مقدمه

به نام خداوند جان آفرین حکیم سخن در زبان آفرین

جزوه‌ای که هم‌اکنون در اختیار شما قرار دارد به عنوان گام آخر در درس سیستم عامل ارائه شده است. در این جزوه سعی شده است که با ترکیب مناسب تست‌ها و متن درس یک مرور کلی بر آنچه تمام داوطلبان کنکور کارشناسی ارشد انجام داده‌اند باشند.

در این جزوه ابتدا به مفاهیم اولیه و پایه سخت افزار و سیستم عامل پرداخته می‌شود سپس به مفاهیم زمان‌بندی پرداخته می‌شود . در ادامه به بحث هم‌زمانی‌ها و در ادامه بن‌بست مورد بررسی قرار می‌گیرد.

در انتها به مفاهیم حافظه و در نهایت زمان‌بندی دیسک پرداخته می‌شود . در انتهای جزوه نیز تست‌های کنکور سراسری کامپیوتر و IT سال ۸۹ با پاسخ تشریحی قرار داده شده است.

از داوطلبان گرامی تقاضا می‌شود که به هرکدام از تستهای قرار داده شده در پایان فصل به چشم کلاس آموزشی نگاه کنند و تمام نکات موجود در حل تشریحی را به دقت کامل مطالعه فرمایید.

امید است که این جزوه بتوانند گام آخر و سازنده برای دانشجویان فهیم، باانگیزه و پر تلاش کنکور کارشناسی ارشد در درس سیستم عامل باشد و در پایان یادمان باشد که :

زندگی آتشگهی دیرینه برپاست اگر بیفروزیش رقص شعله‌اش تا بیکران پیداست
ورنه خاموش است و خاموشی گناه ماست

موفق، موید، پیروز، سربلند
ابوالفضل طرقی حقیقت
کاوه کاویان‌پور

در جدول ذیل دروس به سرفصلهای مهم آن طبقه بندی شده و مشخص شده است که در هر سال از هر مبحث چند تست سوال شده است و دانشجوی محترم می تواند زمان باقیمانده تا کنکور را با توجه به اهمیت مباحث مدیریت نماید.

رشته: کامپیوتر		درس: سیستم عامل						
ردیف	مبحث	۱۳۸۵	۱۳۸۶	۱۳۸۷	۱۳۸۸	۱۳۸۹	مجموع ۵ سال	نسبت از کل
		تعداد تست	تعداد تست	تعداد تست	تعداد تست	تعداد تست		
1	مفاهیم اولیه سیستم عامل و تعاریف آنها	0	1	0	0	0	1	4%
2	فرآیندها و زمانبندی پردازنده ها	1	2	2	2	4	11	44%
3	همروندی انحصار متقابل و همگام سازی	1	1	1	1	0	4	16%
4	بن بست	1	0	0	0	0	1	4%
5	مدیریت IO و دیسک	0	0	0	0	0	0	0%
6	مدیریت حافظه	1	0	0	0	0	1	4%
7	حافظه مجازی	2	2	1	1	1	7	28%
جمع		6	6	4	4	5	25	100%

فصل اول

مفاهیم پایه سخت افزار و سیستم عامل

در ابتدا جهت آشنایی با سیستم عامل به بررسی مهمترین مفاهیم ابتدایی سخت افزار کامپیوتر می‌پردازیم:

وقفه

گاهی رویدادی (در خارج از پردازنده یا حتی درون پردازنده) رخ می‌دهد که لازم است پردازنده روند عادی اجرای دستورالعمل‌های برنامه جاری را موقتاً قطع کرده و به آن رویداد پاسخ دهد و سپس برگردد و به اجرای برنامه قطع‌شده ادامه دهد. این رویداد وقفه نامیده می‌شود. پاسخ به وقفه به معنی اجرای رویه اداره‌کننده وقفه (Interrupt Handler) یا روال سرویس وقفه، ISR (Interrupt Service Routine) می‌باشد. هر وقفه یک ISR مخصوص در هسته سیستم‌عامل دارد. به طور کلی وقفه‌ها بر دو نوع‌اند:

- ✓ وقفه‌های سخت‌افزاری (یا خارجی) که از سوی یک سخت‌افزار خارج از CPU به صورت یک سیگنال ناهمگام (آسنکرون: در یک لحظه تصادفی که از قبل مشخص نیست در کجای برنامه جاری به وقوع می‌پیوندد) به پردازنده ارسال می‌شود. این وقفه‌ها به چند دسته تقسیم می‌شوند:
  ۱. وقفه‌های I/O
  ۲. خطای ماشین (machine Check) یا نقص سخت‌افزاری (Hardware fault)
  ۳. وقفه ساعت (Clock) یا زمان‌سنج (Timer)
  ۴. وقفه Restart
  ۵. سایر وقفه‌های خارجی (External)
- ✓ وقفه‌های نرم‌افزاری (یا داخلی) که به صورت همگام (سنکرون) در اثر اجرای دستورالعمل خاصی از برنامه جاری، خواسته یا ناخواسته، به وقوع می‌پیوندد. این وقفه‌ها نیز به چند دسته تقسیم می‌شوند:

۱. فراخوان‌های سیستمی (System call): این وقفه‌ها که فراخوان راهبری یا سرپرستی (SVC[1])، فراخوان ناظر یا فراخوان هسته نیز نامیده می‌شوند، عموماً وقتی رخ می‌دهند که برنامه سطح کاربر نیاز به استفاده از سرویس‌های سطح هسته سیستم‌عامل داشته باشد.

[1] Supervisor Call

۲. خطای برنامه (Program Check) یا استثناء (Exception): این وقفه‌ها وقتی رخ می‌دهد که دستورالعمل جاری برنامه بخواهد عملی را انجام دهد که اصولاً غیر مجاز است مانند خطای سرریز یا تقسیم بر صفر، یا اینکه دستورالعمل جاری برنامه سطح کاربر بخواهد عمداً یا سهواً عملی را انجام دهد که در مُد کاربر غیرمجاز است و فقط در مُد هسته امکان‌پذیر است، مانند نقص‌های حفاظتی از قبیل اجرای یک دستورالعمل ممتاز در مُد کاربر و مراجعه به یک آدرس نادرست یا غیر مجاز در حافظه.
۳. سیگنال (Signal): یک وقفه نرم‌افزاری است که توسط یک فرایند یا سیستم‌عامل یا کاربر به یک یا چند فرایند ارسال می‌شود.

نکته: سیستم‌های عامل مدرن، مبتنی بر وقفه (Interrupt driven) هستند. یعنی پس از راه‌اندازی کامپیوتر توسط سیستم‌عامل و سپردن CPU به برنامه کاربر، سیستم‌عامل بارها به علت وقوع وقفه‌های گوناگون سخت‌افزاری یا نرم‌افزاری در لابه‌لای اجرای فرایندهای سطح کاربر ظاهر می‌شود و به وظائف خویش می‌پردازد.

نکته: سخت‌افزار، یعنی خود CPU و در بعضی از مواقع، واحدهای سخت‌افزاری دیگری مانند MMU (واحد مدیریت حافظه)، مسئول کشف خطای برنامه (استثناء) است. در این صورت گفته می‌شود که یک وقفه نرم افزاری (تله) رخ داده است و برنامه خطاکار در تله هسته گرفتار شده است. بدیهی است که در این صورت، پردازنده برنامه خطاکار را قطع کرده و با ورود به مُد هسته به اجرای ISR مربوطه می‌پردازد. سیستم‌عامل می‌تواند مثلاً یک رونوشت از حافظه برنامه در یک فایل ایجاد کرده و پس از چاپ یک پیام خطا، فرایند خطاکار را بکُشد (Kill).

نکته: حتی اگر کاری برای اجرا وجود نداشته باشد، پردازنده یک برنامه ساده حاوی یک حلقه انتظار مشغول (Busy waiting) را اجرا خواهد کرد.

نکته: پردازنده‌های امروزی امکانات خاصی را در اختیار طراحان سیستم‌عامل می‌گذارند تا برای حفاظت سیستم‌عامل و نیز حفاظت برنامه‌ها در برابر یکدیگر مورد استفاده قرار دهند.

منابع سیستم:

منابع سیستم به دو نوع مختلف طبقه بندی می‌شوند:

۱) Physical : حافظه ـ دیسک ـ ماوس ـ چاپگر و ...

۲) Logical : اطلاعات ـ فایل‌ها و ...

سیستم عامل دو وظیفه (هدف) اصلی دارد:

۱) سیستم عامل استفاده از کامپیوتر را ساده می‌سازد.

۲) وظیفه دوم سیستم عامل مدیریت منابع (Resource Managment) می‌باشد.

سیستم عامل واقعیت سخت افزار را از دید کاربران (برنامه نویسان) مخفی می‌ساد. از این رو به O.S ماشین توسعه یافته (Extended Machine) یا ماشین مجازی (Vitual Machine) گفته می‌شود.

به قسمت اصلی سیستم عامل که وظایف مهم آن را انجام می‌دهد هسته یا kernel گفته می‌شود.

مختصری در مورد تاریخچه سیستم عامل:

۱) در نسل اول کامپیوترها (۱۹۵۵ ـ ۱۹۴۵) که از لامپ خلاء برای ساخت آن‌ها استفاده می‌شد، زبان‌های برنامه نویسی (حتی اسمبلی) ابداع نشده بودند و سیستم عامل نیز اصلاً وجود نداشت و برنامه نویسان تنها در یک فاصله زمانی مشخص حق استفاده از کامپیوترهای بزرگ و گران قیمت را داشتند.

۲) نسل دوم سستم عامل از ترانزیستور ساخته شدند.

نحوه کار: کاربر ← نوشتن برنامه به زبان ASM یا فرترن ← Card Punch ← روی کارت ← یک job تحویل اپراتور ← کامپیوتر ← خروجی معمولا چاپی

سیستم‌های دسته‌ای Offline spooling

در این سیستم‌ها، کارهای ورودی به وسیله دستگاه کارت خوان یک کامپیوتر کوچک و نسبتاً ارزان مانند IBM 1401 خوانده می‌شد و از طریق یک نوار گردان بر روی یک نوار مغناطیسی ذخیره می‌گشت. از طرف دیگر یک ماشین گران قیمت، مانند IBM 7094 برای پردازش و محاسبات واقعی استفاده می‌شد. وقتی که یک دسته کار اجرا می‌شد، اپراتور نوار خروجی را برمی‌داشت و به ماشین 1401 دیگری منتقل می‌کرد که به چاپگر متصل بود.

نکته: فزایش بهره‌وری CPU گران‌قیمت، سهولت استفاده از راه دور و سادگی عملیات از نقاط قوت سیستم‌های دسته‌ای Offline spooling به شمار می‌روند.

نکته: عدم استفاده بهینه از CPU، ارتباط غیر مستقیم با کاربر، طولانی بودن زمان برگشت کارها، نیاز به سخت‌افزار اضافه و عدم لحاظ کردن اولویت‌ها از معایب اساسی سیستم‌های دسته‌ای Offline spooling می‌باشند.

سیستم‌های دسته‌ای Online spooling

با ظهور دیسک‌های سخت، تحول عظیمی در سیستم‌های عامل پدید آمد. سیستم‌های دسته‌ای جدید، بلافاصله پس از ورود کارها به اتاق کامپیوتر، می‌توانستند کارت‌ها را خوانده و به دیسک منتقل نمایند. بنابراین هرگاه یک کار در حال اجرا به پایان می‌رسد، سیستم‌عامل می‌تواند با سرعت بیشتری یک کار جدید را از روی دیسک برداشته و در فضای کاربر حافظه اصلی بار نماید و سپس آن را به اجرا درآورد. این تکنیک که **spooling**[۲] (عملیات پیوسته، مستقیم و هم‌زمان دستگاه‌های جانبی) نامیده شد، برای خروجی نیز به کار گرفته شد. یعنی اسناد خروجی مستقیماً به چاپگر فرستاده نمی‌شد، بلکه ابتدا بر روی دیسک ذخیره می‌شد تا هرگاه چاپگر، خروجی قبلی را به اتمام برساند در طی یک یا چند مرحله از دیسک به بافر چاپگر منتقل شوند. در Online spooling، ارتباط CPU با دستگاه‌های جانبی مستقیم است و دیگر نیازی به 1401ها و نوارگردان‌های اضافی و حمل نوارها توسط اپراتورها نبود. هرگاه کلمه spooling را به تنهایی شنیدید، منظور همان Online spooling است. در این سیستم‌ها کنترل‌کننده‌ها، بافرها، دیسک و مکانیسم وقفه، چهار عنصر بنیادین سیستم را تشکیل می‌دهند.

نکته: افزایش بهره‌وری یا راندمان CPU و دستگاه‌های I/O، عدم نیاز به سخت‌افزار اضافه و امکان زمان‌بندی دلخواه کارها و لحاظ کردن اولویت‌ها از نقاط قوت این سیستم‌ها به شمار می‌روند.

نکته: یکی از معایب اساسی این سیستم‌ها این است که اگرچه بهره‌وری CPU خیلی بهتر شده است اما هنوز کاملاً بهینه نیست (سربار سوئیچ به دلیل مدیریت حافظه تک برنامگی). همچنین ارتباط با کاربر هنوز غیر مستقیم است و هنوز زمان برگشت کارها طولانی است.

برای جمع بندی مطالب فوق داریم :

معایب سیستم offline Spooling :

یکی از معایب روش offline Spooling زیاد بودن زمان برگشت (گردش) (Turnaroud Time) است.

دومین عیب روش offline Spooling نیاز به داشتن سخت افزار اضافی است.

سومین عیب روش offline Spooling عدم وجود اولویت درآن است.

[2] Simultaneous Peripherals Operation On-Line (SPOOL)

مزایای سیستم offline Spooling :

۱ـ راندمان بهتر

۲ـ عملیات ساده‌تر

۳ـ سهولت برای استفاده از راه دور

سیستم‌های دسته‌ای چندبرنامگی

در این سیستم‌ها، حافظه را به چند تکه تقسیم بندی کرده و یک کار مجزّا را در هر پارتیشن قرار می‌دادند. وقتی که یک کار خاتمه می‌یافت یا برای تکمیل عملیات I/O منتظر می‌ماند، بلافاصله پردازنده به یکی دیگر از کارهای درون حافظه، سوئیچ می‌شد.
نکته: استفاده بهینه از CPU و سایر منابع از نقاط قوت سیستم‌های دسته‌ای چندبرنامگی می‌باشند.
نکته: از معایب این سیستم‌ها می‌توان به این نکات اشاره کرد که ارتباط با کاربر هنوز غیر مستقیم است، هنوز زمان برگشت کارها طولانی است. پیچیدگی سیستم (حاصل از نداشتن ساختار طراحی مناسب)، عیب دیگر آن است.

سیستم‌های اشتراک زمانی (چندبرنامگی تعاملی)

اگر چه سیستم‌های دسته‌ای چندبرنامگی، موجب استفاده بهینه از پردازنده و سایر منابع سیستم می‌شود، اما آنها به هر حال هنوز سیستم‌های دسته‌ای بودند و ارتباطشان با کاربر به صورت غیرمستقیم بود. برای حل این مشکل، نیازمند ارتباط مستقیم (Online) کاربران با کامپیوتر هستیم. ظهور ترمینال‌ها (کنسول‌های ارتباط مستقیم)، بستر سخت‌افزاری لازم برای حل این مشکل را ایجاد کرد و گونه جدیدی از سیستم‌های چندبرنامگی با قابلیت تعامل مستقیم با کاربر (در نقطه مقابل سیستم‌های دسته‌ای) پدید آمد که سیستم‌های اشتراک‌زمانی (Timesharing) یا چندوظیفه‌ای (Multitasking) نامیده شد.
در سیستم‌های اشتراک زمانی از زمان‌بندی غیرانحصاری مبتنی بر کوانتم استفاده می‌شود. این کار با تقسیم وقت CPU به برش‌های زمانی (Time slice) یا کوانتم‌های (Quantum) نسبتاً کوچک و معمولاً (نه الزاماً) مساوی و تخصیص این برش‌های زمانی به فرایندهای آماده صورت می‌گیرد.
نکته: ارتباط مستقیم با کاربر، تضمین زمان پاسخ کوتاه برای کارهای کوچک واشکال‌زدایی ساده برنامه‌ها با اجرای خط به خط آنها و امکان مشاهده مقادیر متغیرها از نقاط قوت سیستم‌های اشتراک زمانی می‌باشند.
نکته: از معایب اساسی سیستم‌های اشتراک زمانی این است که سربار سوئیچ‌های اضافی، مقدار کمی از بهره‌وری پردازنده نسبت به سیستم‌های دسته‌ای چندبرنامگی می‌کاهد، اما این هزینه نسبت به مزایای سیستم قابل قبول است. همچنین این سیستم‌ها نیاز به حفاظت بیشتری دارند.

برای جمع بندی در سیستمهای اشتراک زمانی (Time Sharing) داریم :

- ✓ در چند برنامگی اجرای یک برنامه تا هنگام عملیات I/O ادامه پیدا می‌کند، سپس عمل I/O آن شروع شده و همزمان CPU اجرای برنامه دیگری را آغاز می‌کند. ولی در Spooling می توان چند کار را همزمان اجرا کرد.
- ✓ در سیستم‌های online پردازنده مستقیماً با دستگاه‌های I/O درارتباط است، ولی در سیستم‌های offline یا غیر مستقیم، پردازنده با دستگاه‌های I/O به طور مستقیم در ارتباط نیست.
- ✓ سیستم‌های Time Sharing در نسل سوم کامپیوترها معمول شدند و در واقع تعمیمی از سیستم چند برنامگی هستند.
- ✓ در سیستم time Sharing کاربر به کمک Terminal که شامل Monitor , Keyboard است با کامپیوتر به صورت Interactive رابطه برقرار می‌سازد.

مزایای Time Sharing نسبت به Multi Programming :

۱ـ Debug راحت‌تر

۳ـ زمان پاسخ کمتر (Response Time)

در سیستم اشتراک زمانی فقط یک پردازنده وجود دارد که توسط مکانیزم‌های زمان‌بندی بین برنامه‌های مختلف کاربرها با سرعت زیاد (مثلاً در حد میلی‌ثانیه) Switch می‌شود.

سیستم‌های دسته‌ای برای اجرای برنامه‌های بزرگ که نیاز محاوره‌ای کمی دارند مناسب است. ولی سیستم‌های اشتراک زمانی برای مواردی که Response Time کوتاه لازم است استفاده می‌شوند.

در سیستم اشتراک زمانی وجود یک سیستم فایل ضروری است. زیرا نمی‌توان در هر بار اجرای کار مدارک بزرگی را توسط ترمینال‌ها وارد کامپیوتر کرد.

اهداف سیستم‌عامل‌های اولیه به دلیل گرانی CPU ماکزیمم کردن درصد استفاده CPU بود. اما امروزه با توجه به هزینه اندک سخت افزار هدف سیستم عامل تغییر کرده و به سمت راحتی کاربر رفته است.

Multi Threading:

Threads (نخ‌ها)

در حقیقت، در سیستم‌های مدرن، اغلب وضعیت‌هایی وجود دارد که در آن‌ها علاقمند به وجود چندین رشته کنترلی هستیم که به صورت هم‌روند در یک فضای آدرس در حال اجرا هستند، چنان‌چه گویی فرایندهایی مجزا و مستقلی هستند؛ جز این‌که فضای آدرس و برخی از منابع را به صورت اشتراکی استفاده می‌کنند و می‌توانند از مدیریت کارامدتری برخوردار باشند.

نکته: مزایای سیستم‌های چندنخی عبارت‌اند از:

۱. افزایش سرعت اجرا و زمان پاسخ فرایندهایی که مکرراً مسدود می‌شوند (حتی با وجود یک پردازنده).
۲. افزایش کارایی (سرعت) حاصل از ارتباط و تبادل داده سریع بین نخ‌های درون یک فرایند با استفاده از فضای آدرس (داده‌های) مشترک؛ بدون دخالت هسته و نگرانی‌های امنیتی.
۳. بهره‌گیری کافی از توازی، تسریع در اجرای فرایندهای بزرگ و ایجاد توازن بار در سیستم‌های چندپردازنده.
۴. افزایش ماژولاریتی و درجه دانه‌بندی در برنامه‌های کاربردی بزرگ (شکستن فرایندهای بزرگ به نخ‌های ریزتر به منظور ساده‌سازی طراحی، پیاده‌سازی، عیب‌یابی، نگهداری و افزایش قابلیت اطمینان و انعطاف‌پذیری)
۵. افزایش کارایی به علت تعویض سریع نخ‌ها نسبت به تعویض کند فرایندها در بعضی از سیستم‌های چندنخی
۶. بعضی از برنامه‌ها ذاتاً از بخش‌های کاملاً مستقل تشکیل می‌شوند که جدا نکردن آنها باعث پیچیدگی بیش از حد برنامه‌نویسی می‌شود.

نکات مهم در مورد Multi Threading:

- ✓ در تکنیک چند نخی (Multi Threading) یک فرآیند (Process) که برنامه ای در حال اجراست، می‌تواند به بخش‌ها یا نخ‌هایی تقسیم شود که می‌تواند به صورت همزمان اجرا شوند.
- ✓ برنامه‌هایی که چند وظیفه مستقل از هم را انجام می‌دهند می‌توانند به صورت چند نخی نوشته شوند.
- ✓ گاهی اوقات به سیستم‌های Multi Threading سیستم‌های Multi Tasking هم گفته می‌شود.
- ✓ فرآیند می‌تواند مجموعه‌ای از یک یا چند نخ باشد. کلیه اطلاعات مربوط به هر Process در یکی از جداول سیستم به نام PCB ذخیره می‌شود.

- ✓ سیستم عامل Multi Tasking برای اجرای چند نخ بر روی یک CPU و سیستم عامل Multi Processor برای اجرای چند نخ بر روی چند CPU به کار می‌روند.
- ✓ در چند پردازنده‌ای، CPUها باید بتوانند از حافظه، I/O و Bus سیستم به صورت اشتراکی استفاده کنند.
- ✓ گرچه Multi threading و Multi Processing امکانات مستقلی هستند ولی معمولاً با هم پیاده سازی می‌شوند. حتی در یک ماشین تک پردازنده‌ای، چند نخی کارایی را افزایش می‌دهد.

سایر نکات مهم در مورد مفاهیم سیستم عامل:

- ✓ سیستم‌های بی‌درنگ (Real Time) معمولاً به عنوان یک کنترل کننده در یک کاربرد خاص استفاده می‌شود. سیستم در این حالت می‌بایست در زمانی مشخص و معین حتماً جواب مورد نظر را بدهد.
- ✓ در سیستم‌های بی‌درنگ معمولاً وسایل ذخیره سازی ثانویه وجود ندارد و به جای آن از حافظه‌های ROM استفاده می‌شود. (به دلیل سرعت پائین حافظه جانبی) .
- ✓ سیستم‌های بی‌درنگ با سیستم اشتراک زمانی تناقض دارند، لذا نمی‌توانند هر دو، توأماً وجود داشته باشند. به دلیل نیاز به پاسخ دهی سریع و تضمین شده سیستم‌های بلادرنگ از حافظه مجازی اشتراک زمانی استفاده نمی‌کنند.
- ✓ در برخی کاربردها (مثل کنترل صنعتی) در کامپیوترها ازسیستم عامل استفاده نمی‌شود، زیرا وجود واسطه‌ای مثل O.S باعث کند شدن پاسخ سیستم به یک اتفاق خواهد شد.
- ✓ بخش اعظم سیستم عامل UNIX و همچنین Window SNT به زبان C که قابل حمل (Portable) به کامپیوترهای مختلف است نوشته شده است.
- ✓ عملیات offline Spooling ساده‌تر از online Spooling است.
- ✓ سیستمی که در آن یک پردازش، پردازش دیگری را تولید می‌کند چند وظیفه‌ای (Multi Tasking) است.
- ✓ نقطه ضعف اصلی Multiprogramming در Context Switch بین برنامه‌هاست که سربار سیستم را بالا می‌برد.
- ✓ در یک سیستم عامل Multi Tasking: وقفه‌های خارجی نسبت به سایر وقفه‌ها اولویت بالاتری دارند.
- ✓ تعویض حالت بدین معنی است که پردازنده از حالت کابر به حالت هسته می‌رود.
- ✓ ممکن است تعویض حالت بدون تغییر حالت فرآیند جاری صورت گیرد. اما تعویض فرایند مستلزم تعویض حالت است.
- ✓ اولویت دسترسی DMA به حافظه اصلی بیش از اولویت پردازنده در دسترسی به حافظه است.
- ✓ غیرفعال نمودن وقفه‌ها و تغییر در نقشه حافظه فقط در Kernel Mode مجاز می‌باشند.
- ✓ پردازش وقفه پس از تکمیل دستورالعمل جاری صورت می‌گیرد.
- ✓ در Multi Programming فقط یک پردازنده وجود دارد که به کمک مکانیزم‌های وقفه بین کارهای CPU limited و I/O Limited سوئیچ می‌کند و به ظاهر برنامه‌ها موازی و همزمان اجرا می‌شوند.

تست‌ها

۱. در یک سیستم تک پردازنده‌ای اشتراک زمانی (Time Shared) و قبضه شونده (Preemptive) یکی از n پردازنده موجود به شکل زیر تعریف شده است:

```
L1 : instruction 1
     go to L1
```

زودترین و دیر ترین امکان زمان اجرای دستورالعمل instruction 1 عبارت است از:(کارشناسی ارشد کامپیوتر ـ دولتی ۸۳)

۱) بلافاصله؛ هیچ‌وقت
۲) $(n-1) \times t$ ؛ $n \times t$ (t طول برهه زمانی بر حسب میلی ثانیه)
۳) $(n-1) \times t$ ؛ هیچ‌وقت
۴) بلافاصله ؛ $n \times t$

۲. جدول زیر زمان‌های لازم برای ورود، محاسبه و خروج 3 کار را در یک سیستم دسته‌ای (Batch) با Spooling نشان می‌دهد. حداقل کل زمان مصرفی برای اجرای هر 3 کار به شرط آنکه ترتیب ورود کارها تعیین کننده ترتیب پردازش و ترتیب خروجی آن‌ها باشد، چقدر است؟ (کارشناسی ارشد کامپیوتر ـ آزاد ۷۹)

	زمان ورود	زمان پردازش	زمان خروج
کار 1	5	4	1
کار 2	2	2	3
کار 3	5	3	2

۱) 27 ۲) 10 ۳) 20 ۴) 17

۳. تکنیک Spooling به چه معناست؟ (متفرقه)

۱) به‌کارگیری حافظه ثانویه به عنوان میانگیر حافظه هنگام پر شدن حافظه اصلی.
۲) به‌کارگیری حافظه ثانویه به عنوان میان‌گیر هنگام انتقال داده‌ها بین وسایل جانبی و پردازنده‌ها.
۳) به‌کارگیری حافظه ثانویه جهت ذخیره محاسبات پردازشگرها هنگام پر شدن حافظه اصلی.
۴) به‌کارگیری حافظه اصلی به عنوان یک میانگیر حافظه ثانویه جهت کاهش تاخیرهای پردازش.

۴. مزیت Online spooling نسبت به Offline spooling در چیست؟ (متفرقه)

۱) عملیان آن ساده‌تر است.
۲) دسترسی با اولویت امکان‌پذیر است.
۳) ارتباط مستقیم با کاربر.
۴) در استفاده از راه دور، سهولت آن بیش‌تر است.

پاسخنامه

۱. گزینه ۱ درست است.

