بسمه تعالی

سیستمهای عامل

علی چوداری خسروشاهی

Akhosroshahi@IAUT.ac.ir

دانشگاه آزاد اسلامی

1

مرجع

Operating Systems

نویسنده: Wiliam Stallings

ترجمه: دکتر حسین پدرام

www.akhosroshahi.ir

2

فصل اول

نگاه کلی به سیستم عامل

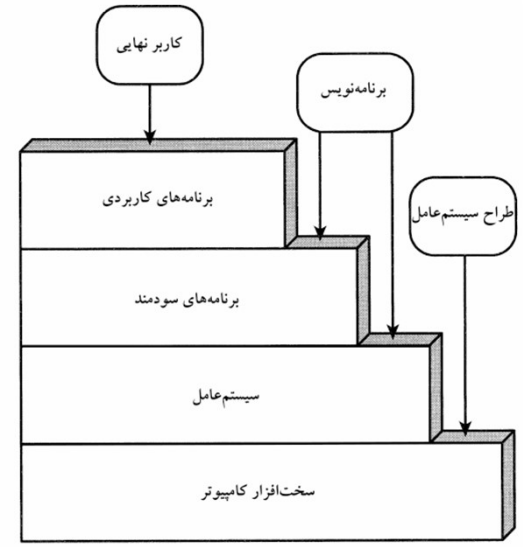
3

اهداف و وظایف سیستم عامل

- سیستم عامل برنامه ای است که اجرای برنامه های کاربردی را کنترل و بصورت رابط کاربر و سخت افزار عمل می کند.
- سه هدف سیستم عامل
  - **سهولت:** سیستم عامل موجب سهولت استفاده از کامپیوتر می شود.
  - **کارآمدی:** سیستم عامل موجب استفاده کارآمد از منابع سیستم کامپیوتری می شود.
  - **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.

4

لایه ها و نمای سیستم کامپیوتری



5

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 خطا
- حسابداری

6

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.

7

تکامل تدریجی سیستم عامل

۱- پردازش ردیفی (Serial Processing).

۲- سیتم های دسته ای ساده (Simple Batch System).

۳- سیستم های چند برنامه ای دسته ای (Multiprogrammed Batch System).

٤- سیستم های اشتراک زمانی (Time Sharing Systems).

8

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.

9

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.

10

سیستم های دسته ای

- اولین سیتم 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.

11

سیستم های دسته ای

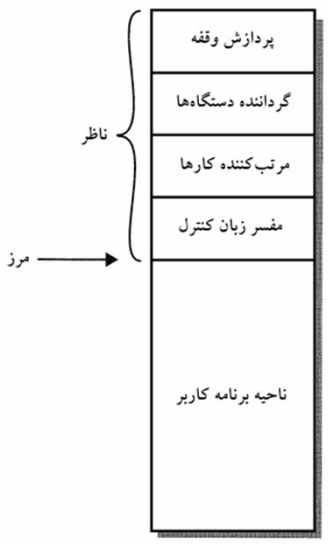
برای آشنایی با عملکرد این طرح، آن را از دو دیدگاه بررسی می کنیم:

- **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.

12

طرح حافظه برای ناظر ماندگار



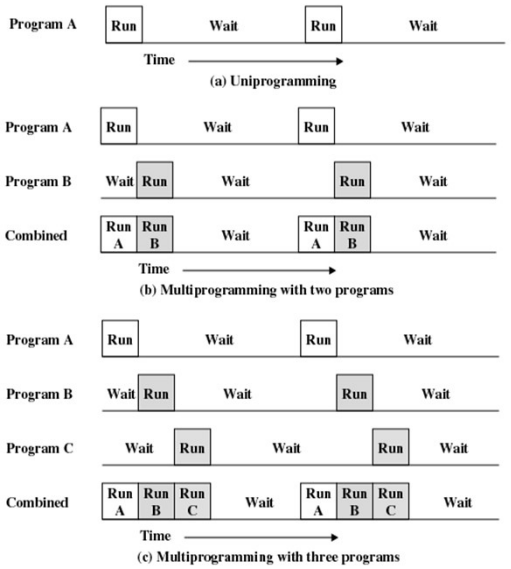
13

سیستم های چند برنامه ای دسته ای

- حتی بامرتب سازی خودکار کارها در سیستم عامل دسته ای ساده، باز هم پردازنده غالباً بی کار است. چون دستگاه های I/O 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

14

مثالی از چندبرنامه ای



15

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 1/N از زمان مفید پردازنده استفاده می کنند.
- عکس العمل انسان کند می باشد.

16

دستاوردهای اصلی

پنج دستاورد توسعه سیستم عامل:

۱- فرآیندها
Process

۲- مدیریت حافظه
Memory management

۳- حفاظت اطلاعات و ایمنی
Information protection and security

٤- زمانبندی و مدیریت منابع
Scheduling and resource management

۵- ساختار سیستم
System structure

17

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.

18

فصل دوم

شرح و کنترل فرآیند

19

حالات فرآیند

- اساسی ترین عمل پردازنده اجرای دستورالعمل های موجود در حافظه است.
- اجرا شامل دنباله ای از دستورالعمل های همان برنامه است.
- به اجرای یک برنامه خاص فرایند یا وظیفه گویند.
- رفتار یک فرآیند به خصوص 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.

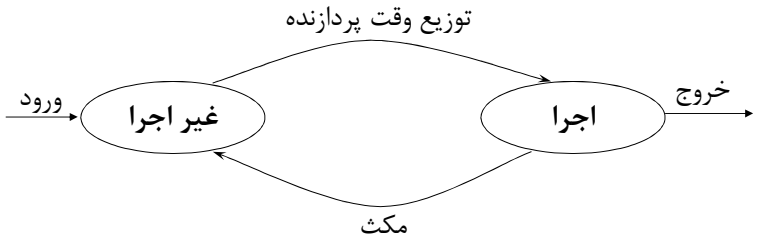
20

مدل دو حالته

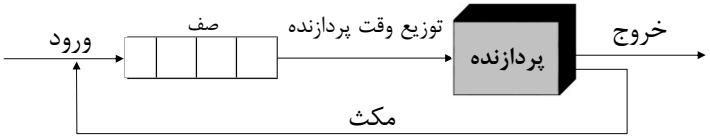
- مسئولیت اصلی سیستم عامل کنترل یک فرآیند است.
- یک فرآیند ممکن است در یکی از دو حالت اجرا و غیر اجرا باشد.
- وقتی یک سیستم عامل فرآیندی را دریافت می کند آن را در حالت غیر اجرا قرار می دهد.
- پس از رسیدن نوبت فرآیند فعلی را به حالت غیر اجرا برده و فرآیند را به حالت اجرا برده.
- در مورد هر فرآیندی باید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

21

مدل دو حالته



نمودار تغییر حالت



نمودار صف بندی

22

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.
- معمولا چهار حادثه موجب به ایجاد فرآیند می گردد

کار دسته اي جديد	سيستم عامل با جرياني از كنترل كار دسته اي سروكار دارد كه معمولاً بر روي نوار يادبسك ذخيره مي شود.وقتي سيستم عامل آماده تحويل گرفتن كار جديد مي شود، دنباله بعدي از فرمان هاي كنترل كار را مي خواند.
ارتباط محاوره اي توسط سيستم عامل ايجاد مي شود تا سرويس ارائه كند. عمل چاپ).	كاربر از طريق پايانه با سيستم ارتباط برقرار مي كند. سيستم عامل مي تواند فرآيندي را ايجاد كند تا عملي را از طرف برنامه كاربر انجام دهد، بدون اين كه كاربر منتظر بماند (مثل فرآيندي براي كنترل
توليد مثل توسط فرآيند موجود	به منظور بهره برداري از خاصيت توازي و مولفه اي بودن، برنامه كاربر مي تواند ايجاد تعدادي از فرآيندها را ديكته كند.

23

پایان فرآیند

- هر کار دسته ای باید حاوی دستورالعمل توقف یا فراخوانی صریح 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 spawing):**
    - فرآیندی با درخواست فرآیند دیگر به وجود می آید
  - **فرزند:**
    - فرایند زاینده را پدر و فرایند ایجاد شده را فرزند گویند

24

دلایل پایان فرآیند

کامل شدن عادي	فرآيند يك فراخواني سرويس سيستم عامل رااجرا مي كند تانشان دهد كه اجرايش كامل شده است.
مهلت زماني	فرآيند بيش از مهلت زماني تعيين شده است. راه هاي مختلفي براي اندازه گيري مهلت زماني وجود دارد. اين راه ها شامل كل زمان مصرفي ("زمان ساعت ديواري")، مدت زماني كه صرف اجرا شده است، و در فرآيند محاوره ايي، مدت زماني كه از آخرين ورودي گذشته است.
فقدان حافظه	حافظه مورد نياز فرآيند بيش از حافظه موجود در سيستم است.
نقض حدود	فرآيند سعي مي كند به محلي از حافظه مراجعه كند كه اجازه دستيابي ندارد.
خطاي حفاظت	فرآيند سعي در استفاده از منبع يا فايلي دارد كه مجوز آن را ندارد، يا سعي در استفاده نادرست از آن دارد، مثل نوشتن در فايلي كه فقط خواندني است.
خطاي محاسباتي	فرآيند سعي در انجام محاسبات غيرمجاز، مثل تقسيم بر صفر دارد، يا سعي در ذخيره اعداد بزرگ تر از ظرفيت سخت افزار دارد.
گذشت زمان	فرآيند بيش از حداكثر زمان لازم براي وقوع يك رويداد، منتظر مانده است.
خطاي I/O	در اثناي ورودي يا خروجي خطايي رخ مي دهد، مثل نيافتن فايل، عدم موفقيت در خواندن يا نوشتن پس از حداكثر تعداد دفعات تلاش (مثل زمان روبه روشدن با نواحي خراب نوار)، يا عمليات نامعتبر (مثل خواندن از چاپگر).
دستورالعمل نامعتبر	فرآيند سعي در اجراي دستورالعملي دارد كه موجود نيست (غالباً وقتي رخ مي دهد كه انشعابي به ناحيه اي از داده ها و تلاش براي اجراي داده ها صورت مي گيرد).
دستورالعمل ممتاز	فرآيند سعي در اجراي دستورالعملي دارد كه براي سيستم عامل درنظر گرفته شده است.
استفاده نادرست از داده	قطعه اي از داده با نوع غلط يا بدون مقدار اوليه.
دخالت سيستم عامل يا اپراتور	سيستم عامل يا اپراتور به دلايلي فرآيند را خاتمه مي دهند (مثل وجود بن بست).
پايان والد	وقتي فرآيند والد خاتمه مي يابد،سيستم عامل ممكن است تمام فرآيندهاي فرزند آن راخاتمه دهد.
درخواست والد	فرآيند والد مجوز پايان دادن به فرآيندهاي فرزند را دارد.

