موسسه بابان

انتشارات بابان و انتشارات راهیان ارشد

درس و کنکور ارشد

سیستم عامل

(حل سوالات کنکور ارشد ۱۴۰۱)

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی کامپیوتر و IT

براساس کتب مرجع

آبراهام سیلبرشاتز، ویلیام استالینگز و اندور اس تننباوم

ارسطو خلیلی‌فر

تست‌های فصل اول: مفاهیم اولیه

۱۰۱- اگر نرخ انتقال اطلاعات بین حافظه اصلی و حافظه مجازی 50 MB / Sec، اندازه هر فرآیند به طور متوسط 10 MB و سیستم عامل چند برنامگی (Multi program) باشد که بتواند فرایندهای زیادی داخل حافظه بارگذاری کرده و همزمان با DMA اجرا نماید و هر فرآیند 200 میلی ثانیه به CPU نیاز داشته باشد، نرخ بهره‌وری CPU به کدام مورد نزدیکتر است؟

(مهندسی کامپیوتر - دولتی ۱۴۰۱)

۱) 100%　　۲) 75%　　۳) 50%　　۴) 25%

عنوان کتاب: سیستم عامل

مولف: استاد خلیلی‌فر

ناشر: انتشارات راهیان ارشد و دکتری

آدرس سایت گروه بابان: BABAN.IR

پاسخ تست‌های فصل اول: مفاهیم اولیه

۱۰۱- گزینه (۱) صحیح است.

در لحظه‌ی اجرای یک فرآیند جاری که اجرای آن 200 میلی‌ثانیه زمان می‌برد، بخش‌های مختلف فرآیند بعدی از حافظه مجازی به حافظه اصلی منتقل می‌گردد. زمان انتقال بخش‌های مختلف یک فرآیند 10 مگابایتی با نرخ انتقال 50 مگابایت در ثانیه از حافظه مجازی به حافظه اصلی 200 میلی‌ثانیه طول می‌کشد، به صورت زیر:

$$T_F = \frac{L}{R} = \text{زمان انتقال} = \frac{\text{اندازه فرآیند}}{\text{نرخ انتقال}} = \frac{10\,MB}{50\,MB\,sec} = \frac{1}{5}sec = \frac{1}{5}\times 10^3 = 200\ M\,sec$$

واضح است که زمان اجرای یک فرآیند جاری (200 میلی‌ثانیه) با زمان ورودی و خروجی فرآیند بعدی (200 میلی‌ثانیه) همپوشانی کامل دارد. بنابراین نرخ بهره‌وری CPU نزدیک 100 درصد است.

توجه: در صورت سوال ذکر شده است که هر فرآیند 200 میلی‌ثانیه عمل CPU دارد. و کارهای مولفه ورودی و خروجی هم توسط DMA به طور موازی انجام می‌شود. ضمن اینکه سیستم عامل هم چندبرنامگی است و کارهای CPU یک فرآیند می‌تواند با کارهای ورودی خروجی یک فرآیند دیگر به طور موازی انجام شود.

به طور کلی برای انتقال ورودی و خروجی سه روش در کامپیوترها قابل استفاده است:

- روش سرکشی یا نمونه‌برداری (Polling)
- روش وقفه (Interupt)
- روش DMA

در روش سرکشی (Polling) هر دستگاه دارای ثباتی کنترلی است که CPU با بررسی آن می‌تواند متوجه شود که آیا عملیات انتقال آن دستگاه تمام شده است یا خیر. لذا در این تکنیک CPU می‌بایست در یک پریود زمانی، مرتبا ثبات‌های کنترلی دستگاه‌های مختلف را به صورت سرکشی بررسی کند که باعث اتلاف وقت CPU و به تبع کاهش بهره‌وری CPU است. با آنکه پیاده‌سازی این روش ساده است و به امکانات سخت‌افزاری خاصی نیاز ندارد ولی سرعت آن پایین است. این روش مانند کلاس درسی است که در آن استاد هر چند دقیقه یکبار از تک تک دانشجویان به ترتیب بپرسد که آیا سوالی دارند یا خیر، همچنین روش سرکشی باعث اتلاف وقت دستگاه‌های IO و به تبع کاهش بهره‌وری دستگاه‌های IO می‌شود، چون برای مثال اگر کار IO یک دستگاه ورودی و خروجی تمام شود باید آنقدر معطل بماند تا نوبت سرکشی آن توسط CPU فرا برسد. اما در روش وقفه (Interupt) هر دستگاه دارای سیگنال کنترلی مخصوص به خود است که به نحوی به CPU ارتباط دارد. هرگاه انتقال داده‌ها توسط آن دستگاه تمام شود، سیگنالی را به نام وقفه به سمت CPU می‌فرستد تا آن را از این موضوع مطلع سازد. در این حالت پردازنده، پردازش

جاری را متوقف ساخته و به سرویس‌دهی این وقفه می‌پردازد. این روش مانند کلاس درسی است که در آن هر دانشجویی که سوال دارد در هر زمان با بالا بردن دست خود این موضوع را به اطلاع استاد می‌رساند. در این حالت استاد در اولین زمان مناسب صحبت‌های جاری خود را قطع کرده و پاسخ آن دانشجو را می‌دهد. سرعت این روش از روش سرکشی بیشتر است. همچنین روش وقفه باعث افزایش بهره‌وری CPU و دستگاه‌های IO می‌شود. تمام کامپیوترها راهکاری را فراهم می‌کنند تا قسمت‌های مختلف کامپیوتر (مانند ورودی و خروجی) روند اجرای دستورالعمل‌ها توسط پردازنده را جهت سرویس‌دهی‌های دیگری قطع کنند. در واقع مکانیزمی را فراهم می‌کند تا اجرای دستورالعمل‌های جاری پردازنده موقتاً متوقف شده و دستورات سرویس‌دهی دیگری اجرا شوند، پس از آن دوباره کنترل به همان برنامه باز می‌گردد. سیستم عامل جهت تسریع در پاسخ دادن به وقفه‌ها، آدرس روال‌های سرویس‌دهنده به آنها را در جدولی به نام جدول توصیف وقفه **(Interrupt Descriptor Table)** نگهداری می‌کند که به ازای هر وقفه یک درایه در این جدول وجود دارد که به آن بردار وقفه **(Interrupt Vector)** می‌گویند.

