
بسم الله الرحمن الرحیم

مؤسسه آموزش عالی بهار
غیرانتفاعی - غیردولتی

سیستم عامل

تهیه کننده: فرزاد زندی

zandi_farzad@yahoo.com

zandi8farzad@gmail.com

آنچه در این درس فرا می‌گیرید:

۱- نگاه کلی به سیستم‌عامل
۲- توصیف و مدیریت فرآیندها
۳- نخ یا ریسمانها
۴- زمان‌بندی و همگام‌سازی فرآیندها
۵- مدیریت حافظه
۶- ساختار حافظه
۷- مدیریت ورودی خروجی
۸- محافظه و امنیت

آنچه از شما انتظار می‌رود:

۱- درک مفاهیم سیستم‌عامل
۲- درک مفاهیم فرآیندها
۳- شناخت زمان‌بندی و همگام‌سازی
۴- آشنایی با ساختار حافظه و درک نحوه مدیریت آن
۵- آشنایی با مدیریت ورودی خروجی
۶- ارائه در کلاس درباره یکی از موضوعات مطرح شده

نحوه ارزیابی نمره:

۱۰ نمره
- ۱- حضور مستمر
- ۲- فعالیت کلاسی
- ۳- انجام تمرینات محوّل شده
- ۴- ارائه در کلاس

۵- امتحان میان ترم ۵ نمره

۶- امتحان پایان ترم ۵ نمره

جلسه دوم:

نگاه کلّی به سیستم‌عامل

آنچه در این جلسه می خوانید:

۱- تعریف سیستم‌عامل
۲- اهداف سیستم‌عامل
۳- وظایف سیستم‌عامل
۴- تکامل سیستم‌عامل
۵- سیستم‌های پردازش ردیفی
۶- سیستم‌های دسته‌ای ساده
۷- سیستم‌های دسته‌ای چندبرنامه‌ای
۸- سیستم‌های اشتراک زمانی
۹- سیستم‌های توزیع شده
۱۰- سیستم‌های چندپردازنده‌ای
۱۱- سیستم‌های بلادرنگ

تعریف سیستم‌عامل:

سیستم‌عامل برنامه‌ای است که اجرای برنامه‌های کاربردی را کنترل کرده و به عنوان واسط بین برنامه‌های کاربردی و سخت‌افزار کامپیوتر عمل می‌کند.

اهداف سیستم‌عامل:

- **سهولت:**

استفاده آسان از کامپیوتر

- **کارآمدی:**

استفاده مفید از منابع سیستم کامپیوتری.

- **قابلیت تکامل:**

سیستم‌عامل طوری باید ساخته شود که به طور مؤثر توسعه, آزمون و معرفی وظایف جدید بدون تداخل با خدمات سیستم را امکان پذیر سازد.

وظایف سیستم‌عامل برای سهولت استفاده:

- توسعهٔ برنامه
- اجرای برنامه
- دسترسی به دستگاههای ورودی خروجی
- دسترسی کنترل شده به فایل
- دسترسی به سیستم
- کشف و پاسخگویی به خطاها
- حسابداری (جمع آوری آمارهای بهره‌برداری از منابع مختلف)
- معماری مجموعه دستورالعملها (تعریف و اجرای دستورالعملهای زبان ماشین)
- ...

وظایف سیستم‌عامل به عنوان مدیر منابع:

- مدیریت و کنترل ذخیره‌سازی, انتقال و پردازش داده‌ها

وظایف سیستم‌عامل در سهولت تکامل:

- **ارتقاء و انواع جدید سخت‌افزار:**

شناخت قطعات جدید

- **خدمات جدید:**

ارائه ابزارهای کنترل و اندازه‌گیری جدید

- **رفع خطاها:**

رفع خطا به مرور زمان

تکامل سیستم‌عامل:

۱- پردازش ردیفی
۲- سیستم‌های دسته‌ای ساده
۳- سیستم‌های دسته‌ای چندبرنامه‌ای
۴- سیستم‌های اشتراک زمانی
۵- سیستم‌های توزیع‌شده
۶- سیستم‌های چندپردازنده‌ای
۷- سیستم‌های بلادرنگ

سیستم‌های پردازش ردیفی:

اواخر دهۀ ۱۹۴۰ تا اواسط دهۀ ۱۹۵۰ سیستم‌عاملی وجود نداشت.
برنامه‌ساز مستقیماً با سخت‌افزار در ارتباط بود.
کاربران به نوبت به کامپیوتر دسترسی داشتند.
کامپیوترها شامل یک کنسول (میز فرمان) دارای چراغهای نمایش, کلیدها, نوعی دستگاه ورودی (کارت خوان) و چاپگر بودند.

مشکلات عمده این نوع سیستم‌ها:

- **زمان‌بندی:**

به انجام نرسیدن یا زودتر تمام شدن کارِ کاربر

- **زمان‌نصب:**

بوجود آمدن خطا در یکی از مراحل تولید برنامه (هنگام پانچ کارت)

سیستم‌های دسته‌ای ساده:

کامپیوترهای اولیه بسیار گران بودند و در نتیجه استفاده حداکثر از پردازنده بسیار مهم و اتلاف وقت بخاطر زمان‌بندی و زمان‌نصب قابل قبول نبود.

برای افزایش استفاده از پردازنده, مفهوم سیستم‌عامل مطرح شد.

در این نوع سیستم‌ها (پردازش ردیفی) کاربر مستقیماً به پردازنده دسترسی نداشت و کار را بر روی کارت یا نوار به اپراتور تحویل می‌داد.

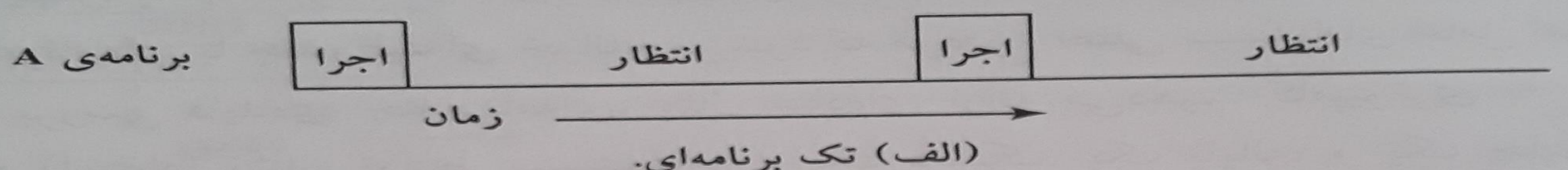
اپراتور کارها را به صورت ردیفی دسته‌بندی کرده و کل دسته را روی یک دستگاه ورودی قرار می‌داد.

زمان‌بندی کارها بر عهده نرم‌افزاری به نام ناظر (همان سیستم‌عامل) بود.

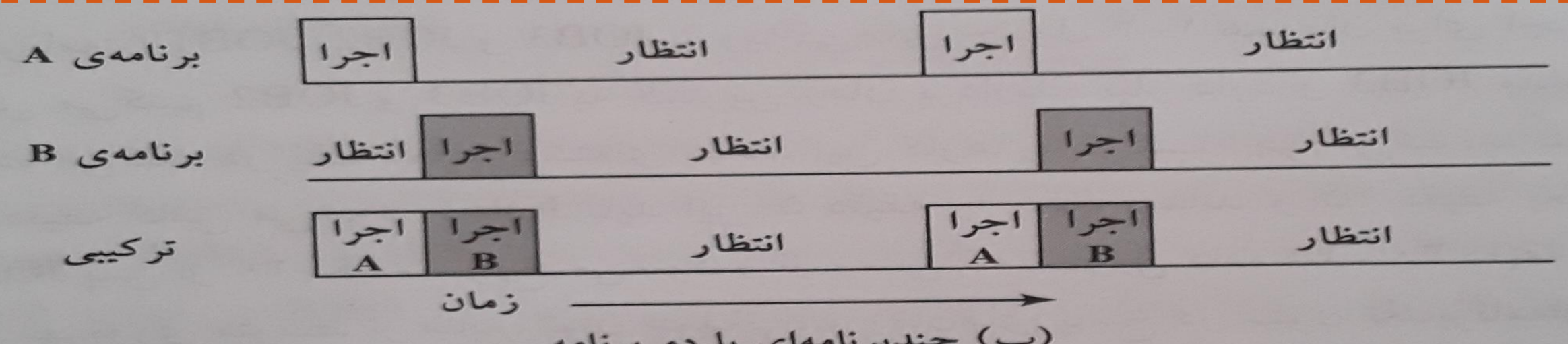
سیستم‌های دسته‌ای چندبرنامه‌ای:

در سیستم‌های دسته‌ای ساده پردازنده غالباً بیکار می‌ماند و در آنها کامپیوتر بیش از ۹۶ درصد از وقت خود را صرف انتظار دستگاه‌های ورودی خروجی می‌کرد تا انتقال داده‌ها به فایل یا از فایل کامل شود.

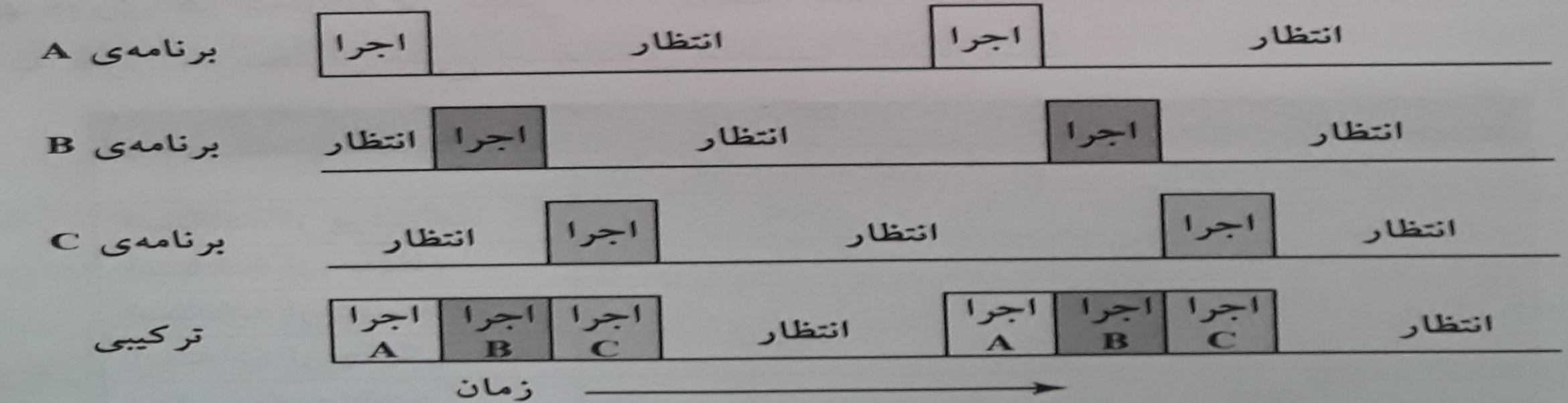
در سیستم‌های دسته‌ای چندبرنامه‌ای, چند برنامه باهم وارد حافظه می‌شود و هنگامی که یک برنامه منتظر ورودی خروجی است برنامه دیگر می‌تواند اجرا شود. این رویکرد چندبرنامه‌ای یا چندوظیفه‌ای خوانده می‌شود.



(الف) تک برنامه‌ای.



(ب) چندبرنامه‌ای با دو برنامه.



سیستم‌های اشتراک زمانی:

در سیستم‌های اشتراک زمانی, همزمان چندین کاربر از طریق پایانه‌ها به سیستم دسترسی دارند. بنابراین اگر n کاربر فعال, همزمان درخواست خدمت داشته باشند, هر کاربر هر کاربر بطور متوسط فقط $\frac{1}{n}$ از ظرفیت مؤثر کامپیوتر را در اختیار می‌گیرد.

در این سیستم‌ها علاوه بر ورودی خروجی (I/O), پردازنده (CPU) نیز بین برنامه‌های مختلف سوییچ می‌شد.

CPU بعد از مدّت زمان معیّنی از برنامهٔ خاصی گرفته و به برنامهٔ دیگر تحویل می‌شد و بدیت ترتیب برنامه‌ها به صورت موازی اجرا می‌شدند.

سیستم‌های توزیع شده:

در این سیستم‌ها قسمتهای مختلف برنامهٔ کاربران بدون آنکه آنها متوجه شوند همزمان در چند کامپیوتر مجزا که ساختار شبکه‌ای دارند اجرا و نتیجه به کامپیوتر اصلی برمی‌گردد.

سیستم‌های چندپردازنده‌ای:

در این سیستم‌ها کامپیوتر دارای چندین CPU است و سیستم‌عامل روی آنها پیاده می‌شود. بنابراین چند قسمت یک برنامه که امکان اجرای موازی دارند می‌توانند روی این سیستم‌ها اجرا شوند.

سیستم‌های بلادرنگ (سخت , نرم):

- **سیستم بلادرنگ سخت:**

در این سیستم‌ها زمان پاسخ باید سریع و تضمین شده باشد, تأخیر توجیه‌پذیر نیست و خسارت بجای می‌گذارد.

سیستم‌عامل پیشرفته نیست و از ROM استفاده می‌شود چرا که خواندن اطلاعات توسط CPU از RAM سرعت را کم می‌نماید.

- **سیستم بلادرنگ نرم:**

در این سیستم‌ها یک وظیفه بی‌درنگ بحرانی نسبت به وظایف دیگر اولویت دارد و تا پایان تکمیل شدن کار آن حفظ می‌شود.

جلسه سوم:

نگاه کلّی به سیستم‌عامل

آنچه در این جلسه می خوانید:

۱- دستاوردهای اصلی
۲- معماری داخلی ویندوز
۳- مؤلفه‌های هسته لینوکس
۴- تکامل سیستم‌عامل
۵- معماری سیستم‌عامل اندروید

دستاوردهای اصلی:

سیستم‌عامل‌ها, پیچیده‌ترین نرم‌افزارهایی هستند که تاکنون نوشته شده‌اند. این پیچیدگی باعث تلاش برای برآورده کردن سه هدف مهم سهولت, کارایی و امکان تکامل شد.

در توسعه سیستم‌عامل‌ها چهار دستاورد مطرح است که بسیاری از نکات طراحی و پیاده‌سازی سیستم‌عامل‌ها جدید را تعیین می‌کند:

- **فرآیندها**

احتمال همگام‌سازی نامناسب, احتمال شکست در انحصار متقابل, احتمال عملکرد غیرقطعی برنامه, احتمال بن‌بست.

- **مدیریت حافظه**

جداسازی فرآیندها, تخصیص و مدیریت خودکار, حفاظت و کنترل دسترسی, حافظه درازمدّت.

