی (thrashing) ■
 - اگر سیستم عامل تکه ای را به داخل بیاورد باید تکه دیگری را خارج کند.
 - اگر تکه ای را درست قبل از اینکه اجرا شود خارج کند، به فاصله خیلی کم دوباره مجبور است آن تکه را به داخل آورد.
 - تکرار زیاد این عمل منجر به پدیده ای به نام **کوبیدگی** می شود.
 - در این حالت پردازنده به جای اجرای دستورالعمل کاربران، بیشترین وقتش را صرف مبادله تکه ها می کند.

اصل محلی بودن

- مراجعات به برنامه و داده ها در حافظه، خوشه ای هستند.
- تعداد محدودی از فرآیند ها در حافظه اجرا می شوند و باید حدس زد که کدام برنامه در آینده اجرا می شود.
- به این ترتیب از کوبیدگی جلو گیری می کنند .
- تاکید در کارآمدی فرآیندها در یک محیط حافظه مجازی است.