دسترسی مستقیم به حافظه (DMA)

اگرچه روش ورودی و خروجی مبتنی بر وقفه از روش سرکشی کارآمدتر است، اما هر دوی این روش‌ها نیاز به دخالت فعال، جز به جز و مستقیم CPU برای انتقال داده‌ها مابین حافظه اصلی و مولفه ورودی و خروجی دارد، انجام امور جز به جز مثل خواندن یک بایت از مولفه ورودی و سپس نوشتن آن داخل حافظه اصلی و تکرار مکرر آن وقت CPU را می‌گیرد و به هدر می‌دهد. انتخاب روش‌های مذکور برای انتقال حجم اندکی از داده‌ها مناسب و قابل تحمل است، اما زمانی که برای جابه‌جایی حجم زیادی از داده‌ها استفاده شوند، مثل خواندن از دیسک و نوشتن در دیسک، سربار زیادی روی CPU ایجاد می‌کند. برای کاهش این سربار و به تبع افزایش بهره‌وری CPU از DMA استفاده می‌شود. DMA سرواژه عبارت Direct Memory Access و به معنی دسترسی مستقیم به حافظه اصلی است. در این روش CPU به جای مدیریت مستقیم و جز به جز نقش نظارتی و کنترلی روی ورودی و خروجی دارد. که موجب افزایش بهره‌وری CPU و همچنین افزایش نرخ انتقال می‌شود.

بدون DMA، زمانی که پردازنده از ورودی وخروجی استفاده می‌کند، به‌طور معمول در تمام مدت عملیات خواندن یا نوشتن مشغول است و در نتیجه برای انجام کارهای دیگر در دسترس نیست. با DMA، پردازنده ابتدا انتقال را آغاز می‌کند، سپس، در مدتی که انتقال در حال پیشرفت است کارهای دیگری انجام می‌دهد، در نهایت زمانی که عملیات به پایان رسید یک وقفه از کنترل‌کننده DMA دریافت می‌کند.

مراحل سه‌گانه عملیات DMA به صورت زیر است:

۱-شروع واگذاری کارهای ورودی و خروجی از CPU به DMA

این واگذاری به معنی برنامه‌ریزی و تنظیم رجیسترهای DMA توسط CPU است. بنابراین DMA خواهد فهمید که چه حجمی از داده را به صورت بایت‌شمار از کجا به کجا باید منتقل کند. سپس انتقال بین مولفه ورودی و خروجی و حافظه اصلی از طریق ارسال درخواست انتقال و گذرگاه انجام می‌شود. در این حین آدرس محل خواندن یا نوشتن باید مشخص شود.

۲- CPU به کارهای دیگر می‌پردازد و به موازات آن DMA مشغول انجام کارهای ورودی و خروجی می‌شود.

۳-پایان ارتباط CPU و DMA از طریق ارسال یک وقفه از DMA به CPU

توجه: گاهی ممکن است، یک فرآیند در بخشی از اجرای خود فقط کارهای CPU و RAM داشته باشد و نیازی به مولفه‌های ورودی و خروجی نداشته باشد، اما گاهی هم ممکن است، یک فرآیند در بخشی از اجرای خود کار با مولفه ورودی و خروجی داشته باشد. در این شرایط کنترل گذرگاه داده در اختیار DMA است. بنابراین فرآیند دیگری که فقط کار CPU و RAM دارد برای تصاحب گذرگاه برای انتقال داده باید صبر پیشه کند.

تست‌های فصل دوم: مدیریت فرآیندها و زمان‌بندی پردازنده

۱۰۲- اگر سه فرآیند متناوب جدول زیر با الگوریتم زمان‌بندی قبضه‌ای اولویت‌دار زمان‌بندی شوند و اولویت با فرآیندی باشد که نسبت تقسیم «مدت زمان CPU» بر «دوره تناوب» آن کمترین است، بهره‌وری CPU چقدر خواهد بود؟ (در لحظه صفر هر سه فرآیند به ترتیب وارد می‌شوند.)

(مهندسی کامپیوتر - دولتی ۱۴۰۱)

P1	P2	P3	
5	20	10	**مدت زمان CPU**
25	50	40	**دوره تناوب**

۱) 0.8 ۲) $3\left(\sqrt[3]{2}-1\right)$ ۳) 0.85 ۴) زمان‌بندی امکان‌پذیر نیست.

پاسخ‌های فصل دوم: مدیریت فرآیندها و زمان‌بندی پردازنده

۱۰۲- گزینه (۳) صحیح است.

در سیستم‌های بی‌درنگ، زمان پاسخگویی به فرآیندها نباید از یک حد مشخصی (مهلت زمانی) بیش‌تر شود، در غیر اینصورت از بین خواهند رفت. سیستم‌های بی‌درنگ معمولا به شیوه‌ای پیاده‌سازی می‌شوند که باید به اتفاقات و حوادث خارجی سریعا پاسخ دهند. این وقایع به دو طبقه متناوب و غیرمتناوب تقسیم می‌شوند:

وقایع متناوب (مساوی): آن دسته از وقایعی که در فواصل زمانی منظم و مساوی رخ می‌دهند، مانند استراحت دستگاه فتوکپی پس از چاپ 500 برگه در فواصل زمانی منظم و مساوی.