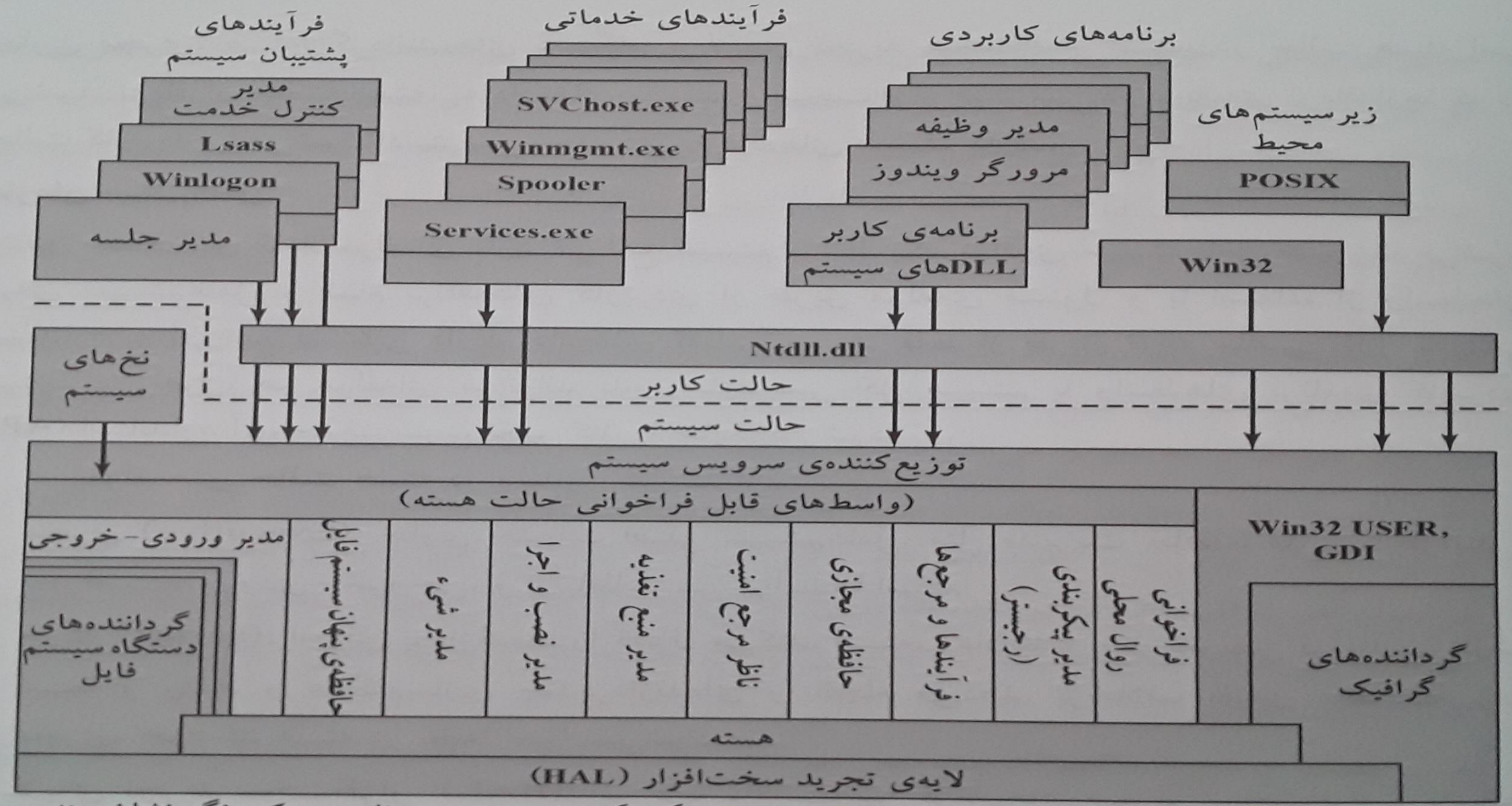
- **امنیت و حفاظت اطلاعات:**

دسترس‌پذیری, محرمانگی, جامعیت داده‌ها, اعتبار.

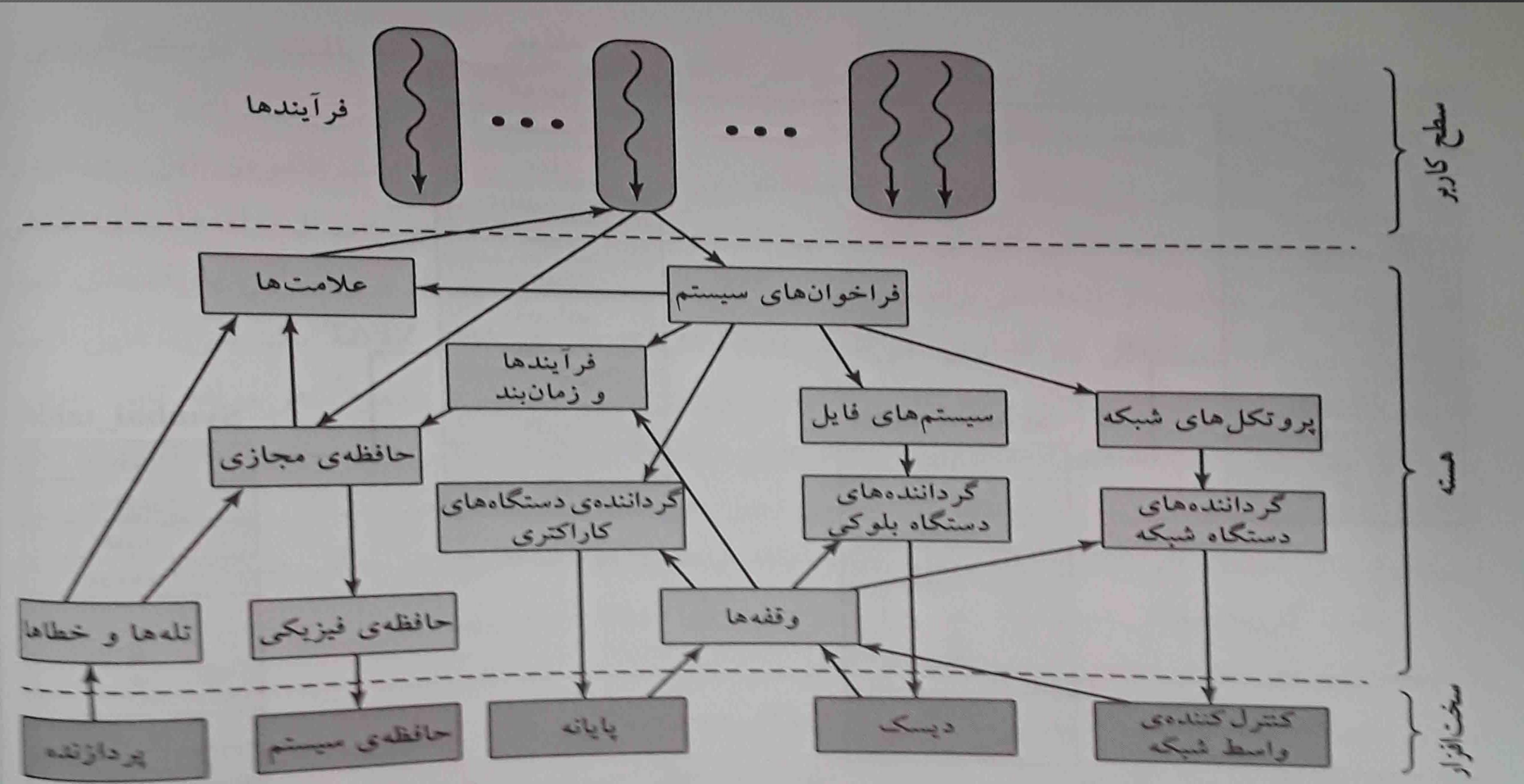
- **زمان‌بندی و مدیریت منابع:**

انصاف, حساسیت در مقابل تفاوت‌ها, کارایی.

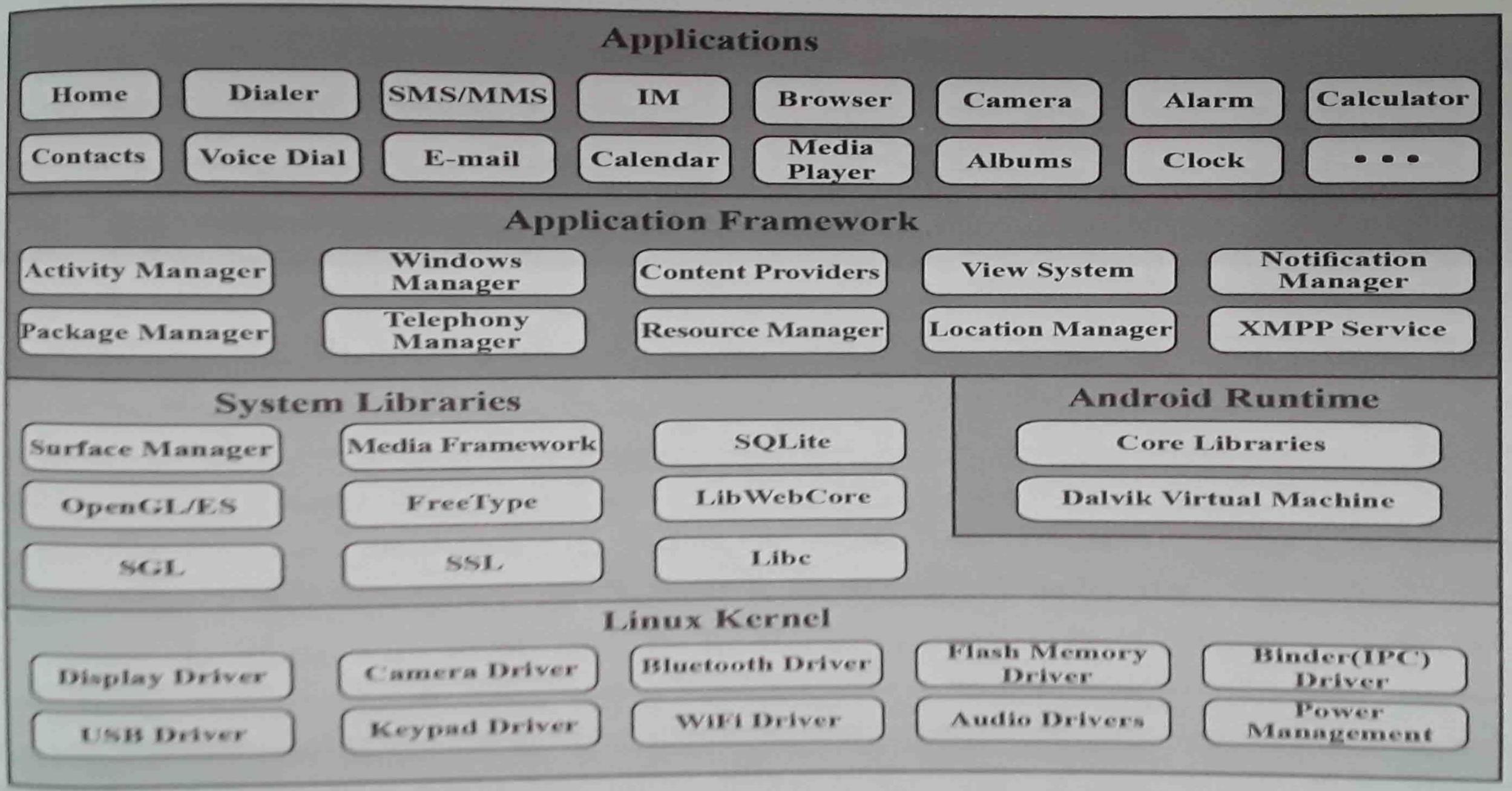
معماری داخی ویندوز:



مؤلفه‌های هسته لینوکس:



معماری سیستم‌عامل اندروید:



جلسه چهارم:

توصیف و کنترل فرآیند

آنچه در این جلسه می خوانید:

۱- فرآیند چیست؟
۲- مشخصات فرآیند
۳- مدل‌های فرآیند
۴- مدل دو حالته برای فرآیند
۵- مدل پنج حالته برای فرآیند

تعریف فرآیند:

- فرآیند برنامهٔ در حال اجرا بر روی یک کامپیوتر و یا موجودیتی که می‌تواند به پردازنده تخصیص داده شود تا اجرا شود, است.
- فرآیند یک واحد فعالیت است که با اجرای دنباله‌ای از دستورالعمل‌ها, حالت جاری و مجموعه‌ای از منابع سیستم مشخص می‌شود.

نحوه ایجاد فرآیند:

چهار رویداد متداول زیر منجر به ایجاد فرآیند می‌شود:

- کار دسته‌ای جدید
- برقراری ارتباط محاوره‌ای
- ایجاد فرآیند توسط سیستم‌عامل برای ارائه خدمت
- زایش توسط فرآیند موجود

نحوه پایان فرآیند:

رویدادهای زیر منجر به پایان فرآیند می‌شوند:

- پایان طبیعی
- سقف زمانی
- کمبود حافظه
- تجاوز از حدود
- خطای حفاظت
- خطای محاسباتی
- گذشت زمان
- خرابی ورودی خروجی
- دستورالعمل نامعتبر
- دستورالعمل ممتاز
- استفادهٔ نادرست از داده‌ها
- دخالت اپراتور یا سیستم عامل
- پایان یافتن پدر
- به درخواست پدر

مشخصات فرآیند:

در هر لحظه, در حالیکه برنامه در حال اجرا است, این فرآیند می‌تواند توسط تعدادی از عناصر, از جمله عناصر زیر بطور یکتا مشخص شود:

- **شناسه:**

هر فرآیند یک شناسه یکتای مربوط به خود دارد.

- **حالت:**

حالت اجرای فرآیند را نشان می‌دهد.

- **اولویت:**

سطح اولویت نسبت به دیگر فرآیندها است.

- **شمارندهٔ برنامه:**

آدرس دستورالعمل بعدی این برنامه که باید اجرا شود.

- **داده‌های متن:**

داده‌هایی که در اَثنای اجرای فرآیند, در ثبات‌های پردازنده وجود دارند.

- **اشاره‌گرهای حافظه:**

حاوی اشاره‌گرهایی به کُد برنامه و داده‌های مربوط به این فرآیند, بعلاوه بلوک‌های حافظهٔ مشترک به دیگر فرآیندها است.

- **اطلاعات وضعیت ورودی خروجی:**

حاوی درخواست‌های معوّق ورودی خروجی, دستگاه‌های ورودی خروجی تخصیص داده شده به فرآیند و لیستی از فایل‌های در حال استفاده توسط فرآیند است.

- **اطلاعات حسابداری:**

حاوی زمان‌های صرف شده ساعت و پردازنده, محدودیت‌های زمانی و غیره است.

بلوک کنترل فرآیند

شناسه

حالت

اولویت

شمارندهٔ برنامه

اشاره‌گرهای حافظه

داده‌های متن

اطلاعات وضعیت ورودی خروجی

اطلاعات حسابداری

مدل‌های فرآیند:

برای اجرای یک برنامه, یک فرآیند یا وظیفه برای آن ایجاد می‌شود. مسئولیت اصلی سیستم عامل کنترل اجرای این فرآیندها است, که شامل تعیین الگوی اجرای آنها در بین یکدیگر و تخصیص منابع به آنها می‌شود.