هدف این سؤال این است که ببیند آیا شما بین مفهوم اشتراک‌زمانی و الگوریتم زمان‌بندی Round Robin فرقی قایل می‌شوید یا خیر. اگر زمان‌بندی عادلانه RR مطرح باشد و فرایند در ابتدای صف باشد، بلافاصله اجرا می‌شود و اگر در انتهای صف باشد، پس از زمان حداکثر $(n-1) \times t$ اجرایش شروع می‌شود. اما اگر برای مثال، سیستم اشتراک زمانی از الگوریتم زمان‌بندی اولویت استفاده کند ممکن است مشکل قحطی رخ دهد و فرایند مورد نظر ما اولویت پایین داشته باشد و فرایندهای با اولویت بالاتر، دایماً پردازنده را در اختیار بگیرند و آن را دچار گرسنگی نمایند.

۲. گزینه ۴ درست است.

در سیستم‌های Spooling (البته Online Spooling و چندبرنامگی دسته‌ای)، امکان پردازش یک کار، همزمان با I/O سایر کارها وجود دارد و در این صورت ورود 3 کار از کارتخوان در زمان‌های (0 تا 5)، (5 تا 7) و (7 تا 12) انجام می‌شود. پردازش کارها در زمان‌های (5 تا 9)، (9 تا 11) و (12 تا 15) صورت می‌گیرد و خروجی آن‌ها در زمان‌های (9 تا 10)، (11 تا 14) و (15 تا 17) انجام می‌شود. خروج آخرین کار در لحظه 17 است.

۳. گزینه ۲ درست است.

در Spooling از دیسک برای ذخیره موقت کارهای ورودی و مدارک خروجی استفاده می‌شود.

۴. گزینه ۲ درست است.

البته مزایای مهم‌تری نیز وجود دارند، مانند عدم نیاز به ماشین‌ها و دستگاه‌های اضافی و حذف نقش اپراتورها در انتقال نوارها و ارتباط مستقیم بین پردازنده و دستگاه‌های جانبی و از همه مهم‌تر ایجاد بستر لازم برای چندبرنامگی و نهایتاً استفاده بهینه از منابع.

فصل دوم

مدیریت فرآیندها

فرایندها

یک **فرایند** (Process)، اساساً یک برنامه در حال اجرا است. منظور از برنامه در حال اجرا، کاری است که توسط زمان‌بند کار انتخاب و وارد گردونه اجرا شده است؛ ولی هنوز پایان نیافته و از سیستم خارج نشده است؛ اما الزاماً در حال حاضر CPU را در اختیار ندارد. معمولاً در اغلب سیستم‌های عامل، مفاد یا متن (Context) یک فرایند شامل موارد زیر است:

- ✓ فضای آدرس حافظه (متشکل از متن برنامه، داده و پشته) یا تصویر حافظه
- ✓ PCB شامل اطلاعات شناسایی و کنترلی فرایند و محتویات کلیه رجیسترهای فرایند به‌ویژه PC و PSW
- ✓ منابع تخصیص یافته به فرایند

البته گاهی از نظر منطقی این اجزاء را به سه بخش کُد (شامل متن برنامه)، اجرایی (شامل داده‌ها، پشته و PCB) و منابع تقسیم می‌کنند. هر فرایند، دارای **فضای آدرس** (Address space) مخصوص به خود است. فضای آدرس شامل برنامه اجرایی، داده‌های برنامه و پشته آن است.

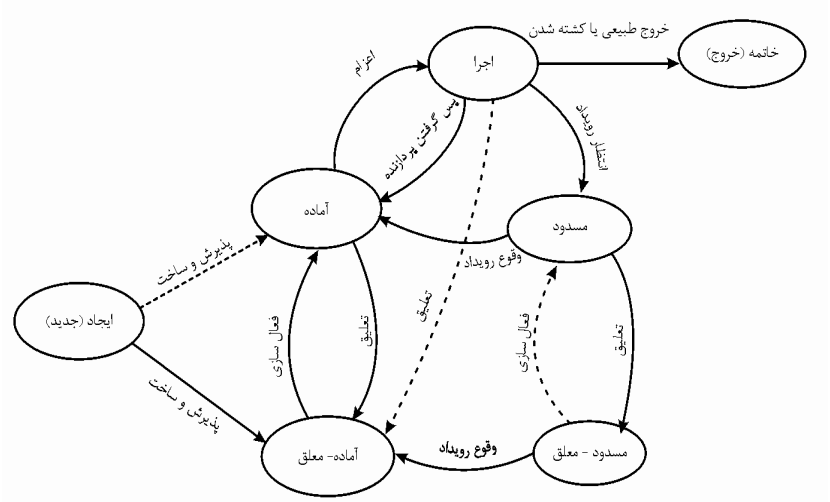
همچنین هر فرایند برای خودش دارای یک مجموعه از رجیسترها است که شامل PC، SP، PSW و دیگر رجیسترهای سخت‌افزاری می‌شود و نیز اطلاعات متنوع دیگری که برای اجرای برنامه مورد نیاز است. این اطلاعات در یکی از جداول سیستم‌عامل به نام **جدول فرایند** (Process table) ذخیره می‌شود. این اطلاعات مهم، شامل شناسه فرایند، شناسه کاربر مربوطه، شناسه گروه مربوطه، اولویت فرایند، وضعیت فرایند، میزان CPU مصرفی، اشاره‌گر برنامه، PSW، اشاره‌گر پشته، سایر رجیسترها و اشاره‌گرهای فایل و غیره می‌باشند. تمامی اطلاعات مربوط به هر فرایند، در یک درایه (مدخل یا سطر) از جدول فرایند در فضای هسته درون حافظه اصلی ذخیره می‌شود. معمولاً این جدول یک آرایه (یا لیست پیوندی) از **ساختار**ها (Structure) است که هر عضو آن مربوط به یکی از فرایندهای درون سیستم می‌باشد. در بعضی از متون سیستم‌عامل به این ساختار **PCB**[3] گفته می‌شود.

[3] Process Control Block

بنابراین هر فرایند، اولاً شامل فضای آدرس مربوط به خود است که معمولاً **تصویر حافظه** (core image) نامیده می‌شود (ریشه این واژه در اینجا است که در گذشته از حافظه‌های چنبره مغناطیسی (Core memory) به عنوان حافظه اصلی استفاده می‌شد نه RAM) و ثانیاً شامل یک درایه مربوط به خود در جدول فرایند می‌باشد که رجیسترهایش (در کنار چیزهای دیگر) در آنجا قرار دارد.

چند فراخوان سیستمی کلیدی برای مدیریت فرایندها وجود دارد که برای ایجاد و خاتمه دادن به فرایندها به کار می‌روند. در برخی از سیستم‌های عامل، یک فرایند اجازه دارد یک یا چند فرایند دیگر را ایجاد نماید که به آنها **فرایند فرزند** (Child process) گفته می‌شود. فرایندهای فرزند نیز می‌توانند یک یا چند فرایند فرزند دیگر ایجاد کنند و به سرعت یک ساختار درختی از فرایندها تشکیل می‌شود.

وضعیت‌های (حالات) فرایند



فرآیند در وضعیت اجرا (Runing) قرار می‌گیرد اگر پردازنده را در اختیار داشته باشد.

فرآیند در وضعیت آماده (Ready) قرار می‌گیرد اگر منابع لازم به آن داده شود و در انتظار ردازنده باشد.

فرآیند در وضعیت مسدود (Wait) قرار می‌گیرد اگر در حال انتظار برای وقوع رویدادی جهت از سر گرفتن اجرای خود باشد، مثلاً در انتظار پایان یافتن یک عمل I/O باشد.

*فرآیند در وضعیت آماده و معلق (Suspend Ready) قرار می‌گیرد اگر حافظه اصلی از آن گرفته شود و بر روی حافظه ثانوی قرار گیرد و به محض باز شدن در حافظه اصلی در وضعیت آماده قرار گیرد.

فرآیند در وضعیت مسدود و معلق (Suspend wait) قرار می‌گیرد اگر در حافظه ثانوی قرار گیرد و منتظر حادثه‌ای باشد.

اختصاص یافتن پردازنده به اولین فرآیند در لیست فرآیندهای آماده را توزیع یا Dispatch انجام می‌گیرد.

معیارهای مقایسه الگوریتم‌های زمان بندی

۱ـ بهره‌وری پردازنده: در صفر زمانی که پردازنده مشغول می‌باشد.

۲ـ توان عملیاتی (Through put) : مقدار کار انجام شده در واحد زمان.

۳ـ زمان بازگشت (Turn Around Time): فاصله زمانی بین لحظهٔ تحویل تا لحظه تکمیل آن می‌باشد.

مدت زمانی که Job در سیستم قرار می‌گیرد.

Finish Time Arriral Time
$RT = F.T - A.T$
Respose Time

۴ـ Response Time (زمان پاسخگویی) : فاصله زمانی ارائه یک تقاضا تا شروع دریافت پاسخ آن می‌باشد.

الگوریتم‌های زمان بندی پردازنده

الف) الگوریتم‌های Non – Preemptive (انحصاری):

۱ـ الگوریتم FIFO یا FCFS:

در این الگوریتم، فرایندها CPU را به ترتیبی که درخواست می‌کنند، در اختیار می‌گیرند. هنگامی که یک فرایند بلوکه، آماده می‌شود مانند کاری که تازه وارد شده، در انتهای صف قرار می‌گیرد.

نکته: اهداف و مزایای الگوریتم FCFS عبارت‌اند از ۱) سادگی ۲) قابلیت پیاده‌سازی ۳) انصاف و عدالت ۴) عدم وجود قحطی (گرسنگی)

نکته: معایب و نقاط ضعف الگوریتم FCFS عبارت‌اند از ۱) میانگین زمان‌های برگشت و انتظار نسبتاً بالا است.

۲ـ الگوریتم Shortest Job First :

وقتی که چند کار با اهمیت یکسان برای اجرا شدن در یک صف ورودی قرارمی‌گیرند، زمان‌بند باید **ابتدا کوتاهترین کار (SJF[4])** را انتخاب نماید.

نکته: این الگوریتم انحصاری است و به کارهای کوچک اولویت می‌دهد.

نکته: میانگین زمان‌های برگشت و انتظار حداقل است (اگر کارها همزمان وارد شده باشند).

نکته: این الگوریتم، مشکل پیاده‌سازی دارد (نیاز به زمان اجرای کارها)

نکته: این الگوریتم، مشکل قحطی (گرسنگی) کارهای طولانی را دارد.

تخمین زمان فرآیند

$$S_{n+1} = (1-\alpha) S_n + \alpha t_n$$

T_n: مقدار واقعی که در مرحله n اجرا کردیم

S_n: مقدار تخمینی برای مرحله n

S_1: را برابر صفر فرض می کنیم $0<\alpha<1$

4_ Shortest Job First

۳ـ الگوریتم Highest Response Ratio Next :

در الگوریتم انحصاری «**بالاترین نسبت پاسخ**» (**HRRN**[5])، هرگاه فرایند جاری تکمیل یا بلوکه گردد، کاری که در بین کلیه کارهای آماده دارای بیشترین مقدار «نسبت پاسخ» باشد، برای اجرا انتخاب می‌شود.

$$\text{نسبت پاسخ هر کار} = \frac{w+s}{s} = \frac{w}{s} + 1$$

که در آن w، معرف زمان انتظار کار «از لحظه ورود تا کنون» و s، نشان‌دهنده زمان سرویس (اجرای) آن کار می‌باشد. در این‌جا نیز مانند الگوریتم SJF، کارهای کوتاه‌تر در شرایط مساوی، زمان انتظار یکسان، نسبت به کارهای طولانی اولویت دارند (مخرج کسر $\frac{w}{s}$ برای آنها کوچک‌تر است) تا با این رفتار SJFگونه خود میانگین زمان انتظار را کاهش دهد. البته برای اینکه این مزیت به قیمت ایجاد قحطی تمام نشود، شبیه الگوریتم FCFS به کارهایی که بیشتر منتظر شده‌اند نیز اهمیت می‌دهد (صورت کسر $\frac{w}{s}$ برای آنها بزرگ‌تر است).

ب) الگوریتم‌های Preemptive (غیرانحصاری):

۱ـ نوبت چرخشی یا Round Robin:

هر فرایند، یک بازه زمانی به نام **کوانتم** (ذره یا تکه کوچک) اختصاص داده می‌شود (که برش زمانی نیز نامیده می‌شود) تا در آن کوانتم بتواند اجرا شود. اگر فرایند در پایان کوانتم، هنوز در حال اجرا باشد، CPU از آن گرفته شده و به فرایند دیگری واگذار می‌شود (زمان‌بندی غیرانحصاری). اگر فرایند قبل از اتمام کوانتم به پایان رسد یا این که بلوکه شود، بدیهی است که بلافاصله عمل سوئیچ CPU به یک فرایند دیگر صورت خواهد گرفت.

نکته: اهداف و مزایای الگوریتم RR عبارت‌اند از ۱) تضمین زمان پاسخ مطلوب برای کارهای معمولی کوچک. ۲) نداشتن قحطی۳) انصاف وعدالت ۴) سادگی

نکته: معایب و نقاط ضعف الگوریتم RR عبارت‌اند از ۱) سربار تعداد زیاد تعویض متن و کاهش بهره‌وری پردازنده ۲) میانگین زمان پاسخ و انتظار بالا ۳) نیاز به دقت زیاد در تعیین اندازه کوانتم.

۲ـ Shortest Remaining Time First (SRTF) :

نسخه غیر انحصاری از «ابتدا کوتاه‌ترین کار»، الگوریتم «**کوتاه‌ترین زمان باقی‌مانده**» (**SRT**[6]) می‌باشد. در این الگوریتم، زمان‌بند فرایندی را انتخاب می‌کند که زمان باقی‌مانده اجرای آن از همه کوتاه‌تر باشد.

نکته: میانگین زمان‌های برگشت و انتظار، حداقل است.

نکته: این الگوریتم، مشکل پیاده‌سازی دارد (نیاز به زمان اجرای کارها)

نکته: این الگوریتم، مشکل قحطی (گرسنگی) کارهای طولانی را دارد.

۳ـ Multi Level feed back Queue &Multi Level Queue :

این روش با ایجاد **صف‌های چندگانه (Multiple Queues)**، کلاس‌های اولویت ایجاد می‌کند. همه فرایندهای تازه وارد در انتهای بالاترین صف (بالاترین کلاس اولویت) قرار می‌گیرند. فرایندها در بالاترین کلاس، به مدت یک کوانتم اجرا می‌شوند. فرایندهای کلاس بعدی ۲ کوانتم، فرایندهای کلاس بعدی ۴ کوانتم و به همین ترتیب، فرایند درون صف Q_n، 2^n کوانتم می‌گیرد. اگر یک فرایند از تمامی

5_ Highest Response Ratio Next (HRRN or **HRN**)

6_ Shortest Remaining Time [in other text books: (**SRTN**: Shortest Remaining Time Next) or (**SRTF**: Shortest Remaining Time First) or (**SRPT**: Shortest Remaining Processing Time) or (**Preemptive-SJF**)]

کوانتم‌های اختصاص یافته به خودش به صورت کامل استفاده نماید به کلاس پایین‌تر راه خواهد یافت (فرایند طولانی است و بنابراین به علت سنگینی‌اش ته‌نشین می‌شود). هنگامی به سراغ فرایندهای یک کلاس می‌رویم که صف تمامی کلاس‌های بالاتر از آن خالی باشد. چنان‌چه مشغول پردازش فرایندها در یکی از صف‌های پایین‌تر باشیم و فرایند جدیدی وارد صف اول شود باید به صف اول بازگردیم و از اجرای فرایندهای بعدی صف جاری خودداری نماییم. این الگوریتم با نام دیگر **صف‌های چندسطحی بازخورد** یا **MLFQ**[۷] (یا **MFQ**) نیز شناخته می‌شود.

مشتقات دیگری از این الگوریتم قدرت‌مند پیشنهاد شده و در سیستم‌های عامل مورد استفاده قرار گرفته‌اند. اگر الگوریتم پایه را به صورت $2^nQ-MLFQ$ بشناسیم که در صف شماره n (Q_n) به هر فرایند ، 2^n کوانتم تخصیص داده می‌شود، الگوریتم‌هایی مانند $nQ-MLFQ$ و $1Q-MLFQ$ نیز وجود دارند که در صف شماره n آن به هر فرایند، به ترتیب، n کوانتم و یک کوانتم تخصیص داده می‌شود.

برای محدود کردن تعداد صف‌ها، روش‌های مختلفی در عمل به‌کار گرفته شده است. مثلاً می‌توانیم سه صف داشته باشیم که در صف Q_0، یک کوانتم و در صف Q_1، دو کوانتم به فرایندها تخصیص دهیم. اما فرایندهای صف سوم را با الگوریتم انحصاری FCFS زمان‌بندی کنیم و آنها را اجرا کنیم تا خاتمه یابند و یا مسدود شده و پس از بیداری به انتهای صف اول بروند.

نکته: مزایای الگوریتم MLFQ (FB) عبارت‌اند از ۱) تضمین زمان پاسخ مطلوب برای کارهای کوچک ۲) کاهش لگاریتمی نرخ تعویض متن ۳) اهمیت دادن به کارهای محدود به I/O ۴) قابلیت پیاده‌سازی ۵) اهمیت دادن به کارهای کوچکتر، بدون نیاز به دانستن زمان سرویس فرایند از قبل یا نیاز به تخمین آن.

نکته: معایب و نقاط ضعف الگوریتم MLFQ (FB) عبارت‌اند از ۱) قحطی (گرسنگی) کارهای طولانی ۲) پیچیدگی ۳) میانگین زمان پاسخ و انتظار بالا

نکته: برای حل مشکل قحطی می‌توانیم سیاست‌های خاصی را به‌کار ببریم و فرایندهای منتظر در صف‌های پایین‌تر را پس از مدت معینی به صف‌های بالاتر منتقل کنیم.

زمان‌بندی بلادرنگ (Real Time) :

CPU در صورتی می‌تواند به این وقایع پاسخ دهد که داشته باشیم:

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \le 1$$

CBT موردنیاز / دوره تناوب

در فرمول بالا P_i, C_i بر حسب برش زمانی هستند یعنی چند TS

- ✓ فرآیندها فضای مشترک آدرس دهی ندارند ولی ریسمان‌های داخل یک فرآیند فضای مشترک آدرس‌دهی دارند.
- ✓ چنانچه 80 درصد CBT های مجموعه‌ای از فرآیندها کوتاه‌تر از کوانتوم زمانی باشد، آنگاه میانگین زمان پاسخ دهی این مجموعه فرآیندها می‌تواند مطلوب باشد. (در RR).
- ✓ در سیستم اشتراک زمانی وقت پردازنده به طور مساوی بین فرایندها تقسیم می‌شوند هنگامی که فرآیندی به اتمام رسید وقت پردازنده بین مابقی آن‌ها به طور مساوی تقسیم می‌گردد.

7- Multiple feedback Queues

- ✓ در سیستم اشتراک زمانی

$$\text{زمان بکارگیری پردازنده} = \frac{TS}{TS + CST}$$ Context Sovitch time

Time Sliue

- ✓ تعویض اجرای فرآیندها، تعویض متن (Context Switch) نام دارد .تعویض متن توسط بخشی از سیستم عامل به نام Dispatcher انجام می‌شود

$$\text{Response Time} \leq \text{Turn Around Time}$$

انواع زمان‌بندها

- ✓ سطح پایین: همگام کردن منطقی برنامه‌ها
- ✓ سطح وسط : تنظیم اولویت و عملیات برش زمانی
- ✓ سطح بالا: ورود کارها به داخل سیستم

در جمع بندی داریم:

- ✓ متوسط زمان انتظار در الگوریتم FIFO غالباً طولانی است و گرسنگی داریم.
- ✓ در الگوریتم SJF کارهای طولانی به تعویق می‌افتند (Starvation) ـ اما زمان میانگین پاسخ‌دهی به فرآیندهایی که در Ready list قرار دارند به حداقل می‌رسد .
- ✓ SRTF دارای سربار بیشتری نسبت به SJF است و همانند SJF دارای گرسنگی است.
- ✓ زمان انتظار و زمان پاسخ SJF بیشتر از SRTF است.
- ✓ در الگوریتم Priority هم پدیده Starvation داریم.
- ✓ در الگوریتم HRRN هرگز پدیده گرسنگی نداریم.
- ✓ در الگوریتم MLFQ پدیده Starvation نداریم.
- ✓ در الگوریتم زمان بندی شانسی (Lottery) تعدادی عدد را سیستم به هر پردازش نسبت می‌دهد. پردازشی که از اولویت بالا برخوردار می‌باشد تعداد اعدادش بیشتر از پردازشی است که اولویتش کمتر است. سپس به طور تصادفی یک عدد انتخاب می‌شود و پردازشی که این عدد را دارد CPU را می‌گیرد. این الگوریتم انحصاری است.
- ✓ LPT (Langest Orocess Time) : همیشه طولانی‌ترین job انتخاب می‌شود. بهینه نیست انحصاری است.

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

زمان پردازشی

دوره تناوب

- ✓ زمان بندی EDF (Earliest Deadline First) مربوط به سیستم‌های بلادرنگ است.

نکته: میانگین زمان اجرا + میانگین زمان انتظار = میانگین زمان برگشت

- ✓ در یک سیستم با n پردازش که همه چیز یکسان فرض می‌شود هر پردازش باید $\frac{1}{n}$ اززمان پردازنده را دریافت نماید. (در زمان بندی Guaranted یا تضمین شده) این الگوریتم عادلانه است.
- ✓ در سیستم کامپیوتری که دارای CPU با سرعت بالا می‌باشد بهتر است سعی کنیم کارهای CPU limited را با I/O Limited ترکیب کنیم.

تست‌ها

۱. درصورتی‌که 4 پردازه A، B، C و D به همین ترتیب در لیست پردازه‌های آماده اجرای یک زمان‌بند قرار داشته باشند، زمان اجرای تخمینی آن‌ها به ترتیب برابر 40، 20، 50 و 30 میلی‌ثانیه باشد، زمان هر Context Switch **بین پردازه‌ها برابر 5 میلی‌ثانیه باشد و از روش زمان‌بندی نوبه‌ای** Round - Robin **با کوانتوم زمانی 20 میلی‌ثانیه استفاده شود، متوسط زمان پاسخگویی**(Turnaround - Time) **و متوسط زمان انتظار** (Waiting - Time) **پردازه‌ها چقدر است؟**

(کارشناسی ارشد کامپیوتر ــ سراسری ۷۶)

۱) متوسط زمان پاسخگویی = 125 میلی‌ثانیه و متوسط زمان انتظار = 90 میلی‌ثانیه

۲) متوسط زمان پاسخگویی = 102.5 میلی‌ثانیه و متوسط زمان انتظار = 67.5 میلی‌ثانیه

۳) متوسط زمان پاسخگویی = 128.75 میلی‌ثانیه و متوسط زمان انتظار = 110 میلی‌ثانیه

۴) متوسط زمان پاسخگویی = 135 میلی‌ثانیه و متوسط زمان انتظار = 105 میلی‌ثانیه

۲. دو پروسس P1 **و** P2 **با مشخصات اجرای زیر در سیستم موجودند. اطلاعات هر سطر، منبع مورد نیاز برای هر پروسس** (Process) **و زمان مورد نیاز را مشخص می‌کند. مثلاً** Net 3 **در سطر چهارم بیانگر این است که پروسس دوم، کارت شبکه را به مدت 3 ثانیه نیاز دارد. اگر پروسس دوم** (P2) **به مدت 2 ثانیه بعد از پروسس** P1 **به سیستم رسیده باشد و سیستم سیاست** SJF (Shortest Job First) **با خاصیت** Preemption **را برای برنامه‌ریزی پروسس‌ها اعمال کند، زمان کل اجرای 2 پروسس مذکور و زمان هدررفتگی وقت** CPU **برحسب ثانیه چقدر است؟** **(کارشناسی ارشد کامپیوتر ــ سراسری ۸۱)**

P_1	P_2
CPU 3	CPU 4
Net 4	Disk 3
CPU 2	CPU 3
Disk 3	Net 3
CPU 5	CPU 3
Disk 2	Net 3
CPU 2	CPU 3

۱)کل زمان 24 ـ هدررفتگی صفر (0)

۲)کل زمان 25 ـ هدررفتگی 1

۳)کل زمان 27 ـ هدررفتگی 2

۴)کل زمان 28 ـ هدررفتگی 3

۳. جدول زیر اطلاعات 4 پروسس در یک سیستم اشتراک‌زمانی را نشان می‌دهد.

پروسس	زمان ورود به سیستم	زمان مورد نیاز پردازش
P_1	0	8
P_2	3	4
P_3	2	9
P_4	3	5

فرض کنید سیستم عامل مورد نظر روش‌های مختلفی را برای زمان‌بندی پروسس‌ها در نظر بگیرد. کدام‌یک از گزینه‌های زیر کمترین (متوسط زمان تکمیل) Turnaround Time **را خواهد داشت؟** **(کارشناسی ارشد کامپیوتر ــ سراسری ۸۵)**

۱) Shortest Job First

۲) Shortest Job First With Preemption

۳) FCFS (First Come First Served)

۴) گردشی Round Robin با میزان زمانی(Quantum time) 1

۴. ۳ پردازه دسته‌ای (Batch) P_1, P_2, P_3 را با زمان اجرا و زمان وارد شدن زیر در نظر بگیرید:

پردازه	اولویت	زمان ورود	زمان اجرا
P_1	2	t	4
P_2	0	t	2
P_3	1	t+3	1

کدام‌یک از گزینه‌های زیر غلط است؟ **(کارشناسی ارشد کامپیوتر ـ سراسری ۸۳)**

۱) زمان متوسط پاسخگویی (Average Turnaround Time) با روش زمان‌بندی SJN (Shortest Job First) برابر است با $\frac{12}{3}$

۲) زمان متوسط پاسخگویی (Average Turnaround Time) با روش زمان‌بندی FIFO (First In First Out) برابر است با $\frac{14}{3}$

۳) زمان متوسط پاسخگویی (Average Turnaround Time) با روش زمان‌بندی SRT (Shortest Remaining Time) برابر است با $\frac{10}{3}$

۴) زمان متوسط پاسخگویی (Average Turnaround Time) با روش اولویت (Priority) برابر است با $\frac{12}{3}$. (توضیح اینکه عدد اولویت بیشتر نشان‌دهنده اولویت بالاتر است.)

۵. کدام‌یک از الگوریتم‌های زمان‌بندی زیر برای فرایندهایی با زمان اجرای کمتر تبعیض قایل نمی‌شود؟

(کارشناسی ارشد IT ـ آزاد ۸۵)

۱) SRT (Shortest Remaining Time)

۲) RR با کوانتوم طولانی

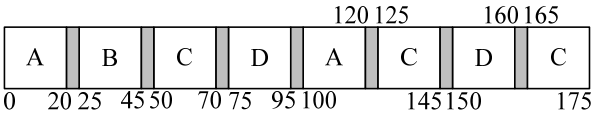
۳) Feedback

۴) SPN (Shortest Process Next)

پاسخنامه

۱. گزینه ۱ درست است.

نمودار گانت:



$$\text{ATT (Average Turnaround Time)} = \frac{1}{4}\left[(120-0)+(45-0)+(175-0)+(160-0)\right] = 125\text{ms}$$

روش اول:

Waiting time = Turnaround time – Service time

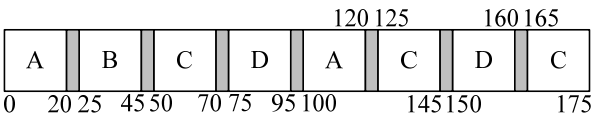
$$\text{AWT(Average Waiting Time)} = \frac{1}{4}\left[(120-40)+(45-20)+(175-50)+(160-30)\right] = 90\text{ms}$$

روش دوم:

$$\text{میانگین زمان انتظار} = \frac{(100-20)+25+[50+(125-70)+(165-145)]+[75+(150-95)]}{4} = 90\text{ms}$$

۲. گزینه ۱ درست است.