وقایع غیرمتناوب (غیرمساوی): آن دسته از وقایعی که در فواصل زمانی نامنظم و غیرمساوی رخ می‌دهند، مانند فشردن پدال ترمز اتومبیل در فواصل زمانی نامنظم و غیرمساوی.

شرط زمان‌بندی

زمان‌بندی تمام وقایع متناوب باهم در یک سیستم بی‌درنگ بر اساس رابطه زیر ممکن است **امکان‌پذیر** باشد:

$$\sum_{i=1}^{m} \frac{t_i}{p_i} \leq 1$$

در رابطه فوق m تعداد وقایع متناوب در سیستم و i رخدادی که اتفاق افتاده است و t_i مدت زمانی که اجرای واقعه به پردازنده نیاز دارد و p_i دوره تناوب واقعه است.

توجه: کلمه **امکان‌پذیر بودن** در عبارت فوق، به معنی حتمی بودن و قطعی بودن نیست، بلکه به این معنی است که زمان‌بندی تمام وقایع متناوب باهم در یک سیستم بی‌درنگ بر اساس برقراری معیار فوق ممکن است شدنی باشد یا نباشد.

توجه: یک سیستم بی‌درنگ که منطبق با معیار فوق نباشد، به طور قطع غیرقابل زمان‌بندی است، اما اگر این معیار را داشته باشد یک الگوریتم خوب **ممکن است** بتواند آنرا زمان‌بندی کرده و همه مهلت‌ها را محقق کند.

سوال: در یک سیستم بی‌درنگ نرم، 3 واقعه متناوب با دوره‌های تناوب 100 و 200 و 500 میلی‌ثانیه وجود دارند، اگر زمان پردازش هر واقعه به ترتیب 50 و 30 و 100 میلی‌ثانیه باشد، آیا این سیستم قابل زمان‌بندی است؟

پاسخ:

$$\sum_{i=1}^{3}\frac{t_i}{p_i}=\frac{50}{100}+\frac{30}{200}+\frac{100}{500}=0.5+0.15+0.2=0.85\le 1$$

از آنجاکه شرط فوق برقرار است، ممکن است زمان‌بندی وقایع فوق در عمل امکان‌پذیر باشد.

توجه: به طور کلی الگوریتم‌های زمان‌بندی سیستم‌های بی‌درنگ به دو طبقه **ایستا** و **پویا** تقسیم می‌شود. در الگوریتم‌های ایستا، اولویت زمان‌بندی قبل از اجرای فرآیندها تعیین می‌شود (پیشا اجرا)، اما در الگوریتم‌های پویا، اولویت زمان‌بندی حین اجرای فرآیندها تعیین می‌شود (حینا اجرا). معروف‌ترین الگوریتم ایستا، نرخ یکنواخت (Rate-Monotonic Scheduling) و معروف‌ترین الگوریتم پویا، ابتدا زودترین مهلت (EDF: Earliest Deadline First) است.

بر اساس فرض صورت سوال، الگوریتم زمان‌بندی بی‌درنگ برای اجرای فرآیندهای متناوب از سیاست اولویت «ایستا» از نوع «غیرانحصاری» (preemptive) یا قبضه‌ای، قابل تخلیه پیش‌هنگام و قابل پس گرفتن استفاده می‌کند. در این الگوریتم، در ابتدای کار به هر فرآیند یک اولویت ایستا، ثابت، غیرقابل تغییر و برای همیشه بر اساس دوره تناوب داده می‌شود. مطابق فرض سوال اولویت با فرآیندی است که نسبت تقسیم «مدت زمان **CPU**» بر «دوره تناوب» آن کمترین باشد، به صورت زیر:

$$\text{اولویت}=\frac{\text{مدت زمان CPU}}{\text{دوره تناوب}}=\frac{t_i}{p_i}$$

توجه: مطابق رابطه فوق فرآیندهای با نتیجه تقسیم کمتر، دارای اولویت بیشتر خواهند بود.

در زمان اجرای فرآیندها، زمان‌بند پردازنده، همیشه فرآیندی که بیشترین اولویت را دارد، انتخاب می‌کند و چنانچه فرآیندی با اولویت بیشتر در صف آماده حضور یابد، بلافاصله فرآیند جاری را قبضه خواهد کرد.

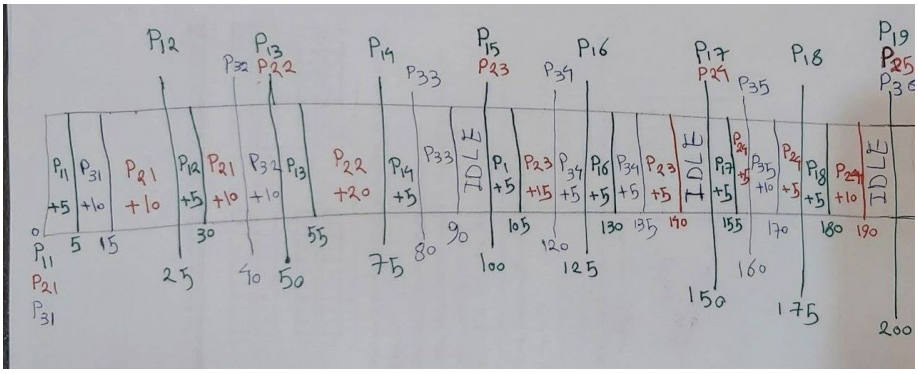
اگر سوال فوق را بر اساس تعریف الگوریتم فرض شده در نظر بگیریم، آنگاه اولویت بیشتر به فرآیند P_1 بعد P_3 و در نهایت P_2 داده می‌شود. به صورت زیر:

P1	P2	P3	
5	20	10	**مدت زمان CPU**
25	50	40	**دوره تناوب**

$$\text{اولویت } P_1 = \frac{5}{25} > \quad \text{اولویت } P_3 = \frac{10}{40} > \quad \text{اولویت } P_2 = \frac{20}{50}$$

$$\sum_{i=1}^{3} \frac{t_i}{p_i} = \frac{5}{25} + \frac{20}{50} + \frac{10}{40} = 0.2 + 0.4 + 0.25 = 0.85 \leq 1$$

واضح است که در مجموع، بهره‌وری فرآیندهای متناوب P_1 ، P_2 **و** P_3 برابر 0.85 است. بنابراین از نظر **تئوری** مهلت‌های زمانی فرآیندها قابل زمان‌بندی است. اما برای بررسی قابل زمان‌بندی بودن آن به طور قطعی و **عملی** باید نمودار گانت آن رسم شود. ترتیب اجرای فرآیندها به صورت شکل زیر است:



واضح است که ابتدا P_1، اجرای خود را آغاز نموده و در لحظه 5 خاتمه می‌یابد. و مهلت زمانی اول فرآیند P_1 هم برآورده می‌شود. در این نقطه، P_3 اجرای خود را آغاز کرده و در لحظه 15 خاتمه می‌یابد و مهلت زمانی اول فرآیند P_3 هم برآورده می‌شود. در این نقطه، P_2 اجرای خود را آغاز کرده و تا لحظه 25 اجرا می‌شود. در لحظه 25 با اینکه 10 میلی‌ثانیه از زمان اجرای فرآیند P_2 باقی‌مانده‌است اما مطابق الگوریتم فرض شده صورت سوال، پردازنده از P_2 (اولویت کمتر) گرفته می‌شود و به P_1 (اولویت بیشتر) داده می‌شود چون در لحظه 25 رویداد راه‌انداز بار دوم برای فرآیند P_1 از راه رسیده‌است. مطابق تعریف الگوریتم مطرح شده همواره اولویت با فرآیند با اولویت بیشتر و به صورت «غیرانحصاری» (preemptive) یا قبضه‌ای است. در ادامه فرآیند P_1 پردازش خود را در نقطه 30 کامل می‌کند که در این لحظه P_2 از پایان اجرای قبلی، کار خود را ادامه می‌دهد. و در لحظه 40 خاتمه می‌یابد و مهلت زمانی اول فرآیند P_2 هم برآورده می‌شود. این روند تا لحظه 200 و برآورده سازی تمامی مهلت‌های زمانی هر سه فرآیند P_1 ، P_2 **و** P_3 ادامه دارد.

دقت کنید که فرآیند P_1 هر 25 میلی‌ثانیه یکبار و در لحظات 0 و 25 و 50 و 75 و ... ، فرآیند P_2 هر 50 میلی‌ثانیه یکبار و در لحظات 0 و 50 و 150 و ... و فرآیند P_3 هر 40 میلی‌ثانیه یکبار و در لحظات 0 و 40 و 80 و ... و این چرخه ادامه دارد و در نهایت مهلت زمانی هر سه فرآیند محقق می‌شود.

توجه: مقدار 200 از ک.م.م سه مقدار 25، 50 و 40 حاصل شده است.

توجه: ممکن است در مسئله‌ای معیار قابلیت زمان‌بندی در تئوری برقرار باشد، یعنی بهره‌وری پردازنده کمتر یا مساوی 1 باشد، اما در عمل و در نمودار گانت مهلت‌های زمانی فرآیندها محقق نشود.

مطابق خواسته سوال، بهره‌وری CPU از رابطه زیر محاسبه می‌شود:

$$\sum_{i=1}^{3}\frac{t_i}{p_i}=\frac{5}{25}+\frac{20}{50}+\frac{10}{40}=0.2+0.4+0.25=0.85$$

اگر به نمودار گانت دقت کنید در مدت زمان 200 میلی‌ثانیه در لحظات 90 تا 100، 140 تا 150 و 190 تا 200 در مجموع 30 میلی‌ثانیه زمان هدر رفته است، یعنی 170 میلی‌ثانیه زمان مفید و 30 میلی‌ثانیه زمان غیرمفید مصرف شده است. که رابطه زیر برای آن برقرار است:

$$\sum_{i=1}^{3}\frac{t_i}{p_i}=\frac{5}{25}+\frac{20}{50}+\frac{10}{40}=\frac{40+80+50}{200}=\frac{170}{200}=0.85$$

تست‌های فصل ششم: مدیریت فرآیندها و نخ‌های هم‌روند

۱۰۳- آسانسور ساختمانی 20 طبقه (از همکف الی طبقه 19) با ظرفیت حمل 1 نفر مفروض است. فرض کنید در هر طبقه 1 نفر زندگی می‌کند و در شبانه‌روز از آسانسور برای رفت و برگشت به دیگر طبقات استفاده می‌کند. الگوریتم حرکت آسانسور خالی برای توقف در طبقه درخواستی، در همان جهتی است که قبلا حرکت می‌کرده است (مثلا اگر هنگام حمل مسافر از طبقه 1 به سمت 4 حرکت کرده است، پس از تخلیه مسافر، آسانسور به سمت طبقات 5 الی 19 حرکت می‌کند تا اگر کسی در این طبقات درخواست داشت، بایستد. سپس از طبقه 19 به سمت همکف حرکت می‌کند و اگر کسی در این طبقات درخواست داشت، می‌ایستد. آسانسور خالی مدام در حالت حرکت و پیمایش طبقات است. در ابتدا خالی بوده و در طبقه همکف (صفر) قرار دارد.) در صورتی که این مسئله، مشابه مسئله ناحیه بحرانی مدنظر باشد طوری که مسافران حکم فرآیند (پردازه) و آسانسور حکم ناحیه بحرانی را داشته باشد، چند شرط از شروط ناحیه بحرانی (انحصار متقابل، پیشرفت، انتظار محدود) نقض می شود؟ **(مهندسی کامپیوتر - دولتی ۱۴۰۱)**

۱) دقیقا 1 شرط نقض می‌شود.
۲) دقیقا 3 شرط نقض می‌شود.
۳) دقیقا 2 شرط نقض می‌شود.
۴) هیچ شرطی نقض نمی‌شود.