مدل‌های مختلفی برای کنترل اجرای فرآیندها وجود دارد:

- مدل دو حالته برای فرآیند
- مدل پنج حالته برای فرآیند
- ...

مدل دو حالته برای فرآیند:

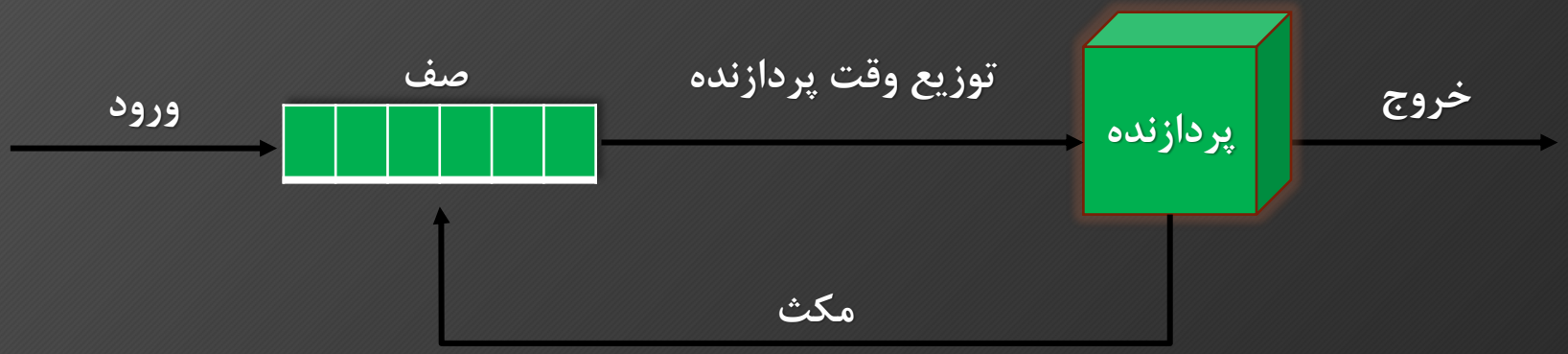
ساده‌ترین مدلی که مطرح است این است که در هر لحظه یک فرآیند توسط پردازنده در حال اجرا هست یا نیست.

در این مدل, هر فرآیند می‌تواند در یکی از دو حالت **اجرا** و یا **غیرِ اجرا** باشد. وقتی سیستم عامل فرآیند جدیدی را ایجاد می‌کند, یک بلوک کنترل فرآیند را برای آن ایجاد کرده و آن فرآیند را در حالت غیر اجرا وارد سیستم می‌کند. فرآیند وجود دارد, برای سیستم‌عامل شناخته شده است و منتظر فرصتی برای اجرا شدن است.

گاهی فرآیند در حال اجرا دچار وقفه می‌شود و بخش توزیع‌کنندهٔ سیستم عامل فرآیند دیگری را برای اجرا شدن انتَخاب می‌کند و فرآیند قبلی از حالت اجرا به حالت غیر اجرا می‌رود و فرآیند دیگر به حالت اجرا می‌رود.

این رفتار توزیع‌کننده را می‌توان صف‌بندی کرد.

مدل دو حالته برای فرآیند:



مدل پنج حالته برای فرآیند:

در این روش حالت غیرِ اجرا به دو حالتِ آماده و مسدود تقسیم می‌شود.
حالت‌های مدل پنج حالته بصورت زیر می‌باشد:

- **اجرا:**

فرآیندی که هم اکنون در حال اجرا است.

- **آماده:**

فرآیندی که وقتی به آن فرصت داده شود, آمادهٔ اجرا است.

- **مسدود/انتظار:**

فرآیند نمی‌تواند اجرا شود تا اینکه رویدادی مثل تکمیل‌شدن یک عمل ورودی خروجی رخ دهد.

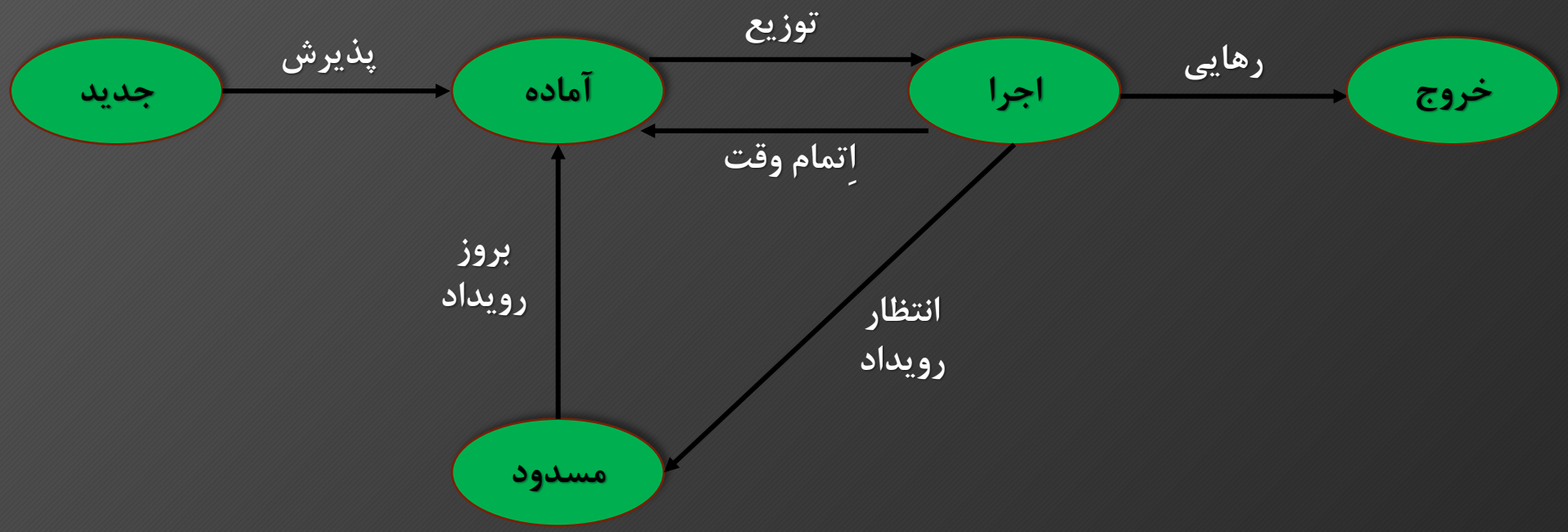
- **جدید:**

فرآیندی که تازه ایجاد شده و هنوز به حافظهٔ اصلی بار نشده است.

- **خروج:**

فرآیندی که توسط سیستم‌عامل از مخزن فرآیندهای قابل اجرا, به دلیل توقف یا قطع اجرای آن, متوقف شده است.

مدل پنج حالته برای فرآیند:



جلسه پنجم:

آشنایی با نَخ‌ها

آنچه در این جلسه می خوانید:

۱- تمایز بین فرآیند و نَخ
۲- چندنَخی
۳- مدل‌های فرآیند تک‌تخی و چندنَخی

تمایز بین فرآیند و نَخ:

مفهوم فرآیند از آنچه که تاکنون مطرح شده پیچیده‌تر و ظریف‌تر است و در واقع دو مفهوم مجزا و مستقل را شامل می‌شود: یکی مفهوم مرتبط با مالکیت منبع و دیگری مرتبط با اجرا.

- مالکیت منبع:

در بلوک کنترل فرآیند مجموعه‌ای از برنامه‌ها, داده‌ها, پشته و صفات تعریف شده است. گاهی ممکن است کنترل یا مالکیت منبع, مثل حافظهٔ اصلی, کانال‌های ورودی خروجی, دستگاه‌های ورودی خروجی و فایل‌ها به فرآیند داده شود. در این هنگام سیستم‌عامل عملیاتی حفاظتی, جهت جلوگیری از تداخل ناخواستهٔ بین فرآیندها برای بدست گرفتن منابع انجام می‌دهد.

- زمان‌بندی و اجرا:

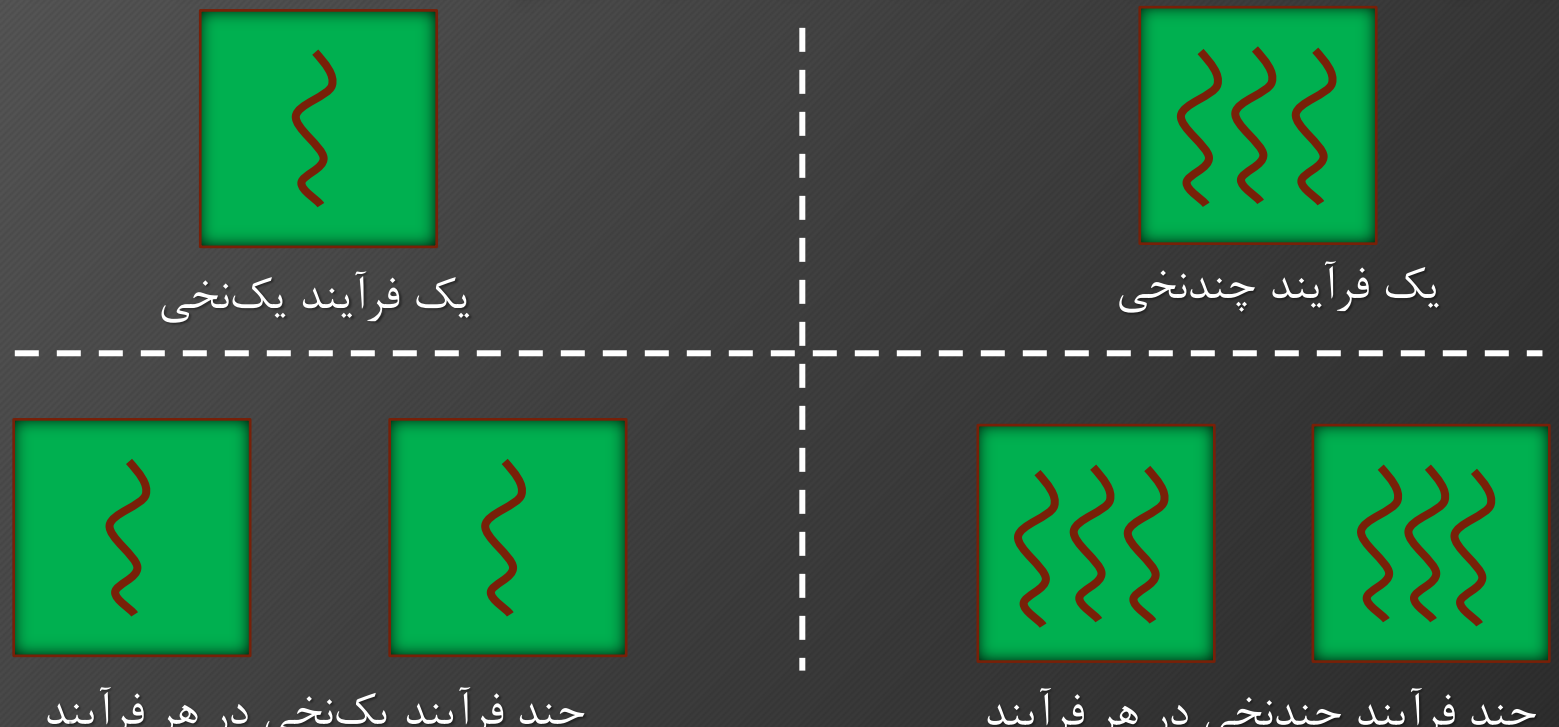
اجرای فرآیند, یک مسیر اجرایی را از طریق یک یا چند برنامه طی می‌کند. ممکن است این اجرا در بین فرآیندهای دیگر قرار گیرد. لذا فرآیند دارای یک حالت اجرا (اجرا, آماده و غیره) و اولویت توزیع وقت پردازنده است که باید توسط سیستم‌عامل زمان‌بندی و وقت پردازنده به آن اختصاص یابد.

واحد توزیع کننده را معمولاً نَخ یا فرآیند سبک‌وزن می‌گویند. واحد مالکیت منبع را معمولاً فرآیند یا وظیفه می‌گویند.

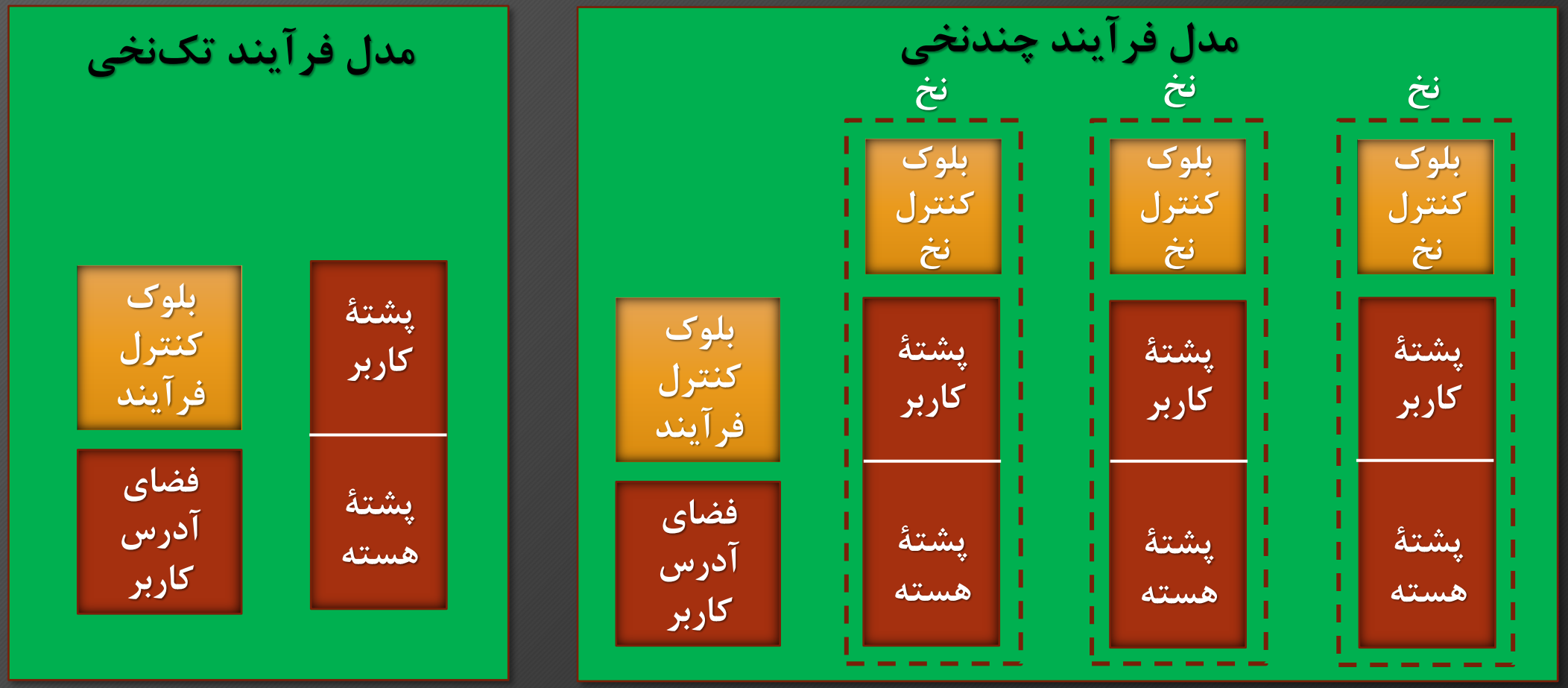
چندنخی:

یک نخ اجرا در هر فرآیند, تک‌نخی نامیده می‌شود.