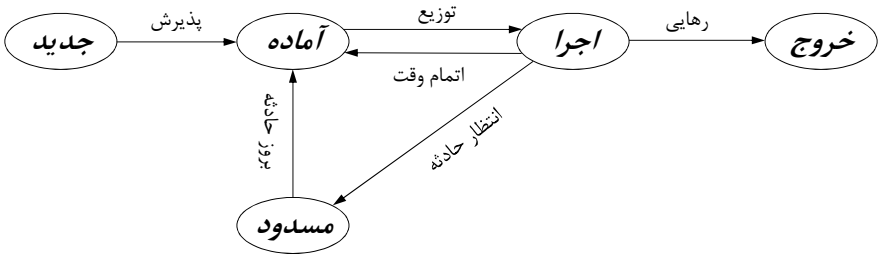
مدل پنج حالته

انواع حالتهای ممکن

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

26

مدل پنج حالته برای فرایند



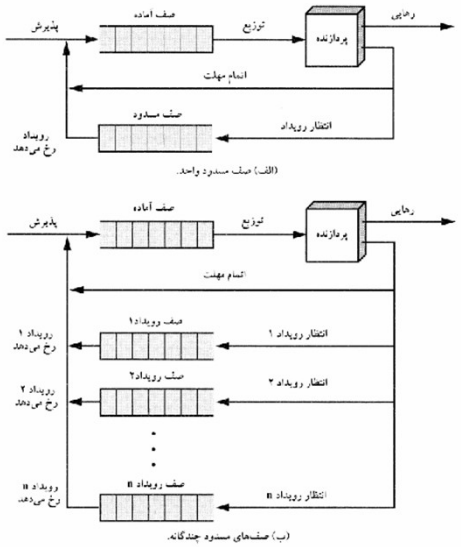
27

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.

28

مدل صف بندی



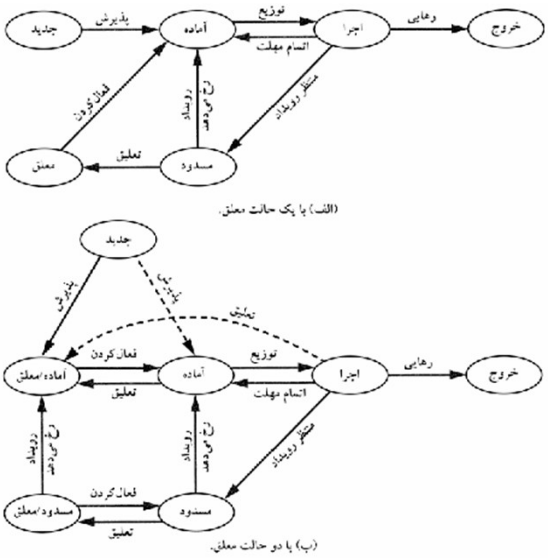
29

فرآیند معلق (Suspend)

- اگر هیچ کدام از فرآیندهای موجود در حافظه در حالت آماده نباشند، سیستم عامل یکی از فرآیندهای مسدود را به صف معلق بر روی دیسک انتقال می دهد(مبادله به خارج). این صف، حاوی فرآیندهایی است که موقتا از حافظه اصلی خارج یا معلق شدند. سپس سیستم عامل فرآیند دیگری را از صف معلق وارد حافظه می کند یا درخواست فرآیند جدید را می پذیرد (مبادله به داخل). فرآیند تازه وارد اجرا می شود.
- وقتی تمام فرآیندهای موجود در حافظه در حالت مسدود باشند، سیستم عامل می تواند فرآیندی را به تعلیق درآورد. برای این منظور، آن را در حالت معلق قرار می دهد و به دیسک منتقل می کند. با خروج فرآیند از حافظه اصلی، می تواند 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 می رسد که بهتر است فرآیند معلق به حافظه آورده شود، زیرا وارد کردن فرآیند جدید به سیستم، بار آن را افزایش می دهد.

30

نمودار تغییر حالت فرآیند، با حالت معلق



31

کاربردهای دیگر تعلیق

به طور کلی فرآیندی را معلق 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 معلق برود تا از اجرای آن جلوگیری شود.
4. تا زمانی که عامل تعلیق صراحتاً دستور ندارد، فرآیند نمی تواند از حالت معلق خارج شود.

32

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 محاورهای:** کاربر به منظور استفاده از منبع برنامه را معلق کند.
- **ترتیب زمانی:** فرآیند به طور دوره ای اجرا شود.
- **درخواست فرآیند پدر:** فرآیند فرزند زا به تعلیق بیندازد.

33

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 معمولا همراه آن به سیستم عامل می آید که به مجموعه این صفات بلوک کنترل فرآیند (Process Control Block) می گویند.
- به مجموعه برنامه، داده ها، پشته و صفات، تصویر فرآیند (Process Image) می گویند و محل آن به طرح مدیریت حافظه بستگی دارد و در حافظه ثانویه نگهداری می شود و برای اجرا باید به حافظه اصلی بار شد.

34

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 فراخوانی مربوط به رویه ها و سیستم ها استفاده می شود.
- **بلوک کنترل فرآیند**
  - داده های مورد نیاز سیستم عامل برای کنترل فرآیند.

35

عناصر بلوک کنترل فرآیند

شناسه فرآیند

شناسه ها

شناسه هاي عددي كه مي توانند با بلوك كنترل فرآيند ذخيره شوند عبارت اند از:

- شناسه اين فرآيند
- شناسه فرآيندي كه اين فرآيند را ايجاد كرد (فرآيند والد)
- شناسه كاربر

اطلاعات حالت فرآيند

ثبات هاي كاربر

ثبات هاي وضعيت و كنترل

اين ها ثبات هايي هستند كه براي كنترل عمل پردازنده به كار مي روند:

- **شمارنده برنامه** حاوي دستورالعمل بعدي است كه بايد واكشي شود.
- **كدهاي وضعيت** نتيجه آخرين عمل محاسباتي يا منطقي (علامت، صفر، رقم نقلي، مساوي، سرريز).
- **اطلاعات وضعيت** شامل پرچم هاي فعال سازي و غيرفعال سازي وقفه،و حالت اجرا است.

اشاره گرهاي پشته

هر فرآيند حاوي يك يا چند پشته سيستم است. پشته براي ذخيره پارامترها و آدرس فراخواني رويه و سيستم است. اشاره گر پشته بر بالاي پشته اشاره مي كند.

اطلاعات كنترل فرآيند

اطلاعات حالت و زمان بندي

سيستم عامل براي زمان بندي به اين اطلاعات نياز دارد. اين اقلام اطلاعاتي عبارت اند از:

- **حالت فرآيند** آمادگي فرآيند براي زمان بندي و اجرا را تعريف مي كند (اجرا، آماده، منتظر، متوقف).
- **اولويت** يك يا چند فيلد ممكن است براي توصيف اولويت زمان بندي فرآيند استفاده شوند.
- **اطلاعات مربوط به زمان بندي** به الگوريتم زمان بندي بستگي دارد. مثال ها عبارت اند از مدت انتظار فرآيند، مدت اجراي فرآيند در آخرين باري كه در حال اجرا بود.
- **رويداد** هويت رويدادي كه فرآيند قبل از ازسرگيري منتظر آن است.

ساختمان داده ها

هر فرآيندي ممكن است به فرآيند ديگري در يك صف، حلقه يا ساختمان ديگري پيوند داده شود.

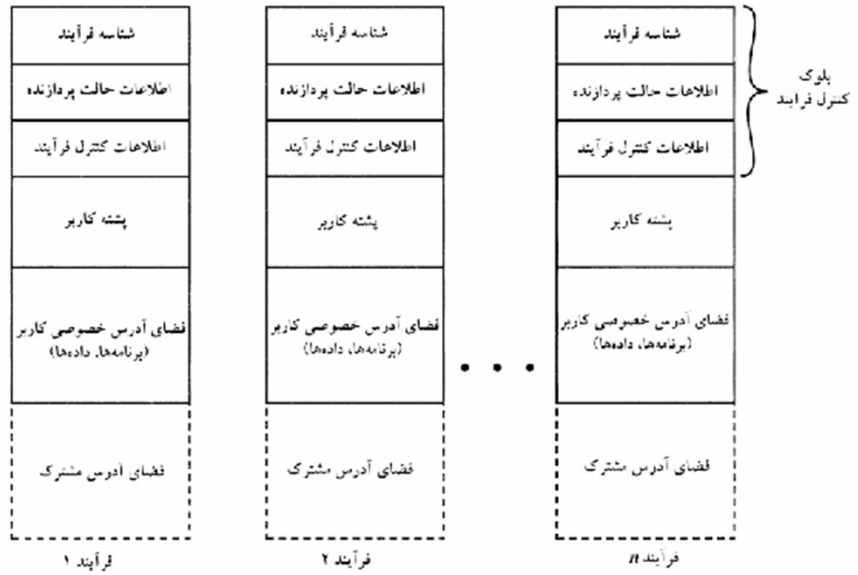
ارتباط بين فرآيندها

امتيازات فرآيند

مديريت حافظه

مالكيت و بهره وري از منبع

فرآیند کاربر در حافظه مجازی



کلمه وضعیت برنامه

Program Statue Word (PSW)

- طراحی تمام پردازنده ها شامل یک ثبات یا مجموعه ای از ثبات ها است که به نام کلمه وضعیت برنامه (PSW) شناخته می شود و حاوی اطلاعات وضعیت است.

38

فصل سوم

نخ

Thread

39

فرایندها و نخها

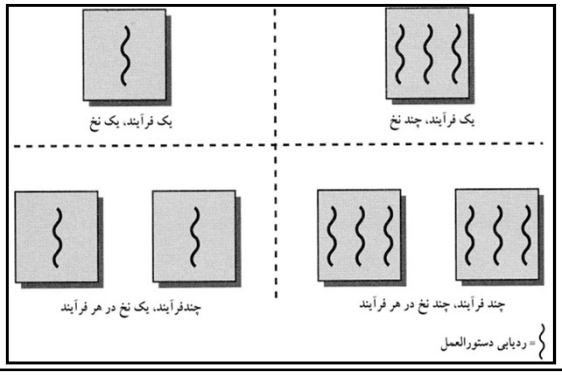
بحثی که تاکنون ارائه شد، دو ویژگی برای فرآیند مطرح کرده است:

- **تملک منبع:** هر فرآیند حاوی یک فضای آدرس مجازی برای نگهداری تصویر فرآیند است
- **توزیع وقت پردازنده/اجرا**: هر فرآیند دارای یک حالت اجرا (درحال 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 **فرایند سبُک وزن** (light weight process) خوانده می شود، در حالی که واحد مالکیت منبع به نام **فرآیند** یا **وظیفه** خوانده می شود.

40

چند نخی

- 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.