عنوان کتاب: سیستم عامل

مولف: استاد خلیلی‌فر

ناشر: انتشارات راهیان ارشد و دکتری

آدرس سایت انتشارات بابان: BABAN.IR

پاسخ‌های فصل ششم: مدیریت فرآیندها و نخ‌های هم‌روند

۱۰۳- گزینه (۱) صحیح است.

شرایط رقابتی (مسابقه)

هرگاه دو یا چند فرآیند همزمان با هم وارد ناحیه‌ی بحرانی (منبع مشترک) شوند، شرایط رقابتی پیش می‌آید. در شرایط رقابتی، نتیجه‌ی نهایی بستگی به ترتیب دسترسی‌ها دارد. در واقع فرآیندهای همکار بر هم اثر دارند و اینکه پردازنده، به چه ترتیبی و در چه زمان‌هایی بین آنها تعویض متن انجام دهد در ایجاد پاسخ نهایی اثرگذار خواهد بود. بنابراین علت شرایط رقابت **تعویض متن پردازنده** بین فرآیندهای همکار است.

برای کنترل شرایط رقابتی، باید راه حلی ارائه شود که سه شرط زیر را به عنوان **معیارهای اخلاقی** در رقابت، رعایت کند:

۱- شرط انحصار متقابل

برای برقرای شرط انحصارمتقابل، عامل مشترک را اسکورت کنید، مانند زمانی که وارد باجه‌ی تلفن همگانی (عامل مشترک) می‌شوید، در را می‌بندید تا مانع ورود شخص دیگری گردید! در عالم انسان‌ها، هیچ دو فردی نباید به طور همزمان وارد عامل مشترک شوند. در عالم فرآیندها نیز هیچ دو فرآیندی نباید به طور همزمان وارد عامل مشترک (ناحیه بحرانی) شوند. استفاده‌ی همزمان از عامل مشترک معنا ندارد! (اخلاقی نیست) بنابراین باید راهی پیدا کنیم که از ورود همزمان دو یا چند فرآیند به ناحیه‌ی بحرانی جلوگیری کند. به عبارت دیگر، آنچه که ما به آن نیاز داریم، انحصار متقابل است که در متون فارسی به آن دو به دو ناسازگاری یا مانعه الجمعی نیز گفته می‌شود، یعنی اگر یکی از فرآیندها در حال استفاده از حافظه‌ی اشتراکی، فایل اشتراکی و یا هر عامل اشتراکی رقابت‌زاست باید مطمئن باشیم که دیگر فرآیندها، در آن زمان از انجام همان کار محروم می‌باشند. در واقع از بین تمام فرآیندها، در هر لحظه تنها یک فرآیند مجاز است، در عامل مشترک باشد. بدین معنی که اگر فرآیندی در ناحیه‌ی بحرانی است، از ورود فرآیندهای دیگر به همان ناحیه‌ی بحرانی جلوگیری شود و تا خارج شدن فرآیند اول منتظر بمانند، زیرا هیچ دو فرآیندی نباید به طور همزمان وارد ناحیه‌ی بحرانی شوند. به یاد داشته باشید که استفاده‌ی همزمان از عامل مشترک معنا ندارد!

بنابراین برای برقراری شرط انحصار متقابل باید ساختاری را طراحی کنیم که در هر لحظه فقط یک فرآیند مجوز ورود به ناحیه‌ی بحرانی را داشته باشد. لذا هر فرآیند برای ورود به بخش بحرانی‌اش باید اجازه بگیرد. بخشی از کد فرآیند که این اجازه گرفتن را پیاده‌سازی می‌کند، بخش ورودی نام دارد. بخش بحرانی می‌تواند با بخش خروجی دنبال شود. این بخش خروجی کاری می‌کند که فرآیندهای دیگر بتوانند وارد ناحیه‌ی بحرانی‌شان، شوند. بقیه‌ی کد فرآیند را بخش باقی‌مانده می‌نامند. بنابراین ساختار کلی فرآیندها برای برقراری شرط انحصار متقابل به صورت زیر می‌باشد:

```
P (int i)  {
while ( TRUE)  {
entry_section () ;  // تلاش برای کسب اجازه‌ی ورود به ناحیه‌ی بحرانی
critical_section (); // ناحیه‌ی بحرانی
exit_section () ; // اعلام خروج از ناحیه‌ی بحرانی
remainder_section () ; // ناحیه‌ی باقی مانده
   }
}
```

توجه: بدترین شرایط وقتی است که یک فرآیند بخواهد بارها و بارها وارد ناحیه بحرانی خود شود، برای اینکه سخت‌ترین شرایط بررسی شود، ناحیه بحرانی را داخل حلقه بی‌نهایت قرار می‌دهیم.