چند نخی به توانایی سیستم‌عامل در حمایت از چندین مسیر اجراییِ همزمان در داخل یک فرآیند گفته می‌شود.



مدل‌های فرآیند تک‌نخی و چندنَخی:



جلسه ششم:

همروندی:
انحصار متقابل و همگام‌سازی

آنچه در این جلسه می خوانید:

۱- همروندی چیست؟

۲- اصول همروندی

۳- ملاحظات سیستم‌عامل در همروندی

۴- محاورهٔ فرآیندها با یکدیگر

۵- رقابت فرآیندها برای منابع

همروندی چیست؟

در هنگام طراحی سیستم عامل , موضوعات مهمی وجود دارد که همگی به مدیریت فرآیندها و نخ‌ها مربوط می شوند. این موضوعات شامل موارد زیر می باشند:

- چندبرنامه ای: مدیریت فرآیندهای متعددِ داخل یک سیستم تک پردازنده ای.
- چندپردازشی: مدیریت فرآیندهای متعددِ داخل یک سیستم چند پردازنده ای.
- پردازش توزیع شده: مدیریت فرآیندهای متعددی که در چندین سیستم کامپیوتری اجرا می شوند.

همروندی, ارتباط بین فرآیندها , اشتراک و رقابت منابع (حافظه , فایل ها و دسترسی به ورودی-خروجی), همگام سازی فعالیتهای فرآیندهای متعدد و تخصیص زمان پردازنده به فرآیندها را در برمی گیرد.

همروندی چیست؟

همروندی در سه زمینه مطرح می شود:

- برنامه های کاربردی متعدد:

چندبرنامه ای به خاطر این ابداع شد که زمان پردازش , به طور پویا بین تعدادی از برنامه‌های کاربردی فعال , به اشتراک گذاشته شود.

- برنامه های کاربردی ساخ تیافته:

برای گسترش اصول طراحی مؤلفه ای و برنامه سازی ساخ تیافته , م یتوان بعضی از برنامه‌های کاربردی را به صورت مجموعه‌ای از فرآیندهای همروند , به طور کارآمد برنامه سازی کرد.

- ساختار سیستم عامل: اغلب سیستم عامل ها به صورت مجموعه ای از فرآیندها یا نخ ها پیاده سازی می شوند. بنابراین ساختار مشخص برای برنامه سازی سیستم نیز قابل اِعمال است.

اصول همروندی:

با توجه به اینکه سرعت اجرای یک فرآیند به فعالیت فرآیندهای دیگر , چگونگی اداره کردن وقفهها توسط سیستم‌عامل و سیاست‌های زمان بندی بستگی دارد , در سیستم‌های چندبرنامه‌ای , مسائل زیر بعلت عدم قابلیت پیش‌بینی سرعت اجرای فرآیندها مطرح می شوند:

- اشتراک منابع سراسری پرمخاطره است.
- مدیریت تخصیص بهینهٔ منابع برای سیستم عامل دشوار است.
- تعیین محل خطای برنامه سازی دشوار می شود.

یک مثال ساده:

روال زیر را در نظر بگیرید:

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

شرط مسابقه:
وضعیتی است که در آن, چندین نخ یا فرآیند یک دادهٔ مشترک را می‌خوانند و می‌نویسند. در این حالت نتیجهٔ نهایی به زمان‌بندی نسبی اجرای آنها بستگی دارد.

ملاحظات سیستم‌عامل در استفاده از همروندی:

استفاده از همروندی, نکات طراحی و پیاده‌سازی زیر را مطرح می سازد:

۱ - سیستم‌عامل باید قادر به نگهداری فرآیندهای مختلف باشد که این کار با استفاده از بلوکهای کنترل فرآیند انجام می‌شود.

۲ - سیستم عامل باید منابع مختلف (زمان پردازنده, حافظه, فایل ها و دستگا ه‌های ورودی-خروجی) را برای هر فرآیند فعال تخصیص دهد و پس بگیرد.

۳ - سیستم‌عامل باید داده‌ها و منابع فیزیکی هر فرآیند را در مقابل مداخلهٔ ناخواستهٔ فرآیندهای دیگر حفاظت کند.

۴ - عملکرد فرآیند و خروجی‌های آن باید مستقل از سرعت پیشرفت اجرای فرآیندهای همروند دیگر باشد.

محاورهٔ فرآیندها باهم:

برای پی‌بردن به چگونگی برخورد با موضوع استقلال فرآیندها, روشهای محاورهٔ فرآیندها باید بررسی گردد.

رو شهای محاوره فرآیندها را می‌توان بر اساس میزان آگاهی آنها از وجود یکدیگر, دسته بندی کرد:

- بی اطلاعی فرآیندها از یکدیگر
- اطلاع غیرمستقیم فرآیندها از یکدیگر
- اطلاع مستقیم فرآیندها از یکدیگر

رقابت فرآیندها برای منابع:

وقتی فرآیندهای همروند برای استفاده از منابع یکسان رقابت میکنند, با هم درگیر می شوند.

فرآیندهای رقیب هیچگونه اطلاعاتی را با هم مبادله نمیکنند. با این حال ممکن است اجرای یک فرآیند بر رفتار فرآیندهای رقیب تأثیر بگذارد.

در مورد فرآیندهای رقیب سه مسأله باید مورد توجه قرار گیرد:

- انحصار متقابل
- بن بست
- گرسنگی

جلسه هفتم:

همروندی:
انحصار متقابل و همگام‌سازی
(ادامه)

آنچه در این جلسه می خوانید:

۱- نیازهای انحصار متقابل

۲- انحصار متقابل, حمایت سخت‌افزار

۳- ویژگی‌های رویکرد دستورالعمل ماشین

۴- راهکارهای متداول همروندی

۵- سِمافورها

نیازهای انحصار متقابل:

هر امکان یا قابلیتی که برای حمایت از انحصار متقابل فراهم می‌شود, باید نیازهای زیر را برآورده کند:

- انحصار متقابل باید اِعمال شود: از بین تمام فرآیندهایی که برای یک منبع واحد یا شئ مشترک دارای بخش بحرانی هستند, در هر زمان تنها یک فرآیند می‌تواند در ناحیهٔ بحرانی‌اش باشد.
- فرآیندی که در بخش غیربحرانی‌اش متوقف می‌شود, باید طوری عمل کند که هیچ دخالتی در فرآیندهای دیگر نداشته باشد.
- برای فرآیندی که درخواست دسترسی به بخش بحرانی را دارد, نباید امکان به تأخیر انداختن نامحدود آن وجود داشته باشد, یعنی بن‌بست و گرسنگی مجاز نباشد.
- وقتی هیچ فرآیندی در بخش بحرانی خود نیست, هر فرآیندی که درخواست ورود به بخش بحرانی خود را دارد, باید بدون تأخیر, مجاز به ورود باشد.
- هیچ فرضی دربارهٔ سرعت نسبی فرآیندها و یا تعداد آنها نمی‌توان داشت.
- هر فرآیند فقط برای مدت محدودی در بخش بحرانی خود می‌ماند.

منبع بحرانی:
هرگاه دو یا چند فرآیند نیازمند دسترسی به منبع واحد اشتراک‌ناپذیر, مثل چاپگر باشند, به چنین منبعی, منبع بحرانی و بخشی از برنامه که از آن استفاده می‌کند را بخش بحرانی می‌گویند.

انحصار متقابل, حمایت سخت‌افزار:

چندین رویکرد سخت‌افزاری جالب برای انحصار متقابل وجود دارد:

۱- از کار انداختن وقفه:

در سیستم تک‌پردازنده‌ای, اجرای فرآیندهای همروند نمی‌تواند همپوشانی داشته باشد, بلکه تنها می‌توانند بین هم قرار گیرند.

یک فرآیند تا زمانی که خدمتی از سیستم‌عامل را فراخوانی نکرده است یا دچار وقفه نشده است, به اجرا ادامه می‌دهد. لذا, برای تضمین انحصار متقابل, کافی است که از وقفه دادن فرآیند جلوگیری شود. چون بخش بحرانی نمی‌تواند دچار وقفه شود, اِعمال انحصار متقابل تضمین شده است.

نکته:

این رویکرد در سیستم چندپردازنده‌ای کار نمی‌کند.

انحصار متقابل, حمایت سخت‌افزار:

۲- دستورالعمل‌های ویژهٔ ماشین:

در سیستم‌های چندپردازنده‌ای, فرآیندهای متعدد در دسترسی به حافظهٔ اصلی, سهیم هستند. در این حالت پردازنده‌ها رفتاری مستقل دارند و راهکار وقفه بین پردازنده‌ها وجود ندارد که انحصار متقابل بتواند بر روی آنها پایه‌گذاری گردد.

در سطح سخت‌افزار, به هنگام دسترسی به مکانی از حافظه, از هر دسترسی دیگر به آن مکان جلوگیری می‌شود. با این وجود, طراحان پردازنده چندین دستورالعمل ماشین را پیشنهاد کردند که در حین اجرای این دستورالعملها, دسترسی به آن مکان حافظه (موقع خواندن و نوشتن), برای تمام دستورالعملهای دیگری که به آن مکان مراجعه می‌کنند, مسدود می‌شود.

دو مورد از دستورالعملهایی که به طور گسترده استفاده می‌شوند عبارتند از:

- دستورالعمل مقایسه و مبادله (compare&swap)
- دستورالعمل معاوضه(exchange)

ویژگی‌های رویکرد دستورالعمل ماشین:

مزایای استفاده از دستورالعمل ماشین:

- برای هر تعدادی از فرآیندها بر روی یک یا چند پردازنده که از حافظهٔ مشترک استفاده می‌کنند, قابل به‌کارگیری است.
- ساده است و وارسی آن آسان می‌باشد.
- می‌تواند برای حمایت از چندین بخش بحرانی استفاده شود, هر بخش بحرانی می‌تواند با متغییر خودش تعریف شود.

معایب استفاده از دستورالعمل ماشین:

- انتظار مشغولی وجود دارد.
- امکان گرسنگی وجود دارد.
- امکان بن‌بست وجود دارد.

راهکارهای متداول همروندی:

سیستم‌عامل و زبان‌برنامه‌سازی, برای فراهم کردن همروندی راهکارهای زیر را ارائه داده‌اند:

- سِمافورها (راهنماها)
- سمافور دودویی
- Mutex
- متغییر شرط
- ناظر
- پرچم‌های رویداد
- صندوق‌های پستی/پیام‌ها
- قفل‌های چرخشی

سِمافورها (semaphore):

یکی از راهکارهایی که سیستم‌عامل ارائه می‌کند تا دو یا چند فرآیند بتوانند به وسیلهٔ علامت‌های ساده‌ای با یکدیگر همکاری کنند سمافور است. برای علامت دادن از متغییرهای ویژه‌ای به نام سمافور استفاده می‌شود.

برای دست یافتن به اثر مطلوب, سمافور را به عنوان متغییری صحیح در نظر می‌گیرند که یک مقدار صحیح داشته باشد و فقط سه عمل بر روی آن تعریف می‌شود:

۱- مقدارگذاری سمافور با یک مقدار صحیح نامنفی.

۲- عمل semiWait برای دریافت یک علامت, که یک واحد از مقدار سمافور می‌کاهد. اگر این مقدار منفی شود آنگاه فرآیندی که semiWait را اجرا می‌کند, مسدود می‌شود. در غیر اینصورت, اجرای فرآیند ادامه می‌یابد.

۳- عمل semiSignal برای انتقال یک علامت, که یک واحد از مقدار سمافور می‌کاهد. اگر نتیجه حاصل کوچک‌تر یا مساوی صفر باشد, آنگاه بلوکی که توسط عمل semiWait مسدود شده بود, آزاد می‌شود.

جلسه هشتم:

همروندی:
انحصار متقابل و همگام‌سازی
(راهکارهای متداول همروندی)

آنچه در این جلسه می خوانید:

۱- سِمافور دودویی

۲- Mutex

۳- متغییر شرط

۴- ناظر با علامت (سیگنال)

۵- پرچم‌های رویداد

۶- صندوق‌های پستی

۷- قفل‌های چرخشی

سمافور دودویی:

نسخه محدودتر سمافور, سمافور دودویی است که فقط مقادیر ۰ و ۱ را می‌پذیرد و می‌تواند توسط سه عمل زیر تعریف شود:

۱- سمافور دودویی می‌تواند مقدار اولیه ۰ یا ۱ را بپذیرد.

۲- عمل semiWaitB مقدار سمافور را بررسی می‌کند. اگر مقدار سمافور صفر باشد, آنگاه فرآیندی که semiWaitB را اجرا کرده است, مسدود می‌شود. اگر مقدار سمافور ۱ باشد, آنگاه به ۰ تبدیل می‌شود و اجرای فرآیند ادامه می‌یابد.

۳- عمل semiSignalB مسدود بودن فرآیندها بر روی این سمافور را (که مقدارش صفر است) تعیین می‌کند. اگر چنین باشد, آنگاه فرآیندی که توسط یک semiWaitB مسدود شده بود, آزاد می‌شود. اگر هیچ فرآیندی مسدود نباشد, آنگاه مقدار سمافور برابر یک می‌شود.

Mutex (قفل انحصار متقابل):

قفل انحصار متقابل (Mutex), یک مفهوم مرتبط با سمافور دودویی است. Mutex یک پرچم برنامه‌سازی است که برای بدست آوردن و آزادسازی یک شئ استفاده می‌شود.

وقتی داده‌هایی بدست می‌آیند که نمی‌توانند به اشتراک گذاشته شوند, یا پردازشی شروع می‌شود که نمی‌تواند همزمان در جای دیگری از سیستم اجرا شود, Mutex قفل می‌شود (معمولاً صفر), و هر تلاش دیگر برای دستیابی به آن مسدود می‌گردد.

متغییر شرط:

متغییر شرط, یک نوع داده است که برای مسدود کردن یک فرآیند یا نخ, تا برقراری شرط خاص استفاده می‌شود.

ناظر با علامت (سیگنال):

ناظر, یک مؤلفهٔ نرم‌افزاری متشکل از یک یا چند روال, دنباله‌ای از مقدارگذاری‌های اولیه و داده‌ای محلی است.

ویژگی‌های اصلی ناظر عبارتند از:

۱- متغییرهای دادهٔ محلی فقط توسط روال‌های ناظر قابل دستیابی است و هیچ روال خارجی به آنها دسترسی ندارد.

۲- هر فرآیند با احضار یکی از روال‌های خود, وارد ناظر می‌شود.