41

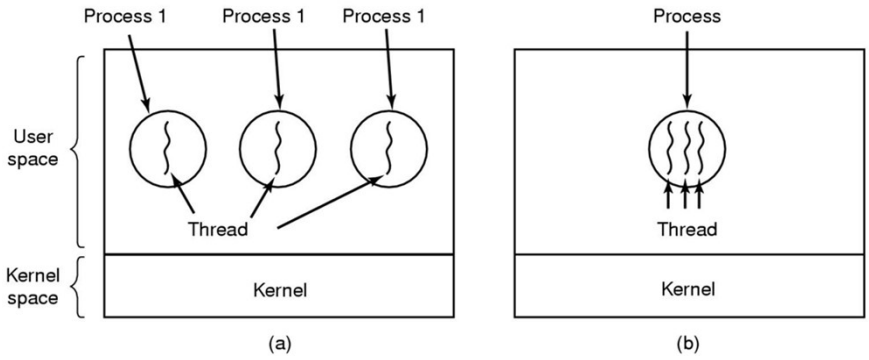
چند نخی (ادامه)

- 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).

42

چند نخی (ادامه)

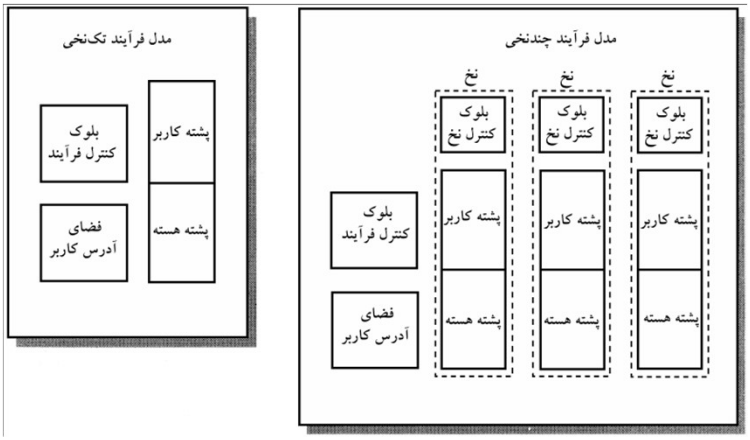
- (a) Three processes each with one thread
- (b) One process with three threads



43

چند نخی (ادامه)

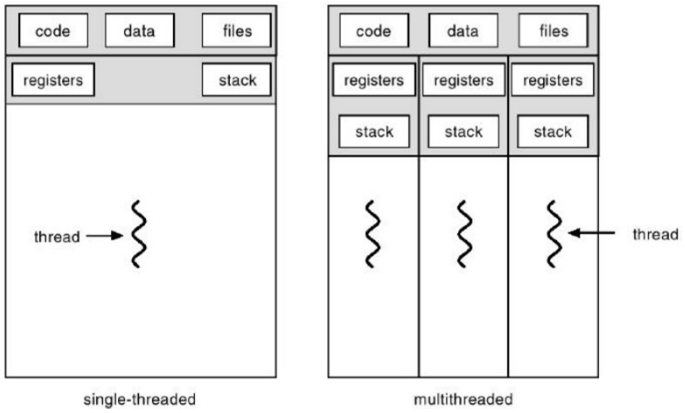
- مدل های تک نخی و چندنخی



44

چند نخی (ادامه)

- مدل های تک نخی و چندنخی



45

چند نخی (ادامه)

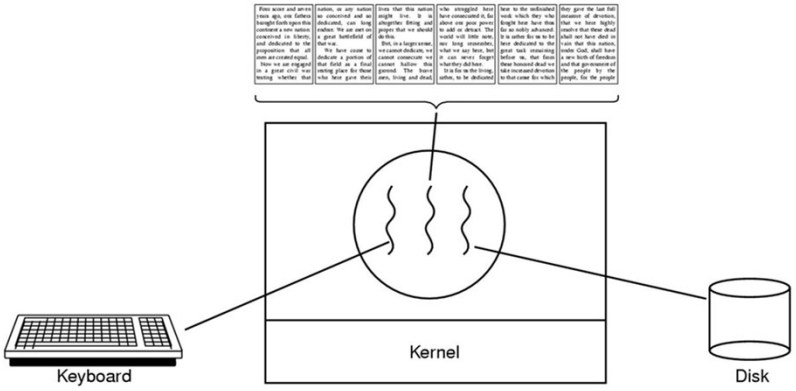
فایده نخ ها، نقش مهم آن ها در کارایی است:

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 تهیه 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.

46

چند نخی (ادامه)

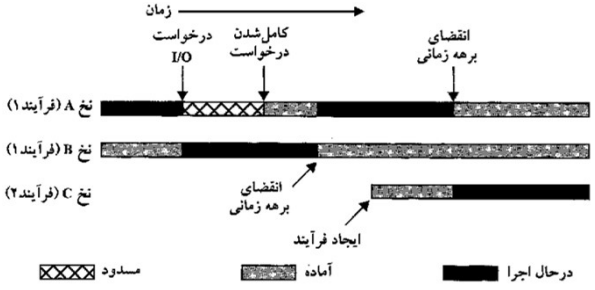


A word processor with three threads

47

عملکرد نخ

- مهم ترین حالت های نخ ها عبارت اند از در حال *اجرا*، *آماده* و *مسدود*. معمولاً نخ ها فاقد حالت *تعلیق* 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 گرفتند. وقتی نخِ در حال اجرا مسدود می شود یا برهه زمانی آن خاتمه می یابد، اجرا به نخ دیگری منتقل می شود.



48

فصل چهارم

زمانبندی تک پردازنده ای

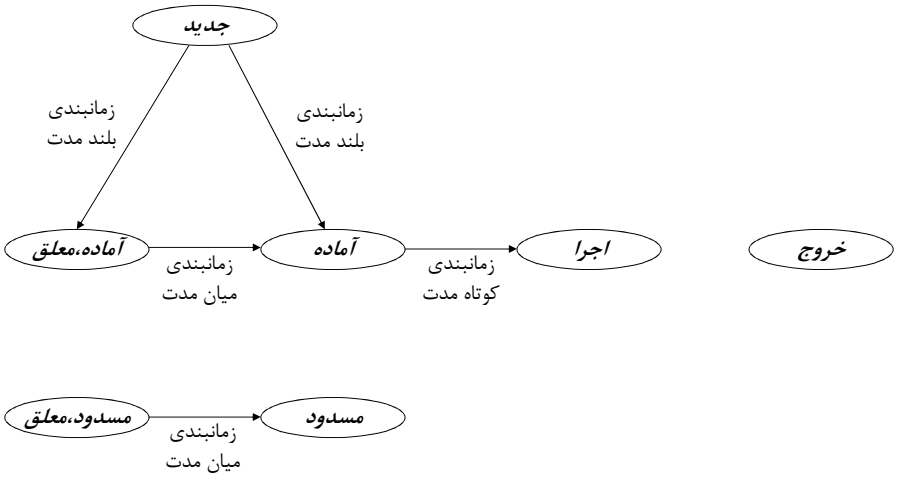
49

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 کارایی 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.**

50

زمانبندی و تغییر حالتهای فرآیند



51

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.

٤- 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- حداکثر شدن Throughput: تعداد کارهایی است که در واحد زمان انجام می شود.

٦- زمان انتظار(Waiting time): مجموع پریودهای زمانی صرف شده در صف آماده می باشد.

52

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.

53

حالت تصمیم گیری

زمانی که تابع انتخاب به اجرا در می آید را مشخص می کند . 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 می تواند هنگام ورود و... ایجاد شود.

54

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.

55

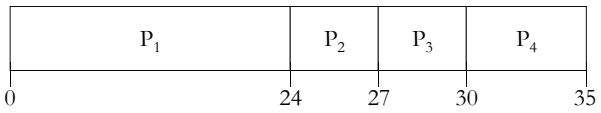
زمانبندی خدمت به ترتیب ورود FCFS

- 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

56

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: $P_1=0$; $P_2=24$; $P_3=27$; $P_4=30$

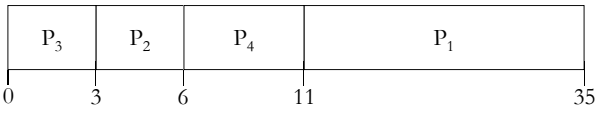
متوسط زمان انتظار: $(0+24+27+30)/4=20.25$

57

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



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

متوسط زمان انتظار: $(11+3+0+6)/4=5$

58

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

59

زمانبندی RR

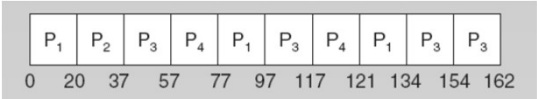
- تابع انتخاب: ثابت
- حالت تصمیم گیری: با قبضه کردن
- توان عملیاتی: اگر برهه زمانی خیلی کوچک باشد کم
- زمان پاسخ : برای فرایند های کوتاه زمان بسیار خوب
- سربار: کم
- تاثیر بر روی فرآیند ها: عملکرد عادلانه
- گرسنگی: خیر

غیر انحصاری ،به هر پردازش مقدار مشخصی cpu می دهند. به عبارت بهتر زمان استفاده از آن به کوانتوم هایی تقسیم می شود.

60

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: $P_1=81; P_2=20; P_3=94; P_4=97$

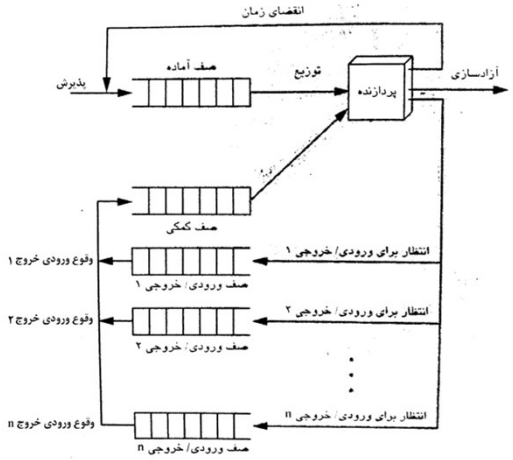
متوسط زمان انتظار: $(81+20+94+97)/4=73$

61

زمانبندی نوبت گردشی مجازی VRR

Virtual Round Robin

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



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

62

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 می‌گذرد 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.

63

زمانبندی SPN

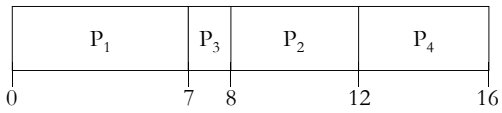
- تابع انتخاب: min[s]
- حالت تصمیم گیری: بدون قبضه کردن
- توان عملیاتی: زیاد
- زمان پاسخ: برای فرایند های کوتاه خوب
- سربار: می تواند زیاد باشد
- تاثیر بر روی فرآیند ها: به فرایند های طولانی صدمه زده
- گرسنگی(starvation): امکان دارد

فرایند هایی که زمان پاسخ کوتاهی دارند اول اجرا می شوند.