۲- شرط پیشرفت

فرآیندی که داوطلب ورود به ناحیه‌ی بحرانی نیست و نیز در ناحیه‌ی بحرانی قرار ندارد، نباید در رقابت برای ورود سایر فرآیندها به ناحیه‌ی بحرانی شرکت کند، به عبارت دیگر، نباید مانع ورود فرآیندهای دیگر به ناحیه‌ی بحرانی شود. در یک بیان ساده‌تر می‌توان گفت، فرآیندی که در ناحیه‌ی باقی‌مانده قرار دارد، حق جلوگیری از ورود فرآیندهای دیگر به ناحیه‌ی بحرانی را ندارد، یعنی نباید در تصمیم‌گیری برای ورود فرآیندها به ناحیه‌ی بحرانی شرکت کند.

۳- شرط انتظار محدود

فرآیندهایی که نیاز به ورود به ناحیه‌ی بحرانی دارند، باید مدت انتظارشان محدود باشد، یعنی نباید به طور نامحدود در حالت انتظار باقی بمانند.

انتظار نامحدود به دو دسته می‌باشد: (۱) قحطی، (۲) بن‌بست، بنابراین نباید در شرایط رقابتی بین فرآیندها، قحطی یا بن‌بست رخ دهد. برای اینکه شرط انتظار محدود برقرار باشد، باید هم قحطی و هم بن‌بست رخ ندهد.

قحطی (گرسنگی)

در عالم زندگی قحطی زمانی رخ می‌دهد که عده‌ای مدام از منابع مشترک استفاده کنند، و عده‌ای دیگر قادر به استفاده از منابع مشترک نباشند. زیرا دسته‌ی اول از اختصاص منابع به دسته‌ی دوم به طور مداوم و بدون رعایت یک حد بالای مشخص جلوگیری می‌کنند. در عالم فرآیندها نیز هرگاه فرآیندی به مدت نامعلوم و بدون رعایت یک حد بالای مشخص در انتظار گرفتن یک منبع بحرانی یا دسترسی به یک عامل مشترک بماند و فرآیندی دیگر مدام در حال استفاده از منبع بحرانی باشد، در این حالت فرآیند اول دچار قحطی شده است. بنابراین در صورت اقدام یک فرآیند برای ورود

به ناحیه‌ی بحرانی، باید محدودیتی برای تعداد دفعاتی که سایر فرآیندها می‌توانند وارد ناحیه‌ی بحرانی شوند، وجود داشته باشد تا قحطی رخ ندهد.

بن‌بست

به وضعیتی که در آن مجموعه‌ای متشکل از دو یا چند فرآیند برای همیشه منتظر یکدیگر بمانند (مسدود) و به عبارت دیگر دچار سیکل انتظار ابدی شوند، بن‌بست گفته می‌شود.

توجه: به تفاوت قحطی و بن‌بست توجه کنید، در قحطی فرآیندی مدام در حال کار و فرآیندی دیگر به مدت نامعلوم در انتظار است. اما در بن‌بست، مجموعه‌ای از فرآیندها در سیکل انتظار ابدی، گرفتار شده‌اند. نه راه پس دارند و نه راه پیش.

توجه: در کنترل شرایط رقابتی، رعایت شرط انحصار متقابل، شرط لازم و رعایت شروط پیشروی و انتظار محدود، شروط کافی برای ارائه‌ی یک راه‌حل **جامع** و **اخلاقی** به شمار می‌آیند.

ابتدا الگوریتم مطرح شده در صورت سوال را برای دو فرآیند P_0 و P_1 به صورت زیر بازنویسی می‌کنیم:

P_0:
```
{
(1) while (turn ! = 0); /*loop*/
/ * critical section */
(2) turn = 1;
/ * reminder section */
}
```

P_1:
```
{
(1) while (turn ! = 1); /*loop*/
/ * critical section */
(2) turn = 0;
/ * reminder section */
}
```

توجه: جهت سهولت در تحلیل، آسانسور را 2 طبقه فرض می‌کنیم که توسط تعریف آرایه و تغییراتی در قطعه کد، قابل تعمیم به طبقات بیشتر است.

توجه: مطابق فرض سوال این آسانسور در ابتدا خالی بوده و در طبقه همکف (صفر) یعنی نوبت فرآیند P_0 قرار دارد. سپس به سمت طبقه اول یعنی نوبت فرآیند P_1 حرکت می‌کند، در ادامه به سمت P_0 و سپس به سمت P_1 حرکت می‌کند و این روند P_0 به P_1 و P_1 به P_0 ادامه دارد.

توجه: ایستادن در هر ایستگاه (طبقه) به معنی دادن CPU به فرآیند است. قطعه کد هر فرآیند زمانی انجام می‌شود که CPU به فرآیند داده شود و CPU زمانی به یک فرآیند داده می‌شود که آسانسور در طبقه همان فرآیند باشد.

توجه: در الگوریتم آسانسور امکان ندارد دو فرآیند با هم وارد ناحیه‌ی بحرانی شوند، علت این امر به خاصیت آسانسور روش نوبت‌بندی مربوط می‌شود. بنابراین شرط انحصار متقابل برقرار است.

توجه: در الگوریتم آسانسور پس از پیاده‌شدن مسافر از آسانسور(خروج از ناحیه باقی‌مانده)، آسانسور به طبقه بعدی می‌رود. یعنی مسافر نمی‌تواند در همان طبقه یکبار پیاده شود و در همان لحظه مجدد سوار آسانسور شود (ورود به ناحیه بحرانی). پس از پیاده شدن مسافر از آسانسور

نوبت به مسافر طبقه بعدی می‌رسد. و همین موضوع عامل نقض شرط پیشرفت در الگوریتم آسانسور است.

توجه: در الگوریتم آسانسور گرسنگی وجود ندارد. زیرا فرآیندها به صورت آسانسوری و نوبتی وارد ناحیه بحرانی می‌شوند و نه تصادفی، این آسیاب به نوبت است. بنابراین گرسنگی رخ نمی‌دهد.