۳- در هر لحظه, تنها یک فرآیند می‌تواند ناظر را در حال اجرا داشته باشد و تمام فرآیندهای دیگری که ناظر را احضار کرده‌اند, تا فراهم شدن ناظر مسدود می‌شوند.

پرچم‌های رویداد:

پرچم‌های رویداد از یک قسمت حافظه به عنوان راهکار همگام‌سازی استفاده می‌کند.

کُد برنامه ممکن است به هر بیت یک پرچم, رویداد متفاوتی را نسبت دهد.

با بررسی یک یا تعدادی از این بیت‌ها, یک نخ می‌تواند برای رویداد واحد, یا ترکیبی از رویدادها منتظر بماند.

در حالت AND تا مقدارگیری تمام بیت‌ها و در حالت OR تا زمانی که حداقل یکی از این بیت‌ها مقدار بگیرد, آن نخ مسدود خواهد شد.

صندوق‌های پستی/پیام‌ها:

صندوق‌های پستی, وسیله‌ای برای تبادل اطلاعات بین دو فرآیند, که برای همگام‌سازی هم می‌تواند مورد استفاده قرار گیرد.

قفل‌های چرخشی:

در این راهکار برای انحصار متقابل, فرآیندِ در انتظار یک متغییر قفل, یک حلقهٔ بی‌پایان را اجرا می‌کند, تا مقدار آن, گویای امکان ورود بَه ناحیهٔ بحرانی باشد.

جلسه نهم:

همروندی:
بن‌بست و گرسنگی

آنچه در این جلسه می خوانید:

۱- اصول بن‌بست

۲- منابع قابل استفادهٔ مجدد

۳- شرایط بن‌بست

۴- پیشگیری از بن‌بست

۵- اجتناب از بن‌بست

۶- کشف بن‌بست

اصول بن‌بست:

بن‌بست را می‌توان مسدود بودن دائمی مجموعه‌ای از فرآیندها که برای منابع سیستم با هم رقابت می‌کنند یا با یکدیگر در ارتباط هستند, تعریف کرد.

مجموعه‌ای از فرآیندها در صورتی در بن‌بست هستند که هر کدام از آنها در انتظار رویدادی (معمولاً آزادسازی یک منبع درخواستی) مسدود شده‌اند و این رویداد تنها توسط فرآیند مسدود دیگر در آن مجموعه, رخ می‌دهد.

بن‌بست دائمی است, زیرا هیچکدام از رویدادها هرگز رخ نمی‌دهند.

برخلاف مسأله‌های دیگر در مدیریت فرآیند همروند, هیچ راه‌حل کارآمدی در حالت کلی برای بن‌بست وجود ندارد.

منابع قابل استفادهٔ مجدد:

منابع را می‌توان به دو دستهٔ کلی تقسیم کرد:

- **منابع قابل استفادهٔ مجدد:**

منبع قابل استفاده مجدد، منبعی است که در هر زمان توسط یک منبع، بدون هیچ آسیبی مورد استفاده قرار می‌گیرد و در اثر استفاده تمام نمی‌شود. (پردازنده، ورودی خروجی، حافظه اصلی و ثانوی، دستگاه‌ها و ساختمان‌داده‌هایی مثل فایل‌ها، پایگاه‌های داده و ...)

- **منابع مصرف شدنی:**

منبع مصرف شدنی، منبعی است که می‌تواند ایجاد (تولید) شود و از بین برود (مصرف شود).

معمولاً روی تعداد منابع مصرف‌شدنی از یک نوع خاص، محدودیتی وجود ندارد.

وقتی منبع در اختیار فرآیند مصرف کننده قرار گرفت، آن منبع از بین می‌رود. (وقفه‌ها، علامت‌ها، سیگنال‌ها، پیام‌ها و ...)

شرایط بن‌بست:

شرایط زیر به عنوان سیاست باید وجود داشته باشد تا بن‌بست امکان‌پذیر گردد:

۱- انحصار متقابل

۲- نگه داشتن و انتظار:

در حالی‌که فرآیند, منبع تخصیص یافته را نگه می‌دارد, منتظر تخصیص منبع دیگری است.

۳- قبضه نکردن:

وقتی فرآیندی منبعی را در اختیار دارد, نتوان آن منبع را به زور پس گرفت.

سه شرط فوق برای وجود بن‌بست لازم است ولی کافی نیست. برای اینکه بن‌بست واقعاً رخ دهد نیاز به شرط زیر می‌باشد:

۴- انتظار حلقوی:

زنجیرهٔ بسته‌ای از فرآیندها وجود دارد به طوری که هر فرآیند حداقل یک منبع مورد نیاز فرآیند بعدی در این زنجیره را در اختیار دارد.

پیشگیری از بن‌بست:

برای پیشگیری از بن‌بست باید سیستم به گونه‌ای باشد که از قبل امکان بن‌بست از بین برده شود.

روشهای پیشگیری از بن‌بست, به دو دسته غیرمستقیم و مستقیم تقسیم می‌شوند.

در روشهای غیرمستقیم برای پیشگیری از بن‌بست, جلوگیری از وقوع یکی از سه شرط شرایط بن‌بست لازم است.

در روش مستقیم برای پیشگیری از بن‌بست, جلوگیری از وقوع شرط انتظار حلقوی لازم است.

- **نکته:**

  - شرط انحصار متقابل را نمی‌توان رد کرد. (اجازه خواندن و عدم اجازه نوشتن فایل)
  - می‌توان از شرط نگه داشتن و انتظار را با مُلزم کردن فرآیند به درخواست یکباره تمام منابع مورد نیاز و مسدود کردن آن فرآیند تا موقعی که تمام منابع در اختیارش قرار گیرد, پیشگیری کرد.
  - برای جلوگیری از قبضه کردن, اگر فرآیندی که منابعی را در اختیار دارد مجاز به درخواست بیشتر نباشد, آن فرآیند منابع قبلی خود را باید رها کند و در صورت لزوم آنها را همراه با منبع اضافی, دوباره درخواست کند.
  - برای پیشگیری از شرط انتظار حلقوی, می‌توان یک تعریف خطی از انواع منابع انجام داد.

اجتناب از بن‌بست:

رویکرد زیرکانه‌تر از پیشگیری از بن‌بست, اجتناب از بن‌بست است. در اجتناب از بن‌بست, انتخاب‌ها طوری عمل می‌کند که هرگز نقطهٔ بروز بن‌بست پیش نیاید.

دو رویکرد برای اجتناب از بن‌بست وجود دارد:

- عدم شروع فرآیندی که درخواست‌های آن منجر به بن‌بست شود.
- عدم پاسخ به درخواست‌های منبع اضافی از فرآیندی که ممکن است با این تخصیص منجر به بن‌بست شود.

کشف بن‌بست:

راهبردهای کشف بن‌بست دسترسی به منابع یا فعالیت عملکرد فرآیند را محدود نمی‌کنند. در کشف بن‌بست هر جا ممکن باشد, منابع درخواست شده در اختیار فرآیندها قرار می‌گیرد. سیستم‌عامل متناوباً الگوریتمی را اجرا می‌کند تا وجود شرط انتظار حلقوی را کشف کند.

بررسی وجود بن‌بست در هر درخواست منبع منجر به کشف زودهنگام بن‌بست می‌شود ولی این بررسی‌های مکرر زمان قابل ملاحظه‌ای از پردازنده را مصرف می‌کند.

- **ترمیم:**

پس از کشف بن‌بست, به راهبردهایی برای ترمیم نیاز است. چند نمونه از این راهبردها در زیر آمده است:

۱- قطع تمام فرآیندهای بن‌بست.

۲- برگشت هر یک از فرآیندهای بن‌بست به یک نقطهٔ بازرسی از قبل تعریف شده و شروع مجدد تمام فرآیندها.

۳- قطع پی در پی فرآیندهای بن‌بست تا دیگر بن‌بستی نباشد.

۴- قبضه‌کردن پی در پی منابع تا اینکه دیگر بن‌بست نباشد.

جلسه دهم:

مدیریت حافظه

آنچه در این جلسه می خوانید:

۱- مقدمه

۲- نیازهای مدیریت حافظه

مقدمه:

در سیستم‌های تک‌برنامه‌ای, حافظه اصلی به دو بخش تقسیم می‌شود: یک بخش برای سیستم‌عامل (ناظر مقیم, هسته) و یک بخش برای برنامهٔ در حال اجرا.

در سیستم‌های چندبرنامه‌ای, بخش "کاربر" حافظه باید به زیر بخش‌های تقسیم شود تا چندین فرآیند را در خود جای دهد. وظیفهٔ تقسیم‌بندی حافظه, به طور پویا توسط سیستم‌عامل انجام می‌گیرد و به این عمل, مدیریت حافظه می‌گویند.

وجود یک مدیریت حافظه کارآمد برای یک سیستم چندبرنامه‌ای حیاتی است. اگر فقط چند فرآیند محدود در حافظه باشند, اغلب اوقات تمام فرآیندها منتظر ورودی-خروجی هستند و پردازنده بیکار خواهد بود. بنابراین حافظه باید طوری اختصاص داده شود که تعداد معقولی از فرآیندهای آماده بتوانند از وقت پردازنده استفاده کنند.

نیازهای مدیریت حافظه:

هنگام بررسی راهکارها و خط‌مشی‌های گوناگون مدیریت حافظه, پرداختن به نیازهایی که مدیریت حافظه باید برآورده نماید, مهم است. این نیازها عبارتند از:

- جابجایی (relocation)
- حفاظت (protection)
- اشتراک (sharing)
- سازمان منطقی (logical organization)
- سازمان فیزیکی (physical organization)

نیازهای مدیریت حافظه:

- جابجایی (relocation):

در سیستم چندبرنامه‌ای, حافظه موجود بین تعدادی فرآیند مشترک است. معمولاً, برای برنامه‌ساز ممکن نیست که از قبل بداند در حین اجرای برنامهٔ او, چه رنامه‌های دیگری در حافظهٔ اصلی مقیم هستند. همچنین باید امکان مبادلهٔ فرآیند فعال به داخل یا خارج از حافظه باشد تا فرآیندهای متعددی آماده اجرا باشند تا از این طریق بهره‌وری پردازنده به حداکثر برسد.

اگر قرار باشد برنامه‌ای که بر وی حافظه جانبی مبادله شده, هنگام برگشت به حافظه اصلی, در همان محل قبلی‌اش قرار گیرد, اینکار محدودیت‌هایی را بوجود می‌آورد. به همین دلیل فرآیند به ناحیهٔ دیگری از حافظه اصلی جابجا می‌شود.

نیازهای مدیریت حافظه:

- حفاظت (protection):

هر فرآیند باید در مقابل دخالت‌های ناخواستهٔ فرآیندهای دیگر محافظت شود, چه این دخالت تصادفی و چه عمدی باشد. لذا, فرآیندهای دیگر نباید قادر باشند به منظور خواندن یا نوشتن, بدون اجازه به مکان‌های حافظه یک فرآیند مراجعه کنند. از طرف دیگر, برآورده کردن نیازهای جابجایی, برآورده کردن نیازهای حفاظتی را دشوارتر می‌کند. چون مکان برنامه در حافظهٔ اصلی قابل پیش‌بینی نیست, بررسی آدرس برای اطمینان از حفاظت ممکن نیست. لذا, تمام مراجعات به حافظه که توسط یک فرآیند انجام می‌گیرد, باید در زمان اجرا بررسی گردد تا اطمینان حاصل شود که آنها فقط به فضای حافظهٔ تخصیص یافته شده به آن فرآیند مراجعه می‌کنند.

- **نکته:**

نیازهای حفاظت از حافظه باید توسط پردازنده (سخت‌افزار) و نه سیستم‌عامل (نرم‌افزار) برآورده شود. زیرا سیستم‌عامل نمی‌تواند تمام مراجعاتی را که یک برنامه مطرح می‌کند, پیش‌بینی کند. حتی اگر هم بتواند, بررسی برای یافتن خطاهای مراجعه به حافظه, بسیار وقت‌گیر است.

نیازهای مدیریت حافظه:

- اشتراک (sharing):

هر راهکار حفاظتی باید انعطاف پذیری لازم را برای اجازه دادن به فرآیندهای متعدد در دسترسی به بخش واحدی از حافظه داشته باشد. اگر تعدادی فرآیند در حال اجرای برنامهٔ واحدی هستند, به جای اینکه هر فرآیند کپی جداگانه‌ای از برنامه را در اختیار داشته باشد, بهتر است تمام فرآیندها به یک کپی از برنامه دسترسی داشته باشند.

فرآیندهایی که برای انجام وظیفه‌ای با یکدیگر همکاری می‌کنند, ممکن است لازم باشد به ساختمان دادهٔ مشترکی دسترسی داشته باشند. لذا سیستم مدیریت حافظه باید با رعایت اصول حفاظتی, دستیابی کنترل شده به حافظهٔ مشترک را اجازه دهد.

نیازهای مدیریت حافظه:

- سازمان منطقی (logical organization):

معمولاً حافظه اصلی در یک سیستم کامپیوتری, به صورت فضای آدرس خطی یا یک بُعدی سازمان‌دهی شده و شامل دنباله‌ای از بایت‌ها یا کلمه‌ها است. حافظه جانبی نیز در سطح فیزیکی, به همین صورت سازمان‌دهی می‌شود. گرچه این سازمان‌دهی با سخت‌افزار ماشین سازگار است, ولی با روش ساخت برنامه‌ها سازگار نیست. اغلب برنامه‌ها به صورت مؤلفه‌ها سازمان‌دهی می‌شوند, که بعضی از آنها غیر قابل تغییر (فقط خواندنی, فقط اجرایی) و بعضی دیگر شامل داده‌های قابل تغییر هستند.

اگر سیستم‌عامل و سخت‌افزار بتوانند برنامه‌های کاربر و داده‌ها را به شکل مؤلفه‌ها اداره کنند, آنگاه:

۱- مؤلفه‌ها را می‌توان به طور مستقل نوشت.

۲- با سربار اندکی, می‌توان درجات مختلفی از حفاظت (فقط خواندنی, فقط اجرایی) را در نظر گرفت.

۳- می‌توان راهکارهایی برای اشتراک مؤلفه‌ها یک فرآیندها فراهم کرد.

نیازهای مدیریت حافظه:

- سازمان فیزیکی (physical organization):

حافظهٔ کامپیوتر در دو سطح سازمان‌دهی می‌شود, حافظه اصلی و حافظه جانبی.

در این طرح دو سطحی, سازمان‌دهی جریان اطلاعات بین حافظهٔ اصلی و جانبی, دغدغه اصلی سیستم است. مسئولیت جریان اطلاعات می‌تواند بر عهدهٔ برنامه‌ساز گذاشته شود ولی به دو دلیل غیرعملی و نامطلوب است:

۱- ممکن است حافظه اصلی موجود, برای برنامه و داده‌های آن کافی نباشد. در این‌صورت, برنامه‌ساز باید درگیر روشی به نام روی هم‌گذاری شود که بسیار وقت برنامه‌ساز را به هدر می‌دهد.

۲- در محیط‌های چندبرنامه‌ای, برنامه‌ساز در زمان کُدنویسی نمی‌داند چه مقدار فضا موجود خواهد بود یا آن فضا در کجاست.