64

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: $P_1=0$; $P_2=6$; $P_3=3$; $P_4=7$

متوسط زمان انتظار: $(0+6+3+7)/4=4$

65

زمانبندی کوتاه ترین زمان باقیمانده
SRT
Shortest Remaining Time

- فرایندی اجرا می شود که کمترین باقی مانده زمان پاسخ را داشته باشد.
- وقتی فرایندی به صف آماده وارد می شود امکان دارد زمان باقیماده کمتری نسبت به فرایند در حال اجرا داشته باشد که با عمل قبضه کردن آن فرایند به اجرا در می آید.
- نوعی SPN با قبضه کردن است.
- مانند SPN باید تخمینی از زمان پردازش داشته باشیم.
- خطر گرسنگی فرایندهای طولانی تر وجود دارد.

66

زمانبندی SRT

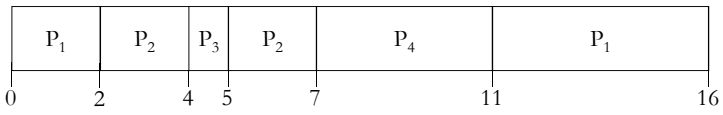
- تابع انتخاب: min[s-e]
- حالت تصمیم گیری: با قبض کردن
- توان عملیاتی: زیاد
- زمان پاسخ: خوب
- سربار: می تواند زیاد باشد
- تاثیر بر روی فرآیند ها: به فرایند های طولانی صدمه زده
- گرسنگی: امکان دارد

فرایندی اجرا می شود که کمتری باقی مانده زمان پاسخ را داشته باشد.

67

زمانبندی SRT

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: $P_1=9; P_2=1; P_3=0; P_4=2$

متوسط زمان انتظار: $(9+1+0+2)/4=3$

68

زمانبندی بالاترین نسبت پاسخ HRRN

Highest Response Ratio Next

- *اولویت داینامیک بوده و کارهای کوتاه تر اولویت بیشتر داشته و زودتر اجرا می شوند.*
- فرمول تشخیص بالاترین پاسخ : max(w+s/s)
  - 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 بایستی قبل از به کارگیری HRRN تخمین زده شود.

69

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.

70

زمانبندی فید بک FB

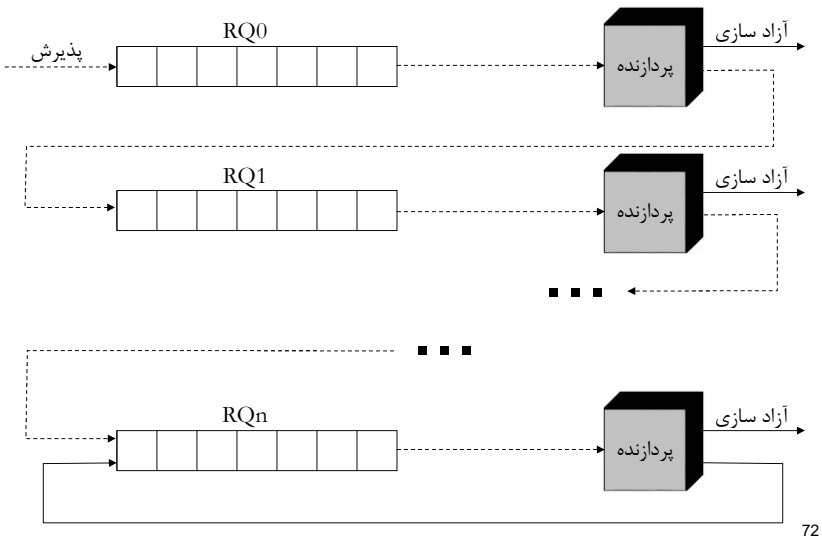
Feedback

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 فرایند 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.

71

زمانبندی فید بک FB



72

زمانبندیFB

- 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.

73

مقایسه کارآمدی

زمان کل turnaround time (TAT)

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

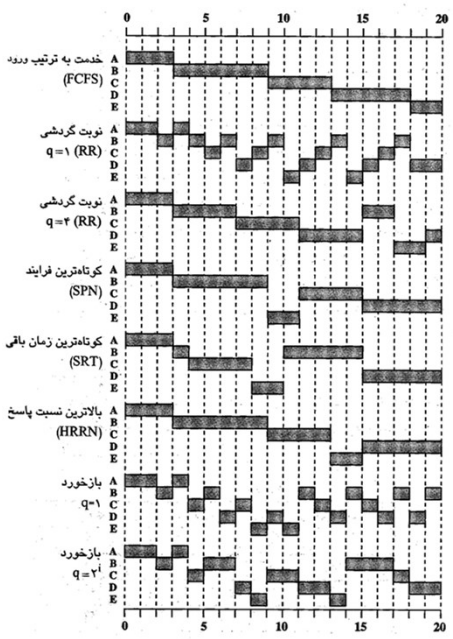
- مثال: زمانبندی فرایند

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

74

مثال: زمانبندی فرایند

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



شکل ۹-۵: مقایسه سیاستهای زمانبندی

مثال: زمانبندی فرایند

	فرایند زمان ورود زمان خدمت (T_s)	A ۰ ۳	B ۲ ۶	C ۴ ۴	D ۶ ۵	E ۸ ۲	میانگین
FCFS	زمان پایان زمان کل (T_r) T_r/T_s	۳ ۳ ۱	۹ ۷ ۱٫۱۷	۱۳ ۹ ۲٫۲۵	۱۸ ۱۲ ۲٫۴۰	۲۰ ۱۲ ۶	۸٫۶ ۲٫۵۶
RR (q=1)	زمان پایان زمان کل (T_r) T_r/T_s	۴ ۴ ۱٫۳۳	۱۸ ۱۶ ۲٫۶۷	۱۷ ۱۳ ۳٫۲۵	۲۰ ۱۴ ۲٫۸۰	۱۵ ۷ ۳٫۵۰	۱۰٫۸۰ ۲٫۷۱
RR (q=4)	زمان پایان زمان کل (T_r) T_r/T_s	۳ ۳ ۱	۱۷ ۱۵ ۲٫۵	۱۱ ۷ ۱٫۷۵	۲۰ ۱۴ ۲٫۸۰	۱۹ ۱۱ ۵٫۵۰	۱۰ ۲٫۷۱
SPN	زمان پایان زمان کل (T_r) T_r/T_s	۳ ۳ ۱	۹ ۷ ۱٫۱۷	۱۵ ۱۱ ۲٫۷۵	۲۰ ۱۴ ۲٫۸۰	۱۱ ۳ ۱٫۵۰	۷٫۶۰ ۱٫۸۴
SRT	زمان پایان زمان کل (T_r) T_r/T_s	۳ ۳ ۱	۱۵ ۱۳ ۲٫۱۷	۸ ۴ ۱	۲۰ ۱۴ ۲٫۸۰	۱۰ ۲ ۱	۷٫۲۰ ۱٫۵۹
HRRN	زمان پایان زمان کل (T_r) T_r/T_s	۳ ۳ ۱	۹ ۷ ۱٫۱۷	۱۳ ۹ ۲٫۲۵	۲۰ ۱۴ ۲٫۸۰	۱۵ ۷ ۳٫۵۰	۸ ۲٫۱۴
FB (q=1)	زمان پایان زمان کل (T_r) T_r/T_s	۴ ۴ ۱٫۳۳	۲۰ ۱۸ ۳	۱۶ ۱۲ ۳	۱۹ ۱۳ ۲٫۶۰	۱۱ ۳ ۱٫۵۰	۱۰ ۲٫۲۹
FB (q=2)	زمان پایان زمان کل (T_r) T_r/T_s	۴ ۴ ۱٫۳۳	۱۷ ۱۵ ۲٫۵۰	۱۸ ۱۴ ۳٫۵۰	۲۰ ۱۴ ۲٫۸۰	۱۴ ۶ ۳	۱۰٫۶۰ ۲٫۶۳

جدول ۹-۵: مقایسه سیاستهای زمانبندی

فصل پنجم

همزمانی :
انحصار متقابل و همگام سازی

77

مقدمه

- همه موضات 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.
  - پردازش توزیعی: مدیریت فرایند فرایندهای متعدد در روی سیستم های کامپیوتری 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.

78

اصول همزمانی

- در یک سیستم تک پردازنده 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.

79

اصول همزمانی

- یک مثال ساده
- رویه زیر را در نظر بگیرید

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

- دنباله زیر را در نظر بگیرید

```
      Process P1                Process P2
   .                         .
   chin = getch ();          .
   .                         chin = getch ();
   .                         chout = chin;
   .                         putch ( chout );
   chout = chin;             .
   putch ( chout );          .
```

80

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 نهایی داده های اشتراکی بر این اساس مشخص می شود که چه فرایندی آخر پایان می پذیرد.

- برای جلوگیری از شرایط مسابقه، فرایندهای همزمان باید هماهنگ شوند.

81

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 فرایند ها 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 فرایند های همزمان دیگر باشد (موضوع این فصل).

82

محاوره فرایندہ ها

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

83

محاوره فرایندہ ها

درجه آگاهی	رابطه	تأثیر فرآیندها بر یکدیگر	مسائل بالقوه کنترل
فرآیندها از یکدیگر بی خبر هستند	رقابت	■ نتایج یک فرآیند مستقل از فعالیت سایر فرآیندها است. ■ امکان تأثیر در زمان بندی فرآیند.	■ انحصار متقابل ■ بن بست(منابع تجدیدشدنی) ■ گرسنگی
فرآیندها به طور غیرمستقیم از یکدیگر خبر دارند (مثلاً از طریق شئ مشترک)	همکاری از طریق اشتراک	■ نتایج فرآیند ممکن است به اطلاعات ناشی از فرآیندهای دیگر بستگی داشته باشد. ■ امکان تأثیر در زمان بندی فرآیند.	■ انحصار متقابل ■ بن بست(منابع تجدیدشدنی) ■ گرسنگی ■ وابستگی داده ها
فرآیندها به طور مستقیم از یکدیگر خبر دارند (فراهم بودن عوامل اولیه ارتباط برای آنها)	همکاری از طریق ارتباط	■ نتایج یک فرآیند ممکن است به اطلاعات ناشی از فرآیندهای دیگر بستگی داشته باشد. ■ امکان تأثیر در زمان بندی فرآیند.	■ بن بست(منابع مصرفی) ■ گرسنگی

جدول ۲_۵ محاوره فرآیندها.

84

رقابت فرایندها برای منابع

- هنگامی که چند فرایند برای استفاده از منابع یکسان رقابت می کنند، به درگیری می رسند.
- 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 region (ناحیه بحرانی) یا Critical Section (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)

85

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.

86

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 تاخیر 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.

87

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.