نمودار گانت:



$$\text{ATT (Average Turnaround Time)} = \frac{1}{4}\left[(120-0)+(45-0)+(175-0)+(160-0)\right] = 125\text{ms}$$

روش اول:

Waiting time = Turnaround time – Service time

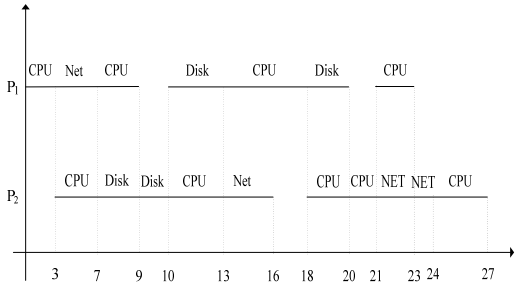
$$\text{AWT(Average Waiting Time)} = \frac{1}{4}\left[(120-40)+(45-20)+(175-50)+(160-30)\right] = 90\text{ms}$$

روش دوم:

$$\text{میانگین زمان انتظار} = \frac{(100-20)+25+[50+(125-70)+(165-145)]+[75+(150-95)]}{4} = 90\text{ms}$$

۳. گزینه ۳ درست است.

در طرح این سؤال عمل Preemption دیده نشده است. به عنوان مثال، اگر P_1 در سومین Burst خود به‌جای 5 واحد، 7واحد CPU لازم داشت در لحظه 16 که P_2 به 3 واحد CPU نیاز دارد باید عمل Preemption صورت می‌گرفت. چون در آن صورت در لحظه 16 زمان باقی‌مانده CPU برای P_1 برابر 4 بود.



نمودار بالا نشان می‌دهد که کل زمان اجرا 27 ثانیه است. زمان‌های (9 تا 10) و (23 تا 24) جمعاً به مدت 2 ثانیه CPU هدر رفته است.

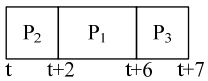
۴. گزینه ۲ درست است.

این مسئله نیاز به حل ندارد. قابل اثبات است که اگر از زمان سوئیچ صرف‌نظر کنیم، الگوریتم SRT همیشه حداقل میانگین زمان انتظار و پاسخ (یا برگشت) را دارد.

۵. گزینه ۴ درست است.

برای هر الگوریتم، نمودار گانت را رسم می‌کنیم:

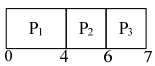
: SJN



$$\text{میانگین زمان پاسخ‌گویی} = \text{ART}_{\text{SJF}} = \frac{[(t+6)-t]+[(t+2)-t]+[(t+7)-(t+3)]}{3} = \frac{12}{3}$$

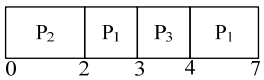
چون زمان t، در زمان ورود تمامی فرایندها نقش مبدأ زمانی ثابت را دارد و در محاسبات زمان پاسخ آن‌ها حذف می‌شود، بنابراین می‌توان آن را 0 فرض کرد:

: FIFO



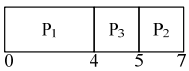
$$\text{میانگین زمان پاسخ‌گویی} = \text{ART}_{\text{FIFO}} = \frac{(4-0)+(6-0)+(7-3)}{3} = \frac{14}{3}$$

: SRT



$$\text{میانگین زمان پاسخ‌گویی} = \text{ART}_{\text{SRT}} = \frac{(7-0)+(2-0)+(4-3)}{3} = \frac{10}{3}$$

: Priority



$$\text{میانگین زمان پاسخ‌گویی} = \text{ART}_{\text{Pr.}} = \frac{(4-0)+(7-0)+(5-3)}{3} = \frac{13}{3}$$

۶. گزینه ۲ درست است.

در الگوریتم‌های زمان‌بندی SPN و SRT اساساً اولویت با فرایندهای کوتاه‌تر است. در الگوریتم Feedback نیز فرایندهای کوتاه‌تر در صف اول (صفی با حداکثر اولویت) خاتمه یافته و به صف‌های پایین‌تر ته‌نشین نمی‌شوند (فرایندهای بزرگ‌تر به صف‌های پایین رفته و ممکن است دچار قحطی شوند). الگوریتم RR با کوانتوم زمانی طولانی، شبیه الگوریتم FCFS رفتار می‌کند و در آن عدالت مهم‌تر از توجه به کارهای کوتاه‌تر است.

فصل سوم

همروندی انحصار متقابل و همگام سازی

ارتباط بین فرایندها (IPC[8])، با سه مبحث مهم در طراحی سیستم‌عامل روبه‌رو می‌شویم:

- ✓ تبادل اطلاعات بین فرایندها
- ✓ همگام‌سازی فرایندها
- ✓ رقابت (مسابقه) فرایندها

گاهی یک فرایند، به نتیجه محاسبات یک فرایند دیگر نیاز دارد. بنابراین به یک مکانیسم جهت ارتباط بین فرایندها نیاز داریم و ترجیح می‌دهیم که این کار با روشی سریع و ساخت‌یافته انجام شود. فرایندها و نخ‌ها با چندین مکانیسم مختلف می‌توانند با یکدیگر **تبادل اطلاعات (Communication)** نمایند. مکانیسم‌های حافظه مشترک، تبادل پیام، لوله (نوعی شبه‌فایل در UNIX) و فایل مشترک از این دسته‌اند.

همگام‌سازی (Synchronization) فرایندها، به این موضوع اشاره دارد که هنگامی‌که وابستگی وجود دارد، ترتیب صحیح انجام کارها اهمیت پیدا می‌کند.

مفهوم سوم، به این بحث مربوط می‌شود که چه کنیم تا مطمئن شویم که دو یا چند فرایند در فعالیت‌های **بحرانی** یکدیگر، برخورد یا مداخله ننمایند و دچار **شرایط رقابتی (Race condition)** نشوند. فرض کنید که دو فرایند داریم که هر دو همزمان با هم سعی می‌کنند تا در یک درایه از یک جدول مشترک، اطلاعات مربوط به فایل خروجی خود را ثبت کنند. اگر با این مشکل، درست برخورد نشود ممکن است اطلاعات خروجی یک فرایند بر روی دیگری بازنویسی شده و اطلاعات خروجی فرایند قبلی گم شود. مانند زندگی عادی انسان‌ها، در فرایندها نیز همیشه برخورد، تداخل و رقابت بر سر عوامل مشترک است. هرگز نمی‌توانیم رقابتی را فرض کنیم که هیچ عامل مشترکی در آن نباشد.

8_ InterProcess Communication

پس به صورت کلی داریم :

- ✓ در ارتباط بین فرایندها، نگران سه مشکل خطرناک رقابت، بن‌بست و قحطی (گرسنگی) هستیم.
- ✓ بخش‌هایی از کُد فرایند که به سراغ منبع بحرانی یا عامل مشترک رقابت‌زای دیگری (مثل متغیرهای مشترک رقابت‌زا) می‌رود را ناحیه بحرانی می‌نامیم. دقت کنید که هر عامل مشترکی رقابت‌زا نیست.
- ✓ هرگاه دو یا چند فرایند همزمان با هم وارد ناحیه بحرانی شوند، شرایط رقابتی (Race condition) پیش می‌آید. در شرایط رقابتی، نتیجه نهایی بستگی به ترتیب دسترسی‌ها دارد.
- ✓ به وضعیتی که در آن مجموعه‌ای متشکل از دو یا چند فرایند برای همیشه منتظر یکدیگر بمانند و به عبارت دیگر دچار سیکل انتظار ابدی شوند، بن‌بست (Deadlock) گفته می‌شود.
- ✓ هرگاه فرایندی به مدت نامعلوم و بدون حد بالای مشخص در انتظار گرفتن یک منبع بحرانی یا دسترسی به یک عامل مشترک یا دریافت اطلاعاتی از فرایند مقابل بماند، دچار قحطی (Starvation) شده است.

باید جهت رفع مشکل وضعیت سابقه سه شرط زیر را در نظر بگیریم:

۱ـ Mutual Exclusion : هنگامی که پردازشی در ناحیه بحرانی‌اش اجرا می‌گردد، هیچ پردازش دیگری نباید در ناحیه بحرانی حضور داشته باشد.

۲ـ شرط پیشروی (Progress) : هیچ پردازشی نباید از بیرون ناحیه بحرانی خود امکان بلوکه کردن پردازش‌های دیگر را داشته باشد.

۳ـ شرط انتظار مقید یا محدود (Bounded Waiting) : یک برنامه منتظر ورود به ناحیه‌ی بحرانی نباید به طور نامحدود در حالت انتظار باقی بمانند. (یعنی starvation نداشته باشیم).

*اگر پردازشی در حال اجرا و پردازش دیگری بلوکه یا آماده باشد و هر دوی آن‌ها در ناحیه بحرانی خود باشند، شرط انحصار متقابل رعایت نشده است. بنابراین هنگام بحث درباره شرط انحصار متقابل وضعیت پردازش‌ها مهم نمی‌باشد.

۱ـ ساده‌ترین راه آنست که هر پردازه بلافاصله پس از ورود به ناحیه بحرانی‌اش کلیه وقفه‌ها را از کار بیندازد. و درست قبل از خروج از ناحیه بحرانی دوباره همه آن‌ها رافعال کند. با خاموش ساختن وقفه‌ها CPU به هیچ عنوان نمی‌تواند از پردازشی به پردازش دیگر سوئیچ کند.

*یک مشکل این روش این است که ممکن است کاربر وقفه‌ها را خاموش کند ولی دوباره آن‌ها را فعال نسازد و بدین ترتیب کل سیستم از کار خواهد افتاد.

*از کار انداختن وقفه‌ها غالباً در خود سیستم عامل استفاده می‌شود و به کارگیری آن برای پردازش‌های User مناسب نیست.

۲ـ استفاده از متغیرهای قفل: هنگامی که پردازشی می‌خواهد وارد ناحیه بحرانی شود، ابتدا Lock را آزمایش می‌کند، اگر Lock=0 بود آن را برابر 1 کرده و وارد ناحیه بحرانی می‌شد. ولی اگر Lock=1 بود باید در یک حلقه منتظر بماند تا Lock برابر صفر شود. (این روش انحصار متقابل ندارد).

۳ـ رویکردهای نرم‌افزاری انحصار متقابل : راه‌حل‌های مطرح شده در این قسمت، همگی بدون حمایت دستورالعمل‌های خاص توسط سخت‌افزار و بدون حمایت سیستم‌عامل و زبان‌های برنامه‌سازی، مستقیماً توسط برنامه‌ها در سطوح مختلف، مورد استفاده قرار می‌گیرند. این راه‌حل‌ها، در سیستم‌های تک‌پردازنده و چندپردازنده با حافظه اشتراکی قابل استفاده‌اند.

پیشنهاد متغیرهای قفل

```
P(int i) {
  while (TRUE) {
     while (lock == 1)  /*loop*/
;
     lock = 1;
     critical_region();
     lock = 0;
     non_critical_region();
  }
}
```

نکته: این روش، شرط انحصار متقابل را رعایت می‌کند.

Deckerراه‌حل نرم‌افزاری

اولین تلاش: تناوب قطعی

```
int turn;

P0(void){
   while (TRUE) {
        while (turn != 0)  /*loop*/ ;
        critical_region();
        turn = 1;
        non_critical_region();
   }
}
```

```
P1(void){
   while (TRUE) {
        while (turn != 1)  /*loop*/
;
        critical_region();
        turn = 0;
        non_critical_region();
   }
}
```

نکته: این روش، مبتنی بر انتظار مشغول است و بدتر از آن اینکه شرط پیشرفت را رعایت نمی‌کند.

دومین تلاش

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

P0 (void) {
   while (TRUE) {
      while (flag [1] ) /* loop */
;
      flag [0] = TRUE ;
       critical_region();
       flag [0] = FALSE ;
       non_critical_region();
   }
}
```

```
P1 (void) {
   while (TRUE) {
      while (flag [0] ) /* loop */ ;
      flag [1] = TRUE ;
       critical_region();
       flag [1] = FALSE ;
       non_critical_region();
   }
}
```

نکته: این روش، شرط انحصار متقابل را رعایت نمی‌کند. همچنین این روش، شرط انتظار محدود را رعایت نمی‌کند، چون امکان قحطی وجود دارد. همچنین، مبتنی بر انتظار مشغول است

سومین تلاش

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

P0 (void) {
   while (TRUE) {
```

```
P1 (void) {
   while (TRUE) {
```

```
        flag [0] = TRUE ;
        while (flag [1] ) /* loop */
;
        critical_region();
        flag [0] = FALSE ;
        non_critical_region();
    }
}
```

```
        flag [1] = TRUE ;
        while (flag [0] ) /* loop */ ;
        critical_region();
        flag [1] = FALSE ;
        non_critical_region();
    }
}
```

نکته: این روش، شرط انتظار محدود را رعایت نمی‌کند، چون امکان بن‌بست وجود دارد. همچنین، مبتنی بر انتظار مشغول است.

چهارمین تلاش

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

P0 (void) {
    while (TRUE) {
        flag [0] = TRUE ;
        while (flag [1] ) {
            flag [0] = FALSE ;

delay_for_a_short_time();
            flag [0] = TRUE ;
        }
        critical_region();
        flag [0] = FALSE ;
        non_critical_region();
    }
}
```

```
P1 (void) {
    while (TRUE) {
        flag [1] = TRUE ;
        while (flag [0] ) {
            flag [1] = FALSE ;
            delay_for_a_short_time();
            flag [1] = TRUE ;
        }
        critical_region();
        flag [1] = FALSE ;
        non_critical_region();
    }
}
```

نکته: این روش، شرط انتظار محدود را رعایت نمی‌کند. مشکل Deadlock حل شده است، اما هنوز امکان Livelock وجود دارد (اما نادر است). می‌توان با تصادفی کردن مدت تأخیر، مشکل Livelock را عملاً حل کرد ولی از نظر تئوری این راه‌حل نیز شرط قطعی‌بودن راه‌حل را رعایت نمی‌کند. در مورد همان شرط انتظار محدود، مشکل قحطی نیز منتفی نیست (اگر چه آن هم نادر است). همچنین، این روش مبتنی بر انتظار مشغول است.

راه‌حل صحیح Decker پس از چهار تلاش ناموفق

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

P0 (void) {
    while (TRUE) {
        flag [0] = TRUE ;
        while (flag [1] )
            if  (turn == 1) {
                flag [0] = FALSE ;
                while (turn == 1) /* loop
*/ ;
                flag [0] = TRUE ;
            }
        critical_region();
        turn = 1;
        flag [0] = FALSE ;
        non_critical_region();
    }
}
```

```
P1 (void) {
    while (TRUE) {
        flag [1] = TRUE ;
        while (flag [0] )
            if  (turn == 0) {
                flag [1] = FALSE ;
                while (turn == 0) /* loop
*/ ;
                flag [1] = TRUE ;
            }
        critical_region();
        turn = 0;
        flag [1] = FALSE ;
        non_critical_region();
    }
}
```

راه حل پترسون (Peterson) :

پردازش‌ها از دو متغیر مشترک turn و Flag استفاده می‌کنند.

کد روش پترسون :

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

P0 (void) {
   while (TRUE) {
      flag [0] = TRUE ;
      turn = 0;
      while (turn == 0 && flag [1] )
/*loop*/  ;
      critical_region();
      flag [0] = FALSE ;
      non_critical_region();
   }
}
```

```
P1 (void) {
   while (TRUE) {
      flag [1] = TRUE ;
      turn = 1;
      while (turn == 1 && flag [0] )
/*loop*/  ;
      critical_region();
      flag [1] = FALSE ;
      non_critical_region();
   }
}
```

روش پترستون تمام شروط ۴ گانه ناحیه بحرانی را ارضا می‌کند.

۴ـ سمافور (Semaphores):

سمافور غالباً در دسته راه‌حل‌های مبتنی بر حمایت سیستم‌عامل قرار می‌گیرد و از اولیه‌های هسته برای پیاده‌سازی آن استفاده می‌شود، اما توجه داشته باشید که پیاده‌سازی آن در زبان‌های برنامه‌سازی و استفاده از توابع کتابخانه سطح بالا به جای اولیه‌های هسته نیز امکان‌پذیر است.

سمافور عبارت است از یک ساختار شامل:

- ✓ یک شمارنده صحیح غیر منفی برای شمارش Signalهایی(upهایی) که به دلیل عدم وجود فرایند بلوکه شده می‌خواهند از دست بروند.
- ✓ و نیز یک صف برای نگه‌داری فرایندهای بلوکه شده بر روی سمافور

تعاریف ساده wait و signal به صورت زیر می‌باشد.

Wait (s) : while (S ≤ 0) ; // loop
S = S-1

Signal (s) : S = S + 1 :

به جای wait از نام‌های Down یا P و به جای Signal از نام‌های UP یا V نیز استفاده می‌گردد.

تعاریف ساده فوق برای wait , signal سمافورها، مشابه راه حل پترسون و TSL مشکل Busy Waitin را دارد. برای رفع این مشکل تعریف سمافور و wait و signal را کامل‌تر می‌کنیم. در این حالت زمانی که پروسس اجازه ورود به ناحیه بحرانی‌اش را ندارد، بلوکه یا مسدود (block) می‌شود. (برای آن که در یک حلقه while چرخ بزند) بدین ترتیب آن پروسس به حالت تعلیق می‌رود تا پروسس دیگری آن را بیدار (wake up) کند.

عمل block یا sleep پردازشی که آن را صدا زده است را مسدود می‌کند (از حالت اجرا به حالت انتظار می‌برد) و عمل wake up (p) اجرای پردازش مسدود شده p را از سر می‌گیرد (از حالت انتظار به حالت آماده می‌برد) این دو عملیات توسط سیستم عامل فراهم می‌گردند.

مشکل بن بست سمافورها

فرض کنید P_1, P_0 به دو سمافور S و Q با مقادیر اولیه 1 دسترسی دارند و ابتدا wait (s) و سپس wait (Q) اجرا می‌شود.

P_1	P_0
Wait (Q)	Wait (S)
Wait (S)	Wait (Q)
Signal (Q)	Signal (S)
Signal (S)	Signal (Q)

مسائل کلاسیک IPC : هر الگوریتم پیشنهادی جدی در رابطه با همزمانی پردازش‌ها باید بتواند این مسائل کلاسیک را به درستی حل کند: Produeer & Consumer ـ Readers 8 writers مسأله غذا خوردن فیلسوف‌ها ـ مسأله آرایشگر خوابیده.

حل مسأله کلاسیک تولید کننده ـ مصرف کننده با Semaphore

دو پردازش یک بافر معمولی با اندازه ثابت N را به اشتراک می‌گذارند. یکی از آن‌ها (producer) اطلاعات را در بافر قرار می‌دهد و دیگری (Consumer) اطلاعات را از بافر بر می‌دارد.

برای حل مسأله از سمافور باینری Mutex با مقدار اولیه 1 و دو سمافور شمارش full و empty با مقادیر اولیه به ترتیب CD و N استفاده می‌کنیم.

Counting Semaphore : $\text{full} = Q_1 \quad \text{empty} = N$

Binary Semaphore : $\text{mutex} = 1$

```
Producer                    Consumer
{                           {
Produce on item             Down (full)
Down (empty)                Down (mutex)
Down (mutex)                Remove item
Insert (item)               Up (mutex)
Up (mutex)                  Up (Empty)
Up (full)                   Consume the item
}                           }
```

سمافوری که برای مسأله ناحیه بحرانی به کار می‌رود غالباً مقدار اولیه آن برابر 1 می‌باشد.

حل مسأله کلاسیک Writers & Readers با سمافور

چند پردازش (Readers) همزمان می‌توانند پایگاه داده را بخوانند، ولی اگر یک پردازش (writer) در حال تغییر پایگاه داده باشد، پردازش‌های دیگر (حتی خوانندگان) نبایدبه پایگاه داده دستیابی داشته باشند.

```
Semaphore mutex =1 , db = 1 :
Int  rc = 0 :
Reader
 {    while (TRUE)  {
 Down (mutex) :
 Rc ++ ;
 If (rc == 1) down (db)
 Up (mutex)
 Read – Data Base ( ) ;
 Down (mutex)
 Rc - - ;
 If (rc == 0)   up (db)
 Up (mutex)
```

```
Writer
{ while (TRUE)  {
 Think up – data ( ) ;
 Down (db)
 Write – data base ( ) ;
  Up (db)
}
}
```

در این مسأله تا زمانی که خواننده‌ای وجود داشته باشد به نویسنده اجازه کار داده نمی‌شود. اگر مرتباً خوانندگان جدیدی وارد سیستم شوند به آن‌ها سرویس داده شده و نویسنده همچنان معلق باقی می‌ماند. این مسئله باعث starvation می‌شود.

حل مسأله کلاسیک غذا خوردن فیلسوف‌ها با سمافور

```
Void philosopher (int)  {
  While (TRUE)  { thine ( ) ;
  Take – forks (i) ; cat ( )  : put – forks ( ) ; }}
Void  take – forks (int i) {down (8 mutex) ;
  State [i] = HUNGRY ; test (i) ; up (8 mutex) ;
  Down (8 S [i]) ; }

Void put – forks {down (8 mutex) ;
  State [i] = thinking : Test (8 left); Test (Right)
  Up (8 mutes) ; }

Void test (i)
  {
  If (state [i] = = HUNGRY 88 State [left] ! = EATING 88
     State [Right] ! = Eating)
  { state [i] = Eating ; up (8 S[i])
  }
 {
```

۵- مانیتورها

همگام‌سازی فرایندها با سمافور پیچیده است و اگر برنامه‌نویس دقت نکند خطرناک خواهد بود. برای این‌که نوشتن برنامه‌های صحیح ساده‌تر باشد،یک ابزار سطح بالاتر، انتزاعی‌تر و راحت‌تر برای همگام‌سازی به نام **مانیتور** (Monitor) ابداع شد. مانیتور یک ساختار زبان برنامه‌سازی است و به صورت یک ماژول برنامه‌نویسی در گرامر بعضی از زبان‌های برنامه‌نویسی پیشرفته تعریف می‌شود. این ماژول، متشکل است از:

۱. متغیرها و ساختمان داده‌ها شامل
   - ✓ متغیرهای شرطی
   - ✓ متغیرهای محلی
   - ✓ و البته کد مقداردهی اولیه متغیرها
   - ✓ تعدادی از توابع و رویه‌ها

```
monitor mon_name          ✓
    var                   ✓
        integer i; real z;  ✓
        condition     c;    ✓

                            ✓
    procedure p1(…);        ✓
    begin                   ✓
        …                   ✓
    end;                    ✓
                            ✓
    function f1(…);         ✓
    begin                   ✓
        …                   ✓
    end;                    ✓
                            ✓
    initialization code;    ✓
end monitor;                ✓
```

نکته: مانیتور دو خاصیت مهم دارد:

۱. محصورسازی (Encapsulation)

۲. انحصار متقابل (Mutual exclusion)

خاصیت اول، بیان‌گر این نکته است که فرایندها نمی‌توانند از طریق رویه‌های خارج از مانیتور، مستقیماً به ساختمان داده‌های داخل مانیتور دسترسی داشته باشند. تنها راه دسترسی (خواندن یا نوشتن) به متغیرهای داخل مانیتور، فراخوانی رویه‌های داخل آن مانیتور است. این قانون در زبان‌های شیءگرای پیشرفته مانند جاوا عمومیت دارد.

خاصیت مهم دوم، بیان می‌کند که در هر لحظه فقط یک فرایند می‌تواند در یک مانیتور فعال باشد. به کلمه فعال دقت کنید. اگر یک فرایند با فراخوانی یک رویه درون مانیتور، وارد آن مانیتور شود هیچ فرایند دیگری نمی‌تواند با فراخوانی همان رویه یا رویه‌های دیگر، وارد آن مانیتور شود، مگر آنکه فرایند اول در مانیتور بخوابد و دیگر فعال نباشد.

برای همگام‌سازی و خوابیدن و بیدار کردن در داخل مانیتور فقط باید از **متغیرهای شرطی** (Condition variables) استفاده کرد که در مورد آن دو عملیات wait و signal نیز تعریف می‌شود.

۶ - تکنیک تبادل پیغام (Message Passing)

*در این تکنیک ارتباط بین پردازش‌ها توسط دو ابزار اولیه به نام‌های Send و Receive انجام می‌پذیرد که مانند سمافورها و برخلاف مانیتورها فراخوانی‌های سیستمی بوده و ویژگی زبان‌های برنامه نویسی نمی‌باشند.

*مصرف کننده با فرستادن N پیام خالی به تولید کننده کار را آغاز می‌کند. هرگاه که تولید کننده داده‌ای را تولید می‌کند آن را در یکی از پیام‌های خالی گذاشته و به سمت مصرف کننده بر می‌گرداند.

*سیستم عامل UNIX برای ارتباط بین پردازش‌های خود (پردازش‌هایی که خود سیستم عامل را می‌سازند) از تکنیک تبادل پیام استفاده می‌کند.

*منظور از اجرای Interleared این است که هنگام اجرای یک پردازش در هر زمان امکان تعویض متن به پروسس دیگر وجود دارد.

*در الگوریتم پترسون فرقی نمی‌کند که Turn=I نوشته شود یا Turn=1-I

Flag [I] = true
Turn = 1 – i
While [flag [1-i] 88
Turn = 1-1)

نکته : مانیتور یک ابزار نرم افزاری سطح بالا در سیستم است که از مجموعه‌ای از رویه‌ها و متغیرهای داخلی تشکیل شده و چنانچه بخواهد در یک زبان برنامه نویسی مورد استفاده قرار گیرد باید هم زبان برنامه نویسی و هم سیستم عامل مربوطه مانیتور را پشتیبانی کنند. (یعنی خود مانیتور به تنهایی نمی‌تواند همزمانی پردازش‌ها را پشتیبانی کند.)

تست‌ها

۱. اگر دو پروسس P_1 و P_2 آمده در زیر به طور هم‌روند اجرا شوند، کدام‌یک از موارد ذیل درست است؟

(کارشناسی ارشد IT ـ سراسری ۸۵)

```
N1,N2: integer := 0;
task body P2 is:
begin
  loop
    non-critical-section-2;
    N2 := 1;
    N2 := N1 + 1;
    loop exit when N1=0 OR N2<N1; end loop;
    critical-section-2;
  end loop;
end P2;
```