جلسه یازدهم:

روش‌های مدیریت حافظه

آنچه در این جلسه می خوانید:

۱- روش‌های مدیریت حافظه
۲- بخش‌بندی حافظه
۳- بخش‌بندی ثابت
۴- بخش‌بندی پویا
۵- سیستم رفاقتی
۶- صفحه‌بندی
۷- قطعه‌بندی

روش‌های مدیریت حافظه:

- بخش‌بندی ثابت
- بخش‌بندی پویا
- صفحه‌بندی ساده
- قطعه‌بندی ساده
- صفحه‌بندی حافظهٔ مجازی
- قطعه‌بندی حافظهٔ مجازی

بخش‌بندی حافظه (Memory partitioning):

عمل اصلی مدیریت حافظه, آوردن فرآیندها به حافظهٔ اصلی برای اجرا توسط پردازنده است. تقریباً در تمام سیستم‌های چندبرنامه‌ی جدید, این وظیفه شامل طرح پیچیده‌ای به نام حافظهٔ مجازی است.

حافظهٔ مجازی به نوبهٔ خود نیازمند استفاده از یک یا هر دو روش قطعه‌بندی و صفحه‌بندی است.

بخش‌بندی, در بعضی از سیستم‌عامل‌های قدیمی, به شکل‌های گوناگونی به کار گرفته شده است.

دو روش قطعه‌بندی ساده و صفحه‌بندی ساده به کار گرفته نمی‌شوند.

بخش‌بندی ثابت:

در اغلب طرح‌های مدیریت حافظه, می‌توان فرض کرد که سیستم‌عامل بخش ثابتی از حافظهٔ اصلی را اِشغال می‌کند و بقیهٔ حافظهٔ اصلی می‌تواند در اختیار چندین فرآیند قرار گیرد.

ساده‌ترین طرح مدیریت این حافظهٔ موجود, بخش‌بندی آن به ناحیه‌هایی با مرزهای ثابت است.

در بخش‌بندی ثابت, هر فرآیندی که اندازه‌اش کمتر یا مساوی با اندازهٔ بخش باشد, می‌تواند به داخل هر بخش موجود بار شود. اگر تمام بخش‌ها پر باشند و هیچ فرآیندی در حالت آماده یا اجرا نباشد, سیستم‌عامل می‌تواند فرآیندی را از یکی از بخش‌ها خارج و فرآیند دیگری را به آن بار کند کاری برای پردازنده وجود داشته باشد.

بخش‌بندی ثابت می‌تواند بخش‌هایی با اندازه مساوی و یا بخش‌هایی با اندازه نامساوی باشد.

سیستم‌عامل OS/MFT (چند برنامه‌ای با تعداد ثلبتی از وظیفه‌ها) در کامپیوتر بزرگ IBM از بخش بندی ثابت استفاده می‌کرد.

مشکلات استفاده از بخش‌بندی با اندازه مساوی:

- بزرگ‌تر بودن فرآیند از یک بخش.

(راه حل: برنامه‌ساز باید برنامه را طوری طراحی کند که در هر لحظه فقط بخش مورد نیاز به حافظه اصلی بار شود)

- بهره‌وری حافظه به شدت ناکارآمد است.

برنامه‌های کوچک‌تر از یک بخش, مابقی فضای بخش را به هدر می‌دهند.

الگوریتم جاگذاری (Palcement algorithm):

با استفاده از بخش‌های هم‌اندازه, جایگذاری فرآیندها در حافظهٔ اصلی ساده است.

وقتی اندازهٔ بخش‌ها نامساوی باشد, تخصیص هر فرآیند به کوچکترین بخشی که در آن جای می‌گیرد, صورت می‌پذیرد. دو راه برای تخصیص فرآیندها به بخش وجود دارد:

۱- در ساده‌ترین راه, برای هر بخش نیاز به یک صف زمان‌بندی است.

۲- روش دیگر در نظر گرفتن صفی واحد برای تمام فرآیندها است.

- **نکته**:

موارد دیگری را نیز می‌توان در نظر گرفت, مثل اولویت یا تقدم فرآیند مسدود شده در مقابل فرآیند آماده.

مشکلات استفاده از بخش‌بندی با اندازه نامساوی:

- تعداد بخش‌های مشخص شده در زمان تولید سیستم, تعداد فرآیندهای فعال در سیستم را محدود می‌سازد.
- چون اندازهٔ بخش در زمان تولید سیستم تعیین می‌شود, کارهای کوچک به طور مؤثر از فضای بخش استفاده نمی‌کنند.

بخش‌بندی پویا:

در طرح بخش‌بندی پویا, بخش‌ها طول متغییر دارند و تعداد آنها نیز متغییر است. وقتی فرآیندی به حافظه آورده می‌شود, دقیقاً همان اندازه که حافظه نیاز دارد به آن اختصاص می‌یابد.

- مشکل استفاده از بخش‌بندی پویا:

این روش به خوبی شروع می‌شود اما سرانجام به وضعیتی می‌رسد که تعداد زیادی حفرهٔ کوچک حافظه وجود دارد که بهره‌وری حافظه را کاهش می‌دهد.

- رفع مشکل تکه تکه شدن حافظه:

برای مقابله با تکه تکه شدن می توان از فشرده‌سازی استفاده کرد. در این تکنیک سیستم‌عامل فرآیندها را انتقال می‌دهد تا کنار هم قرار گیرندو کل حافظهٔ آزاد در یک قسمت قرار گیرد.

- مشکل استفاده از فشرده‌سازی:

وقت‌گیر است و وقت پردازنده را هدر می‌دهد.

سیستم‌عامل OS/MVT (چند برنامه‌ای با تعداد متغییری از وظیفه‌ها) در کامپیوتر بزرگ IBM از بخش بندی پویا استفاده می‌کرد.

الگوریتم جاگذاری (Palcement algorithm):

چون فشرده‌سازی حافظه وقت‌گیر است, طراح سیستم‌عامل باید در تصمیم‌گیری برای تخصیص حافظه به فرآیندها (نحوهٔ پر کردن حفره‌ها) هوشمندانه عمل کند.

سه الگوریتم جاگذاری مهم عبارتند از:

- بهترین برازش (Best fit): (بدترین)

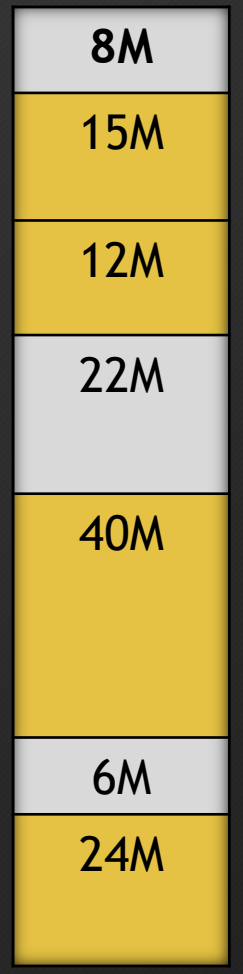
این الگوریتم بلوکی را انتخاب می‌کند که اندازهٔ آن از همهٔ به حافظهٔ درخواستی نزدیک‌تر باشد.

- اولین برازش (First fit): (ساده‌ترین, بهترین و سریع‌ترین)

حافظه را از ابتدا مرور می‌کند و اولین بلوک با اندازهٔ کافی را انتخاب می‌نماید.

- برازش بعدی (Next fit):

حافظه را از مکان آخرین جاگذاری به بعد, مرور می‌کند و اولین بلوک با اندازهٔ کافی را انتخاب می‌کند.



سیستم رفاقتی (Buddy system):

در این طرح, اندازهٔ بلوک حافظه 2^K است که در آن $L \le K \le U$ است.

2^L: اندازهٔ کوچک‌ترین بلوک تخصیص یافته است.

2^U: اندازهٔ بزرگ‌ترین بلوک تخصیص یافته است.

بلوک ۱ مگا بایتی

1M			
A=128K	128K	256K	512K

A = فرآیندی ۱۲۸ کیلوبایتی

B = فرآیندی ۲۴۰ کیلوبایتی

C = فرآیندی ۶۴ کیلوبایتی

D = فرآیندی ۲۵۶ کیلوبایتی

B آزادسازی

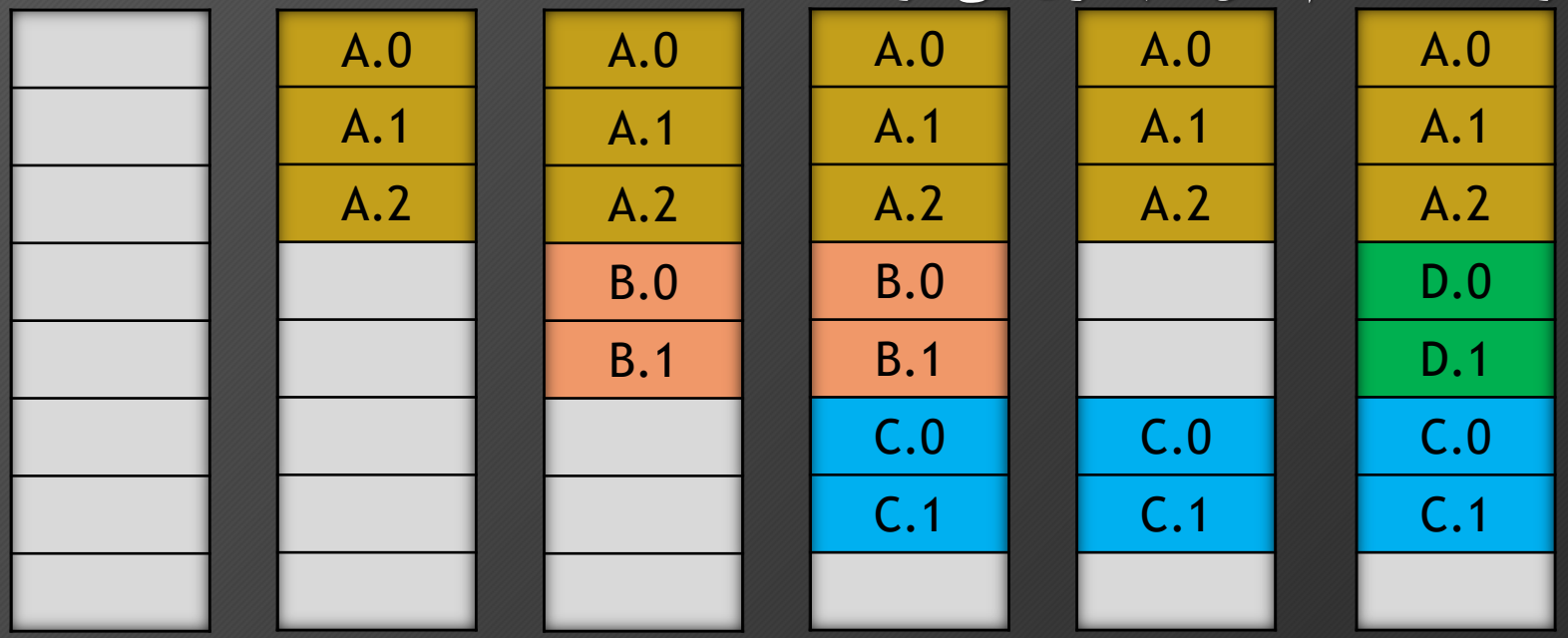
E = فرآیندی ۷۵ کیلوبایتی

شکل اصلاح شده سیستم رفاقتی برای تخصیص حافظهٔ هسته یونیکس استفاده می‌شود.

صفحه‌بندی (Paging):

در این روش حافظهٔ اصلی به بخش‌های نسبتاً کوچک با اندازه‌های ثابت مساوی, و هر فرآیند نیز به بخش‌های هم‌اندازه با آنها تقسیم می‌شود. آنگاه بخش‌های فرآیند که صفحه نام دارند, می‌توانند به بخش‌های موجود حافظه, به نام قاب یا قاب صفحه تخصیص داده شوند.

لیستی از قاب‌های آزاد توسط سیستم‌عامل نگهداری می‌شود.



قطعه‌بندی (Segmentation):

در این روش برنامه کاربر می‌تواند قطعه‌بندی شود, که در آن, برنامه و داده‌های مربوط به آن به تعدادی قطعه تقسیم می‌شوند. لازم نیست تمام قطعات همهٔ برنامهها طول یکسانی داشته باشند, گرچه حداکثری برای طول قطعه وجود دارد.

به دلیل استفاده از قطعات نامساوی, قطعه‌بندی شبیه بخش‌بندی پویا است.

تفاوت قطعه‌بندی با بخش‌بندی در این است که در قطعه‌بندی, هر برنامه ممکن است بیش از یک بخظ را اشغال کند و لزومی ندارد این بخش‌ها پیوسته باشند.

جلسه دوازدهم:

زمان‌بندی تک‌پردازنده‌ای

آنچه در این جلسه می خوانید:

۱- مقدمه

۲- انواع زمان‌بندی پردازنده

۳- زمان‌بندی کوتاه مدت

۴- سیاست‌های زمان‌بندی

۵- الگوریتم خدمت به ترتیب ورود

مقدمه:

در سیستم‌های چندبرنامه‌ای, چندین فرآیند در حافظهٔ اصلی نگهداری می‌شود. هر فرآیند یا در حال استفاده از پردازنده است یا منتظر کامل شدن اجرای ورودی خروجی یا وقوع رویداد دیگری است.

پردازنده از طریق اجرای یک فرآیند, در حالیکه فرآیندهای دیگر در حال انتظار هستند, فعال نگه داشته می‌شود.

کلید چندبرنامه‌ای, زمان‌بندی است.

انواع زمان‌بندی:

هدف زمان‌بندی پردازنده, تخصیص فرآیندها به پردازنده جهت اجرا است> به طوری که اهداف سیستم از قبیل زمان پاسخ, توان عملیاتی و کارایی پردازنده برآورده شود.

- زمان‌بندی بلندمدت:

تصمیم‌گیری در مورد افزودن به مجموعهٔ فرآیندها برای اجرا.

- زمان‌بندی میان‌مدت:

تصمیم‌گیری در مورد افزودن به تعداد فرآیندهایی که تمام یا بخشی از آنها در حافظهٔ اصلی است.