توجه: در الگوریتم آسانسور بن‌بست وجود ندارد. زیرا فرآیندها به صورت آسانسوری و نوبتی وارد ناحیه بحرانی می‌شوند و نه تصادفی و موازی و هم‌روند، این آسیاب به نوبت است. بنابراین بن‌بست رخ نمی‌دهد.

توجه: همانطور که گفتیم در الگوریتم آسانسور گرسنگی و بن‌بست رخ نمی‌دهد، بنابراین شرط انتظار محدود برقرار است.

توجه: پُرواضح است که گزینه اول پاسخ سوال است، زیرا فقط شرط پیشرفت برقرار نیست.

حال شرایط رقابتی را برای این الگوریتم بررسی می‌کنیم:

شرط انحصار متقابل:

برای کنترل برقراری شرط انحصار متقابل، شرط پیشرفت و شرط انتظار محدود (گرسنگی و بن‌بست) از آزمون‌های زیر استفاده می‌کنیم:

توجه: ما نام این آزمون‌ها را به عنوان مبدع آن «**قوانین ارسطو**» نام‌گذاری کردیم، این قوانین به «**قوانین چهارگانه ارسطو**» نیز موسوم است.

قانون اول ارسطو (آزمون اول شرط انحصار متقابل)

(گام ۱) فرآیند اول را داخل ناحیه بحرانی خودش قرار بده، **(گام ۲)** سپس فرآیند دوم هم تصمیم بگیرد وارد ناحیه بحرانی خودش شود، اگر موفق شود که وارد ناحیه بحرانی خودش شود، در این حالت شرط انحصار متقابل نقض شده است.

(گام ۱): فرآیند اول را داخل ناحیه بحرانی خودش قرار بده، یعنی:

فرض کنید فرآیند P_0 قصد دارد وارد ناحیه بحرانی خودش شود، به صورت زیر:

P_0:
(1) while (turn ! = 0);

توجه: هم اکنون turn = 0 است.

شرط حلقه FALSE است، پس کنترل برنامه از حلقه خارج شده و داخل ناحیه بحرانی فرآیند P_0 قرار می‌گیرد، به صورت زیر:

P_0:
/*critical section*/

همانطور که در (گام ۱) گفتیم قرار شد که فرآیند اول را داخل ناحیه بحرانی خودش قرار بدهیم، خب قرار دادیم. حال در ادامه آزمون اول وارد (گام ۲) می‌شویم. هم اکنون پردازنده در ناحیه بحرانی فرآیند P_0 مشغول حرکت است.

(گام ۲): فرآیند دوم هم تصمیم بگیرد وارد ناحیه بحرانی خودش شود، اگر موفق شود که وارد ناحیه بحرانی خودش شود، در این حالت شرط انحصار متقابل نقض شده است، یعنی:

در ادامه پردازنده را از فرآیند P_0 بگیرید و به فرآیند P_1 بدهید.

فرض کنید فرآیند P_1 نیز قصد دارد وارد ناحیه بحرانی خودش شود، به صورت زیر:

P_1:

(1) while (turn ! = 1);

توجه: هم اکنون turn = 0 است.

شرط حلقه TRUE است، پس کنترل برنامه از حلقه خارج نشده و داخل ناحیه بحرانی فرآیند P_1 قرار نمی‌گیرد، این حلقه مدام تکرار می‌شود و فرآیند P_1 در یک حلقه انتظار مشغول پشت ناحیه بحرانی خود می‌ماند و می‌چرخد تا مادامی که کوانتوم آن تمام شود. این پدیده به انتظار مشغول (Busy Waiting) موسوم است.

همانطور که در (گام ۲) گفتیم قرار شد که فرآیند دوم هم تصمیم بگیرد وارد ناحیه بحرانی خودش شود، اگر موفق شود که وارد ناحیه بحرانی خودش شود، در این حالت شرط انحصار متقابل نقض شده است، خب موفق نشد. فرآیند دوم نتوانست وارد ناحیه بحرانی خودش بشود. بنابراین شرط اول انحصار متقابل برقرار است.

قانون دوم ارسطو(آزمون دوم شرط انحصار متقابل)

فرآیند اول و دوم را به طور همروند در سیستم تک پردازنده‌ای و یا موازی در سیستم چند پردازنده‌ای حرکت بدید اگر هر دو باهم توانستند وارد ناحیه بحرانی خودشان شوند، آنگاه در این حالت شرط انحصار متقابل نقض شده است.

توجه: در الگوریتم آسانسور بن‌بست وجود ندارد. زیرا فرآیندها به صورت آسانسوری و نوبتی وارد ناحیه بحرانی می‌شوند و نه تصادفی و موازی و هم‌روند، این آسیاب به نوبت است. بنابراین بن‌بست رخ نمی‌دهد.

همانطور که گفتیم قرار شد که فرآیند اول و دوم را به طور همروند در سیستم تک پردازنده‌ای و یا موازی در سیستم چند پردازنده‌ای حرکت بدهیم اگر هر دو باهم توانستند وارد ناحیه بحرانی خودشان شوند، آنگاه در این حالت شرط انحصار متقابل نقض شده است، خب هر دو باهم موفق نمی‌شوند. فرآیند اول و دوم نمی‌توانند هر دو باهم وارد ناحیه بحرانی خودشان بشوند. بنابراین شرط دوم انحصار متقابل نیز برقرار است.

توجه: برای برقرار بودن شرط انحصار متقابل باید **قانون اول ارسطو** (آزمون اول شرط انحصار متقابل) و **قانون دوم ارسطو** (آزمون دوم شرط انحصار متقابل) هر دو باهم برقرار باشند. بنابراین شرط انحصار متقابل در سوال مطرح شده برقرار است.