88

انحصار متقابل: رویکرد نرم افزاری

اولین تلاش

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

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

- 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**
  - (int turn=0;) مقدار اولیه برابر صفر
- رویه انتظار مشغول (Busy wating) 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 پردازنده 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 کندتر 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.

91

انحصار متقابل: رویکرد نرم افزاری

دومین تلاش

- متغیر های اشتراکی
  - boolean **flag[2]**;
  - مقدار اولیه 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);
```

92

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.

93

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.

94

انحصار متقابل: رویکرد نرم افزاری

چهارمین تلاش

- متغیر های اشتراکی
- boolean **flag[2]**;
- مقدار اولیه flag[0]=flag[1]=false;
- برای پردازنده 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.

95

انحصار متقابل: رویکرد نرم افزاری

الگوریتم DEKKER:

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

96

```
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;
      /*reminder section*/
  }
}
```

الگو ریتم Dekker

```
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;
      /*reminder 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؟

مسأله ۵-۶ کتاب

98

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 متغییر 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؟

99

انحصار متقابل: حمایت سخت افزار

از کار انداختن وقفه

- در یک ماشین تک پردازنده ،هم زمان نمی شود فرایند 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 */
}
```

100

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.

101

انحصار متقابل: حمایت سخت افزار

دستور العملهای ویژه ماشین

- دستورالعمل آزمون و مقدار گذاری
  - دراین روش هر فرایند در هنگام وارد شدن به ناحیه بحرانی مقدار متغییر 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.

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

102

انحصار متقابل: حمایت سخت افزار

دستور العملهای ویژه ماشین

- انحصار متقابل با دستورالعمل آزمون و مقدار گذاری
  - داده مشترک

boolean lock = false;

  - فرایند P_i

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


انحصار متقابل: حمایت سخت افزار

دستور العملهای ویژه ماشین

- انحصار متقابل با دستورالعمل آزمون و مقدار گذاری

```
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) )
}
```

انحصار متقابل: حمایت سخت افزار

دستور العملهای ویژه ماشین

- دستورالعمل معاوضه
- دستور العمل معاوضه را می‌توان بصورت زیر نوشت:

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

105

انحصار متقابل: حمایت سخت افزار

دستور العملهای ویژه ماشین

- انحصار متقابل با دستورالعمل معاوضه

```
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) )
}
```

106

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.

107

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 می‌توانند به وسیله سیگنال‌های ساده همکاری کنند، به طوری که فرآیند می تواند در نقطه خاصی مجبور به توقف شود و منتظر سیگنالی بماند.
- برای علامت دادن از متغییر ویژه ای به نام راهنما استفاده می کند.

108

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 ان، معلق می ماند.

109

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. اگرمقدار مثبت نباشد، فرآیند مسدود شده توسط Wait، از حالت مسدود خارج می شود.

110

راهنماها (semaphore)

```
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;
        }
}
```

111

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;
    }
}
```

112

راهنماها (semaphore)

انحصار متقابل با Semaphore

- داده مشترک

semaphore mutex; //initially *mutex* = 1

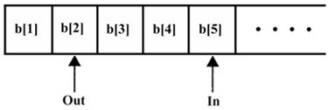
- فرایند P_i

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

113

مسئله تولید کننده و مصرف کننده

- یک یا چند تولیدکننده، 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 تولیدکننده 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 */
}
```

114

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, costumer);
}
```

117

مسئله تولید کننده و مصرف کننده

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

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

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

118

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, costumer);
}
```

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

119

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) دستورالعمل 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) وقفه‌ها.

دو پیاده سازی ممکن برای سمافورها.

120

ناظرها (monitor)

- ایجاد یک برنامهٔ درست از سمافورها، دشوار است.
- ناظر یک مولفه نرم افزاری است که از یک یا چند رویه، دنباله مقداردهی اولیه، و داده های محلی تشکیل شده است.
- ویژگی های مهم ناظر عبارت اند از:
  - متغیرهای محلی فقط توسط رویه های ناظر قابل دستیابی اند و رویه های خارجی نمی توانند به آن ها دستیابی داشته باشند.
  - فرآیند با فراخوانی یکی از رویه های ناظر وارد آن می شود.
  - در هر زمان فقط یک فرآیند می تواند در ناظر اجرا شود. سایر فرآیندهایی که ناظر را فراخوانی می کنند معلق می شوند و منتظر می مانند تا ناظر مهیا شوند.

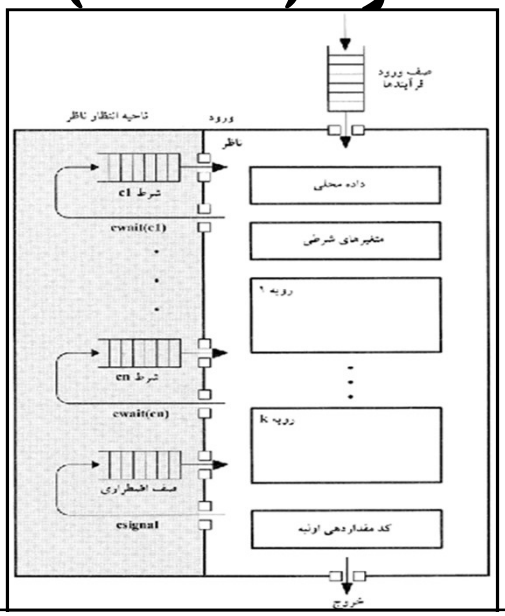
121

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.
- دوعمل برای متغیرهای شرطی انجام می شوند:
  - cwait(c) اجرای فرآیند فراخوان را در شرایط c معلق می کند. اکنون فرآیند دیگر می تواند از ناظر استفاده کند.
  - csignal(c) 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.

122

ناظرها (monitor)



123

```
monitor bouendedbuffer;
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 consmer()
{
    char x;
    while( true )
    {
       take(x);
       consume( x );
    }
}

void main()

{
    nextin=0; nextout=0; count=0;

    Parbegin(producer, cunsumer);

}
```

راه حلی برای مساله تولید کننده و مصرف کننده با میانگیر محدود، با استفاده از یک ناظر

124

فصل ششم

همزمانی :
بن‌بست و گرسنگی

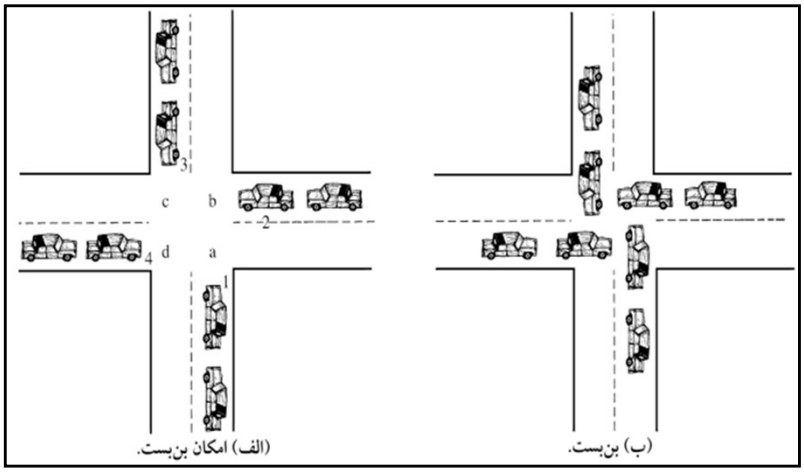
125

اصول بن‌بست

- بن بست را به صورت مسدود بودن دائمی مجموعه ای از فرآیند ها که برای منابع سیستم رقابت می کنند یا با یکدیگر در ارتباط هستند .
- راه حل کارامدی برای بن بست وجود ندارد.
- تمام بن بست ها در اثر نیازهای متضاد دو یا چند فرآیند به منابع رخ می دهند.

126

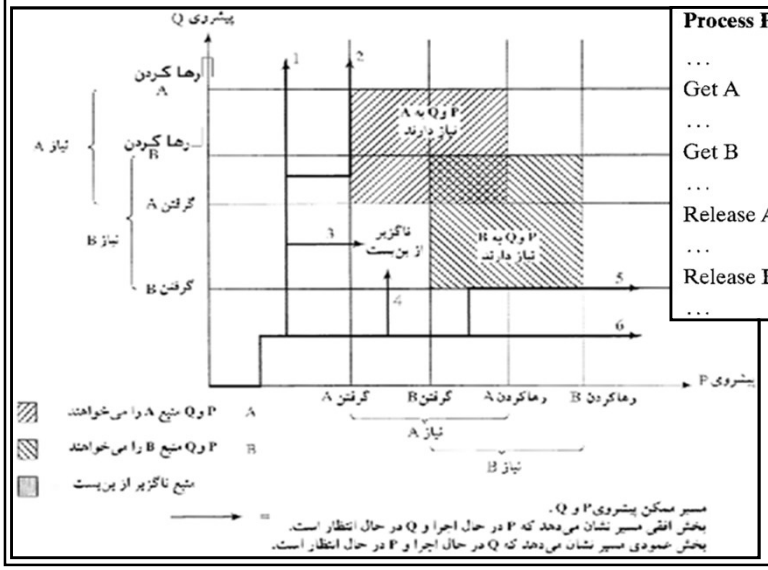
اصول بن‌بست



نمایش بنبست

127

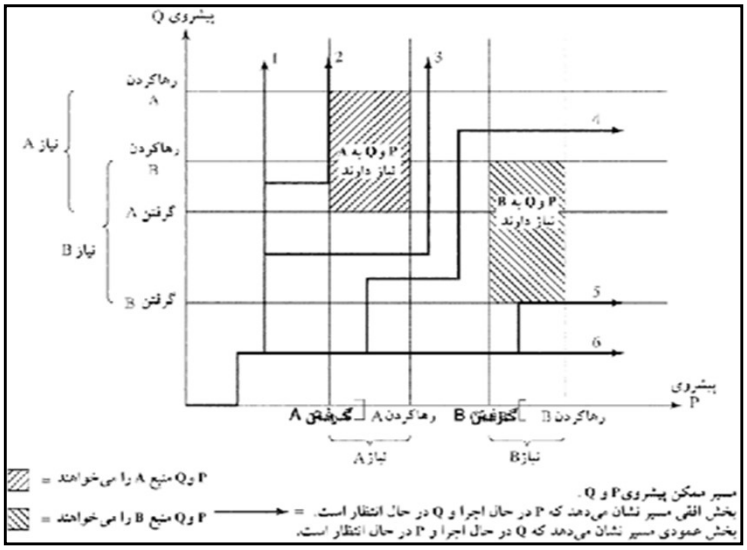
اصول بن‌بست



مثال بن‌بست

128

اصول بن‌بست



مثال عدم بن‌بست

129

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.

130

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

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

131

انواع منابع

- به عنوان مثالی از بن بست ناشی از منابع مصرف شدنی، دو فرآیند زیر را درنظر بگیرید که هرفرآیند سعی می کند پیامی را از فرآیند دیگر دریافت کند وسپس پیامی را به فرآیند دیگر بفرستد:

```
P1                          P2
...                         ...
Receive (P2);               Receive (P1);
...                         ...
Send (P2, M1);              Send (P1, M2);
```

- 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.

132

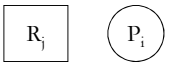
شرایط بن‌بست

- شرایط 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 می کند.
    - در این صورت یک منبع به مدت طولانی در اختیار یک فرآیند خواهد بود.
  - قبضه نکردن: (شرط لازم)
    - منبع را نمی توان به زور پس گرفت.
    - تنها وقتی از بن بست جلوگیری می کند :که منابع دارای اولویت باشند.
  - انتظار مدور: (شرط لازم و کافی)
    - چند فرآیند درخواست منبع هایی می کنند که در اختیار منبع دیگر است
    - و به همین سبب در انتظار چرخشی قرار می گیرند.
    - موجب کند کردن فرآیند ها و رد کردن غیر ضروری دسترسی منابع می شود

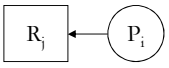
133

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.

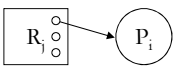


- یال گراف از یک فرآیند به یک منبع، نشان می دهد که فرآیند آن منبع را درخواست کرد ولی هنوز آن را به دست نیاورده است.



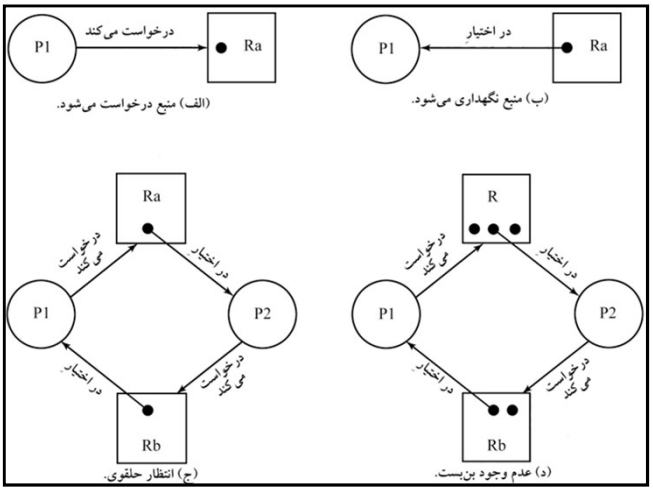
- در داخل یک گره منبع، برای هر نمونه از آن منبع یک نقطه قرار داده شده است.

- یال گراف از گره منبع به یک فرآیند، تقاضایی را نشان می دهد که برآورده شده است. (منبع به فرآیند اختصاص یافته).



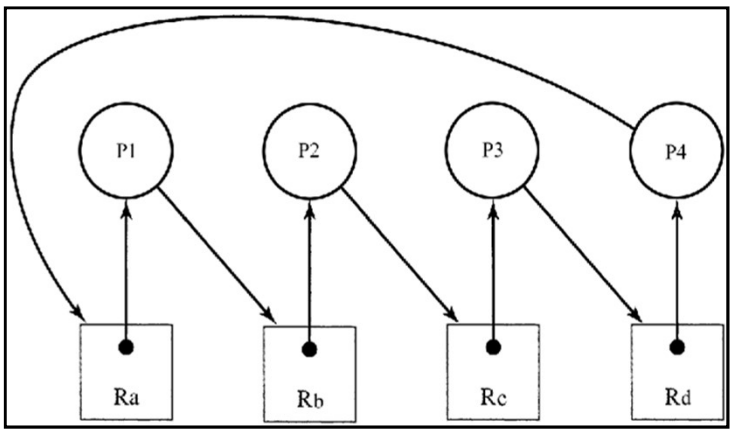
134

گراف تخصیص منابع



مثال هایی از گراف های تخصیص منابع.

گراف تخصیص منابع



مثالی از گراف های تخصیص منابع.

رویکرد های 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.

137

پیشگیری از بن بست

- روش غیر مستقیم
  - پیشگیری از حالتهای ۱و۲و۳ (انحصار متقابل، نگهداشتن و انتظار، قبضه نکردن)
- روش مستقیم
  - پیشگیری از بروز انتظار مدور

138

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، درحالی 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.

139

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 رادرخواست 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 اِعمال 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.

140

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 انتحاب 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.

141

عدم شروع فرآیند

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

$\text{Resource} = R = (R_1, R_2, \ldots, R_m)$ کل منابع موجود در سیستم

$\text{Available} = V = (V_1, V_2, \ldots, V_m)$ تعداد کل هر منبعی که به فرآیندی تخصیص نیافت و موجود است

$\text{Claim} = C = \begin{vmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{22} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nm} \end{vmatrix}$ C_{ij} = نیاز فرآیند i به منبع j

$\text{Allocation} = A = \begin{vmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{vmatrix}$ A_{ij} = 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 فرایند است.
برای این که اجتناب از بن بست امکان پذیر باشد، فرآیند باید این اطلاعات را از قبل اعلان کند.

142

عدم شروع فرآیند

روابط زیر برقرار است:

- برای تمام iها 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 kها و iها داریم:

$$C_{ki} \le R_i$$

  - معنایش این است که هیچ فرآیندی نمی تواند منابعی بیش از منابع موجود در سیستم را درخواست کند.
- برای تمام kها و iها داریم:

$$A_{ki} \le C_{ki}$$

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

143

عدم شروع فرآیند

- فرایند را وقتی شروع کنید که برای تمام iها داشته باشیم:

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

  - 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 می‌کند تمام فرآیندها همزمان حداکثر نیاز خود را درخواست می کنند.

144

عدم تخصیص منبع

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

145

عدم تخصیص منبع

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

$$C_{ij} - A_{ij} \le V_i$$ برای تمام j ها

146

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	6	1	2
P3	2	1	1
P4	0	0	2

ماتریس Allocation

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

C — A

R1	R2	R3
0	1	1

بردار Available

R1	R2	R3
9	3	6

بردار Resource

(الف) حالت اولیه.

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

ماتریس Claim

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

ماتریس Allocation

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

C — A

R1	R2	R3
6	2	3

بردار Available

R1	R2	R3
9	3	6

بردار Resource

(ب) P2 به‌طور کامل اجرا می‌شود.

...

عدم تخصیص منبع

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

ماتریس Claim

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

ماتریس Allocation

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

C — A

R1	R2	R3
7	2	3

بردار Available

R1	R2	R3
9	3	6

بردار Resource

(ج) P1 به‌طور کامل اجرا می‌شود.

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

ماتریس Claim

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

ماتریس Allocation

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

C — A

R1	R2	R3
9	3	4

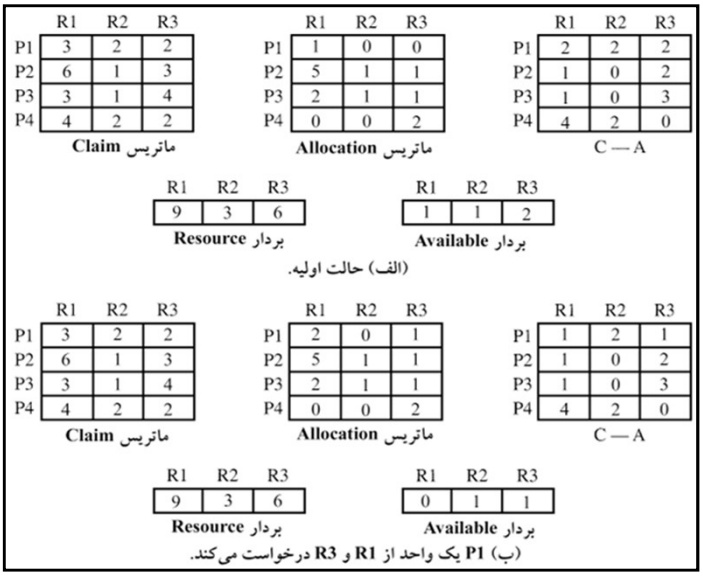
بردار Available

R1	R2	R3
9	3	6

بردار Resource

(د) P3 به‌طور کامل اجرا می‌شود.

عدم تخصیص منبع



149

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.

150

کشف بن‌بست

- راهبرد های کشف بن بست، دستیابی به منابع یا فعالیت های فرآیند را محدود نمی کنند.
- در کشف بن بست، منابع درخواستی در صورت امکان به فرآیندها تحویل می شوند.
- سیستم عامل متناوباً الگوریتمی را اجرا می کند که انتظار حلقوی (شرط ۴ از شرایط مذکور) را تشخیص می دهد.

151

کشف بن‌بست

الگوریتم کشف بن بست

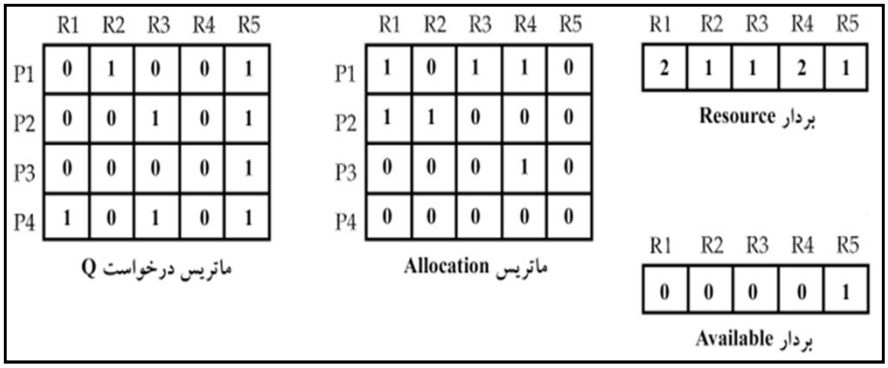
- از ماتریس Allocation و بردار Available استفاده می شود.
- علاوه براین، ماتریس درخواست Qطوری تعریف می شود که Q_{ij} مبین مقدار درخواست فرایند i از منبع j می باشد.
- این الگوریتم، فرآیندهایی را که در بن بست نیستند علامت گذاری می کند. در آغاز، تمام فرآیندها بدون علامت هستند. سپس مراحل زیر انجام می شود:

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 فعلاً علامت دار نیست و سطر iام Q کمتر یا مساوی W است. یعنی، برای $1\leq k\leq m$ ، داشته باشیم $Q_{ik}\leq W_k$. اگر چنین سطری پیدا نشد، الگوریتم را خاتمه بده.
4. اگر چنین سطری 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.

152

کشف بن‌بست

- 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

R1	R2	R3	R4	R5
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 کنترلیِ از قبل تعریف شده، و شروع مجدد آن ها.
3. قطع پی درپی فرآیندهای بن بست تا حدی که دیگر بن بست وجود نداشته باشد.
4. قبضه کردن پی درپی 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 ویژگی های 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.

154

بن‌بست

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

روش	خط مشی تخصیص منبع	طرح‌های مختلف	مزایای عمده	معایب عمده
پیشگیری	محافظه‌کاری؛ درگیر کردن کمتر منابع	درخواست یکباره منابع	➢ برای فرآیندهایی مناسب است که فعالیت انبوهی را انجام می‌دهند. ➢ نیاز به قبضه کردن نیست.	➢ ناکارآمدی ➢ تأخیر شروع به کار فرایند ➢ اطلاع از منابع ضروری بعدی
		قبضه کردن	➢ سهولت به کارگیری در منابعی که ذخیره و بازیابی حالت آن‌ها آسان است.	➢ قبضه کردن بیش از حد لازم ➢ در معرض شروع مجدد حلقوی
		مرتب کردن منابع	➢ در زمان ترجمه قابل بررسی است. ➢ عدم نیاز به محاسبه زمان اجرا، زیرا در طراحی سیستم حل شده است.	➢ قبضه کردن نه چندان موثر ➢ مجازنداشتن درخواست افزایشی منابع
اجتناب	بین پیشگیری و کشف	دستکاری برای یافتن حداقل یک مسیر امن	➢ عدم نیاز به قبضه کردن	➢ اطلاع از منابع ضروری بعدی ➢ امکان مسدودشدن طولانی فرایندها
کشف	بسیار آزاد است، به‌طوری‌که منابع درخواستی در صورت امکان تخصیص می‌یابند	بررسی مداوم وجود بن‌بست	➢ عدم تأخیر در شروع به کار فرآیند. ➢ تسهیل اداره کردن online.	➢ ضررهای قبضه کردن

155

فصل هفتم

مدیریت حافظه

156

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.

157

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.

158

بخش بندی حافظه

بخش بندی ثابت

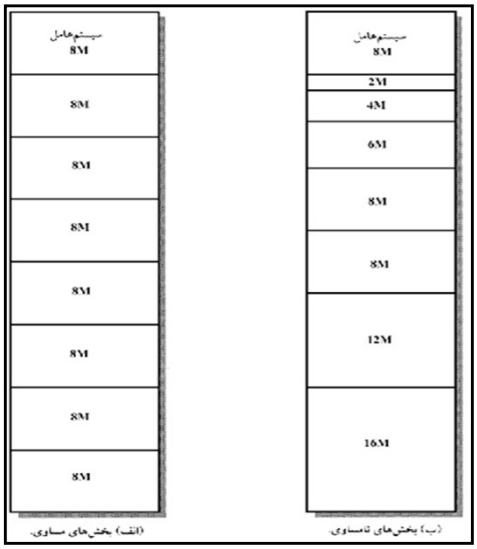
- در اغلب طرح های مدیریت حافظه، می توانیم فرض کنیم که سیستم عامل بخش ثابتی از حافظه اصلی را اشغال می کند و بقیه حافظه اصلی برای استفاده فرآیندها مهیا است.
- ساده ترین طرح مدیریت بر این حافظه موجود، بخش بندی آن به ناحیه هایی با مرزهای ثابت است.

اندازه بخشها

- 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

159

بخش بندی حافظه



مثالی از بخش بندی ثابت در حافظه 64M.

160

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.

161

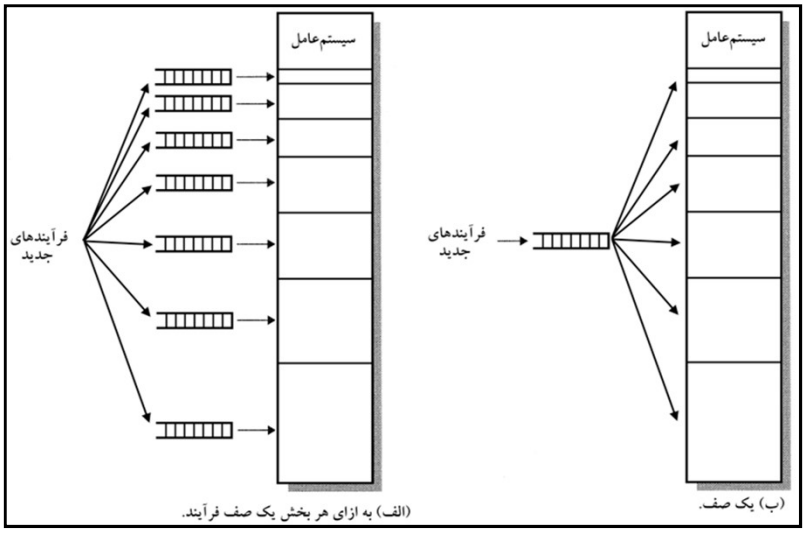
بخش بندی حافظه

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 صف زمان بندی است تا فرآیندهایی که از آن بخش به خارج مبادله شدند درآن 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، فرآیند ی با اندازه بین ۱۲و ۱۶مگابایت وجود ندارد.در این حالت،از بخش ۱۶مگابایتی استفاده نمی شود، گرچه فرآیند کوچک تری می توانست به آن نخصیص یابد.
  - روش مناسب تر این است که برای تمام فرآیندها فقط یک صف در نظر گرفته شود.

162

بخش بندی حافظه



تخصیص حافظه برای بخش بندی ثابت.

163

بخش بندی حافظه

- مزایای بخش بندی ایستا
  - به حداقل نرم افزار سیستم عامل و سربار 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 بخشها در زمان ایجاد سیستم انتخاب شده اند، کارهای کوچک نمی توانند با کارایی از فضای بخش استفاده کنند.

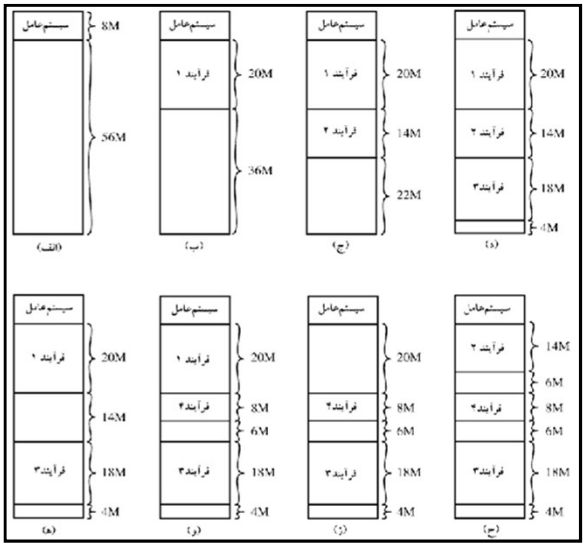
164

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 fragmentetion) گویند.
  - یک تکنیک غلبه بر تکه تکه شدن خارجی، **فشرده سازی** است.

165

بخش بندی پویا



اثر بخش بندی پویا.

166

بخش بندی پویا

الگوریتم جاگذاری

- چون فشرده سازی حافظه وقت گیر است. طراح سیستم عامل باید در تخصیص حافظه به فرآیندها (چگونگی پرکردن حفره ها) هوشمندانه عمل کند.
- اگر در هنگام بارکردن 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)

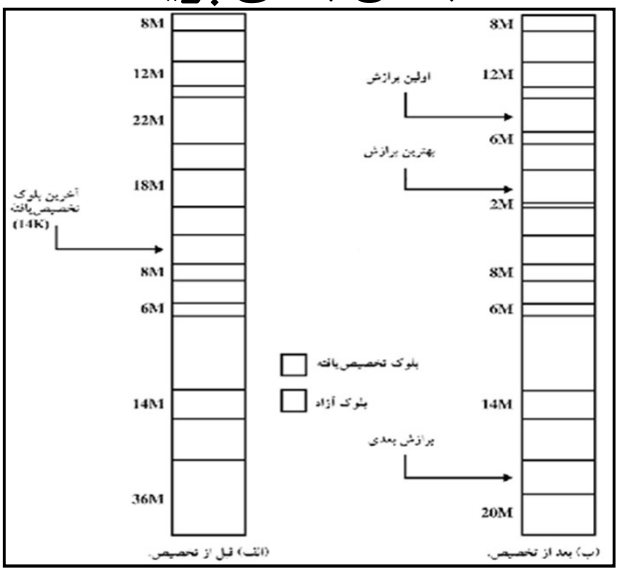
167

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 بر می 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.

168

بخش بندی پویا



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

169

بخش بندی پویا

الگوریتم جایگزینی

- در سیستم چند برنامه ای که از بخش بندی پویا استفاده می شود، زمانی فرا می رسد 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 می کند 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.

170

سیستم رفاقتی (buddy system)

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

171

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. وگرنه، 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 ازدو رفیق تخصیص می یابد. درغیر این صورت، یکی از رفیق ها دوباره تقسیم می شود.
- این روند ادامه می یابد تا کوچک ترین بلوکِ 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 $i + 1$ 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 $(i + 1)$ ترکیب می شوند.

172

سیستم رفاقتی

- با تقاضایی به اندازه K که $2^{i-1}<K\leq 2^i$ 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>;
}
```