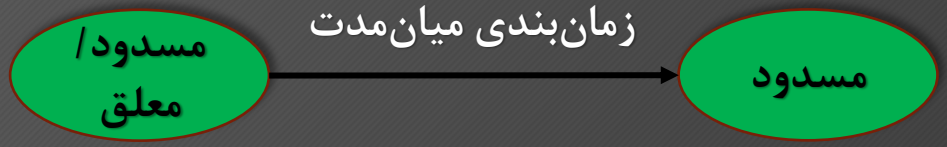
- زمان‌بندی کوتاه‌مدت:

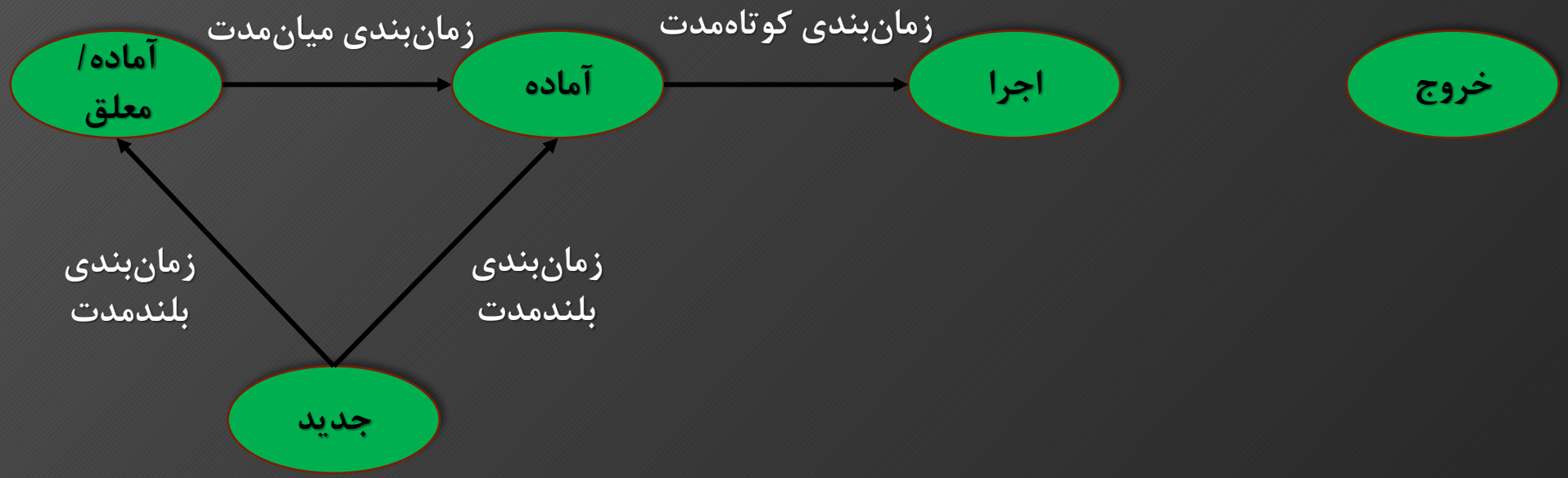
تصمیم‌گیری در مورد اینکه کدام فرآیند موجود در حافظه اصلی برای اجرا انتخاب گردد.

- زمان‌بندی ورودی خروجی:

تصمیم‌گیری در مور اینکه کدام درخواست ورودی خروجی فرآیندها, توسط دستگاه ورودی خروجی بیکار اداره شود.

انواع زمان‌بندی:





سیاست‌های زمان‌بندی:

- خدمت به ترتیب ورود (FCFS) یا خروج به ترتیب ورود (FIFO)
- اولویت
- نوبت گردشی (RR)
- کوتاه‌ترین فرآیند (SPN)
- کوتاه‌ترین زمان باقیمانده (SRT)
- بالاترین نسبت پاسخ (HRRN)
- ...

الگوریتم خدمت به ترتیب ورود:

ساده‌ترین سیاست زمان‌بندی, خدمت به ترتیب ورود است که خروج به ترتیب ورود یا طرح صف‌بندی دقیق نیز نامیده میشود.

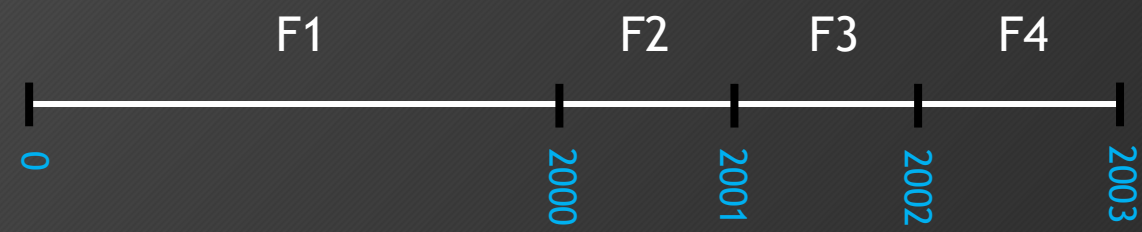
وقتی فرآیندی آماده می‌شود, در صف آماده قرار می‌گیرد.

وقتی فرآیند در حال اجرا متوقف می‌شود, فرآیندی که بیش از همه در صف بود برای اجرا انتخاب می‌شود.

این الگوریتم برای فرآیندهای طولانی بهتر عمل می‌کند.

خدمت به ترتیب ورود:

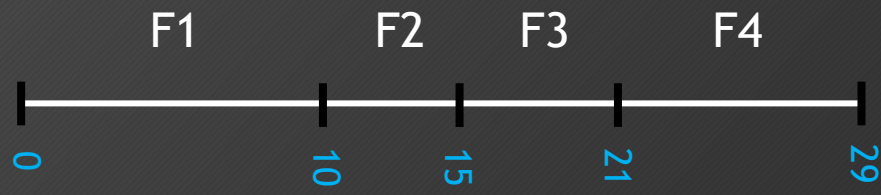
فرآیند	زمان مورد نیاز
F1	2000s
F2	1s
F3	1s
F4	1s



زمان برگشت: زمان اتمام کار منهای زمان شروع اولین فرآیند
زمان انتظار: زمان منتظر ماندن فرآیند برای شروع
میانگین زمان برگشت: جمع تمامی زمان‌های برگشت تقسیم بر تعداد آنها
میانگین زمان انتظار: جمع تمامی زمان‌های انتظار تقسیم بر تعداد آنها

خدمت به ترتیب ورود:

فرآیند	زمان مورد نیاز
F1	10s
F2	5s
F3	6s
F4	8s



زمان برگشت F1:

زمان انتظار F1:

زمان برگشت F2:

زمان انتظار F2:

زمان برگشت F3:

زمان انتظار F3:

زمان برگشت F4:

زمان انتظار F4:

زمان برگشت کل:

زمان انتظار کل:

جلسه سیزدهم:

زمان‌بندی تک‌پردازنده‌ای

(الگوریتم‌های زمان‌بندی)

آنچه در این جلسه می خوانید:

۱- الگوریتم نوبت گردشی
۲- الگوریتم اولویت
۳- الگوریتم کوتاه‌ترین فرآیند
۴- الگوریتم بالاترین نسبت پاسخ

الگوریتم نوبت گردشی (Round Robin):

این الگوریتم از ایدهٔ FIFO استفاده می‌کند و ایراد آنرا ندارد.

هر کاری که ابتدا وارد می‌شود, ابتدا CPU را در اختیار می‌گیرد اما رها کردن آن دلخواه نیست. اگر در یک بازهٔ زمانی خودش CPU را رها نکرد, برش زمانی که به پایان برسد الزاماً عمل تعویض کار صورت می‌گیرد و پردازش جاری به انتهای صف بر می‌گردد تا بعداً اجرایش را دنبال کند.

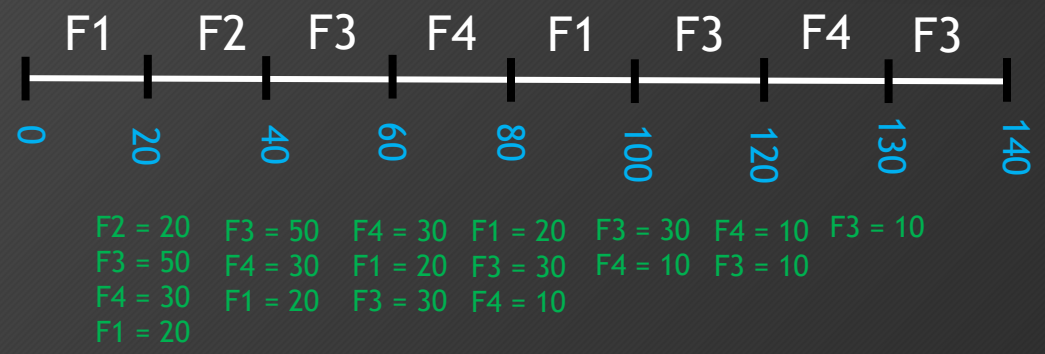
- **عیب:**

میانگین زمان برگشت کارها طولانی است زیرا کارها سوئیچ می‌شوند.

نوبت گردشی (RR):

فرآیند	زمان مورد نیاز
F1	40s
F2	20s
F3	50s
F4	30s

برش زمانی
20s



زمان برگشت F1: 0 - 100 = 100

زمان برگشت F2: 0 - 40 = 40

زمان برگشت F4: 0 - 130 = 130

زمان برگشت F3: 0 - 140 = 140

زمان انتظار F1: (80-20) + 0 = 60

زمان انتظار F2: 20

زمان انتظار F3: (130-120) + (100-60) + 40 = 90

زمان انتظار F4: (120-80) + 60 = 100

نوبت گردشی (RR):

- **تمرین:**

مثال صفحه قبل را وقتی که برش زمانی 15s است حل کنید.

الگوریتم اولویت (Priority):

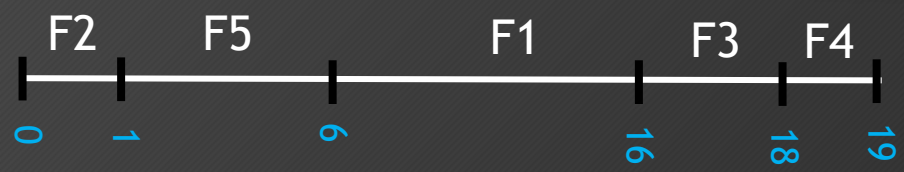
در این روش, هر فرآیندی که اولویت بیشتری داشته باشد زودتر CPU را در اختیار می‌گیرد.
این الگوریتم به دو دسته تقسیم می‌شود: ۱- با بازپس‌گیری. ۲- بدون بازپس‌گیری.

- **عیب:**

ممکن است به بعضی از پردازش‌ها که دارای اولویت کمتری هستند هرگز CPU نرسد.

الگوریتم اولویت (Priority):

فرآیند	زمان مورد نیاز	اولویت
F1	10s	3
F2	1s	1
F3	2s	3
F4	1s	4
F5	5s	2



زمان برگشت F1: 0 - 16 = 16

زمان برگشت F2: 0 - 1 = 1

زمان برگشت F3: 0 - 18 = 18

زمان برگشت F4: 0 - 19 = 19

زمان برگشت F5: 0 - 6 = 6

زمان انتظار F1: 6

زمان انتظار F2: 0

زمان انتظار F3: 16

زمان انتظار F4: 18

زمان انتظار F5: 1

الگوریتم اولویت (Priority):

- **تمرین:**

زمان‌های برگشت و انتظار را برای فرآیندهای روبرو محاسبه و خط زمانی آن را رسم کنید.

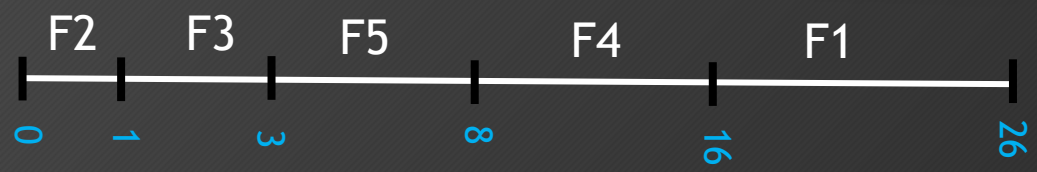
فرآیند	زمان مورد نیاز	اولویت
F1	10s	3
F2	17s	1
F3	12	5
F4	6s	4
F5	5s	2

الگوریتم کوتاه‌ترین فرآیند (Shortest Job First):

در این روش، فرآیندهای کوتاه‌تر اولویت بیشتری دارند و زودتر CPU را در اختیار می‌گیرد.

الگوریتم کوتاه‌ترین فرآیند (Shortest Job First):

فرآیند	زمان مورد نیاز
F1	10s
F2	1s
F3	2s
F4	8s
F5	5s



زمان برگشت F1: 0 - 26 = 26

زمان برگشت F2: 0 - 1 = 1

زمان برگشت F3: 0 - 3 = 3

زمان برگشت F4: 0 - 16 = 16

زمان برگشت F5: 0 - 8 = 8

زمان انتظار F1: 16

زمان انتظار F2: 0

زمان انتظار F3: 1

زمان انتظار F4: 8

زمان انتظار F5: 3

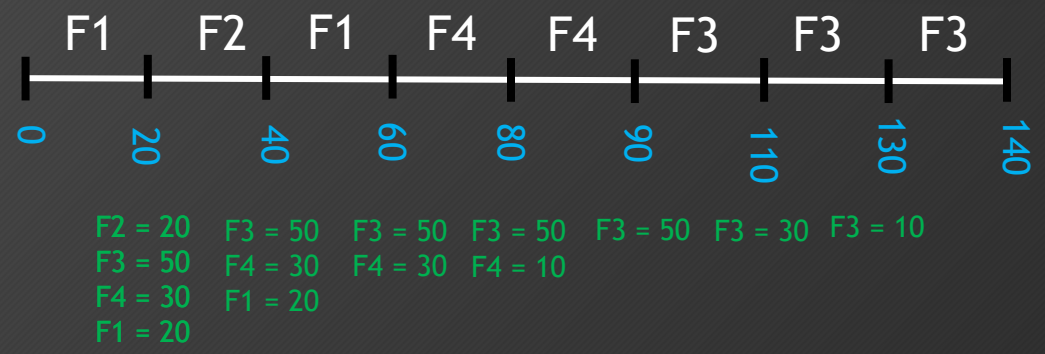
الگوریتم کوتاه‌ترین زمان باقی‌مانده (Shortest Remain Time):

در این روش, کوچک بودن زمان پردازش باقیمانده دارای اولویت بیشتری است.
فرآیندی که زمان اجرای کوتاه‌تری دارد ابتدا انتخاب می‌شود ولی CPU قابل باز پس‌گیری است.

الگوریتم کوتاه‌ترین زمان باقی‌مانده (Shortest Remain Time):

فرآیند	زمان مورد نیاز
F1	40s
F2	20s
F3	50s
F4	30s

برش زمانی
20s



زمان برگشت F1: 0 - 60 = 60

زمان برگشت F2: 0 - 40 = 40

زمان برگشت F3: 0 - 140 = 140

زمان برگشت F4: 0 - 90 = 90

زمان انتظار F1: (40-20) + 0 = 20

زمان انتظار F2: 20

زمان انتظار F3: 90

زمان انتظار F4: 60

الگوریتم کوتاه‌ترین زمان باقی‌مانده (Shortest Remain Time):

- **تمرین:**

مثال صفحه قبل را وقتی که برش زمانی 15s است حل کنید.

الگوریتم بالاترین نسبت پاسخ (Highest Response Ratio Next):

این روش, یک نوع زمان‌بندی اولویت است که اولویت‌بندی پویا دارد.

اگر دارای زمان‌اجرای کم باشد, اولویت آن زیاد است اما برای کارهای با زمان اجرای زیاد اولویت آن کم است و به تدریج هر چه زمان انتظار زیاد می‌شود, بر اولویت آنها افزوده می‌گردد.