```
task body P1 is:
begin
  loop
    non-critical-section-1;
    N1 := 1;
    N1 := N2 + 1
    loop exit when N2=0 OR N1<N2;end loop;
    critical-section-1;
  end loop;
end p1;
```

۱) انحصار متقابل تأمین می‌شود.

۲) انحصار متقابل تأمین نمی‌شود.

۳) انحصار متقابل تأمین می‌شود ولی امکان بن‌بست وجود دارد.

۴) انحصار متقابل تأمین می‌شود ولی امکان گرسنگی وجود دارد.

۲. اگر مقادیر اولیه تمام سمافورهای به‌کاررفته در فرایندهای زیر صفر باشد، کدام گزینه در مورد ترتیب اجرای فرایندها امکان‌پذیر نخواهد بود؟ عملگر P معادل Wait و عملگر V معادل Signal است و ترتیب اجرا از چپ به راست است.

(کارشناسی ارشد IT ـ سراسری ۸۳ و کارشناسی ارشد کامپیوتر ـ آزاد ۸۰)

P_1:	P_2:	P_3:	P_4:	P_5:	P_6:
.	$P(s_{12})$	$P(s_{13})$	$P(s_{24})$	$P(s_{25})$	$P(s_{36})$
.	.	.	.	$P(s_{35})$	$P(s_{46})$
.	.	.	.	.	.
$V(s_{12})$	.	.	.	.	.
$V(s_{13})$	$V(s_{24})$	$V(s_{35})$	$V(s_{46})$	.	.
	$V(s_{25})$	$V(s_{36})$			

۱) $P_1, P_2, P_3, P_4, P_6, P_5$

۲) $P_1, P_2, P_4, P_3, P_5, P_6$

۳) $P_1, P_3, P_2, P_5, P_4, P_6$

۴) $P_1, P_3, P_5, P_2, P_4, P_6$

۳. الگوریتم زیر یک راه‌حل نرم‌افزاری برای حل مسئله ناحیه بحرانی است. در این الگوریتم: (کارشناسی ارشد IT ـ سراسری ۸۳)

```
var C1,C2: boolean; turn: integer;
C1 := C2 := true;
cobegin
P1:  loop
        C1 := false; turn := 1;
        while (not C2) and (turn=1) do;
         CS1;
         C1 := True;
    end loop
P2:  loop
       C2 := false; turn := 2;
       while (not C1) and (turn=2) do;
        CS2;
        C2 := True;
    end loop
```

۱) این الگوریتم درست است.

۲) Deadlock وجود دارد.

۳) Starvation وجود دارد.

۴) شرایط ناحیه بحرانی برآورده نمی‌شود (استفاده انحصاری).

۴. در یک سیستم تک‌پردازنده‌ای اشتراک زمانی سه برنامه P_1، P_2 و P_3 با قطعه کدهای زیر مفروض است. در صورت اجرای همزمان سه برنامه کدام خروجی اصلاً رخ نمی‌دهد؟ سمافورهای A، B و C به ترتیب دارای مقدار اولیه 0، 1 و 2 هستند.

(کارشناسی ارشد کامپیوتر ـ آزاد ۸۳)

P_1:	P_2:	P_3:

while(1){
wait(A);
printf("A");
signal(C);
}
	while(1){
	wait(B);
	printf("B");
	wait(B);
	printf("B");
	signal(A);
	}
		while(1) {
		wait(C);
		printf("C");
		signal(B);
		}

۱) CBBA ۲) CCBC ۳) BCBA ۴) BCCA

۵. در حل مشکل غذاخوری فیلسوف‌ها به روش مقابل کدام گزینه درست است؟ تعداد فیلسوف‌ها 5 است.

(کارشناسی ارشد IT ـ سراسری ۸۶)

```
Void philosophers(int  i)

  Think( );              //فکر کردن
  take-forks(i);         //برداشتن همزمان دو چنگال
  eat( );                //خوردن
  put-fork-left(i);      //گذاشتن چنگال چپ
  put-fork-right(i);     //گذاشتن چنگال راست
```

۱) بن‌بست ندارد ولی امکان قحطی دارد.

۲) بن‌بست دارد و احتمال قحطی ندارد.

۳) هم بن‌بست دارد هم احتمال قحطی.

۴) نه بن‌بست دارد نه احتمال قحطی.

۶. فرض کنید دو فرایند زیر به صورت هم‌روند در یک سیستم تک‌پردازنده‌ای اجرا می‌شوند. متغیر مشترک X مقدار اولیه صفر دارد و دستور X++ درحقیقت به صورت زیر انجام می‌شود: (کارشناسی ارشد کامپیوتر ـ آزاد ۸۱)

```
Load  R, X   // load X into a register
INC   R      // R++
Store X, R   // store reg.value to X
```

بعد از تکمیل اجرای هر دو فرایند تمام مقادیر ممکن برای متغیر X کدام گزینه است؟