173

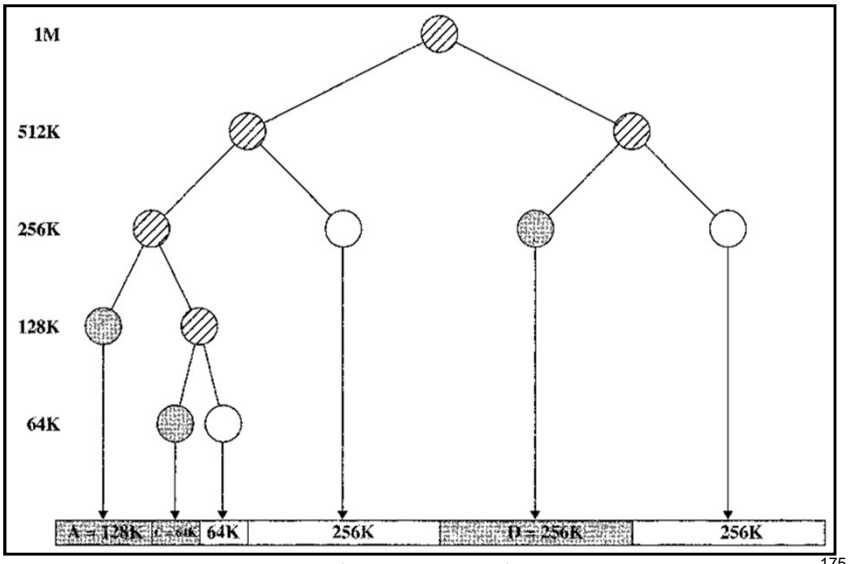
سیستم رفاقتی

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

نمونه ای از سیستم رفاقتی.

174

سیستم رفاقتی



نمایش درختی سیستم رفاقتی.