جلسه چهاردهم:

مدیریت ورودی خروجی

آنچه در این جلسه می خوانید:

۱- دستگاه‌های ورودی خروجی
۲- تفاوت‌های عمدهٔ بین دسته‌ها
۳- سازمان عمل ورودی خروجی
۴- تکامل عمل ورودی خروجی
۵- دسترسی مستقیم به حافظه

دستگاه‌های ورودی خروجی:

دستگاه‌های خارجی که با ورودی خروجی سروکار دارند به سه دسته تقسیم می‌شوند:

۱- قابل خواندن توسط انسان:

برای برقراری ارتباط کاربر با کامپیوتر مناسب است. (چاپگر, مانیتور, کیبرد, ماوس و ... .)

۲- قابل خواندن توسط ماشین:

برای برقراری ارتباط با تجهیزات الکترونیکی مناسب است. (گرداننده‌های دیسک و نوار و ... .)

۳- ارتباطات:

برای برقراری ارتباط با دستگاه‌های راه‌دور مناسب است. (مودم و ... .)

تفاوت‌های عمدهٔ بین دسته‌ها:

- **نرخ داده:**

از کم به زیاد: صفحه کلید, ماوس, مودم, چاپگر, اسکنر, دیسک نوری و سخت, نمایش گرافیکی.

- **کاربردها:**

دیسکی برای دسترسی فایل‌ها در مقابل دیسکی برای ذخیرهٔ پشتیبان. (مؤثر در الگوریتم‌های زمان‌بندی)

- **پیچیدگی کنترل:**

دیسک نسبت به چاپگر به رابط کنترل پیچیده‌تری نیاز دارد.

- **واحد انتقال:**

داده‌ها ممکن است به صورت جریانی از بایت‌ها یا کاراکترها یا بلوک‌های بزرگ انتقال یابند.

- **نمایش داده‌ها:**

در دستگاه‌های مختلف, طرح‌های کدگذاری مختلفی برای داده‌ها استفاده می‌شود.

- **شرایط خطا:**

ماهیت خطاها, شیوهٔ گزارش آنها, اثرات آنها و حوزهٔ پاسخ آنها از دستگاهی به دستگاه دیگر متفاوت است.

سازمان عمل ورودی خروجی:

سه روش انجام ورودی خروجی مطرح است:

۱- ورودی خروجی برنامه‌سازی شده:

پردازنده از طرف فرآیند, یک فرمان ورودی خروجی را به مؤلفهٔ ورودی خروجی صادر می‌کند. سپس آن فرآیند به حالت انتظار مشغولی می‌رود تا عمل ورودی خروجی کامل شود.

۲- ورودی خروجی مبتنی بر وقفه:

پردازنده یک فرمان ورودی خروجی را از جانب فرآیند صادر می‌کند. اگر دستورالعمل ورودی خروجی از جانب فرآیند مسدود کننده نباشد, پردازنده می‌تواند به اجرای دستورالعمل ادامه دهد. اگر دستورالعمل ورودی خروجی مسدود کننده باشد, آنگاه دستورالعمل بعدی که پردازنده اجرا می‌کند,فرآیند جاری را در حالت مسدود قرار می‌دهد و فرآیند دیگری را زمان‌بندی می‌کند.

۳- دسترسی مستقیم به حافظه (DMA): (مهمترین روش انتقال داده‌ها)

مؤلفهٔ DMA, مبادلهٔ داده‌ها بین حافظهٔ اصلی و مؤلفهٔ ورودی خروجی را کنترل می‌کند.

پردازنده درخواست انتقال بلوکی از داده‌ها را به مؤلفهٔ DMA صادر می‌کند و فقط پس از انتقال کامل بلوک دچار وقفه می‌شود.

تکامل عمل ورودی خروجی:

با تکامل سیستم‌های کامپیوتری, پیچیدگی و پیشرفت هر یک از مؤلفه‌ها افزایش یافت که این تغییر ار عمل ورودی خروجی بیشتر نمایان است.

- **مراحل تکامل:**

۱- پردازنده مستقیم دستگاه جانبی را کنترل می‌کند.

۲- یک کنترل‌کننده یا مؤلفهٔ ورودی خروجی اضافه می‌شود.

۳- یک کنترل‌کننده یا مؤلفهٔ ورودی خروجی اضافه می‌شود و وقفه اِعمال می‌گردد.

۴- کنترل حافظه از طریق DMA به مؤلفهٔ ورودی خروجی داده می‌شود.

۵- مؤلفه ورودی خروجی تا حد یک پردازندهٔ مستقل تکامل یافت.

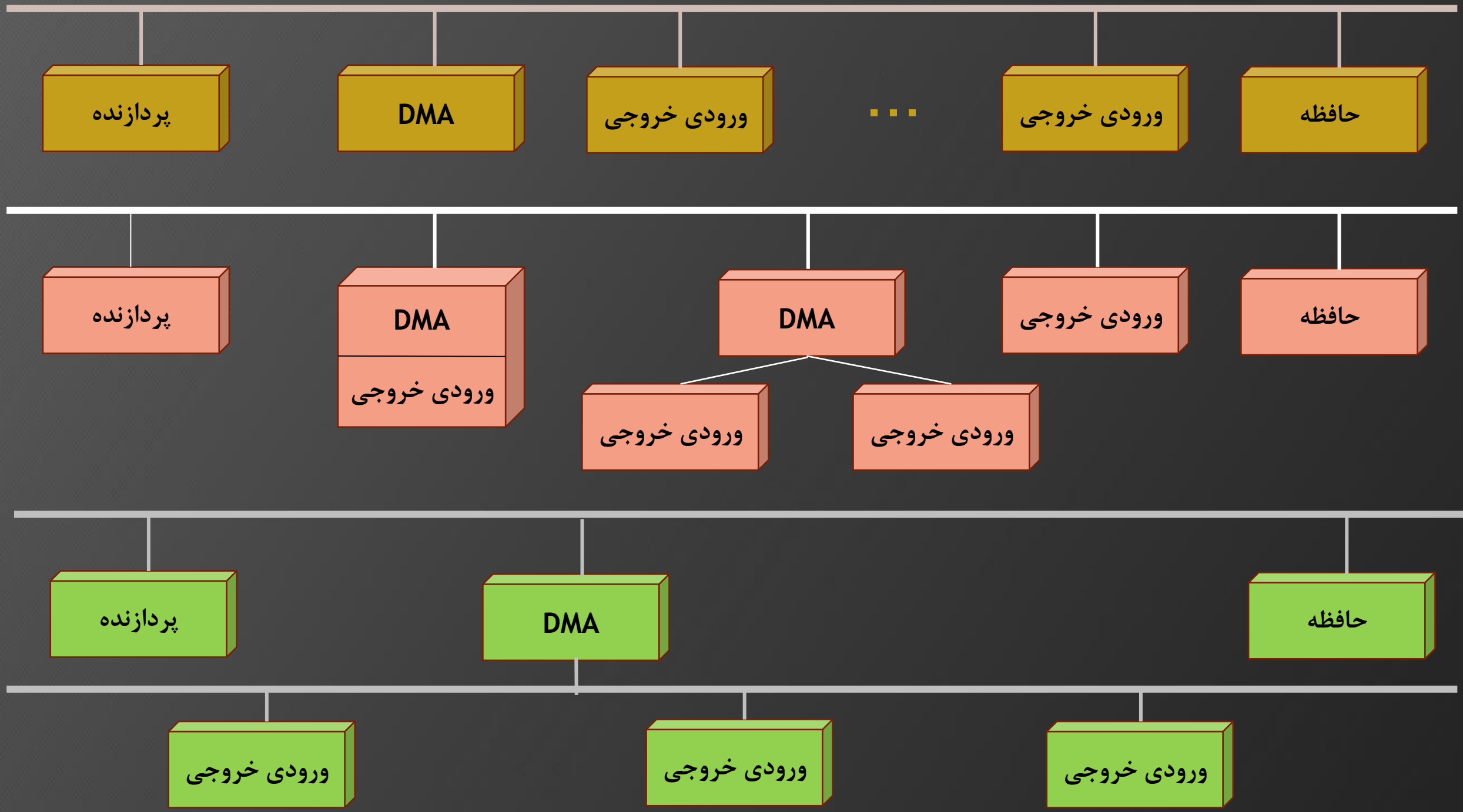
۶- مؤلفهٔ ورودی خروجی, حافظهٔ محلی خاص خودش را دارد.

دسترسی مستقیم به حافظه (Direct Memory Access):

عملکرد DMA:

وقتی پردازنده می‌خواهد بلوکی از داده‌ها را بخواند یا بنویسد, از طریق ارسال اطلاعات زیر به مؤلفهٔ DMA فرمانی را به آن صادر می‌کند:

- با استفاده از خط کنترلی خواندن یا نوشتن بین پردازنده و مولفهٔ DMA مشخص می‌شود درخواست خواندن است یا نوشتن.
- آدرس دستگاه ورودی خروجی مورد نظر از طریق خط داده‌ها ارسال می‌شود.
- آدرس محل شروع خواندن یا نوشتن در حافظه از طریق خط داده‌ها ارسال شده و مؤلفهٔ DMA آن را در ثبات خود ذخیره می‌کند.
- تعداد کلماتی که باید خوانده یا نوشته شود از طریق خطوط داده ارسال شده و در ثبات شمارش داده‌ها ذخیره می‌شود.


پردازنده
DMA
ورودی خروجی
ورودی خروجی
حافظه
پردازنده
DMA
ورودی خروجی
DMA
ورودی خروجی
ورودی خروجی
ورودی خروجی
حافظه
پردازنده
DMA
حافظه
ورودی خروجی
ورودی خروجی
ورودی خروجی

جلسه پانزدهم:

زمان‌بندی دیسک

آنچه در این جلسه می خوانید:

۱- مقدمه

۲- پارامترهای کارایی دیسک

۳- سیاست‌های زمان‌بندی دیسک

۴- RAID

مقدمه:

در طول سال‌های گذشته, افزایش سرعت پردازنده‌ها و حافظهٔ اصلی بیشتر از افزایش سرعت دسترسی به دیسک بوده است.

کارایی زیر سیستم حافظهٔ دیسک بسیار مورد توجه است و تحقیقات زیادی برای ارائه طرح‌هایی جهت بهبود کارایی انجام شده است.

پارامترهای کارایی دیسک:

جزییات واقعی عملیات ورودی خروجی دیسک به سیستم کامپیوتری, سیستم‌عامل و ماهیت کانال ورودی خروجی و سخت‌افزار کنترل‌کننده بستگی دارد.

در یک گرداننده دیسک سرعت خواندن و نوشتن به موارد زیر وابسته است:

- زمان پیگرد
- تأخیر چرخشی (درنگ دوران)
- زمان دسترسی
- زمان انتقال

سیاستهای زمان‌بندی دیسک:

نام	توصیف	ملاحظات
انتخاب بر اساس درخواست‌کننده		
Random	زمان‌بندی تصادفی	برای تحلیل و شبیه‌سازی
FIFO	خروج به ترتیب ورود	عادلانه‌تر از همه
PRI	اولویت برای فرایند	کنترل در خارج از مدیریت صف دیسک
LIFO	خروج به ترتیب عکس ورود	حداکثر محلی بودن و بهره‌وری از منابع
انتخاب بر اساس عنصر درخواست‌شده		
SSTF	کوتاه‌ترین زمان خدمت	بهره‌وری بالا, صف‌های کوتاه
SCAN	جلو و عقب رفتن بر روی دیسک	توزیع بهتر خدمات
C-SCAN	یک‌طرفه با بازگشت سریع	تغییر اندک خدمات
N-step-SCAN	پیمایش N رکورد در هر زمان	تضمین خدمات
F-SCAN	پیمایش N (اندازه صف در آغاز چرخه SCAN) رکورد در هر زمان	حساس به بار

:RAID (Redundant Array of Independent Disks)

با استفاده از چند مؤلفهٔ موازی, می‌توان کارایی بیشتری بدست آورد.

در مورد دیسک, این موضوع منجر به آرایه‌هایی از دیسک‌ها شده است که به طور مستقل و موازی عمل کنند. با استفاده از دیسک‌های چندگانه, تا زمانی‌که داده‌های مورد نیاز در دیسک‌های جداگانه‌ای قرار داشته باشند, درخواست‌های ورودی خروجی متعدد می‌توانند به طور موازی انجام شوند.

برای این منظور طرح RAID (آرایهٔ اضافیِ دیسک‌های مستقل) طراحی شد.

RAID شامل ۷ سطح است که از صفر تا ۶ شماره‌گذاری می‌شود.

طبقه	سطح	توصیف	دیسک‌های مورد نیاز	دسترس‌پذیری داده‌ها	ظرفیت انتقال دادهٔ ورودی خروجی بزرگ	نرخ درخواست ورودی خروجی کوچک
باریکه‌بندی	۰	بدون افزونگی	N	کمتر از یک دیسک	بسیار بالا	بسیار بالا برای خواندن نوشتن
آینه‌سازی	۱	آینه‌سازی شده	2N	بیشتر از RAID سطوح 4 ,3 ,2 و کمتر از سطح 6	بیشتر از یک دیسک برای خواندن, شبیه یک دیسک برای نوشتن	تا دو برابر یک دیسک برای خواندن, مشابه با یک دیسک برای نوشتن
دستیابی موازی	۲	افزونگی از طریق کُد همینگ	N + m	خیلی بیشتر از یک دیسک, بیش از RAID سطوح 4 ,3و 5	بیش از تمام موارد	تقرایباً دو برابر یک دیسک
	۳	در بین هم قرار گرفتن بیت توازن	N + 1	خیلی بیشتر از یک دیسک, قابل مقایسه با RAID سطوح 4 ,2 یا 5	بیش از تمام موارد	تقریباً دو برابر یک دیسک
دستیابی مستقل	۴	در بین هم قرار گرفتن بلوکی توازن	N + 1	خیلی بیشتر از یک دیسک, قابل مقایسه با RAID سطوح 3 ,2 یا 5	مشابه با RAID سطح صفر برای خواندن, خیلی کمتر از یک دیسک برای نوشتن	مشابه با RAID سطح صفر برای خواندن, خیلی کمتر از یک دیسک برای نوشتن
	۵	در بین هم قرار گرفتن بلوکی توازن توزیعی	N + 1	خیلی بیشتر از یک دیسک, قابل مقایسه با RAID سطوح 3 ,2 یا 4	مشابه با RAID سطح صفر برای خواندن, کمتر از یک دیسک برای نوشتن	مشابه با RAID سطح صفر برای خواندن, معمولاً کمتر از یک دیسک برای نوشتن
	۶	در بین هم قرار گرفتن بلوکی توزیعی دوگانه توازن	N + 2	بیش از تمام موارد	مشابه با RAID سطح صفر برای خواندن, کمتر از RAID سطح 5 برای نوشتن	مشابه با RAID سطح صفر برای خواندن, خیلی کمتر از RAID سطح 5 برای نوشتن

پایان