```
Process A:              Process B:
    for i=1 to 5 do         for j=1 to 5 do
        X++                     X++
```

۱) {2, 3, 4, 5, 6, 7, 8, 9, 10}
۲) {5, 6, 7, 8, 9, 10}
۳) {10}
۴) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

۷. مانیتور mon به صورت زیر تعریف شده است. CS_1 و CS_2 دو بخش بحرانی متفاوت و مستقل از یکدیگر هستند. اگر دو فرایند P_1 و P_2 به صورت هم‌روند در حال اجرا باشند کدام شرط زیر نقض می‌شود؟ (کارشناسی ارشد IT ـ آزاد ۸۴)

Monitor mon
procedure enter1;
Begin
CS_1;
end;
procedure enter2;
Begin
CS_2;
End
End
	P_1:
	while(true){
	Mon.enter1;
	}
	P_2:
	while(true){
	Mon.enter2;
	}

۱) شرط انتظار مشغول (Busy Waiting)
۲) شرط پیشرفت (Progress)
۳) شرط دوبه‌دو ناسازگاری (Mutual Exclusion)
۴) هیچ‌یک از شرایط بالا نقض نمی‌شود.

پاسخنامه

۱. گزینه ۱ درست است.

اگر حالت‌های مختلف اجرای این دو فرایند را بررسی کنید خواهید دید که شرط انحصار متقابل تأمین شده و دو فرایند نمی‌توانند همزمان وارد ناحیه بحرانی خود شوند؛ همچنین امکان بن‌بست و گرسنگی وجود نخواهد شد. حالت اول:

فرایند P_1		فرایند P_2
$N_1 = 1;$		
$N_1 := N_2 + 1 = 0 + 1 = 1$	$\rightarrow$	
		$N_2 := 1;$
	$\leftarrow$	$N_2 := N_1 + 1 = 1 + 1 = 2$
Loop exit when $N_2=0$ or $N_1 \le N_2$; end loop;	$\rightarrow$	
		loop exit when $N_1=0$ or $N_2<N_1$; end loop;

چون مقدار N_1 کوچک‌تر از N_2 است، P_1 وارد ناحیه بحرانی خواهد شد و P_2 در حلقه می‌چرخد. حال اگر ترتیب اجرا را برعکس کنیم فرایند P_2 وارد ناحیه بحرانی خود شده و فرایند P_1 در حلقه خواهد ماند. حالت دوم:

فرایند P_1		فرایند P_2
$N_1 = 1;$	$\rightarrow$	
	$\leftarrow$	$N_2 := 1;$
$N_1 := N_2 + 1 = 1 + 1 = 2$		
Loop exit when $N_2=0$ or $N_1 \le N_2$; end loop;	$\rightarrow$	
		$N_2 := N_1 + 1 = 2 + 1 = 3$
		loop exit when $N_1=0$ or $N_2<N_1$; end loop;

در این حالت نیز چون مقدار N_1 کوچک‌تر از N_2 است P_1 وارد ناحیه بحرانی خود خواهد شد و P_2 در حلقه خواهد ماند. اگر ترتیب اجرا را برعکس کنیم این بار P_2 است که وارد ناحیه بحرانی شده و P_1 در حلقه خواهد ماند. حالت سوم:

فرایند P_1		فرایند P_2
$N_1 = 1;$	$\rightarrow$	
		$N_2 := 1;$
	$\leftarrow$	$N_2 := N_1 + 1 = 1 + 1 = 2$
$N_1 := N_2 + 1 = 2 + 1 = 3$		
Loop exit when $N_2=0$ or $N_1 \le N_2$; end loop;	$\rightarrow$	
		loop exit when $N_1=0$ or $N_2<N_1$; end loop;

در این حالت نیز چون $N_2 < N_1$ است P_2 وارد ناحیه بحرانی خود خواهد شد و P_1 منتظر خواهد ماند. با ترتیب عکس، P_1 وارد ناحیه بحرانی شده و P_2 دچار انتظار مشغول خواهد شد. حالت چهارم:

فرایند P_1		فرایند P_2
$N_1 = 1;$		
$N_1 := N_2 + 1 = 0 + 1 = 1$	$\rightarrow$	
Loop exit when $N_2=0$ or $N_1 \le N_2$; end loop;		
		$N_2 := 1;$
		$N_2 := N_1 + 1 = 1 + 1 = 2$
		loop exit when $N_1=0$ or $N_2<N_1$; end loop;

در این حالت نیز P_1 وارد ناحیه بحرانی شده و P_2 در حلقه خواهد ماند و مانند حالت‌های قبلی تغییر ترتیب اجرای دو بخش بالا تنها ترتیب ورود فرایندها را به ناحیه بحرانی تغییر خواهد داد. البته حالت‌های دیگری را با تغییر ترتیب اجرای شرط حلقه می‌توان ایجاد کرد که بدیهی است به نتایج مشابهی خواهیم رسید. همان‌طور که دیده می‌شود در هیچ‌یک از حالات ذکرشده شرط انحصار متقابل نقض نشده و

در هر لحظه حداکثر یک فرایند در ناحیه بحرانی خود قرار دارند. درضمن هیچ‌گاه هر دو گرفتار حلقه انتظار نمی‌شوند و بن‌بست پیش نمی‌آید.

بررسی احتمال قحطی: اگر فرایند درون ناحیه بحرانی خارج شود و دوباره بخواهد وارد ناحیه بحرانی شود، باید این اطمینان حاصل شود که اگر فرایند مقابل قبلاً درخواست ورود به ناحیه بحرانی را داده و تاکنون منتظر مانده باشد، از ورود مجدد فرایند مذکور به ناحیه بحرانی جلوگیری شود و نوبت فرایند منتظر نادیده گرفته نشود. مثلاً فرض کنید P_1 در ناحیه بحرانی قرار دارد و P_2 درخواست ورود به ناحیه بحرانی را دارد. در این لحظه، مانند حالت چهارم $N_1=1$ و $N_2=2$ است. در صورت خروج فرایند P_1، اگر پردازنده در اختیار این فرایند بماند تا اینکه دوباره قصد ورود به ناحیه بحرانی را داشته باشد، با اجرای کد زیر:

$N_1 = 1;$
$N_1 := N_2 + 1 = 2 + 1 = 3$
loop exit when $N_2=0$ or $N_1 \le N_2$; end loop;

N_1 برابر 3 شده و با توجه به اینکه مقدار N_2 قبلاً 2 بوده شرط حلقه P_1 برقرار نبوده و این فرایند نمی‌تواند وارد ناحیه بحرانی شود. اما شرط حلقه در فرایند P_2 برقرار بوده و می‌تواند وارد ناحیه بحرانی شود. پس عدالت در استفاده از ناحیه بحرانی رعایت می‌شود و قحطی به وجود نمی‌آید.

۲. گزینه ۴ صحیح است.

با توجه به اینکه مقدار اولیه کلیه سمافورهای دودویی برابر با 0 است، برای اینکه «بدنه فرایند» بتواند اجرا شود (نه اینکه CPU بگیرد و با فراخوانی P مسدود شود) باید ببینیم سمافورهایی که در آن فرایند P (Down) می‌شوند در کدام فرایندها V می‌شوند (پیش‌نیاز) و قبل از اجرای آن فرایند، فرایندهای پیش‌نیاز حتماً اجرا شوند.

فرایند	فرایندهای پیش‌نیاز
P_1	-
P_2	P_1
P_3	P_1
P_4	P_2
P_5	P_2, P_3
P_6	P_3, P_4

گزینه ۴ نادرست است. چون فرایند P_5 باید بعد از فرایند P_2 اجرا شود.

۳. گزینه ۱ درست است.

این الگوریتم همان راه‌حل Peterson است که فقط نام متغیرها عوض شده و منطق Flagها معکوس شده است و همان‌طور که در متن درس گفته است این راه‌حل تمام شرایط اصلی را برآورده می‌کند.

۴. گزینه ۲ درست است.

اگر فرایند P_3، دو دور بزند و دو بار کاراکتر "C" را چاپ کند مقدار سمافور C، 0 شده و در دور سوم حتماً مسدود می‌شود تا اینکه فرایندP_1 حداقل یک دور بزند و پس از چاپ کاراکتر "A"، فرایند P_3 را بیدار کند تا بتواند مجدداً اجرا شود و کاراکتر "C" را برای سومین بار چاپ کند.

۵. گزینه ۱ درست است.

این سؤال تکراری است. چون در این راه‌حل، یا هر دو چنگال را همزمان برداشته و یا هیچ‌کدام را بر نمی‌دارد، بن‌بست پیش نخواهد آمد. اما همه چیز تصادفی است و عدالت و نوبت رعایت نمی‌شود و ممکن است یک فیلسوف به مدت نامعلوم گرسنه بماند. برای توضیحات بیشتر به الگوریتم غذا خوردن فیلسوف‌ها در متن کتاب مراجعه کنید.

۶. گزینه ۱ درست است.

فرض کنید A ابتدا اجرا شده و مقدار X را در رجیستر خود قرار می‌دهد و قبل از عمل INC، وقفه می‌آید و به B سوئیچ می‌شود. سپس B اجرا شده و چهار دور می‌زند که در این حالت مقدار X برابر 4 خواهد شد. حال دوباره به A برمی‌گردیم. اما A قبلاً مقدار X (برابر با 0) را خوانده و در رجیستر R خود قرار داده است. فرض کنید A دستورالعمل INC را اجرا کند و R را در X ذخیره کند. در این صورت مقدار X برابر 1 می‌شود و در این لحظه با آمدن وقفه دوباره به B برمی‌گردیم. با فرض اینکه پس از بارگذاری مقدار X و قرار گرفتن مقدار 1 در رجیستر R فرایند B، مجدداً وقفه بیاید و به فرایند A برگردیم و این فرایند چهار دور باقی‌مانده خود را به طور کامل اجرا کرده و خارج شود، مقدار X برابر با 5 خواهد شد. حال به B برمی‌گردیم که مقدار 1 را در رجیستر R خود دارد. در انتها فرایند B، دستورالعمل INC را اجرا کرده و R را که برابر 2 است در X ذخیره می‌کند و خارج می‌شود. بنابراین مقدار نهایی X برابر 2 خواهد بود. این حداقل مقدار x پس از اجرای کامل دو فرایند است.

X	Process A:	R_A	Switch	Process B:	R_B
0	Load R, X;	0			
			→	دور اول:	
0		0		Load R, X;	0
0		0		INC R;	1
1		0		Store X, R;	1
.					.
.				دور چهارم:	.
.					.
3		0		Load R, X;	3
3		0		INC R;	4
4		0		Store X, R;	4
	ادامه دور اول:		←		
4	INC R;	1			4
1	Store X, R;	1			4
			→	دور پنجم:	
1		1		Load R, X	1
	دور دوم:		←		
1	Load R, X;	1			1
1	INC R;	2			1
2	Store X, R;	2			1
.					.
.					.
.	دور پنجم:				.
4	Load R, X;	4			1
4	INC R;	5			1
5	Store X, R;	5			1
			→	ادامه دور پنجم:	
5				INC R;	2
2				Store X, R;	2

به دست آوردن حداکثر مقدار ممکن برای X یعنی 10 که از اجرای متوالی دو فرایند به طور کامل به دست می‌آید بسیار ساده است. از همین اطلاعات می‌توان گزینه‌های ۲، ۳ و ۴ را رد کرد.
شما به عنوان تمرین می‌توانید مقادیر 3 تا 9 را با تغییر در ترتیب اجرا و محل وقفه‌ها به دست آورید. مثلاً در مورد عدد 3 تنها کافی است که B در ابتدا به‌جای چهار دور، سه دور بزند و دو دور از آن باقی بماند که باعث می‌شود در مرحله آخر، مقدار X برابر 3 بشود.

۷. گزینه ۴ درست است.
مانیتور ذاتاً دوبه‌دو ناسازگار است (گزینه ۳) و فرایندی که خارج مانیتور باشد مانع از پیشروی فرایند مقابل و ورود آن به مانیتور نمی‌شود (گزینه ۲) و فرایندهای منتظر ورود به مانیتور مسدود شده (خوابیده) و در صف انتظار قرار می‌گیرند و با انتظار مشغول موجب اتلاف CPU و مشکلاتی مانند اولویت معکوس نمی‌شوند (گزینه ۱).

فصل چهارم

بن بست

«یک مجموعه از فرایندها در صورتی منجر به بن‌بست می‌شوند که هر یک از فرایندهای درون مجموعه منتظر رویدادی باشد که فقط فرایند دیگری از همین مجموعه می‌تواند باعث ایجاد آن شود.»

می‌توان بن‌بست را به چند دسته تقسیم کرد:

۱. بن‌بست منابع : مثلاً فرایند P1 منتظر منبع R1 است که اکنون در اختیار فرایند P2 است و فرایند P2 نیز منتظر منبع R2 است که اکنون در اختیار فرایند P1 است.

۲. بن‌بست ارتباطی: مثلاً فرایند P1 در انتظار رسیدن یک پیام از طرف فرایند P2، فراخوان مسدودکننده receive را صدا زده است و فرایند P2 نیز با یک فراخوان سیستمی wait بر روی سمافور S مسدود شده و فقط فرایند P1 باید یک up بر روی این سمافور انجام دهد.

۳. بن‌بست ترکیبی: مثلاً فرایند P1 در انتظار یک signal از طرف فرایند P2، بر روی متغیر شرطی C، wait کرده است و فرایند P2 نیز منتظر منبع R2 است که اکنون در اختیار فرایند P1 است.

سیستم عامل در جدولی همواره ثبت می‌کند که آیا منبعی آزاد است و یا به کدام پردازش تخصیص داده شده است. اگر فرایندی درخواست منبعی را بکند که هم اکنون در اختیار فرایند دیگری است در این حال می‌تواند به صف پردازش‌های منتظر آن منبع اضافه گردد.

سیستم عامل برای مدیریت هر نوع منبع به ۴ عمل نیاز دارد:

۱ـ Resource Status

۲ـ Scheduling

۳ـ Allocation

۴ـ Release

*بدیهی است که اگر دستورات پردازش‌ها به صورت ترتیبی اجرا شوند هیچ گاه بن بست روی نخواهد داد.

*برای رخ دادن یک بن بست هر چهار شرط زیر باید برقرار باشد.

۱ـ انحصار متقابل (Mutual Exculsion) : یعنی حداقل یک منبع باید غیرقابل اشتراک باشد. به عبارت دیگر فقط یک پردازش در هر زمان می‌تواند از منبع استفاده کند.

۲ـ گرفتن و منتظر ماندن (Hold and wait) : باید پردازشی وجود داشته باشد که حداقل یک منبع را گرفته و در حال انتظار برای کسب منابع اضافه‌تر باشد.

۳ـ انحصاری بودن (No Preemption) : منبع را نمی‌توان به اجبار از پردازش گرفت و پردازش پس از اتمام کارش داوطلبانه آن را رها می‌کند.

۴ـ انتظار چرخشی (Circulor Wait) : بایستی مجموعه‌ای از پردازش‌ها $\{P_0 , P_1 , ... , P_n\}$ وجود داشته باشد به طوری که P_0 منتظر منبعی از P_1 ، P_1 منتظر منبع از $P_2,...,P_n$ منتظر منبعی از P_0 باشد.

نکات گراف تخصیص منبع

- ✓ اگر گراف دارای هیچ سیکلی (حلقه یا Loop) نباشد هیچ پردازشی در سیستم در بن بست نخواهد بود. از سوی دیگر اگر گراف دارای سیکلی باشد احتمال دارد بن بست وجود داشته باشد. پس وجود حلقه در گراف شرط لازم برای بن بست است و نه شرط کافی.
- ✓ اگر در گراف، هر منبع دقیقاً یک نمونه داشته باشد آنگاه اگر گراف حلقه داشته باشد بدین معناست که حتماً بن بست رخ داده است. ولی اگر هر نوع منبع نمونه‌های متعددی داشته باشد آنگاه حلقه الزاماً به معنای وقوع بن بست نیست.

نحوه اداره بن بست

۴ روش برای برخورد با بن بست وجود دارد.

۱ـ پیشگیری یا جلوگیری از بن بست (Deadlock Prevention) : منظور این است که روشی را به کار بریم که یکی از چهار شرط لازم برای وقوع بن بست پدید نیاید. بدین ترتیب بن بست هرگز رخ نخواهد دارد.

۲ـ اجتناب از بن بست (Dead Avoidance) : در این روش درخواست‌ها و رهایی منابع آتی بررسی می‌شوند و تصمیم گرفته می‌شود که آیا درخواست جاری اگر پاسخ داده شود منجر به بن بست خواهد شد یا خیر. در صورتی که پاسخ به درخواست جاری در آینده منجر به بن‌بست شود این درخواست به تعویق انداخته می‌شود.

۳ـ آشکار سازی و بازیافت (Detection & Recovery) : در این سیستم نه الگوریتم‌های پیشگیری و نه الگوریتم‌های اجتناب به کار گرفته می‌شود. لذا ممکن است بن بست رخ دهد. در این روش سیستم بررسی شده و درصورت بروز بن بست تشخیص داده شده و الگوریتم دیگری سیستم را از بن بست خارج می‌کند.

۴ـ صرف نظر کردن از بن بست (الگوریتم Ostrich) : در این سیستم در واقع هیچ عملی در مقابل بن بست انجام داده نمی‌شود. در صورتی که بن بست منجر به از کار افتادن سیستم شود (Hang) آنگاه سیستم به صورت دستی Reset می‌شود.

در اکثر سیستم عامل‌های امروزی از روش چهارم استفاده می‌شود. چون بن‌بست به ندرت رخ می‌دهد.

پیشگیری از بن بست (Deadlock Prevention)

در جدول زیر چهار شرط کافمن برای پیشگیری از بن بست بررسی می‌گردد:

شرط	روش	امکان پیاده‌سازی عملی (در سیستم های همه‌منظوره)	هزینه روش
انحصار متقابل	spool کردن همه منابع بر روی دیسک. مانند spool کردن چاپگر	اولاً غیر ممکن است. همه منابع قابل spool نمی‌باشند؛ مثلاً رکوردهای فایل یا PCB. ثانیاً برای منابعی مانند چاپگر نیز به علت محدودیت فضای spool بر روی دیسک شاید بن‌بست رخ دهد.	
نگهداری و انتظار	درخواست تمام منابع در ابتدای اجرای فرایند	غیر ممکن است. چون فرایند ها در ابتدا باید نیاز آینده خود به منابع را پیش‌بینی کنند.	اولاً باعث هدر دادن منابع از لحظه درخواست تا زمان استفاده می‌شود. ثانیاً موجب انتظار طولانی فرایندها می‌شود و اگر بد عمل کنیم، قحطی پیش می‌آید.
	پس دادن منابع فعلی و درخواست مجدد آنها با منابع جدید مورد نیاز	غیر ممکن است. چون منابع انحصاری‌اند؛ در غیر این صورت بن‌بست به وجود نمی‌آید.	
انحصاری	باز پس‌گیری پیش هنگام منابع	غیر ممکن است. چون منابع انحصاری‌اند؛ گرفتن این منابع معادل با کشتن فرایند است.	
انتظار چرخشی	هر فرایند در هر لحظه فقط یک منبع در اختیار داشته باشد.	غیر ممکن است. چون ممکن است فرایندها در هر لحظه به چندین منبع نیاز داشته باشند.	هدر دادن شدید منابع بی‌کار
	شماره گذاری منابع و درخواست آنها به ترتیب (مثلاً صعودی)	غیر ممکن است. چون دسترسی فرایندها به منابع، ترتیب خاصی ندارد. درخواست پیشاپیش منابع برای حفظ ترتیب نیز نیاز به پیش‌بینی آینده دارد.	درخواست پیشاپیش منابع برای حفظ ترتیب، منابع را هدر می‌دهد.
	شماره گذاری منابع و درخواست منابعی که شماره آن ها بزرگ‌تر از منابع فعلی فرایند است.	غیر ممکن است. چون دسترسی فرایندها به منابع، ترتیب خاصی ندارد. درخواست پیشاپیش منابع برای حفظ ترتیب نیز نیاز به پیش‌بینی آینده دارد.	درخواست پیشاپیش منابع برای حفظ ترتیب، منابع را هدر می‌دهد.

اجتناب از بن بست با گراف تخصیص منبع

اگر در سیستمی از هر نوع منبع فقط یک نوع داشته باشیم می‌توانیم از این روش استفاده کنیم، در غیر این صورت نمی‌توانیم.

- ✓ در این گراف علاوه بر کمان‌های درخواست و تخصیص،کمان ادعا (Claim) نیز وجود دارد که با خط چین نمایش داده می‌شود و بیانگر آن است که پردازش P_i ممکن است در آینده منبع R_j را درخواست کند.
- ✓ جهت اجتناب از بن بست هنگامی که پردازش P_i منبع R_j را درخواست می کند این درخواست به شرطی اعطا می‌شود که تبدیل کمان درخواست به کمال تخصیص منجر به تشکیل یک حلقه در گراف نگردد.

اجتناب از بن بست با الگوریتم بانکدار (Banker's Algorithm)

Nتعداد پردازش‌ها و m تعداد انواع منابع است.

- ✓ بردار Available به طول m که تعداد منابع آزاد از هر نوع را نشان می‌دهد. Available [j] = x یعنی از منبع R_j به تعداد x نمونه در دسترس وجود دارد.
- ✓ ماتریس Max به ابعاد $n \times m$ ماکزیمم نیاز هر پردازش را تعریف می‌کند.
- ✓ ماتریس Allocation به ابعاد $n \times m$ که تعداد منابع از هر نوع که در حال حاضر به هر پردازش تخصیص یافته است را مشخص می‌کند.
- ✓ ماتریس Need به ابعاد $n \times m$ که مشخص می‌سازد هر پردازش چه نیازی در حال حاضر دارد.
- ✓ بدیهی است که $Need[i,j] = Max[i,j] - Allocation[i,j]$

الگوریتم

۱ـ در ماتریس نیازها به دنبال سطری بگردید که کوچکتر از بردار منابع در دسترس (Available) باشد. اگر چنین سطری وجود ندارد سیستم در حالت ناامن است و الگوریتم تمام می‌شود.

۲ـ اگر سطری کوچکتر از بردار Available بود منابع را به پردازش اختصاص می‌دهیم تا تمام شود و تمام منابع در دسترس خود را آزاد سازد . حال منابع آزاد شده را به بردار Available اضافه می‌کنیم و پردازش را به عنوان Terminated علامت می‌زنیم.

۳ـ مراحل ۱و۲ را مرتباً تکرار می‌کنیم تا زمانی که همه پروسس‌ها Terminated شوند یا این که در مرحله ۱ از الگوریتم خارج شویم که در این صورت سیستم در بن بست است.

- ✓ اگر در یک سیستم در لحظه t پردازشی درخواست از چند منبع را بکند این درخواست را به ماتریس Alloc در سطر آن پردازش اضافه می‌کنیم و این سطر را از ماتریس Need و بردار Available کم می‌کنیم. حال بررسی می‌کنیم که سیستم به بن بست می‌رسد یا خیر.
- ✓ مجموع درایه‌های ستون j از ماتریس Alloc و همان درایه از بردار Available در هر وضعیت تعداد کل نمونه‌ها از منبع j را مشخص می‌سازد.
- ✓ یک ایراد الگوریتم بانکدار این است که ممکن است یک پردازش از ابتدا حداکثر نیازهای خودش را نتوانند اعلام کند.
- ✓ اگر در یک سیستم n پردازش و m منبع از یک نوع موجود باشد و شرط زیر برقرار باشد هیچگاه بن بست رخ نمی‌دهد.

$$\sum_{i=1}^{n} Request[i] < m + n$$

آشکار سازی یا تشخیص بن بست (Deadlock Detection)

روش‌های تشخیص بن بست به دو دسته تقسیم می‌شوند:

۱ـ وقتی از هر منبع یک نمونه جود دارد.

۲ـ وقتی از هر منبع چند نمونه وجود دارد.

تشخیص بن بست برای حالت یک نمونه از هر منبع

در این روش از روی گراف تخصیص منبع گراف انتظار (wait – for – groph) را به دست می‌آوریم. برای به دست آوردن گراف انتظار گره‌های منبع را از گراف تخصیص حذف کرده و کمان‌های مناسبی را با هم ترکیب می‌کنیم.

اگر در گراف انتظار (wait for graph) حلقه وجود داشته باشد آنگاه سیستم به بن بست رسیده است.

تشخیص بن بست برای حالت چند نمونه از هر منبع

در این روش به جای ماتریس Need از ماتریس $n \times m$ به نام Request استفاده می‌شود که نیازهای فعلی هر پردازش را نشان می‌دهد. درایه‌های هر ستون ماتریس Alloc را با هم جمع می‌کنیم و از کل منابع کم می‌کنیم تا برای Available را به دست آوریم. حال می‌بینیم که اگر به Requestها پاسخ دهیم سیستم به بن بست می‌رسد یا خیر.

Deadlock Recovery : وقتی سیستم عامل وجود بن بست را تشخیص داد یک روش آن است که سیستم عامل به کاربر اطلاع دهد که بن بست رخ داده تا خود کاربر به صورت دستی آن را اداره کند. ولی اگر خود سیستم عامل بخواهد آن را بازیافت کند دو راه وجود دارد. یا آن که پردازش‌هایی را خاتمه دهد تا انتظار چرخشی شکسته شود یا این که منابعی را از پردازش‌ها پس بگیرد.

۱ـ خاتمه دادن به پردازش‌ها : در این روش کلیه منابع اختصاص یافته به پردازش‌هایی که خاتمه می‌یابند به سیستم برگردانده می‌شوند. در این روش یا می‌توان پردازش‌های درگیر بن بست را و یا فقط یکی از آن‌ها را خاتمه داد. خاتمه دادن تمام پردازش‌های درگیر بن بست هزینه سنگینی دارند.

روش خاتمه دادن یکی یکی نیز سربار زیادی دارد زیرا سیستم پس از حذف هر پردازش با یک الگوریتم تشخیص بن‌بست را صدا بزند تا مشاهده کند آیا بن بست رفع شده یا خیر.

۲ـ پس گرفتن منابع : در این روش منابعی از یک پردازش گرفته شده و در اختیار پردازش دیگری قرار داده می‌شود. برای اینکار سه موضوع باید مشخص گردد:

الف) انتخاب منبع و پردازش‌های مورد نظر

ب) باز گرداندن به عقب (Roll back)

ج) قحطی زندگی،یعنی منابع همواره از یک پردازش خاص گرفته شود.

نکات اضافی بن بست

۱)مرتبه اجرای الگوریتم بانکدار $m \times n^2$ می‌باشد (n تعداد پردازش‌ها و m تعداد منابع) .

۲ـ الگوریتم تشخیص حلقه در گراف از مرتبه $O\left(n^2\right)$ می‌باشد که n تعداد رئوس گراف است.

- ✓ علت اصلی استفاده از Spooling در مبحث بن بست این است که دستگاه I/O به طور اشتراکی توسط چند Process استفاده می‌شود.
- ✓ هنگامی که سیستم در بن بست قرار می‌گیرد الگوریتم‌هایی برای کشف بن بست وجود دارند. ولی در حالتی که سیستم در بن بست قرار ندارد هیچ الگوریتمی برای بررسی پردازش‌های بلوکه شده وجود ندارد.
- ✓ در حالت نا امن ممکن است بعضی پردازه‌ها منابع خود را آزاد کرده و سیستم به بن بست نرسد، لذا در حالت ناامن وقوع بن بست قطعی نیست.
- ✓ امن بودن سیستم به تعداد پردازش‌ها بستگی ندارد.

تست‌ها

۱. یک سیستم کامپیوتری دارای شش عدد Tape Drive **است که n پردازه برای دستیابی به آن‌ها رقابت می‌کنند. هر پردازه به درایو نیاز دارد. این سیستم به ازای حداکثر چه ارزش‌هایی از n فاقد بن‌بست است؟ (کارشناسی ارشد کامپیوتر ـ سراسری ۷۶)**

۱) $n < 4$ ۲) $n \leq 3$ ۳) $n < 6$ ۴) $n \leq 2$

۲. در سیستمی با 5 پردازه (Process) **و 3 نوع منبع** (Resource) **وضعیت تخصیص منابع به شکل زیر است:**

(کارشناسی ارشد کامپیوتر – سراسری ۷۶)

	تخصیص‌یافته			حداکثر مورد نیاز			موجودی اولیه		
	A	B	C	A	B	C	A	B	C
P_0	0	1	2	3	6	8	8	6	10
P_1	2	0	3	7	3	6			
P_2	3	2	0	5	3	3			
P_3	1	0	2	4	5	9			
P_4	1	1	0	2	3	3			

اگر در این وضعیت، درخواستی برای یک واحد دیگر از منبع A توسط پردازه P_3 صادر شود:

۱) پس از انجام درخواست بالا وقوع بن‌بست قطعی است.

۲) پس از انجام درخواست بالا احتمال وقوع بن‌بست وجود دارد.

۳) قبل از درخواست بالا احتمال وقوع بن‌بست وجود دارد.

۴) قبل از انجام درخواست بالا وقوع بن‌بست قطعی است.

۳. سیستمی شامل 5 فرایند هم‌روند و 2 منبع یکسان، قابل استفاده مجدد، Mutual Exclusive **(دوبه‌دو ناسازگار یا دارای انحصار متقابل) و** preemptive **(غیر انحصاری) را در نظر بگیرید. به شرط آنکه فرایند حداکثر به 2 منبع نیاز داشته باشد، تعداد وضعیت‌های بن‌بست** (Deadlock States) **در این سیستم به خاطر منبع مذکور حداکثر چند حالت است؟**

(کارشناسی ارشد IT ـ سراسری ۸۴)

۱) 20 حالت ۲) 10 حالت ۳) 5 حالت ۴) 0 (صفر) حالت

۴. در یک سیستم 4 فرایند و 5 نوع دستگاه موجود است. تعداد دستگاه‌های تخصیص‌یافته و حداکثر نیاز فرایندها در زیر آمده است. اگر موجودی فعلی (1 1 x 0 0) **باشد، حداقل مقدار x چند باشد که سیستم در وضعیت مطمئن** (Safe) **قرار داشته باشد؟**

(کارشناسی ارشد IT ـ سراسری ۸۶)

	اختصاص‌یافته						حداکثر نیاز				
A	1	1	0	2	1	1	3	1	2	1	1
B	2	0	1	1	0		0	2	2	1	2
C	1	1	0	1	0		0	1	3	1	2
D	1	1	1	1	0		1	1	3	2	1

۱) x=0 ۲) x=1 ۳) x=2 ۴) x=3

۵. سیستمی با 5 فرایند و 4 منبع مطابق ماتریس‌های زیر مفروض است. مطابق الگوریتم بانکداران چند فرایند حتماً دچار بن‌بست می‌شوند یا در بن‌بست هستند؟ (کارشناسی ارشد IT ـ آزاد ۸۵)

فرایند	MAX			
	R_0	R_1	R_2	R_3
P_0	2	2	2	1
P_1	2	1	0	2
P_2	3	0	3	0
P_3	2	3	1	3
P_4	4	1	1	0

USED			
R_0	R_1	R_2	R_3
0	0	1	1
1	1	0	1
2	0	2	0
2	1	1	1
1	1	0	0

Available	2	0	1	0

۱) 2 ۲) 3 ۳) 4 ۴) صفر

۶. برای وقوع بن‌بست (Dead Lock) باید همزمان 4 شرط اتفاق افتاده باشد. کدام گزینه مشخص‌کننده این چهار شرط است؟ (کارشناسی ارشد کامپیوتر ـ سراسری ۸۱)

۱) Partial allocation, Circular wait, Preemption, Multi exclusion

۲) Hold and wait, Circular wait, Nonpreemption, Mutual exclusion

۳) Total allocation, Circular wait, Nonpreemption, Mutual exclusion

۴) Partial allocation, Circular wait, Nonpreemption , Non mutual exclusion

۷. سیستمی دارای 5 پردازه (Process) و 4 منبع (Resource) در حالت (State) زیر قرار دارد:

تعداد کل منابع اولیه

R_0	R_1	R_2	R_3
6	3	4	2

منابعی که هنوز مورد نیازند

R_0	R_1	R_2	R_3
1	1	0	0
0	1	1	2
3	1	0	0
0	0	1	0
2	1	1	0

منابع تخصیص‌یافته

	R_0	R_1	R_2	R_3
P_0	3	0	1	1
P_1	0	1	0	0
P_2	1	1	1	0
P_3	1	1	0	1
P_4	0	0	0	0

در چه صورتی وقوع بن‌بست حتمی است؟ (کارشناسی ارشد کامپیوتر ـ سراسری ۷۸ و ۸۵)

۱) پردازه P_0 یک واحد از منبع R_2 را درخواست کند.

۲) پردازه P_1 یک واحد از منبع R_2 و پردازه P_3 آخرین واحد باقی‌مانده R_2 را درخواست کند.

۳) پردازه P_1 یک واحد از منبع R_2 و پردازه P_4 آخرین واحد باقی‌مانده R_2 را درخواست کند.

۴) پردازه P_3 یک واحد از منبع R_2 و پردازه P_4 تمام منابع مورد نیازش را درخواست کند.

پاسخنامه

۱. گزینه ۳ درست است.

در متن درس دیدیم که اگر Max[i, j] حداکثر نیاز فرایند i به منبع j باشد و E[j] کل موجودی منبع j را نشان دهد، شرط عاری بودن سیستم از بن‌بست عبارت است از:

$$\sum_{i=1}^{n} Max[i, j] < n + E[j]$$

چون فقط یک نوع منبع داریم رابطه ساده‌شده به صورت زیر خواهد بود:

$$\sum_{i=1}^{n} Max[i] < n + E$$

$$\sum_{i=1}^{n} 2 < n + 6 \Rightarrow n \times 2 < (6 + n) \Rightarrow n < 6$$

یعنی تا حداکثر 5 فرایند سیستم هرگز دچار بن‌بست نخواهد شد و این نکته بدیهی است و به صورت ذهنی هم قابل درک است چون در بدترین حالت اگر هر یک از فرایندها یکی از منابع را در اختیار بگیرند، باز هم یک منبع آزاد باقی می‌ماند که می‌تواند باعث اجرای کامل یکی از فرایندها شود ولی اگر 5 منبع داشتیم و هر فرایند یک منبع را می‌گرفت و درخواست منبع دوم را داشت بن‌بست پیش می‌آمد.

۲. گزینه ۴ درست است.

یکی از 4 شرط بن‌بست منابع، انحصاری بودن (Non-preemptive) منابع است. چون همه منابع در این سیستم غیر انحصاری هستند، بن‌بست منابع رخ نمی‌دهد.

۳. گزینه ۳ درست است.

Need = MAX – Allocation

	Need				
A	0	1	0	0	2
B	0	2	1	0	0
C	1	0	3	0	0
D	0	0	2	1	1

A=(0, 0, x, 1, 1)

با توجه به منابع آزاد تنها فرایندی که می‌تواند به عنوان شروع مسیر امن، کل منابع مورد نیازش را در اختیار بگیرد، فرایند D است و برای تحقق این امر، حداقل x باید برابر 2 باشد. البته در این سؤال با همین مقدار می‌توان مسیر امن را تا انتها پیش برد ولی یادتان باشد که همیشه لزوماً این طور نخواهد بود و یک‌بار مسیر امن را تا انتها چک کنید:

$$(0,0,2,1,1) \xrightarrow[+(1,1,1,1,0)]{D} (1,1,3,2,1) \xrightarrow[+(1,1,0,1,0)]{C} (2,2,3,3,1) \xrightarrow[+(2,0,1,1,0)]{B} (4,2,4,4,1)$$

البته صورت این سؤال باید اصلاح شود و حداکثر نیاز فرایند A به منبع پنجم به‌جای 3 باید 2 قرار گیرد چون موجودی اولیه آن منبع 2 است.

۴. گزینه ۴ درست است (ولی ممکن است طراح اشتباه کرده و گزینه ۲ را انتخاب کرده باشد).

☜سخنی با طراح تست:

اگر منظور شما گزینه ۲ است، گزینه انتخابی شما قطعاً غلط است. چون الگوریتم بانکدار فقط وضعیت سیستم (امن یا ناامن بودن) را مشخص می‌کند و ناامنی نه به معنی وقوع بن‌بست در گذشته و نه به معنی وقوع حتمی بن‌بست در آینده است بلکه به معنی احتمال وقوع بن‌بست در آینده (ریسک بن‌بست) است.

به‌هرحال اگرچه نیازی به حل تست نیست اما نشان می‌دهیم که این حالت ناامن است.

Need

	R_0	R_1	R_2	R_3
P_0	2	2	1	0
P_1	1	0	0	1
P_2	1	0	1	0
P_3	0	2	0	2
P_4	3	0	1	0

$$(2,0,1,0)\xrightarrow[+(2,0,2,0)]{P_2}(4,0,3,0)\xrightarrow[+(1,1,0,0)]{P_4}(5,1,3,0)$$

این منابع نمی‌توانند نیاز 3 فرایند P_0، P_1 و P_3 را برآورده سازند. پس مسیر امن وجود ندارد و این حالت ناامن است. اما خیلی از دوستان فکر می‌کنند که 3 فرایند مذکور دچار بن‌بست شده‌اند یا حتماً دچار خواهند شد که این درست نیست. اگر خوش‌بین باشیم حتی همین 3 فرایند ممکن است منابع در اختیار خود را رها کنند و لزومی ندارد که فکر کنید فقط درخواست داریم و نه آزادسازی! این مورد از بدیهیات سیستم عامل است و متأسفانه سال‌هاست که هم برخی از طراحان تست و هم بعضی از نویسندگان در مورد آن اشتباه می‌کنند.

۵. گزینه ۲ درست است.

کاش طراح سؤال به بن‌بست «منابع» اشاره می‌کرد.

۶. گزینه ۳ درست است.

اولاً این تست با این بیان که «در چه صورت وقوع بن بست حتمی است» غلط است. چون با اطلاعات این تست می‌توان تشخیص داد که وضعیت امن است یا خیر و می‌دانید که وضعیت ناامن به معنی حتمی بودن وقوع بن‌بست نیست بلکه ریسک و احتمال وقوع بن‌بست را نشان می‌دهد. پس باید سؤال می‌شد که «در کدام گزینه در صورت تخصیص درخواست‌ها وضعیت ناامن خواهد شد؟»

راه تستی: در این تست‌ها معمولاً وضعیت قبل از درخواست و تخصیص، امن بوده است در غیر این صورت همه وضعیت‌های بعد از درخواست‌ها نیز ناامن خواهد بود. ابتدا توصیه می‌شود آغاز مسیرهای امن را پیدا کنیم.

$$P = \sum \text{Allocation} = (5,3,2,3)$$

$$A = E - P = (6,3,4,2) - (5,3,2,2) = (1,0,2,0)$$

$$(1,0,2,0)\xrightarrow[+(1,1,0,1)]{P_3}(2,1,2,1)\xrightarrow{P_0 \text{ or } P_4}\dots$$

نکته: راه نجات (مسیر امن) با P_3 شروع می‌شود و چون P_3 در آینده فقط 1 منبع R_2 می‌خواهد و 2 عدد از این منبع آزاد است، با تخصیص این 2 منبع به فرایندهایی غیر از P_3 راه فرار بسته خواهد شد.

گزینه ۲ تقاضای تخصیص هر 2 منبع R_2 را دارد اما چون یکی از آن‌ها را به P_3 می‌دهد آغاز راه فرار را سد نمی‌کند.

در گزینه ۴ یکی از منابع R_2 آزاد به P_3 داده می‌شود و بنابراین آغاز راه فرار را سد نمی‌کند.

در گزینه ۱، ازآنجاکه مقدار منبع درخواست‌شده، کمتر از باقی‌مانده نیاز فرایند P_0 است، پس با این درخواست موافقت نمی‌شود (خطا).

گزینه ۳ راه فرار را می‌بندد چون پس از تخصیص منابع درخواستی، هیچ منبع R_2 باقی نمی‌ماند و نمی‌توان مسیر امن را با P_3 شروع کرد و درنتیجه وضعیت ناامن خواهد شد.

راه کلاسیک:

در گزینه ۱، ازآن‌جاکه مقدار منبع درخواست‌شده، کمتر از باقی‌مانده نیاز فرایند P_0 است، پس با این درخواست موافقت نمی‌شود (خطا).

گزینه ۲: در صورت پذیرش درخواست‌ها ماتریس‌ها و بردار A به صورت زیر به‌روزرسانی می‌شوند:

فرایند	Allocation				Need			
	R_0	R_1	R_2	R_3	R_0	R_1	R_2	R_3
P_0	3	0	1	1	1	1	0	0
P_1	0	1	❶	0	0	1	⓿	2
P_2	1	1	1	0	3	1	0	0
P_3	1	1	❶	1	0	0	⓿	0
P_4	0	0	0	0	2	1	1	0

A	1	0	⓿	0

این حالت امن است چون برای مثال تعدادی از مسیرهای امن به صورت زیر خواهد بود. دقت کنید وقتی همه فرایندهای باقی‌مانده قابل اجرا باشند نیازی به ادامه راه‌حل نیست و پس از آن تمامی ترکیب‌های مختلف، مسیرهای امن محسوب می‌شوند:

$$(1,0,0,0) \xrightarrow[+(1,1,1,1)]{P_3} (2,1,1,1) \xrightarrow[+(3,0,1,1)]{P_0} (5,1,2,2) \xrightarrow{P_1 \text{ or } P_2 \text{ or } P_4} \dots$$

گزینه ۳: در صورت پذیرش درخواست‌ها ماتریس‌ها و بردار A به صورت زیر به‌روزرسانی می‌شوند:

فرایند	Allocation				Need			
	R_0	R_1	R_2	R_3	R_0	R_1	R_2	R_3
P_0	3	0	1	1	1	1	0	0
P_1	0	1	❶	0	0	1	⓿	2
P_2	1	1	1	0	3	1	0	0
P_3	1	1	0	1	0	0	1	0
P_4	0	0	❶	0	2	1	⓿	0

A	1	0	⓿	0

این حالت ناامن است چون بردار A پاسخگوی نیاز باقی‌مانده هیچ‌یک از فرایندها نیست. اگرچه نیازی به ادامه راه‌حل نیست فقط به عنوان تمرین گزینه ۴ را هم رد می‌کنیم.

گزینه ۴: در صورت پذیرش درخواست‌ها ماتریس‌ها و بردار A به صورت زیر به‌روزرسانی می‌شوند:

فرایند	Allocation				Need			
	R_0	R_1	R_2	R_3	R_0	R_1	R_2	R_3
P_0	3	0	1	1	1	1	0	0
P_1	0	1	0	0	0	1	1	2
P_2	1	1	1	0	3	1	0	0
P_3	1	1	❶	1	0	0	⓿	0
P_4	0	0	0	0	2	1	1	0

A	1	0	❶	0

در این حالت، منابع مورد نیاز P_4 فعلاً آزاد نیست و این فرایند مسدود می‌شود ولی اشتباه نکنید بن‌بستی رخ نداده و وضعیت امن است چون برای مثال تعدادی از مسیرهای امن به صورت زیر خواهد بود:

$$(1,0,1,0) \xrightarrow[+(1,1,1,1)]{P_3} (2,1,2,1) \xrightarrow[+(3,0,1,1)]{P_0} (5,1,3,2) \xrightarrow{P_1 \text{ or } P_2 \text{ or } P_4} \dots$$

فصل پنجم

مدیریت حافظه

منظور از حافظه اصلی حافظه‌ای است که پردازنده برای دستیابی به دستورالعمل‌ها و داده‌ها مستقیماً به آن رجوع می‌کند.

به طور کلی برای مدیریت حافظه می‌توان ۴ وظیفه در نظر گرفت:

۱ـ ثبت وضعیت هر یک از مکان‌های حافظه

۲ـ تعیین سیاست و خط مشی تخصیص حافظه

۳ـ تکنیک تخصیص حافظه اصلی

۴ـ تکنیک آزاد کردن حافظه اصلی

*هر پیوند نگاشتی از یک فضای آدرس به فضای آدرس دیگر می‌باشد. پیوندها در زمان‌های زیر صورت می‌گیرد:

۱ـ در زمان Compile : اگر در زمان ترجمه معلوم گردد که فرآیند در چه بخشی از حافظه قرار خواهد گرفت.