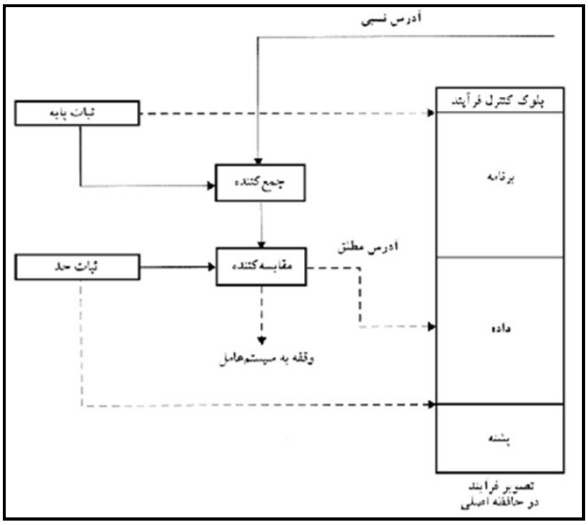
175

جابه 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 محتویات ثبات دستورالعمل، آدرس های ناشی از دستورالعمل های انشعاب و فراخوانی، و آدرس داده ها در دستورالعمل های ذخیره و بارکردن است.

176

جابه جایی



پشتیبانی سخت افزار برای جابه جایی.

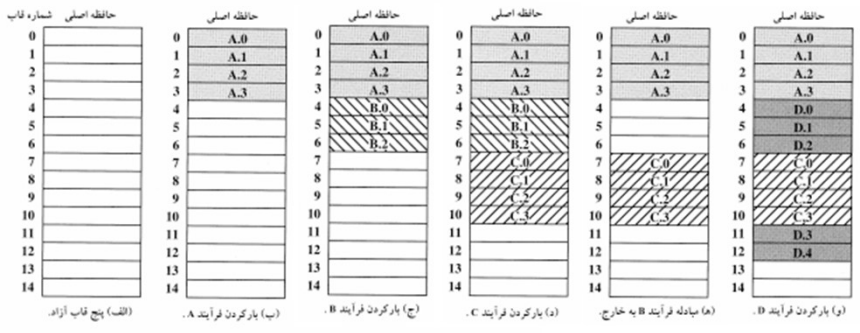
177

صفحه بندی (Paging)

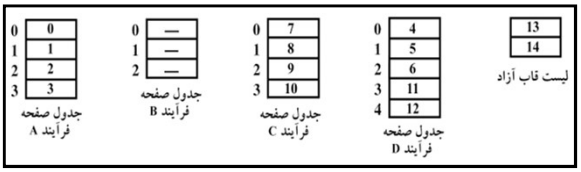
- حافظه اصلی به بخشهای نسبتا کوچک هم اندازه تقسیم می شود.
- هر فرآیند به تکه های هم اندازه با آنها تقسیم می شود.
- به تکه های هر فرآیند صفحه (Page) می گویند.
- به تکه های موجود در حافظه قاب (Frame) گویند.
- اتلاف حافظه برای هر فرآیند به خاطر تکه تکه شدن داخلی است که فقط در آخرین صفحه آن فرآیند به وجود می آید.
- تکه تکه شدن خارجی وجود ندارد.
- یک ثبات پایه کافی نیست.
- سیستم عامل یک جدول صفحه (Page Table) برای هر فرآیند ایجاد می کند.
- تبدیل آدرس منطقی به فیزیکی توسط سخت افزار انجام می شود.

178

صفحه بندی



تخصیص صفحات فرآیند به قاب های آزاد.



ساختمان داده های مربوط به شکل بالا

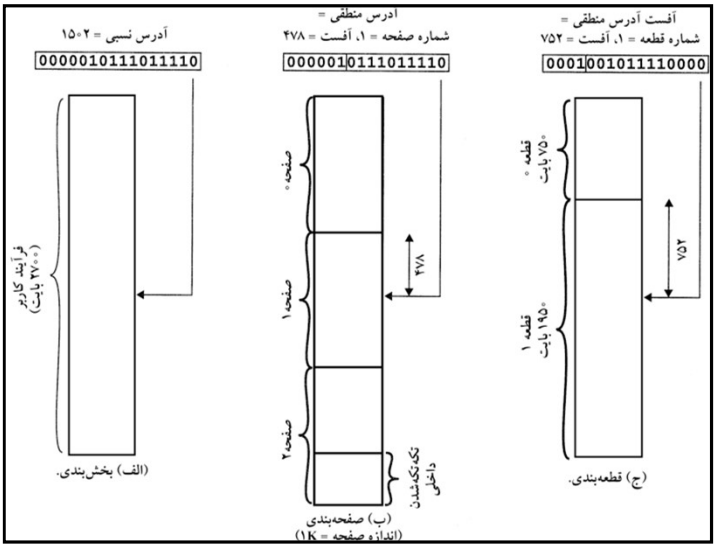
179

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 $k\times2^m$ و آدرس فیزیکیِ 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.

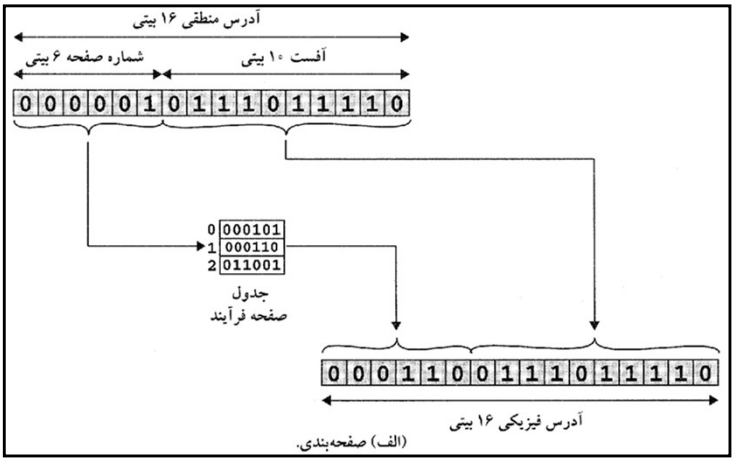
180

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.
  - گر چه 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.

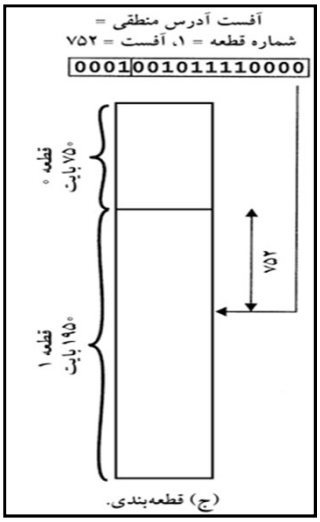
183

قطعه بندی

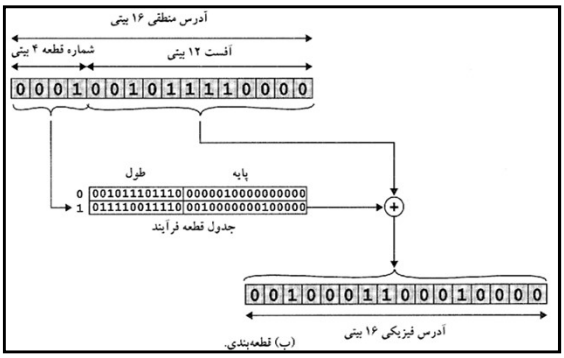
- آدرس $n + m$ بیتی را درنظر بگیرید که n بیت سمت چپ شماره قطعه و mبیت سمت راست آفست است.
- برای ترجمه آدرس مراحل زیر باید انجام گیرند:
  - استخراج n بیتی سمت چپ آدرس منطقی به عنوان شماره قطعه.
  - استفاده از شماره قطعه به عنوان شاخصی در جدول قطعه برای یافتن آدرس شروع قطعه.
  - مقایسه آفست (m بیت سمت 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.

184

قطعه بندی



آدرس منطقی.



ترجمه آدرس منطقی به فیزیکی.

185

فصل هشتم

حافظه مجازی

(Virtual Memory)

186

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 کنند.
- 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.

187

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.

188

حافظه مجازی

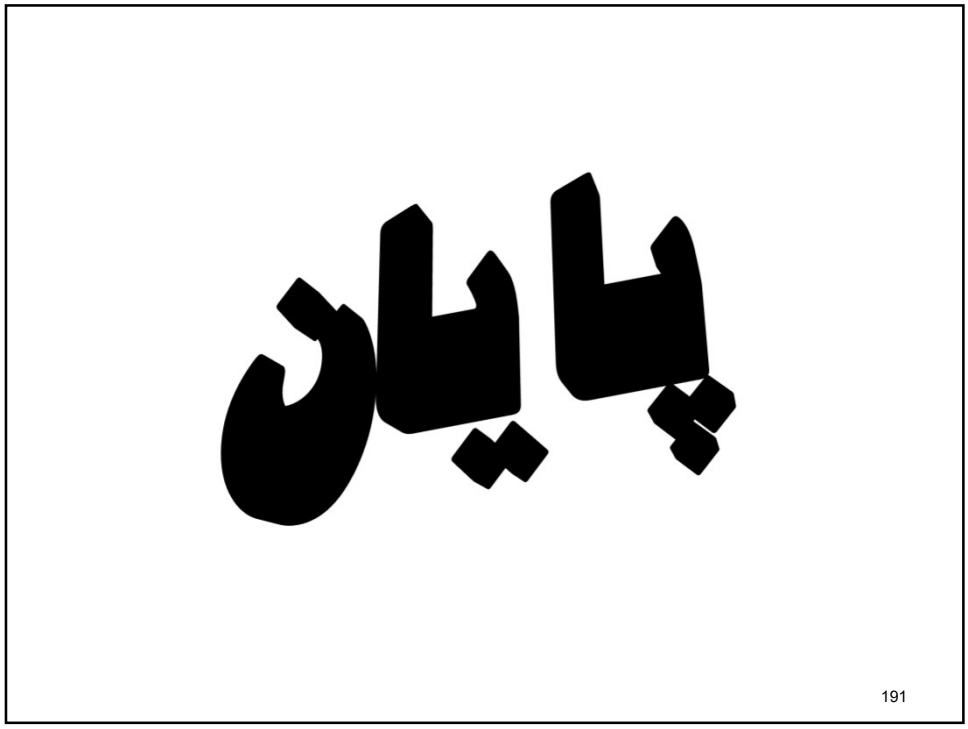
- کوبیدگی (thrashing)
  - اگر سیستم عامل تکه ای را به داخل بیاورد باید تکه دیگری را خارج کند.
  - اگر تکه ای را درست قبل از اینکه اجرا شود خارج کند، به فاصله خیلی کم دوباره مجبور است آن تکه را به داخل آورد.
  - تکرار زیاد این عمل منجر به پدیده ای به نام کوبیدگی می شود.
  - در این حالت پردازنده به جای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

189

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.

190


پایان
191