۲ـ زمان بارگذاری (Load) : در این حالت Compiler کد Relocatable تولید می‌کند آدرس نسبی شروع بعداً معلوم می‌شود.

۳ـ زمان اجرا (Run Time) : چنانچه فرآیند در زمان اجرایش بتواند تغییر مکان دهد،آنگاه پیوند آدرس‌ها تا زمان اجرا به تعویق می‌افتد. در این حالت سخت افزار خاصی مورد نیاز است.

آدرس منطقی همان آدرس تولید شده توسط CPU می‌باشد در حالی که آدرس فیزیکی آدرس قابل رؤیت توسط واحد حافظه است. مجموعه آدرس‌های منطقی به نام فضای آدرس منطقی و مجموعه آدرس‌های فیزیکی به نام فضای آدرس فیزیکی نامیده می‌شوند.

جایگذاشت‌ها (Overlays): به منظور آن که فرآیندی محدودی به اندازه حافظه فیزیکی نگردد و بتواند بزرگتر از حافظه تخصیص یافته به آن اجرا شود تکنیکی به نام جایگذاشت به کار می‌رود. این Overlay آن است که در هر زمانی که داده و یا کدی از برنامه مورد نیاز می‌باشد در حافظه باز گردد. این روش به پشتیبانی مستقیم O.S نیاز ندارد و می تواند کاملاً توسط کاربر پیاده سازی شود.

مدیریت حافظه یکپارچه: در این شیوه پردازش در هنگام زمانبندی کل حافظه را به خود اختصاص می‌دهد. هنگامی که فرآیند انجام شد کل حافظه به وضعیت آزاد بازگردانده می‌شود. در این صورت برنامه‌ها از لحاظ اندازه محدود به مقدار حافظه اصلی هستند. اما این امکان وجود دارد که بااستفاده از جایگذاری برنامه‌های بزرگتر از حافظه اصلی را اجرا نمود.

* این روش نیازی به پشتیبانی سخت افزار ندارد. در چنین سیستم‌هایی عملکرد چند برنامگی وجود ندارد.

*وظیفه اصلی مدیریت حافظه انتقال برنامه به داخل حافظه جهت اجرا توسط پردازنده می‌باشد.

*در تمام O.S های جدید چند برنامگی انجام این وظیفه همراه با طرح حافظه مجازی صورت می‌پذیرد.

*حافظه مجازی به نوبه خود به دو روش اساسی قطعه بندی و صفحه بندی مبتنی است.

*روش بخش بندی ایستا: در این روش سیستم عامل بخش ثابتی از حافظه اصلی را اشغال می‌نماید و بقیه حافظه می‌تواند به دو صورت مختلف بخش‌بندی گردد :

۱)بخش‌های با اندازه‌های مساوی که در این حالت هر فرآیندی که اندازه آن کمتر یا مساوی انجام فشرده سازی برای رفع مشکل Extemal Frag در صورتی امکان پذیر است که کد برنامه‌ها Relocateable باشند.

اندازه بخش باشد می‌تواند به داخل هر بخش موجود بار شود. در این حالت به دو مشکل بر می‌خوریم:

الف) ممکن است برنامه بزرگتر از یک بخش باشد. در این صورت روش overlay می‌تواند استفاده شود.

ب) استفاده از حافظه اصلی ناکارآمد می‌شود. هر برنامه صرف نظر از اندازه‌اش فضای کامل را اشغال می‌کند. فضای به هدر رفته را تکه تکه شدن داخلی (Internal Fragmentation) می‌گویند.

۲) امکان دوم استفاده از بخش‌هایی با اندازه نامساوی که در این حالت هر دو مشکل مطرح شده می‌توانند کاهش یابند. در این روش تخصیص هر فرآیند می‌تواند به کوچکترین بخشی باشد که در آن جا می‌گیرد.

*به طور کلی روش بخش‌بندی ایستا ساده است و حداقل به نرم افزار سیستم عامل نیاز دارد.

***روش بخش‌بندی پویا:** در این روش سیستم عامل جدول شامل بخش‌های حافظه آزاد و اشغال را نگه می‌دارد. در ابتدا کل حافظه همانند یک بلوک آزاد بزرگ در نظر گرفته می‌شود. وقتی فرآیندی درخواست حافظه می‌نماید فضایی به اندازه کافی بزرگ را به آن تخصیص می‌دهیم و باقیمانده فضا را برای درخواست‌های آتی فرآیند نگه می‌داریم.

*حافظه در این روش به مرور به قطعاتی تقسیم می‌شود که ممکن است این قطعات به قدری کوچک باشند که مورد استفاده هیچ برنامه‌ای قرار نگیرد. این نوع تکه تکه شدن را External Fragmentation گوییم.

*سیستم عامل باید از وضعیت حافظه شامل محل شروع، طول یک بخش اشغال شده و برنامه‌هایی که این بخش‌ها را در اختیار دارند. مطلع باشد بدین منظور می‌توان از دو ساختار استفاده نمود.

الف) روش Bitmap: متناظر با هر واحد تخصیص یک بیت در نظر گرفته می‌شود:

0 : آزاد

1 : اشغال

ب) روش linked list: یک لیست پیوندی از قطعات تخصیص یافته و ازاد.

*یک روش برای مقابله با External Fragment فشرده سازی (Compacation) می‌باشد.

الگوریتم‌های مکان‌یابی و تخصیص حافظه

وقتی فرایندها و حفره‌ها در یک لیست مرتب شده بر اساس آدرس قرار می‌گیرند، الگوریتم‌های مختلفی جهت تخصیص حافظه به یک فرایند جدید وجود دارد.

اولین برازش

ساده‌ترین الگوریتم، **اولین برازش (First fit)** نام دارد: «از ابتدای حافظه شروع کن. فرایند را در اولین حفره‌ای قرار بده که در آن جا می‌شود».

نکته: این الگوریتم ساده و سریع است. کارایی آن نیز مناسب است.

برازش بعدی

نوع دیگری از همین الگوریتم، **برازش بعدی (Next fit)** نام دارد: «از محل آخرین تخصیص شروع کن. فرایند را در اولین حفره‌ای قرار بده که در آن جا می‌شود».شبیه‌سازی‌های انجام شده توسط Bays (در سال ۱۹۷۷) نشان می‌دهد که کارآیی Next fit کمی کمتر از First fit است.

نکته: این الگوریتم ساده و سریع است. و عیب آن شکستن سریع‌تر حفره‌های بزرگ انتهای حافظه و ایجاد مشکل در ورود فرایندهای بزرگ بعدی است. به همین دلیل کارایی (بهره‌وری) پایین‌تر از First fit دارد.

بهترین برازش

الگوریتم مشهور بعدی، **بهترین برازش (Best fit)** نام دارد: «تمام لیست را جستجو کن. فرایند را در کوچکترین حفره‌ای قرار بده که در آن جا می‌شود».

نکته: این اگوریتم سرعت پایین (به دلیل نیاز به جستجو در لیست) داشته و در حافظه تعداد زیادی حفره ریز بی‌مصرف ایجاد می‌کند. کارایی آن (بهره‌وری حافظه) در کل مناسب است.

بدترین برازش

برای این که مشکل به وجود آمدن حفره‌های بسیار کوچک در حافظه برطرف گردد، الگوریتم **بدترین برازش (Worst fit)** پیشنهاد شده است: «تمام لیست را جستجو کن. فرایند را در بزرگ‌ترین حفره موجود قرار بده؛ البته اگر در آن جا می‌شود!».

نکته: این الگوریتم سرعت پایین (به دلیل نیاز به جستجو در لیست) و کارایی پایین (بهره‌وری پایین حافظه) دارد.

برازش سریع

هم‌چنین الگوریتم تخصیص دیگری به نام **برازش سریع (Quick fit)** وجود دارد که برای هر دسته از فرایندها با اندازه‌های متداول، یک لیست جداگانه تهیه می‌کند. اگرچه یافتن حفره مناسب سریع است اما هرگاه یک فرایند خاتمه می‌یابد عمل ترکیب حفره‌ها بسیار وقت‌گیر خواهد بود.

حافظه مجازی:

قبلاً برای اجرای برنامه‌های بزرگ‌تر از حافظه فیزیکی، یک راه‌حل ضعیف به نام Overlay معرفی شد که اداره اموری مانند تقسیم برنامه به Overlayها و فراخوانی آنها را به عهده برنامه‌نویس می‌گذاشت که این کار برای برنامه‌نویس بسیار وقت‌گیر و خسته کننده بود. سپس راه حل بهتری پیشنهاد شد. این روش **حافظه مجازی** (Virtual memory) نام داشت.

حافظه مجازی در یک سیستم چندبرنامگی می‌تواند از یکی از سه تکنیک زیر استفاده کند:

- ✓ صفحه‌بندی (Paging): فقط صفحات مورد نیاز فرایندها در حافظه بار می‌شود.
- ✓ قطعه‌بندی (Segmentation): فقط قطعات مورد نیاز فرایندها در حافظه بار می‌شود.

۱. ترکیبی (قطعه‌بندی صفحه‌بندی‌شده): فقط صفحات لازم از قطعات مورد نیاز فرایندها در حافظه بار می‌شود.

۲. صفحه (page)	۳. قطعه (segment)
۴. اندازه یکسان	۵. اندازه متفاوت
۶. اندازه ثابت	۷. اندازه متغیر
۸. اندازه کوچک (معمولاً 0.5 KB تا 1 MB)	۹. می‌تواند بسیار بزرگ باشد (مثلاً تا چند صد مگابایت).
۱۰.معمولاً تعداد بسیار زیادی دارد.	۱۱.معمولاً تعداد نسبتاً کمی دارد.
۱۲.از دید برنامه‌نویس پنهان است.	۱۳.توسط خود برنامه‌نویس تعریف می‌شود.

صفحه‌بندی

در صفحه‌بندی، حافظه منطقی هر فرایند به تکه‌های کوچک با اندازه ثابت و یکسان تقسیم می‌شود که به هر تکه یک **صفحه** (Page) گفته می‌شود. همچنین حافظه اصلی (فیزیکی) به تکه‌هایی برابر با اندازه صفحه تقسیم می‌شود که **قاب صفحه** (Page frame) یا به طور خلاصه قاب (Frame) نامیده می‌شوند.

این تکنیک، فاقد تکه‌تکه‌شدن خارجی است و **تکه‌تکه‌شدن داخلی** را به حداقل می‌رساند. میزان تکه‌تکه‌شدن داخلی، با توجه به اندازه کوچک صفحات، ناچیز و قابل صرف‌نظر است، زیرا فقط در آخرین صفحه هر فرایند، به طور میانگین $\frac{p}{2}$ حافظه هدر می‌رود که p اندازه صفحه است. حداقل اتلاف تکه‌تکه‌شدن داخلی برابر 0 بایت (اگر اندازه فرایند مضرب صحیحی از اندازه صفحه باشد) و حداکثر آن برابر $p-1$ خواهد بود.

پراکندگی صفحات و ترجمه آدرس

سخت‌افزار MMU (واحد مدیریت حافظه) برای نگاشت آدرس‌های منطقی (نسبی) فرایندها به آدرس‌های فیزیکی نیاز به جداولی به نام **جدول صفحه** (Page table) دارد که سیستم‌عامل برای هر فرایند به طور مجزا تشکیل می‌دهد و در حافظه اصلی، در دسترس MMU قرار می‌دهد.

مدیریت حافظه صفحه بندی ساده (Simple Paging)

در مدیریت حافظه صفحه بندی شده هر فضای آدرسی متناظر با یک فرآیند به بخش‌های مساوی به نام صفحه تقسیم می‌گردد. همچنین حافظه فیزیکی نیز خود به نواحی به اندازه یکسان با صفحه به نام قاب یا Frame شکسته می‌شود.

*اندازه صفحه مانند اندازه Frame توسط سخت افزار معین می‌گردد.

وقتی فرآیندی باید اجرا گردد کلیه صفحاتش به داخل قاب‌های آزاد حافظه که لزوماً یکپارچه و پیوسته نیستند بارگذاری می‌شوند.

*درون برنامه هر آدرس منطقی متشکل از شماره صفحه و offset می‌باشد. این آدرس نسبی یکسان می‌باشد.

*برای ساده‌تر شدن طرح صفحه‌بندی تأکید می کنیم که اندازه صفحه و در نتیجه اندازه قاب باید توانی از 2 باشد.

*یک آدرس مجازی (منطقی) n+m بیتی را در نظر بگیرید، n بیت سمت چپ شماره صفحه و m بیت سمت راست انحراف را نشان می‌دهد. مراحل زیر برای ترجمه آدرس لازم است:

۱ـ استخراج شماره صفحه (n بیت سمت چپ آدرس منطقی).

۲ـ به کارگیری شماره صفحه به عنوان شاخص به جدول صفحه برای استخراج شماره قاب (f)

۳ـ برای به دست آوردن آدرس فیزیکی شماره قاب را به جای شماره صفحه در قاب آدرس منطقی قرار می‌دهیم. M بیت سمت راست که offset را نشان می‌دهد تغییر نمی‌کند.

- ✓ در صفحه بندی ساده External Fragmentation نداریم ولی ممکن است Internal Fragmentation داشته باشیم.
- ✓ ساختمان جدول صفحه: اکثر کامپیوترهای امروزی جداول صفحه بسیار بزرگی دارند. در این ماشین‌ها جدول صفحه در حافظه اصلی نگهداری می‌شود و یک رجیستر مبنای جدول صفحه (PTBR) به جدول صفحه اشاره می‌کند.
- ✓ اگر بخواهیم مکانی در صفحه i را دسترسی نمائیم ابتدا باید شماره قاب را در جدول صفحه به دست آوریم و سپس به محل مورد نظر در حافظه دسترسی نمائیم. بنابراین سرعت دسترسی به حافظه با ضریب احتمال پارگی خارجی در Best Fit از همه الگوریتم‌های انباره بیشتر است ۲ کاهش می‌یابد.راه حل مشکل استفاده از سخت افزار Associative Register یا TLB (Transition Look Aside Buffer) می‌باشد.
- ✓ در هنگام تبدیل آدرس اگر شماره صفحه و شماره قاب در رجیستر انجمنی موجود باشد، وضعیت برخورد رخ می‌دهد. در غیر این صورت وضعیت عدم برخورد (Miss) پدید می‌آید که نسبت این در وضعیت به نام نسبت اصابت (Hit Ratio) از رابطه زیر محاسبه می‌شود.

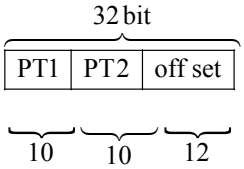
$$\text{Hi Ratio} = \frac{\text{Hit}}{\text{Hit} + \text{Miss}}$$

- ✓ زمان دسترسی موثر به حافظه (Effective Access Time)

$$EAT = t_{ar} + HR * t_{mem} + (1 - HR) * 2 * t_{mem}$$

- ✓ به منظور رفع مشکل داشتن جداول صفحه بزرگ در حافظه بسیاری از سیستم‌ها از جدول صفحه چند سطحی استفاده می‌نمایند.

مثلاً :



- ✓ جدول صفحه معکوس: اگر به ازای هر صفحه مجازی بخواهد یک درایه وجود داشته باش مثلاً اگر فضای آدرس 2^{64} بایتی و صفحات 2K باشند اندازه جدول صفحه بیش از 10^{15} بایت خواهد بود که اصلاً به صرفه نیست. برای حل این مشکل از (Inverted page table) استفاده می‌کنیم. به جای این که به ازای هر صفحه مجازی یک درایه در جدول داشته باشیم. به ازای هر قاب در صفحه اصلی یک درایه در جدول صفحه معکوس وجود دارد.

مدیریت حافظه قطعه بندی ساده (Simple Segmentation)

یک قطعه را می‌توان به عنوان مجموعه منطقی از اطلاعات تعریف کرد، به عنوان مثال زیر روال ، آرایه ناحیه داده‌ای می‌توانند به عنوان قطعه محسوب گردند. بنابراین فضای آدرس هر فرآیند در واقع متشکل از اجتماعی از قطعات می‌باشد. در نتیجه قطعه بندی روشی است که برای مدیریت این قطعات به کار می‌رود.

- ✓ قطعات فرآیند دارای ندازه یکسان نمی‌باشند، اگرچه حداکثری برای طول قطعه در نظر گرفته می‌شود.
- ✓ همانند صفحه بندی آدرس منطقی در قطعه بندی شامل شماره قطعه و آفست است.
- ✓ صفحه بندی از دید برنامه ساز مخفی است ولی قطعه بندی قابل رویت و عامل تسهیل سازماندهی برنامه‌ها و داده‌ها می‌باشد.

ترجمه آدرس منطقی به آدرس فیزیکی در قطعه بندی

اگر آدرس منطقی n+m باشد (m بیت شماره Segment و n بیت برای offset):

۱ـ استخراج شماره قطعه و قطعه از n بیت سمت چپ آدرس منطقی .

۲ـ استفاده از شماره قطعه به عنوان شاخص به جدول قطعه فرآیند برای یافتن آدرس فیزیکی شروع قطعه.

۳ـ مقایسه طول offset باطول قطعه. اگر offset بزرگتر یا مساوی طول قطعه باشد آدرس معتبر نیست.

۴ـ مجموع آدرس فیزیکی شروع قطعه و انحراف آدرس فیزیکی مورد نظر را تولید می‌کند.

اشتراک، یکی از مزایای قطعه بندی می‌باشد.

قطعه بندی با صفحه بندی: قطعات تولید شده توسط مترجم صفحه بندی می‌شوند و به ازای هر فرآیند یک جدول قطعه و به ازای هر قطعه یک جدول صفحه جداگانه در نظر گرفته می‌شود.

مثلاً در یک سیستم با آدرس 34 بیتی که 8 بیت شماره قطعه و 16 بیت offset است، 2^{16} عدد بزرگی است. لذا قطعات را به صفحات 1K تقسیم می‌کنیم:

34 bit Logical Address

شماره قطعه 18 بیت	شماره صفحه 6 بیت	انحراف از صفحه 10 بیت

مدیریت حافظه صفحه بندی بر حسب نیاز (Demand Paging)

- ✓ آدرسی را که توسط فرآیند در حال اجرا مورد ارجاع واقع می‌شود آدرس مجازی نامند. آدرس‌های موجود در حافظه اصلی را آدرس‌های حقیقی گویند.
- ✓ هنگامی که فرآیندی اجرا می‌شود آدرس‌های مجازی باید به آدرس‌های حقیقی تبدیل شوند و این عمل باید به سرعت انجام گیرد. در غیر این صورت کارایی کامپیوتر کاهش می‌یابد.
- ✓ هنگامی که فرآیندی برای اجرا زمانبندی گردد معمولاً فقط اولین صفحه آن به حافظه آورده خواهد شد. سایر صفحات که مورد نیاز فرآیند باشند متعاقباً برحسب تقاضا Load خواهند شد. این امر سبب می‌گردد و تضمین می‌کند صفحه که نیازی به آن نیست در حافظه بارگیری نگردد. لذا به این روش Demand paging می‌گوییم.
- ✓ روش مدیریت حافظه Demand Paging فضاهای آدرس را محدود به حافظه فیزیکی نمی‌کند.

وظایف مدیریت حافظه Demand Paging

۱ـ ثبت وضعیت مکان های حافظه (جداول صفحه، جدول بلوک حافظه، جدول نگاشت فایل)

۲ـ سیاست و خط مشی (برای در اختیار قرار دادن حافظه)

۳ـ تخصیص

۴ـ بازپس گیری.

فیلدهای (Page Map Table) PMT

۱ـ بیت مقیم (بیت وقفه صفحه): نشان می‌دهد که صفحه داخل حافظه قطعه اصلی است.

۲ـ بیت رجوع: نشان می‌دهد که به صفحه رجوع شده است.

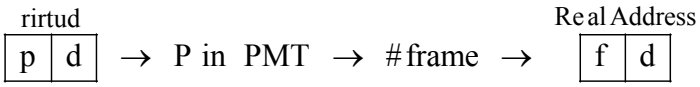
۳ـ بیت تغییر: هر زمانی که مکانی در یک قاب صفحه تغییر نماید 1 می‌شود.

۴ـ بیت‌های حافظت: نشان می‌دهد که چه عملی روی این صفحه مجاز است (RD / WR / EXEC).

۵ـ بیت‌های مربوط به قاب صفحه: به قاب داخل حافظه حقیقی اشاره دارد.

روتین‌های مدیریت حافظه به وسیلهٔ وقفه نقص صفحه (Page Fault) فراخوانی می‌شوند، بنابراین در مدیریت حافظه صفحه بندی شده بر حسب نیاز ارتباط و اثر متقابل بسیار نزدیکی بین سخت افزار و نرم افزار به چشم می‌خورد.

۱ـ تبدیل آدرس صفحه بندی با استفاده از نگاشت مستقیم



۲ـ تبدیل آدرس صفحه بندی شده با استفاده از نگاشت رجیسترهای انجمنی (AR)

کل جدول صفحه را در AR ذخیره می‌کنیم اگر $V.A=(P,d)$ باشد هر درایه در ARبه طور همزمان برای صفحه P جستجو می‌شود و به سرعت f باز گردانده می‌شود جمع f با d آدرس حقیقی را تشکیل می‌دهند. مشکل اینجاست که ARها بسیار گران قیمت هستند.

۳ـ تبدیل آدرس صفحه بندی شده با استفاده از نگاشت ترکیبی رجیسترهای انجمنی / مستقیم

رجیسترهای انجمنی قادر هستند فقط درصد کمی از جدول صفحه یک فرآیند را در خود نگه دارند. درایه‌های صفحه که در این جدول قرار می‌گیرند با صفحاتی که از قبل دفعات زیادی مورد ارجاع واقع می‌شدند متناظر می‌باشد. برنامه در حال اجرا آدرس مجازی $V=(P.d)$ را مورد ارجاع قرار می‌دهد. اگر صفحه P در AR بود که f پیدا می‌شود و $f+d=r$. در غیر این صورت P در جدول صفحه مستقیم جستجو می‌شود و f به دست می‌آید.

الگوریتم‌های جایگزینی صفحه

هنگامی که یک نقص صفحه اتفاق می‌افتد و همه قاب‌های صفحه پر است، سیستم‌عامل باید یک صفحه را انتخاب کرده و از حافظه خارج نماید تا جا برای صفحه جدید باز شود. کدام صفحه؟

الگوریتم «بهینه»

الگوریتم بهینه (**Optimal**) به شرح زیر است: «صفحه‌ای را خارج کن که در آینده دورتری به آن مراجعه خواهد شد».

الگوریتم «خروج به ترتیب ورود» (FIFO)

الگوریتم بعدی **FIFO**[9] نام دارد: «قدیمی‌ترین صفحه درون حافظه را خارج کن» ؛ یعنی صفحه‌ای که مدت بیشتری مقیم یک قاب در حافظه اصلی بوده است.

الگوریتم «حداقل استفاده در گذشته اخیر» (LRU)

یکی از پرطرفدارترین الگوریتم‌های جایگزینی صفحه، الگوریتم **LRU**[10] است: «صفحه‌ای را خارج کن که در گذشته دورتری به آن مراجعه شده است»؛ یعنی از آخرین مراجعه به آن صفحه مدت بیشتری گذشته است.

الگوریتم «عدم استفاده در گذشته اخیر» (NRU)

الگوریتم **NRU**[11] می‌گوید: «صفحه‌ای را خارج کن که در گذشته اخیر (از ابتدای دوره جاری تا کنون) استفاده نشده است».

نکته: مفهوم بیت R این است که از آخرین پالس ساعت تا کنون (در دوره جاری) به صفحه مراجعه شده است یا خیر و مفهوم بیت M این است که از ابتدای ورود صفحه به حافظه تا کنون، این صفحه تغییر کرده است یا خیر.

در الگوریتم NRU، موقعی که یک نقص صفحه به وجود می‌آید، سیستم‌عامل به طور تصادفی صفحه‌ای را انتخاب می‌کند که بیت R آن 0 باشد. اگر بیت R همه صفحات 1 بود باز هم یک صفحه به طور تصادفی انتخاب می‌شود.

الگوریتم NRU پیشرفته

الگوریتم **NRU پیشرفته (Enhanced NRU)** می‌گوید: «صفحه‌ای را خارج کن که در درجه اول، در گذشته اخیر (از ابتدای دوره جاری تا کنون) استفاده نشده است و در درجه دوم، از هنگام ورود به حافظه تا کنون تغییر نکرده است».

در این الگوریتم، موقعی که یک نقص صفحه به وجود می‌آید، سیستم‌عامل بر اساس وضعیت بیت‌های M و R صفحات را به چهار کلاس تقسیم می‌کند و به طور تصادفی یک صفحه را از کلاسی انتخاب می‌کند که شماره کمتری دارد و خالی نیست.

R	M	کلاس
0	0	0
0	1	1
1	0	2
1	1	3

الگوریتم «شانس دوباره»

الگوریتم بعدی، **شانس دوباره** یا **دومین شانس (Second chance)** نامیده می‌شود: «قدیمی‌ترین صفحه‌ای را خارج کن که اخیراً مورد استفاده قرار نگرفته است».

این الگوریتم مانند FIFO یک لیست پیوندی می‌سازد. در هنگام نقص صفحه، ابتدا بیت R قدیمی‌ترین صفحه (در ابتدای لیست) بررسی می‌شود. در صورتی که این بیت 0 باشد، صفحه مورد نظر هم قدیمی و هم بدون استفاده است و بنابراین بلافاصله با صفحه جدید جایگزین می‌شود. اما اگر بیت R مربوطه 1 باشد، آن را 0 کرده و صفحه را به انتهای لیست منتقل می‌کنیم (به آن صفحه دوباره شانس می‌دهیم، انگار که تازه وارد حافظه شده است)، سپس جستجو ادامه می‌یابد.

---

[9]- First-In, First-Out (FIFO)
[10]- Least Recently Used (LRU)
[11]- Not Recently Used (NFU)

الگوریتم «ساعت»

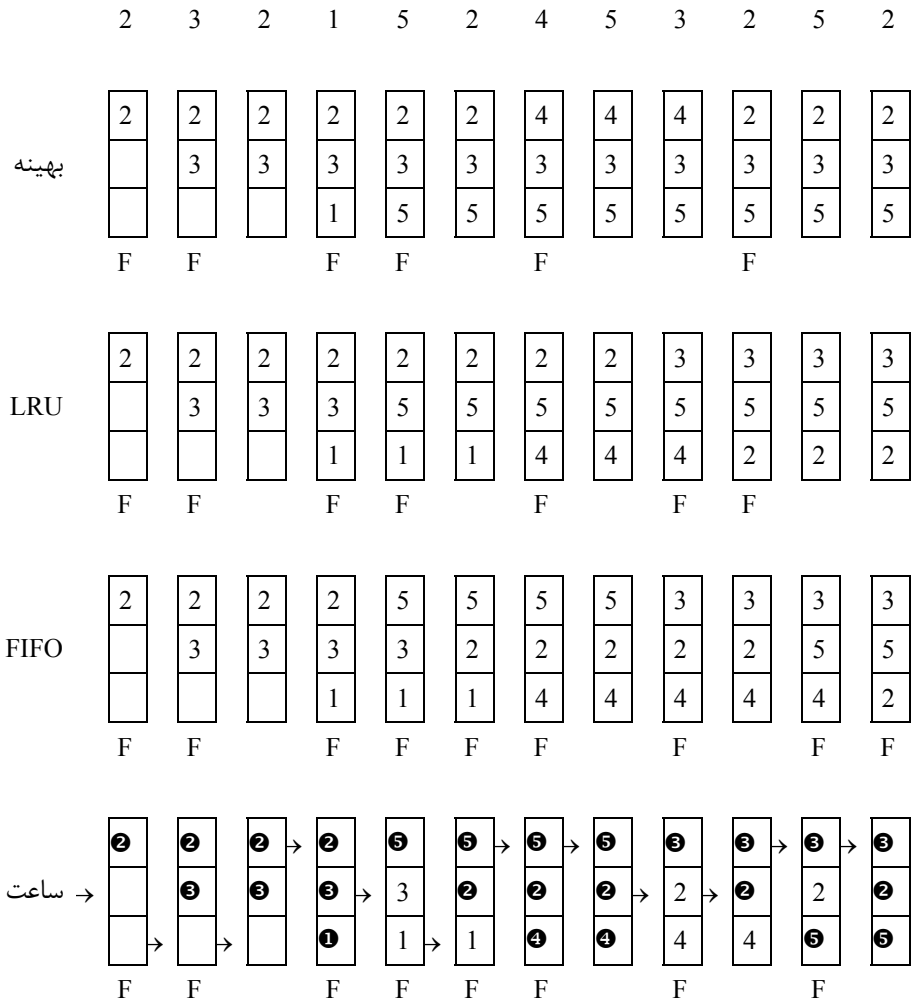
در الگوریتم شانس دوباره، حذف مکرر صفحات از ابتدای لیست و اضافه‌کردن آنها به انتهای لیستپرهزینه است و الگوریتم را بیهوده ناکارآمد می‌سازد. خوشبختانه این مشکل به راحتی در الگوریتم **ساعت** (**Clock**) حل شده است: «عیناً همان رفتار و نتیجه الگوریتم شانس دوباره را دارد، فقط در پیاده‌سازی از لیست چرخشی استفاده می‌کند و سریع‌تر به همان جواب می‌رسد»

مثال: مقایسه رفتار الگوریتم‌های بهینه، FIFO، LRU و ساعت

فرض کنید حافظه اصلی فقط گنجایش 3 صفحه را دارد (یعنی 3 قاب آزاد داریم) و دنباله مراجعات به صفحات حافظه به ترتیب از چپ به راست، به صورت زیر باشد:

2 3 2 1 5 2 4 5 3 2 5 2

با اجرای الگوریتم‌های بهینه، FIFO، LRU و ساعت، تعداد نقص صفحه و جایگزینی‌ها با توجه به شکل زیر به‌دست می‌آید.



شبیه‌سازی LRU در نرم‌افزار: الگوریتم سالمندی(Aging)

در این الگوریتم از فیلدی به نام سن (Age) در جدول صفحه استفاده می‌شود که مثلاً می‌تواند 8 بیتی باشد. مقدار این فیلد در هنگام بارگذاری صفحه (ابتدا) 0 است. در هر وقفه ساعت، سیستم‌عامل برای تمام صفحات موجود در حافظه، فیلد سن آنها را یک بیت به سمت راست شیفت داده و پس از آن، بیت R را در بیت انتهایی سمت چپ (با ارزش‌ترین بیت یا MSB) این فیلد قرار می‌دهد. سپس فیلد R همه صفحات را 0 می‌کند. در صورت وقوع نقص صفحه، صفحه‌ای که شمارنده آن دارای کمترین مقدار است، جهت جایگزینی انتخاب می‌شود.

الگوریتم «عدم استفاده مکرر» (NFU) یا «حداقل استفاده مکرر» (LFU)

الگوریتم بعدی یک الگوریتم شمارشی است که «**عدم استفاده مکرر**» (**NFU**[12]) نام دارد: «صفحه‌ای را خارج کن که تعداد دوره مراجعه به آن در گذشته، حداقل بوده است».

در این الگوریتم به هر یک از صفحات یک شمارنده نرم‌افزاری (مثلاً 8 بیتی) اختصاص می‌دهیم. به صورتی که مقدار همه این شمارنده‌ها در ابتدای کار 0 باشد. در هر وقفه ساعت، سیستم‌عامل برای تمام صفحات موجود در حافظه، مقدار بیت R آن صفحه را (چه 0 باشد و چه 1) با مقدار شمارنده مربوطه جمع می‌کند. سپس فیلد R همه صفحات را 0 می‌کند. بنابراین هر شمارنده نشان می‌دهد که یک صفحه در چند دوره زمانی (زمان بین دو پالس ساعت متوالی) مورد دسترسی قرار گرفته است (البته نمی‌دانیم در هر دوره چند بار) و در صورت وقوع نقص صفحه، صفحه‌ای که شمارنده آن دارای کمترین مقدار است، جهت جایگزینی انتخاب می‌شود. نام دیگر این الگوریتم «**حداقل استفاده مکرر**» (**LFU**[13]) است.

۹-۶-۱-۸ الگوریتم MFU

مشکل الگوریتم NFU (یا LFU) این است که گذشته دور در آن فراموش نمی‌شود. به همین دلیل، الگوریتم «**حداکثر استفاده مکرر**» (**MFU**[14]) را پیشنهاد کرده‌اند که نقطه مقابل LFU است و در صورت وقوع نقص صفحه، صفحه‌ای که شمارنده آن دارای بیشترین مقدار است، جهت جایگزینی انتخاب می‌شود؛ با این استدلال که صفحاتی که دارای مقدار شمارنده کوچک هستند، احتمالاً تازه وارد حافظه شده‌اند. البته این دو الگوریتم نیز افراط و تفریط را نشان می‌دهند و هیچ‌کدام کارایی مطلوبی ندارند.

الگوریتم «بافرکردن صفحه»

در الگوریتم **بافرکردن صفحه** (**Page buffering**) یک وضعیت جدید برای صفحات در نظر گرفته می‌شود: درون بافر. به عبارت دیگر، تعدادی قاب صفحه به عنوان بافر رزرو می‌شود و صفحات درون آن، از نظر MMU و جدول صفحه، غایب در نظر گرفته می‌شوند و با مراجعه به آنها نقص صفحه رخ می‌دهد. در این روش سه لیست صفحه داریم که به ترتیب ورود مرتب شده‌اند:

۱- لیست صفحات درون حافظه
۲- لیست صفحات بدون تغییر درون بافر
۳- لیست صفحات تغییریافته درون بافر

هنگامی که یک صفحه با یک الگوریتم ساده و سریع، مانند FIFO، به عنوان قدیمی‌ترین صفحه از ابتدای لیست شماره 1 برای جایگزینی انتخاب می‌شود، واقعاً از حافظه خارج نمی‌شود؛ بلکه از لیست 1 خارج و در جدول صفحه (ظاهراً) به عنوان غایب علامت‌گذاری شده و به انتهای لیست 2 یا 3 اضافه می‌شود. این صفحه به طور موقت در همان قاب صفحه می‌ماند و به جای آن، یک صفحه قدیمی از ابتدای لیست 2 یا 3 خارج شده و واقعاً از حافظه محو می‌گردد.

الگوریتم «ساعت مجموعه‌کاری» (WS-Clock)

الگوریتم بعدی، **ساعت مجموعه‌کاری** (**WS-Clock**) نام دارد. «قدیمی‌ترین صفحه‌ای را خارج کن که متعلق به مجموعه کاری فرایند نباشد».

می‌دانیم که فرایندها بر اساس اصل **مراجعات محلی** کار می‌کنند. به این معنی که در طی هر فاز از اجرای فرایند، فقط به کسر نسبتاً کوچکی از صفحات آن مراجعه می‌شود. مجموعه صفحاتی که فرایند در حال حاضر از آنها استفاده می‌کند، **مجموعه کاری** (**Working set**) نام دارد.

اطلاع از مجموعه کاری می‌تواند در بهبود الگوریتم ساعت موثر باشد. در الگوریتم ساعت وقتی عقربه به صفحه‌ای اشاره می‌کند که بیت R آن 0 است، آن صفحه بیرون برده می‌شود. برای بهترکردن کار می‌توان بررسی کرد که آیا آن صفحه در مجموعه کاری فرایند قرار دارد یا خیر. اگر عضوی از مجموعه کاری باشد، جهت جایگزینی انتخاب نخواهد شد و عقربه رو به جلو حرکت خواهد کرد.

---

[12]- Not Frequently Used (NFU)
[13]- Least Frequently Used (LFU)
[14]- Most Frequently Used (MFU)

برای پیاده‌سازی این الگوریتم، سیستم‌عامل نیاز به یک روش سیستماتیک برای شناخت صفحات تشکیل دهنده مجموعه کاری هر فرایند دارد. به این منظور می‌توان از فیلد سن (Age) جدول صفحه استفاده نمود که در الگوریتم سالمندی (شبیه‌سازی نرم‌افزاری LRU) نیز به‌کار رفت. هر صفحه‌ای که بین n بیت مرتبه بالای (سمت چپ) فیلد سن آن، حداقل یک بیت 1 وجود داشته باشد، عضوی از مجموعه کاری محسوب می‌شود.

جایگزینی سراسری: فرآیند می‌تواند یک بلوک را از بین کلیه بلوک‌ها حتی آن‌هایی که در حال حاضر به فرآیند دیگری اختصاص داده شده‌اند برای جایگزین شدن انتخاب کند.

جایگزین محلی: هر فرآیند بلوک جایگزین شونده را تنها از مجموعه بلوک‌های اختصاص داده شده به خودش می‌تواند انتخاب نماید.

اندازه بهینه صفحه : فرض کنید میانگین اندازه یک فرآیند P_1 بایت و میانگین اندازه صفحه P_2 بایت باشد و هر درایه از جدول صفحه را بایت نیاز داشته باشد، آنگاه:

$$P_2 = \sqrt{2 * P_1 * e}$$

هرگاه تعداد بلوک‌های اختصا داده شده به هر فرآیند کاهش یابد سرعت و تعداد شکست صفحه افزایش خواهد یافت که نتیجه آن کاهش سرعت اجرایی فرآیند است.

حداقل تعداد بلوک‌های حافظه باید به وسیله معماری سیستم مشخص گردد.

الگوریتم‌های تخصیص بلوک‌ها به فرآیندها

۱ـ اختصاص حافظه به طور مساوی: اگر m بلوک و n فرآیند داشته باشیم به هر کدام $\frac{m}{n}$ بلوک‌ها را اختصاص دهیم.

۲ـ اختصاص حافظه متناسب: به هر فرآیند مطابق اندازه نهایی آن فرآیند حافظه اختصاص داده شود.

در هر یک از روش‌های تخصیص حافظه به طور مساوی یا متناسب فرآیندی که از اولویت بالا برخوردار است و فرآیندی که اولویت پائین یا کمتری دارد یکسان تلقی خواهند شد.

:Thrashing

اگر تعداد بلوک‌های حافظه اختصاص داده شده به یک فرآیند از تعداد حداقلی که معماری کامپیوتر نیاز دارد کمتر گردد، اجرای فرآیند باید به تعویق بیفتد. بدین ترتیب باید باقیمانده صفحات فرآیند را نیز از حافظه خارج نمود و کلیه بلوک‌های حافظه متعلق به آن فرآیند را آزاد کرد.

- ✓ فرآیندی در حالت Thrashing واقع است که به جای آن که زمان پردازنده را برای اجرا اختصاص دهد زمان زیادی را صرف انجام عملیات صفحه بندی می‌نماید.
- ✓ برای جلوگیری از Thrashing باید برای فرآیندها به تعدادمورد نیاز بلوک‌های حافظه‌ای مهیا نمائیم.
- ✓ Locality : هم مکانی مجموعه‌ای از صفحات است که به طور فعال با یکدیگر مورد استفاده قرار می‌گیرند. به طور کلی یک برنامه ترکیبی است از چندین هم مکانی مختلف که ممکن است هم پوشانی داشته باشند.
- ✓ یک استراتژی که نشان می‌دهد یک فرآیند چه تعداد بلوک حافظه‌ای نیاز دارد Working Set است.
- ✓ Working Set : مجموعه کاری ترکیبی است از صفحات یک فرآیند که به طور فعال مورد ارجاع قرار می‌گیرند. برای اجرای موثر یک برنامه مجموعه کاری متناظر با آن باید در حافظه اصلی نگهداشته شود. در غیر این صورت ممکن است پدیده Thrashing رخ دهد.

فرکانس نقص صفحه (Page Foult) : روش فرکانس نقص صفحه (PFF) راهی است برای جلوگیری از بروز پدیده – Thrashing Thrashing دارای سرعت نقص صفحه بالایی است،بنابراین برای جلوگیری از آن باید سرعت شکسته صفحه یا نقص صفحه را کنترل نمود.

در صورتی که سرعت نقص صفحه افزایش یابد اما هیچ بلوک حافظه آزادی وجود نداشته باشد، باید فرآیندی انتخاب شود و موقتاً معوق گردد.

مدیریت حافظه قطعه بندی شده: یک قطعه را می‌توان به عنوان مجموعه منطقی از اطلاعات تعریف کرد، مثال زیر روال، آرایه و ناحیه داده‌ای.

تفاوت اصلی و بزرگ دو روش صفحه بندی و قطعه بندی در این است که قطعه یک واحد منطقی از اطلاعات ـ مشهود برنامه کاربر ـ می‌باشد و دارای اندازهٔ اختیار است. در صورتی که صفحه یک واحد فیزیکی از اطلاعات است که صریحاً برای مدیریت حافظه استفاده می‌شود و از نظر برنامه کاربر نامشهود است و دارای اندازهٔ ثابتی است.

مزایای مدیریت حافظه قطعه بندی شده

۱ـ حذف مشکل Memory Fragmentation

۲ـ ایجاد حافظه مجازی

۳ـ امکان رشد قطعات به طور پویا یا بررسی خودکار مرزهای قطعه

۵ـ بار کردن (Load) و پیوند (Link) پویا

۵ـ تسهیل استفاده از قطعات مشترک

۶ـ دستیابی کنترل شده

قالب آدرس در Segmentation

#Segment	#offset

درایه جدول نگاشت قطعه (SMT) :

r	a	L	R	W	E	A	Sl

بیت‌های حفاظتی طول قطعه

S1 : آدرس مبنای قطعه (در صورتی که قطعه در حافظه باشد)

R: دستیابی از طریق خواندن

W : دستیابی از طریق نوشتن

E : دستیابی از طریق اجرا کردن

A : دستیابی از طریق الحاق کردن

a : آدرس در حافظه مجازی (در صورتی که قطعه در حافظه نباشد)

r : بیت حضور قطعه در حافظه اصلی (ایمنی هست).

اگر میزان offset یک آدرس از L (طول قطعه)بیشتر باشد نقص Segment over flow رخ می‌دهد.

قطعه‌های مشترک: ممکن است رویه‌های مختلفی را به اطلاعات یک قطعه احتیاج داشته باشند. در این صورت مدیریت حافظه قطعه بندی با به اشتراک گذاردن این قطعه بین رویه‌ها از افزونگی جلوگیری می‌کند.

ترکیب قطعه‌بندی و صفحه‌بندی: یک روش برای به دست آوردن فواید منطقی قطعه بندی و حذف بسیاری از معایب آن ترکیب کردن قطعه بندی و صفحه بندی با یکدیگر می‌باشد.

#Segment	#Page	#Offset

قالب آدرس $\rightarrow$

$$E.A.T = H_{tlb}\left(tlb + t_m\right) + \left(1 - H_{tlb}\right)\left(PF\right)\left(tlb + t_{sv} + t_m\right)$$
$$+ \left(1 - H_{tlb}\right)\left(1 - PF\right)\left(tlb + 2t_m\right)$$

working set در زمان اجرای پردازش و توسط سیستم عامل تعیین می‌شود.

$$EAT = tlb + t_m + Hiss\, tlb\left(t_m + PF.t_{srv}\right)$$

مدیریت فایل :

فایل‌ها توسط سیستم عامل بر روی دستگاه‌های فیزیکی نگاشته می‌شوند.

سیستم عامل در رابطه با مدیریت فایل این وظایف را دارد:

۱ـ ایجاد و حذف فایل‌ها و فهرست‌ها

۲ـ مهیا کردن ابزار کار با فایل‌ها

۳ـ نگاشت فایل روی حافظه جانبی

۴ـ ایجاد پشتیبانی از فایل‌ها

۵ـ مکانیزم حفاظتی برای کنترل دستیابی کاربران مختلف به فایل‌ها.

*سیستم فایل متشکیل از دو قسمت مختلف است:

۱ـ مجموعه‌ای از فایل‌های حقیقی که هر یک متشکل از اطلاعات مرتبط هستند.

۲ـ ساختار فهرست‌ها که اطلاعاتی درباره تمام فایل‌ها در سیستم را در بر می‌گیرند.

در هر درایه فهرست فایل می‌توان اطلاعات زیر را ثبت کرد

۱ـ اسم فایل

۲ـ نوع فایل

۳ـ مکان فایل

۴ـ اندازه فایل

۵ـ موقعیت فعلی

۶ـ حفاظت

۷ـ میزان استفاده

۸ـ زمان تاریخ و معرف فرآیند

عملیات فایلی که سیستم عامل فراهم می‌کند

۱ـ ایجاد

۲ـ نوشتن

۳ـ خواندن

۴ـ بازگرداندن

۵ـ حذف

شیوه‌های دستیابی به اطلاعات فایل

۱ـ دسترسی ترتیبی

۲ـ دسترسی مستقیم

۳ـ دسترسی از طریق شاخص (Index Sequential)

سازماندهی ساختار فهرست

۱ـ فهرست تک سطحی: کلیه فایل‌ها در یک فهرست یکسان قرار می‌گیرند.

۲ـ فهرست دو سطحی :هر کاربر دارای فهرست فایل خالص خودش می‌باشد.

۳ـ فهرست‌های باساختار Hierarchical : امکان می‌دهد کاربران فهرست‌های خاص خود را ایجاد نمایند.

انواع مختلفی از عملیات وجود دارند که می‌توانند به صورت کنترل شده اجرا کردند

۱ـ خواندن

۲ـ نوشتن

۳ـ ااجرا

۴ـ الحاق اطلاعات جدید

۵ـ حذف فایل

شیوه‌های حفاظت فایل

۱ـ نامگذاری

۲ـ کلمات عبور

۳ـ لیست‌های دستیابی

۴ـ گروه‌های دستیابی

گروه‌های کاربری مرتبط با فایل عبارتند از

۱ـ Owner

۲ـ Group

۳ـ Other

تست‌ها

۱. یک سیستم چندبرنامه‌ای و دسته‌ای (Batch Multiprogramming) دارای چهار ناحیه (Partition) با اندازه‌های ثابت B_3=50KB ,B_2=30KB ,B_1=30KB ,B_0=20KB, دسته کارهای زیر قرار است در این حافظه زمان‌بندی شوند:

شماره کار	10	9	8	7	6	5	4	3	2	1
حافظه مورد نیاز(KB)	31	21	16	19	49	14	33	10	29	18
زمان ماندگاری در حافظه(ثانیه)	10	15	10	5	10	15	15	20	5	10

هر کار می‌تواند در هر ناحیه‌ای از حافظه با اندازه کافی زمان‌بندی شود؛ فقط به شرط آنکه کارهای بزرگ‌تری که بتوانند در این ناحیه جای گیرند، بلوکه نشوند. زمان پردازش تمام کارها چند ثانیه است؟ (کارشناسی ارشد کامپیوتر ـ سراسری ۸۱)

۱) 115 ۲) 70 ۳) 60 ۴) 35

۲. فرض کنید تحت یک مدیریت حافظه مجازی هر صفحه 512 بایت داشته باشید، برنامه‌نویسی ماتریس 256 × 256 عضوی را از نوع درست (دو بایتی) مطابق کد ذیل، آغازسازی می‌کند:

```
for i=1 to 256 do
    for j=1 to 256 do
        A[i][j]=0;
```

اگر ماتریس به صورت سطر - سطر در حافظه اصلی ذخیره شود و حافظه اصلی فقط دو قاب صفحه خالی برای داده‌ها داشته باشد، به این منظور خطاهای صفحه چقدر است؟ (Page Fault) (کارشناسی ارشد کامپیوتر ـ سراسری ۸۵)

۱) 512 ۲) 256 ۳) 128 ۴) 64

۳. حافظه اصلی کامپیوتری دارای چهار قاب صفحه است. زمان بار شدن (Load)، زمان آخرین دسترسی، بیت R (Reference)، بیت M (Modify) مربوط به هر یک از صفحات در جدول زیر آمده است. اگر خطای صفحه (Page Fault) روی صفحه مجازی شماره 4 در زمان 319 رخ دهد. تحت الگوریتم‌های جایگزینی LRU و NRU به ترتیب محتویات کدام‌یک از قاب صفحه‌ها باید جابه‌جا شوند. (کارشناسی ارشد کامپیوتر ـ سراسری ۸۶)

شماره صفحه مجازی	قاب صفحه	زمان بار شدن	زمان آخرین دسترسی	بیت R	بیت M
2	0	125	278	0	1
1	1	229	239	1	0
0	2	119	271	1	0
3	3	159	318	1	1

۱) یک و دو ۲) یک و صفر ۳) دو و یک ۴) سه و صفر

۴. در یک سیستم حافظه صفحه‌بندی با یک جدول صفحه حاوی 64 مدخل 11 بیتی (شامل یک بیت اعتبار / عدم اعتبار) و صفحه‌های با اندازه هر یک 512 بایت، یک آدرس منطقی و یک آدرس فیزیکی چند بیت است؟ (کارشناسی ارشد IT ـ سراسری ۸۳)

۱) آدرس منطقی 19 بیت – آدرس فیزیکی 15 بیت
۲) آدرس منطقی 15 بیت - آدرس فیزیکی 19 بیت
۳) آدرس منطقی 6 بیت - آدرس فیزیکی 15 بیت
۴) آدرس منطقی 10 بیت - آدرس فیزیکی 6 بیت

پاسخنامه

۱. گزینه ۴ درست است.

زمان / حافظه	0-5	5-10	10-15	15-20	20-25	25- 30	30-35
20KB	7(19)	8(16)	8(16)	3(10)	3(10)	3(10)	3(10)
30KB	9(21)	9(21)	9(21)	-	-	-	-
30KB	2(29)	1(18)	1(18)	5(14)	5(14)	5(14)	-
50KB	6(49)	6(49)	4(33)	4(33)	4(33)	10(31)	10(31)

۲. گزینه ۲ درست است.

هر سطر ماتریس 256 عنصر درست 2 بایتی، یعنی 512 بایت است که برابر اندازه صفحه است. چون ماتریس ردیفی ذخیره شده است هر سطر ماتریس در یک صفحه قرار دارد و چون برنامه مذکور، ماتریس را سطر به سطر مقداردهی می‌کند در کل 256 نقص صفحه خواهیم داشت.

۳. گزینه ۲ درست است.

تست جالبی است. روش LRU صفحه‌ای را خارج می‌کند که در گذشته دورتری به آن مراجعه شده است و با توجه به فیلد «زمان آخرین دسترسی» قاب شماره 1 در گذشته دورتری (زمان 239) مورد استفاده قرار گرفته و برای جایگزینی انتخاب می‌شود. در روش NRU به بیت‌های R و M نگاه می‌کنیم و چهار قاب صفحه به ترتیب در کلاس‌های 01، 10، 10 و 11 قرار دارند و چون کلاس 00 خالی است قاب 0 از کلاس 01 برای جایگزینی انتخاب می‌شود.

۴. گزینه ۲ درست است.

اندازه حافظه منطقی = اندازه هر صفحه × تعداد مدخل‌های جدول صفحه $= 2^6 \times 2^9 = 2^{15}$

مدخل‌های جدول صفحه 11 بیتی است که 1 بیت آن برای اعتبار یا عدم اعتبار استفاده می‌شود و بنابراین 10 بیت برای شماره قاب صفحه (PF#) در نظر گرفته شده است (از بقیه فیلدها صرف‌نظر می‌کنیم چون راجع به آن‌ها صحبت نشده است). از طرفی چون اندازه صفحه و قاب صفحه $2^9 = 512$ بایت است، پس افست 9 بیتی است.

طول آدرس فیزیکی = طول PF# + طول افست $= 10 + 9 = 19$

چون جدول صفحه حاوی $2^6 = 64$ مدخل است بنابراین 6 بیت برای شماره صفحه (P#) در نظر گرفته شده است:

طول آدرس منطقی = طول P# + طول افست $= 6 + 9 = 15$

فصل ششم

زمانبندی دیسک

پارامترهای کارآیی دیسک

۱ـ Seek Time: زمان لازم برای قرار گرفتن Head دیسک روی شیار را زمان پیگیرد یا Seek Time می‌گویند.

۲ـ Rotationel Time : پس از انتخاب شیار، صفحه دیسک طوری چرخش می‌کند که ابتدای sector در مقابل هر قرار می‌گیرد. این زمان را Rotational Time گویند.

۳ـ Transmision Time : زمان لازم برای پیموده شدن رکورد توسط هد.

متوسط کل زمان دسترسی

$$T_{access} = T_{seek} + \left(\frac{1}{2r}\right) + \left(\frac{b}{r\,N}\right)$$

T_{access}: متوسط زمان پیگرد — T_{seek} — $\frac{1}{2r}$: سرعت چرخش بر حسب rps — $\frac{b}{r\,N}$: تعداد بایت هایی که باید منتقل شود / تعداد بایت های یک شیار

هر کلاستر از تعدادی سکتور مجاور هم تشکیل شده است.

الگوریتم‌های زمانبدی دیسک عبارتند از: FCFS ـ SSTF ـ SCAN ـ C.SCAN ـ LOOK ـ CLOOK ـ N.STEP SCAN ـ FAST SCAN .

۱ـ FCFS (FIFO) : ساده‌ترین روش است و عادلانه‌ترین روش می‌باشد. اما غالباً سریع‌ترین روش نیست.

۲ـ SSTF (Shortest Seek Time First) : در این روش درخواستی با حداقل زمان جستجو نسبت به موقعیت فعلی هد انتخاب می‌شود. این روش رایج است. ولی مشکل گرسنگی دارد.

۳ـ SCAN (Elevator Algorithm) : در این روش (پویش) هد دیسک مرتباً از یک انتهای دیسک به سمت انتهای دیگر حرکت می‌کند و هر بار که به سیلندری برسد که نیاز به سرویس دهی دارد به آن سرویس می‌دهد. علت نامگذاری آسانسور این است که این روش مثل آسانسور یک ساختمان عمل می‌کند.

۴ـ C.SCAN (Circular Scan یا پویش دورانی) : این روش نسبت به روش آسانسوری زمان انتظار یکنواخت تری را پدید می‌آورد. در روش C-SCAN مانند SCAN هد در یک جهت حرکت کرده و در مسیر خود به تمام درخواست‌ها سرویس می‌دهد. ولی هنگامی که به انتهای دیسک رسید سریعاً به اول دیسک بر می‌گردد و در این حرکت برگشتی سریع هیچ سرویس دهی انجام نمی‌دهد.

۵و ۶ـ Look و Circular Look : این دو روش اصلاح شده روش‌های SCAN و C.SCAN می‌باشند که در آن‌ها الزاماً حرکت از ابتدای دیسک شروع نمی‌شود و تا آخرین سیلندر نیز ادامه نمی‌یابد، بلکه از اولین درخواست شروع شده و به آخرین درخواست ختم می‌گردد.

۷ـ N.STEP.SCAN (پویش N گاهی): سیستم عامل صف درخواست دیسک را به صف‌هایی به طول N می‌شکند. هنگامی که صفی در حال پردازش است درخواست‌های جدید بایدبه انتهای صف دیگر اضافه شوند.

۸ـ F.SCAN (پویش سریع): سیاستی است که دو زیر صف را به کار می‌گیرد. هنگامی که یکی از صف‌ها آغاز می‌شود تمامی درخواست‌ها در صف دیگر قرار می‌گیرند.

نکته: در SSTF جهت اولیه حرکت Head مهم نیست.

SSTF بهینه نیست.

در الگوریتم SCAN اگر درخواستی جلوی هد برسد به آن پاسخ داده می‌شود ولی اگر پشت هد بیفتد باید منتظر بماند تا هد به انتها برود و برگردد.

Over laying : برای غلبه بر محدودیت حافظه فیزیکی به کار می‌رود و Inter leaving برای بهینه سازی حافظه ثانویه مثل دیسک.

تست‌ها

۱. علت Interleave کردن دیسک‌های سخت..... **(کارشناسی ارشد کامپیوتر - آزاد ۷۱)**

۱) به خاطر سرعت پایین دیسک گردان‌ها نسبت به سرعت پردازنده اصلی است.

۲) به خاطر کند بودن Channelهای انتقال اطلاعات به حافظه اصلی است.

۳) به خاطر عدم توانایی کنترلرهای این دیسک‌ها در خواندن اطلاعات و اتصال آن‌ها به حافظه اصلی است.

۴) ممکن است به خاطر سرعت پایین حافظه اصلی در ذخیره‌سازی اطلاعات خوانده‌شده از روی دیسک باشد.

۲. یک دستگاه دیسک‌خوان با استفاده از روش Shortest Seek First (SSF) سیلندرها را جستجو کرده و عمل خواندن را انجام می‌دهد. اگر تقاضاهایی به ترتیب برای سیلندرهای 10، 22، 20، 2، 40، 6، 38 به آن داده شود و Head دستگاه روی سیلندر 20 باشد (در شروع کار) و 6 میلی‌ثانیه طول بکشد تا Head از یک سیلندر به سیلندر بعدی برود، کل زمان جست‌جو برای این سیلندرها چقدر است؟ (کارشناسی ارشد کامپیوتر - سراسری ۷۹)

۱) 360 میلی‌ثانیه ۲) 876 میلی‌ثانیه ۳) 892 میلی‌ثانیه ۴) 3480 میلی‌ثانیه

پاسخنامه

۱. گزینه ۲ درست است.

۲. گزینه ۱ درست است.

$$20 \xrightarrow{2} 22 \xrightarrow{12} 10 \xrightarrow{4} 6 \xrightarrow{4} 2 \xrightarrow{36} 38 \xrightarrow{2} 40$$

$$T_{seek\_total} = 6 \times (2 + 12 + 4 + 4 + 36 + 2) = 6 \times 60 = 360 \text{m sec}$$

آزمون‌های کارشناسی ارشد کامپیوتر و *IT* ـ سراسری ۸۹

۱. با توجه به بحث Copy-On-Write بین فرآیندهای پدر (Parent) و فرزند (Child) در فراخوان سیستمی fork، در راستای افزایش کارایی، کدام جمله در مورد تقدم و تأخر اجرای این فرآیندها در لحظه ایجاد فرایند فرزند درست است؟

۱) با توجه به آگاهی زمان‌بند از محتوای (برنامه‌ی) فرآیند پدر، بهتر است فرآیند فرزند زودتر اجرا شود.
۲) با توجه به آگاهی زمان‌بند از محتوای (برنامه‌ی) فرآیند پدر، بهتر است فرآیند فرزند پدر زودتر اجرا شود.
۳) با توجه به عدم آگاهی زمان‌بند از محتوای (برنامه‌ی) فرآیند فرزند، بهتر است فرآیند پدر زودتر اجرا شود.
۴) با توجه به عدم آگاهی زمان‌بند از محتوای (برنامه‌ی) فرآیند فرزند، بهتر است فرآیند فرزند زودتر اجرا شود.

۲. فرض کنید سیستمی با 3 فرایند با مشخصات زیر داشته باشیم:

- e: execution time **(زمان اجرا)**
- r: release time **(زمان رسیدن)**
- R: number of required resources **(تعداد منابع مورد نیاز)**

$P_1 : r = 0 \quad , \quad e = 4 \quad , \quad R = 2$
$P_2 : r = 1 \quad , \quad e = 2.5 \quad , \quad R = 3$
$P_3 : r = 3 \quad , \quad e = 4 \quad , \quad R = 4$

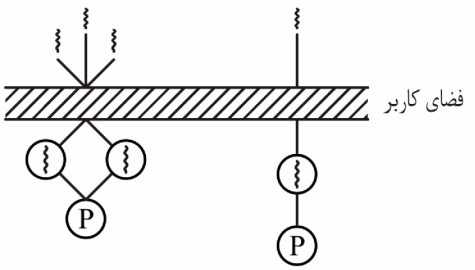
با فرض وجود 4 منبع همسان در سیستم و این فرض که هر فرآیند در لحظات ε ، $\varepsilon+1$ و .... پس از اجرا (ε یک عدد بسیار کوچک است) منابع انحصاری (Non Preemotive) خود را یکی‌یکی درخواست می‌کند و زمان‌بندی براساس الگوریتم Preemptive LCFS (LCFS غیرانحصاری یا قبضه‌ای) انجام می‌شود. کدام عبارت درست است؟

(LCFS: Last-Come First Served)

۱) سیستم حدود لحظه 4 دچار بن‌بست (deadlock) می‌شود.
۲) سیستم دچار بن‌بست (deadlock) نمی‌شود و متوسط زمان کامل (Turnaround Time) برای آن 7 است.
۳) سیستم حدود لحظه 5 دچار بن‌بست (deadlock) می‌شود.
۴) سیستم دچار بن‌بست (deadlock) نمی‌شود و متوسط زمان کامل (Turnaround Time) برای آن 6.17 است.

۳. در یک سیستم کامپیوتری نحوه استفاده از نخ (Thread) در لایه کاربر و در لایه کرنل به صورت مقابل نشان داده شده است. کدام عبارت درست است؟

فرآیند Ⓟ
نخ در لایه کاربر
نخ در لایه کرنل Ⓣ



۱) فراخوانی‌های سیستمی از نوع مسدود (Blocking) با تأمین همروندی حمایت می‌شوند و برای فراخوانی‌های سیستمی از نوع غیرمسدود (Non- Blocking) درجه همروندی پایین‌تر است.
۲) فراخوانی‌های سیستمی از نوع مسدود (Blocking) بدون تأمین همروندی اجرا می‌شوند و فراخوانی‌های سیستمی از نوع غیرمسدود (Non-Blocking) همروندی را تأمین می‌کنند.
۳) فراخوانی‌های سیستمی از نوع مسدود (Blocking)، با تأمین همروندی حمایت می‌شوند و برای فراخوانی‌های سیستمی غیرمسدود (Non-Blocking) درجه همروندی بالاتر است.
۴) فراخوانی‌های سیستمی از نوع مسدود (Blocking) بدون تأمین همروندی اجرا می‌شوند و فراخوانی‌های سیستمی از نوع غیرمسدود (Non-Blocking) نیز با مشکل همروندی مواجه هستند.

۴. آدرس منطقی 0001010010111010 را در نظر بگیرید. با مدیریت صفحه‌بندی 256 صفحه‌ای برای یک حافظه با 256 قاب (frame) و استفاده از جدول صفحه‌ای که در آن هر شماره قاب $\frac{1}{4}$ شماره صفحه باشد، کدام گزینه در مورد مدیریت این حافظه و آدرس فیزیکی متناظر با آدرس منطقی فوق درست است؟

۱) اگرچه این روش نگاشت صفحه مشکل دارد ولی آدرس فیزیکی متناظر 0000101010111010 است.

۲) اگرچه این روش نگاشت صفحه مشکل دارد ولی آدرس فیزیکی متناظر 0000010110111010 است.

۳) این روش نگاشت صفحه بدون مشکل کار می‌کند و آدرس فیزیکی متناظر 0000010110111010 است.

۴) این روش نگاشت صفحه بدون مشکل کار می‌کند و آدرس فیزیکی متناظر 0000101010111010 است.

پاسخنامه

۱. گزینه ۴ درست است.

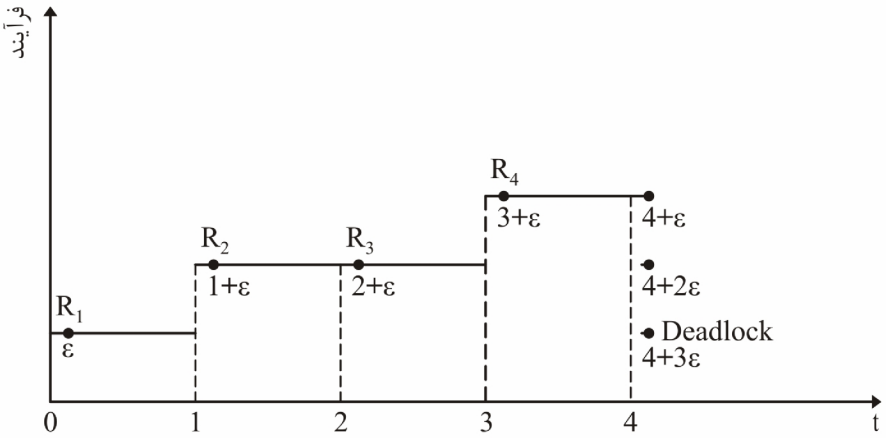
می‌دانیم که اصولاً سیستم‌عامل و بخش‌های مختلف آن مانند زمان‌بندها از محتوا و هدف برنامه‌های کاربر مطلع نیستند و بنابراین گزینه‌های ۱ و ۲ نادرست هستند.

به‌طور کلی تکنیک Copy-on-write به این معناست که اگر بخشی از فضای آدرس حافظه بین چند فرایند به اشتراک (Share) گذاشته شود، چنانچه فرایندی بخواهد تغییری در این فضا ایجاد کند، باید ابتدا کپی جداگانه‌ای برای آن ایجاد شود تا تغییرات صورت گرفته توسط این فرایند بر روی سایر فرایندها تأثیر نگذارد.

با توجه به اینکه در فراخوان سیستمی fork در Unix، فرزند ایجاد شده در ابتدا دقیقاً مانند پدر است، برای افزایش کارایی می‌توان از تکنیک حافظه اشتراکی استفاده کرد و مثلاً کُد پدر و فرزند را به اشتراک گذاشت تا زمانی‌که فرزند بخواهد execve نماید و تصویر حافظه خود را با یک فرایند جدید جایگزین نماید. اما با توجه به اینکه فرزند به فرمان پدر و در راستای خدمت به وی ایجاد شده و خیلی از اوقات پدر باید با فراخوانی waitpid منتظر شود تا فرزندش وظیفه محوله را به اتمام رساند بهتر است ابتدا فرزند اجرا شود.

۲. گزینه ۱ درست است.

نمودار زمانی اجرای فرایندها طبق الگوریتم Preemptive-LCFS و لحظات درخواست و تخصیص منابع به هر فرایند در شکل زیر دیده می‌شود.



همانطور که می‌بینید، ابتدا فرایند P_1 اجرا شده و در زمان ε منبع اول خود را دریافت می‌کند، اما در زمان 1، قبل از گرفتن منبع دوم، CPU از این فرایند گرفته می‌شود و فرایند P_2 اجرا می‌شود (توجه شود که الگوریتم LCFS غیرانحصاری است). P_2 نیز پس از گذشت مدت ε از شروعش (یعنی در زمان $1+\varepsilon$) منبع اول خود را می‌گیرد و پس از گذشت مدت $1+\varepsilon$ از شروعش (یعنی در زمان $2+\varepsilon$) منبع دومش را می‌گیرد. اما در زمان 3 (به علت ورود فرایند P_3) CPU از این فرایند نیز گرفته شده و فرایند P_3 آغاز به کار می‌کند. این فرایند هم پس از مدت ε از شروعش (یعنی در زمان $3+\varepsilon$) منبع اول خود را می‌گیرد که در واقع چهارمین و آخرین منبع آزادِ موجود است. این فرایند نیز پس از گذشت مدت $1+\varepsilon$ از شروعش (یعنی در زمان $4+\varepsilon$) منبع دومش را درخواست می‌کند که موفق به اخذ آن نمی‌شود، چرا که دیگر منبع آزادی باقی نمانده است. در این لحظه همه منابع موجود در سیستم، اشغال شده‌اند (اگرچه در صورت سؤال به این نکته اشاره شده است که منابع انحصاری‌اند و سیستم عامل نمی‌تواند آنها را قبل آزادسازی پس بگیرد ولی این نکته به معنی آن نیست که فرایندها نیز منابع خود را تا لحظه خاتمه اجرا نگه می‌دارند و بهتر بود این نکته نیز به صورت سؤال اضافه می‌شد که البته با توجه به مجهول بودن لحظه آزادسازی منابع، قابل حدس است). دقت کنید هنوز بن‌بست نیست و فقط P_3 خوابیده است. حال دوباره به P_2

سوئیچ می‌شود و این فرایند پس از گذشت مدت ε (که $\varepsilon+2$ از شروع اولیه‌اش گذشته است یعنی در زمان $\varepsilon 2+4$) منبع سومش را درخواست می‌کند که موفق به اخذ آن نمی‌شود و این فرایند نیز مسدود شده و بلافاصله CPU به P_1 سوئیچ می‌شود و این فرایند هم پس از گذشت مدت ε (که $\varepsilon+1$ از شروع اولیه‌اش گذشته است یعنی در زمان $\varepsilon 3+4$) منبع دومش را درخواست می‌کند که این فرایند نیز موفق به اخذ آن نمی‌شود و مسدود می‌شود. در نتیجه در لحظه $\varepsilon 3+4$ (یعنی حدوداً ثانیه چهارم) هر سه فرایند خوابیده و بن‌بست شده است.

۳. گزینه ۳ درست است.

چون در این سیستم هم از نخ‌های سطح کاربر و هم از نخ‌های سطح هسته پشتیبانی می‌شود، رویکرد ترکیبی استفاده شده است و در این رویکرد، به این دلیل که هسته نخ‌های داخل یک فرایند را می‌شناسد می‌تواند این نخ‌ها را به موازات هم بر روی چندین پردازنده اجرا کند و نیز با وقوع یک فراخوان سیستمی مسدود کننده (Blocking) از طرف یک نخ، هسته اشتباه نکرده و کل فرایند را مسدود نخواهد کرد و در نتیجه هم‌روندی نخ‌های داخل فرایند تأمین می‌شود و گزینه‌های ۱ و ۴ نادرست‌اند. البته از جمله انتهایی گزینه ۴ قبلاً به وضوح مشخص بود که این گزینه نادرست است.

از طرف دیگر می‌دانیم که برای فراخوان‌های سیستمی غیر مسدودکننده (Non-Blocking) درجه همروندی بالاتر است و گزینه ۱ نیز نادرست است، زیرا حتی نخی که با یک فراخوان سیستمی درخواستی می‌کند (مانند یک درخواست I/O از یک دستگاه کُند) که باید منتظر نتیجه آن بماند، به جای اینکه بخوابد می‌تواند به اجرا ادامه دهد و به سایر امورش که البته به پاسخ دستگاه I/O مربوط نیست بپردازد و به علت افزایش درجه همروندی زودتر خاتمه یابد.

۴. گزینه ۲ درست است.

چون در این سیستم 256 صفحه وجود دارد (و $2^8 = 256$ است) 8 بیت با ارزش آدرس منطقی داده شده بیانگر شماره صفحه است (ربطی به تعداد قاب ندارد) یعنی:

$$P\# = 00010100$$

$$Offset = 10111010$$

در ترجمه آدرس سیستم‌های صفحه‌بندی شده، افست ثابت مانده و شماره صفحه با شماره قاب صفحه جایگزین می‌شود که آن هم چون در این سیستم 256 قاب وجود دارد (و $2^8 = 256$ است) 8 بیتی است. چون در این سیستم شماره قاب برابر $\frac{1}{4}$ شماره صفحه است، کافی است شماره صفحه را 2 بیت به راست شیفت دهیم تا شماره قاب صفحه به دست آید پس شماره قاب متناظر با این آدرس برابر 5 (یعنی 00000101) خواهد شد. با قرار دادن شماره قاب در سمت چپ افست، آدرس فیزیکی متناظر مشخص می‌شود که برابر 0000010110111010 خواهد بود.

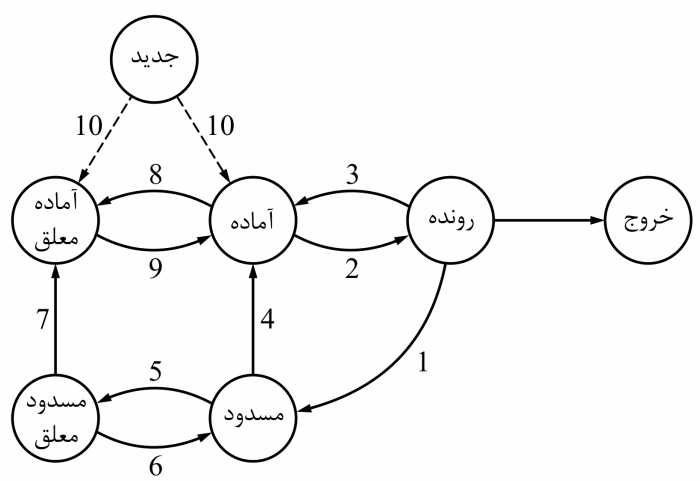
اما این روش نگاشت مشکل دارد، زیرا مثلاً صفحات شماره 2,1,0 و 3 همگی به شماره قاب 0 و صفحات شماره 6,5,4 و 7 همگی به قاب شماره 2 نگاشت می‌شوند و نمی‌توانند همزمان با هم درون حافظه فیزیکی قرار داده شوند.

۱. یک حافظه مجازی با این مشخصات در نظر بگیرید. زمان دسترسی حافظه 50 ns زمان دستیابی TLB, 2 ns نسبت اصابت TLB (hit ratio)، 98% و احتمال خطای صفحه برای کل دسترسی‌ها به حافظه 2×10^{-6} است. زمان انتقال صفحه از دیسک را 10 ms فرض کنید. برای سرعت بخشیدن به این حافظه از حافظه پنهان (cache) با این مشخصات استفاده شده است. زمان دسترسی حافظه پنهان 10 ns نسبت اصابت حافظه پنهان 90% جریمه هر عدم اصابت در حافظه پنهان 100 ns است. میانگین زمان دسترسی به حافظه برای هر آدرس به کدام یک از گزینه‌های زیر نزدیک‌تر است؟

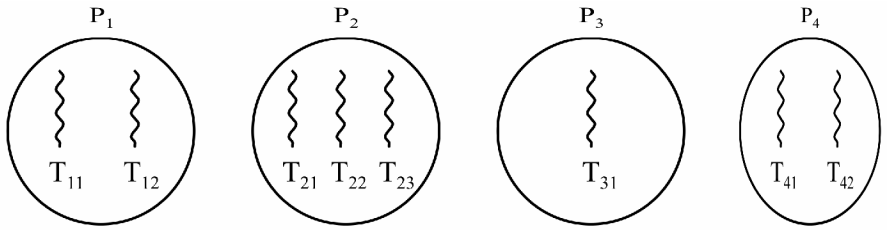
۱) 75 ns ۲) 45 ns ۳) 73 ns ۴) 43 ns

۲. شکل زیر تغییر حالت‌های یک فرایند را نشان می‌دهد. تغییر حالت از رونده به خروج، امکان دارد باعث کدام تغییر حالت‌ها شود؟

۱) 10 , 7 , 4
۲) 2 , 4 , 1
۳) 8 , 5 , 2
۴) 9 , 6



۳. سیستمی شامل 4 فرایند است که داخل هر فرایند می‌تواند بیش از یک نخ (Thread) اجرایی وجود داشته باشد. در لحظه صفر وضعیت این 4 فرآیند و تعداد نخ‌های اجرایی آنها در شکل و جدول زیر مشخص شده است؟



زمان لازم برای اجرای نخ‌ها

فرایند	P_1		P_2			P_3	P_4	
نخ	T_{11}	T_{12}	T_{21}	T_{22}	T_{23}	T_{31}	T_{41}	T_{42}
زمان اجرا (ms)	12	9	7	8	8	9	7	8

سهم زمانی هر فرایند 10 ms است و از روش نوبت گردشی (RR) استفاده می‌شود. همچنین داخل هر فرایند از روش FIFO برای تعویض نخ‌ها استفاده می‌شود و تا زمان اجرایی یک نخ تمام نشده نوبت به نخ بعدی نمی‌رسد. برای تعویض فرآیند 1 ms و برای تعویض نخ در داخل فرایند 0.5 ms زمان لازم است. زمان پایان نخ‌های T_{12} و T_{22} چقدر است؟

۱) 73 ms و 59.5 ms ۲) 70 ms و 45.5 ms ۳) 45.5 ms و 70.5 ms ۴) 18 ms و 29 ms

۴. به فرآیندی 4 قاب (frame) تخصیص یافته است. (تمام اعداد دهدهی هستند و همه شماره‌گذاری‌ها از صفر شروع شده است) زمان آخرین بار شدن یک صفحه در یک قاب، زمان آخرین دستیابی به صفحه، شماره صفحه مربوط به هر قاب، بیت‌های مراجعه (R) و تغییر (M) برای هر قاب در جدول زیر نشان داده شده‌اند (زمان‌ها برحسب ضربان ساعت و از شروع فرایند در زمان صفر است)

شماره صفحه	شماره قاب	زمان بار شدن صفحه در حافظه	زمان آخرین مراجعه	بیت مراجعه (R)	بیت تغییر (M)
2	0	60	160	0	1
1	1	130	161	0	0
0	2	26	163	1	0
3	3	20	162	1	1

یک خطای صفحه برای صفحه 4 رخ داده است. برای هر یک از سیاست‌های مدیریت حافظه FIFO و LRU و clock (اشاره‌گر روی صفحه صفر است و اولویت با انتخاب صفحه تغییر نیافته است) به ترتیب چه صفحه‌ای برای جایگزینی انتخاب می‌شوند؟

۱) 0 و 1 و 1 ۲) 1 و 0 و 2 ۳) 3 و 2 و 1 ۴) 1 و 1 و 2

۵. مدیریت حافظه در یک سیستم فرضی به صورت قطعه‌بندی صفحه‌بندی شده(paged segmentation) است و اندازه هر صفحه 4 کیلو بایت است) هر درایه (entry) جدول قطعه دارای 3 بایت و به صورت زیر است:

بایت سوم	بایت دوم بایت اول
LIMIT	PTBA

و هر درایه جدول صفحه یک بایتی است و نشان دهنده شماره قاب (frame) است. در PCB یک فرآیند برای آدرس پایه جدول قطعه (STBA) مقدار 0AFEH دیده می‌شود. اگر در این فرآیند آدرس منطقی $[02H, 3456H]$ تولید شود، آدرس فیزیکی نظیر چه خواهد بود. بخش اول آدرس منطقی شماره قطعه است. حرف H به معنی Hex است. محتویات حافظه به صورت زیر است:

0B00H	08H	0B08H	08H
0B01H	09H	0B09H	08H
0B02H	03H	0B0AH	00H
0B03H	0AH	0B0BH	0BH
0B04H	0BH	0B0CH	0AH
0B05H	09H	0B0DH	0CH
0B06H	05H	0B0EH	04H
0B07H	0BH	0B0FH	05H

۱) 08345H
۲) 0A456H
۳) 0B560H
۴) 08456H

۶. فرض کنید 5 فرایند با مشخصات زیر به یک سیستم با زمان‌بند چرخشی (Round-Robin) با برش زمانی $q = 1$ وارد شوند: (e: execution time, r: release time)

با فرض این که همیشه بین فرایندی که در لحظه t برش زمانی خود را به پایان می‌رساند و فرآیند ورودی در لحظه t اولویت با فرآیند قبلی موجود در سیستم است و در شرایط کاملاً یکسان بین دو فرآیند، اولویت با فرآیند با شماره کوچک‌تر است. میانگین زمان انتظار فرآیندهای فوق کدام است؟

$P_1 : r = 0 \quad , \quad e = 2$
$P_2 : r = 0 \quad , \quad e = 3$
$P_3 : r = 1 \quad , \quad e = 2$
$P_4 : r = 1 \quad , \quad e = 3$
$P_5 : r = 2 \quad , \quad e = 2$

۱) 6.4
۲) 5.8
۳) 6.2
۴) 5.6

۷. کدام یک از موارد زیر نادرست است؟

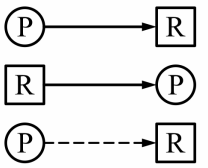
۱) تنظیم زمان سیستم در مود کاربر انجام می‌شود.

۲) کامپیوتر در هنگام روشن شدن در مود کرنل قرار می‌گیرد.

۳) تغییر اولویت فرآیندها در مود کرنل انجام می‌شود.

۴) خواندن ساعت سیستم در مود کاربر انجام می‌شود.

۸. در یک گراف مربوط درخواست و تخصیص منابع یال‌های زیر را تعریف می‌کنیم.



(نوع 1) فرآیند P متقاضی منبع R است.

(نوع 2) منبع R در اختیار فرآیند P است.

(نوع 3) فرآیند P مدعی استفاده از منبع R در آینده است.

با فرض وجود یک نمونه (instance) **از هر منبع کدام عبارت درست است؟**

۱) وجود حلقه در گراف شامل یال‌های نوع 1 و 2 به منزله احتمال وجود بن‌بست و وجود حلقه در گراف شامل یال‌های نوع 2,1 و 3 به منزله وجود بن‌بست است.

۲) وجود حلقه در گراف شامل یال‌های نوع 1 و 2 به منزله احتمال وجود بن‌بست و وجود حلقه در گراف شامل یال‌های نوع 2,1 و 3 به منزله احتمال وجود بن‌بست است.

۳) وجود حلقه در گراف شامل یال‌های 1 و 2 به منزله وجود بن‌بست و وجود حلقه در گراف شامل یال‌های نوع 2,1 و 3 به منزله احتمال وجود بن‌بست است.

۴) وجود حلقه در گراف شامل یال‌های نوع 1 و 2 به منزله وجود بن‌بست و وجود حلقه در گراف شامل یال‌های 2,1 و 3 به منزله وجود بن‌بست است.

پاسخنامه

۱. گزینه ۴ درست است.

چون در صورت سؤال آمده است که احتمال خطای صفحه برای تمامی دسترسی‌ها به حافظه $P_{PF} = 2\times10^{-6}$ است، رابطه کل زمان دسترسی به صورت زیر خواهد بود:

$$T_{Access} = T_{Translation} + T_{Cached\_Mem} + P_{PF} \times T_{Disk}$$

$$T_{Translation} = T_{TLB} + (1 - H_{TLB}) \times T_{Cached\_Mem}$$

$$T_{Cached\_Mem} = T_{Cache} + (1 - H_{Cache}) \times T_{Penalty}$$

با جای‌گذاری مقادیر صورت مسئله در روابط بالا خواهیم داشت:

$$T_{Cached\_Mem} = 10 + (1 - 0.9) \times 100 = 20ns$$

$$T_{Translation} = 2 + (1 - 0.98) \times 20 = 2.4ns$$

$$T_{Access} = 2.4 + 20 + (2\times10^{-6}) \times (10\times10^{-3}) = 2.4 + 20 + 20 = 42.4ns \approx 43ns$$

۲. گزینه ۴ درست است.

واقعاً تست ناپخته‌ای است و گزینه‌های عجیبی دارد. اگرچه گزینه انتخابی طراح ۴ است اما گزینه ۱ هم می‌تواند درست باشد.

با تغییر حالت از رونده به خروج یک فرایند از سیستم خارج می‌شود. در نتیجه اولاً CPU آزاد می‌شود. ثانیاً بخشی از حافظه آزاد می‌شود. بنابراین تغییر حالت‌های ممکن به شرح زیر خواهند بود:

تغییر حالت شماره 2 امکان پذیر است. چون CPU آزاد شده و یک فرایند آماده باید به حالت اجرا در بیاید.

تغییر حالت شماره 10 امکان پذیر است. چون یک فرایند خارج شده و جا برای یک فرایند جدید باز شده است.

تغییر حالت‌های شماره 4 و 7 امکان پذیرند. چون ممکن است مثلاً فرایند پدر با فراخوانی waitpid منتظر خاتمه و exit فرزندش بوده و با خروج فرزندش بیدار شده شود.

تغییر حالت‌های شماره 6 و 9 نیز امکان پذیرند. چون بخشی از حافظه آزاد می‌شود و ممکن است زمان‌بند حافظه تصمیم بگیرد یک فرایند معلق را در حافظه اصلی بارگذاری کند.

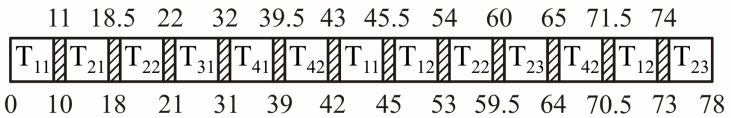
از طرفی تغییر حالت‌های غیر ممکن به شرح زیر خواهند بود:

تغییر حالت‌های شماره 1 و 3 امکان ناپذیرند. چون فقط یک فرایند در حالت رونده بوده که آن هم خارج شده است. بنابراین گزینه ۲ نادرست است.

تغییر حالت‌های شماره 5 و 8 به این موضوع ربطی ندارند. چون دچار کمبود حافظه نشده‌ایم که بخواهیم فرایندی را از حافظه خارج کنیم. بنابراین گزینه ۳ نادرست است.

۳. گزینه ۱ درست است.

نمودار گانت به شکل زیر خواهد بود:



بنابراین اجرای دو نخ T_{12} و T_{22} به ترتیب در زمان‌های 73 و 59.5 خاتمه می‌یابند.

۴. گزینه ۳ درست است.

الگوریتم FIFO صفحه‌ای را خارج می‌کند که قدیمی‌تر است و زودتر وارد حافظه شده است که در اینجا صفحه شماره 3 اخراج می‌شود که در زمان 20 بارگذای شده است. از همین ابتدا معلوم شد که گزینه ۳ درست است!

طبق الگوریتم LRU صفحه‌ای که زمان آخرین مراجعه به آن در گذشته دورتری بوده است خارج می‌شود. در اینجا صفحه شماره 2 که در زمان 160 به آن مراجعه شده است برای اخراج انتخاب می‌شود.

در الگوریتم Clock همان لیست FIFO (به ترتیب ورود) را داریم تنها با این تفاوت که در زمان اخراج بیت ارجاع (R) صفحه‌ای که اشاره‌گر به آن اشاره دارد، بررسی می‌شود و اگر این مقدار 1 باشد، مقدارش 0 شده و اشاره‌گر یکی جلو می‌رود و اگر بیت ارجاع آن 0 باشد، آن صفحه برای اخراج انتخاب می‌شود. این لیست (از چپ به راست) به صورت زیر است:

$3 \to 0 \to 2 \to 1$

R : 1 1 0 0

M : 1 0 1 0

اکنون طبق صورت سؤال، اشاره‌گر به صفحه 0 اشاره می‌کند که بیت R آن 1 است، آنرا 0 می‌کنیم و یکی به جلو می‌رویم. اگرچه برای صفحه 2 بیت R برابر 0 است و طبق الگوریتم ساعت معمولی باید خارج شود اما دقت کنید که در صورت مسئله آمده است که اولویت با انتخاب صفحه تغییر نیافته است $(M=0)$ و یک الگوریتم ساعت تغییریافته (پیشرفته) داریم که بیت M را نیز در نظر می‌گیرد که در اینجا 1 است. فعلاً به سراغ صفحه بعدی (1) می‌رویم که بیت R آن برابر 0 و بیت M آن نیز 0 است و این صفحه برای اخراج نسبت به صفحه 2 اولویت دارد. پس پاسخ صفحه شماره 1 است. اگر یک دور می‌زدیم و هیچ صفحه‌ای هر دو بیتش 0 نبود همان صفحه 2 انتخاب می‌شد.

۵. گزینه ۲ درست است.

چون در آدرس منطقی شماره قطعه برابر $(02)_H$ و هر 3 بایت پشت سر هم در حافظه نشان‌گر یک درایه از جدول قطعه باشد، باید $2\times 3 = 6$ بایت از ابتدای جدول قطعه به جلو برویم. می‌دانیم آدرس شروع جدول قطعه فرایند جاری (STBA) برابر $(0AFE)_H$ است. یعنی آدرس درایه قطعه مورد نظر برابر است با:

```
 0AFE +
    6
---------
 0B04
```

یعنی $(0B04)_H$. اگر ممکن است در جمع هگزادسیمال اشتباه کنید این کار را در مبنای 2 انجام دهید.

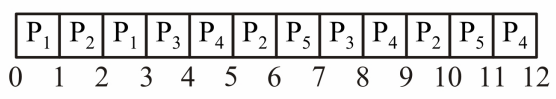
با مراجعه به آدرس $(0B04)_H$ و برداشتن 3 بایت پشت سر هم به 0B0905 می‌رسیم که طبق صورت سؤال 0B09 بیانگر PTBA و 05 بیانگر LIMIT است.

چون هر قطعه صفحه‌بندی شده است، بخش دوم آدرس منطقی، یعنی $(3456H)$ از P#,offset تشکیل شده و با توجه به اینکه صفحات 4 کیلو بایتی $(2^{12}=4KB)$ هستند، پس 12 بیت کم ارزش آن (3 رقم سمت راست هگزادسیمال) که برابر 456H است بیانگر افست و 3 نشان‌دهنده شماره صفحه خواهد بود.

به ابتدای جدول صفحه یعنی آدرس 0B09 می‌رویم و چون هر درایه جدول صفحه یک بایتی است و صفحه شماره 3 را می‌خواهیم 3 بایت به جلو می‌رویم و به آدرس 0B0C می‌رسیم. محتوای این آدرس یعنی 0A بیانگر شماره قاب صفحه است و کافی است در سمت چپ افست (456) قرار گیرد تا آدرس فیزیکی $(0A456)$ را بسازند. یعنی گزینه ۲ درست است.

۶. گزینه ۲ درست است.

با توجه به مفروضات تست، نمودار گانت اجرای این 5 فرایند به شکل زیر خواهد شد:



با توجه به نمودار فوق، میانگین زمان انتظار خواهد شد:

$$\frac{(0+1)+(1+3+3)+(2+3)+(3+3+3+)(4+3)}{5}=\frac{29}{5}=5.8$$

۷. گزینه ۱ درست است.

۸. گزینه ۳ درست است.