قانون سوم ارسطو (آزمون شرط پیشرفت)

(گام ۱) فرآیند اول را داخل ناحیه بحرانی خودش قرار بده، **(گام ۲)** سپس همان فرآیند اول را داخل ناحیه باقی‌مانده خودش قرار بده، **(گام ۳)** در ادامه فرآیند دوم را داخل ناحیه بحرانی خودش قرار بده، **(گام ۴)** سپس همان فرآیند دوم را داخل ناحیه باقی‌مانده خودش قرار بده، **(گام ۵)** در نهایت همان فرآیند دوم به ابتدای برنامه برگردد و مجددا تصمیم بگیرد وارد ناحیه بحرانی خودش شود، اگر موفق شود که مجددا وارد ناحیه بحرانی خودش شود، آنگاه در این حالت شرط پیشرفت برقرار است، در غیر اینصورت شرط پیشرفت برقرار نیست.

(گام ۱): فرآیند اول را داخل ناحیه بحرانی خودش قرار بده، یعنی:

```
P0:                                   P1:
 {                                     {
(1) while (turn ! = 0); /*loop*/      (1) while (turn ! = 1); /*loop*/
/ * critical section */                / * critical section */
(2) turn = 1;                         (2) turn = 0;
/ * reminder section */                / * reminder section */
}                                     }
```

فرض کنید فرآیند P_0 قصد دارد وارد ناحیه بحرانی خودش شود، به صورت زیر:

```
P0:
(1) while (turn ! = 0);
```

توجه: هم اکنون $turn = 0$ است.

شرط حلقه FALSE است، پس کنترل برنامه از حلقه خارج شده و داخل ناحیه بحرانی فرآیند P_0 قرار می‌گیرد، به صورت زیر:

```
P0:
/*critical section*/
```

همانطور که در (گام ۱) گفتیم قرار شد که فرآیند اول را داخل ناحیه بحرانی خودش قرار بدهیم، خب قرار دادیم. حال در ادامه آزمون سوم وارد (گام ۲) می‌شویم. هم اکنون پردازنده در ناحیه بحرانی فرآیند P_0 مشغول حرکت است.

(گام ۲): همان فرآیند اول را داخل ناحیه باقی‌مانده خودش قرار بده، یعنی:

فرض کنید فرآیند P_0 از بخش خروج از ناحیه بحرانی خودش عبور کند، به صورت زیر:

```
P0:
```

/*critical section*/

(2) turn = 1;

حال در ادامه فرآیند P_0 پس از عبور از بخش خروج از ناحیه بحرانی خودش در ناحیه باقی‌مانده خودش قرار می‌گیرد، به صورت زیر:

P_0:

/*remainder_section*/

همانطور که در (گام ۲) گفتیم قرار شد که همان فرآیند اول را داخل ناحیه باقی‌مانده خودش قرار بدهیم، خب قرار دادیم. حال در ادامه آزمون سوم وارد (گام ۳) می‌شویم. هم اکنون پردازنده داخل ناحیه باقی‌مانده فرآیند P_0 مشغول حرکت است.

(گام ۳): فرآیند دوم را داخل ناحیه بحرانی خودش قرار بده، یعنی:

در ادامه پردازنده را از فرآیند P_0 بگیرید و به فرآیند P_1 بدهید.

فرض کنید فرآیند P_1 نیز قصد دارد وارد ناحیه بحرانی خودش شود، به صورت زیر:

P_1:

(1) while (turn ! = 1);

توجه: هم اکنون turn = 1 است.

شرط حلقه FALSE است، پس کنترل برنامه از حلقه خارج شده و داخل ناحیه بحرانی فرآیند P_1 قرار می‌گیرد، به صورت زیر:

P_1:

/*critical section*/

همانطور که در (گام ۳) گفتیم قرار شد که فرآیند دوم را داخل ناحیه بحرانی خودش قرار بدهیم، خب قرار دادیم. حال در ادامه آزمون سوم وارد (گام ۴) می‌شویم. هم اکنون پردازنده در ناحیه بحرانی فرآیند P_1 مشغول حرکت است.

(گام ۴): همان فرآیند دوم را داخل ناحیه باقی‌مانده خودش قرار بده، یعنی:

فرض کنید فرآیند P_1 از بخش خروج از ناحیه بحرانی خودش عبور کند، به صورت زیر:

P_1:

/*critical section*/

(2) turn = 0;

حال در ادامه فرآیند P_1 پس از عبور از بخش خروج از ناحیه بحرانی خودش در ناحیه باقی‌مانده خودش قرار می‌گیرد، به صورت زیر:

P_1:

/*remainder_section

همانطور که در (گام ۴) گفتیم قرار شد که همان فرآیند دوم را داخل ناحیه باقی‌مانده خودش قرار بدهیم، خب قرار دادیم. حال در ادامه آزمون سوم وارد (گام ۵) می‌شویم. هم اکنون پردازنده داخل ناحیه باقی‌مانده فرآیند P_1 مشغول حرکت است.

(گام ۵): فرآیند دوم به ابتدای برنامه برگردد و مجددا تصمیم بگیرد وارد ناحیه بحرانی خودش شود، اگر موفق شود که مجددا وارد ناحیه بحرانی خودش شود، آنگاه در این حالت شرط پیشرفت برقرار است، در غیر اینصورت شرط پیشرفت برقرار نیست. یعنی:

فرض کنید فرآیند P_1 نیز قصد دارد مجددا وارد ناحیه بحرانی خودش شود، به صورت زیر:

P_1:

(1) while (turn ! = 1);

توجه: هم اکنون turn = 0 است.

شرط حلقه TRUE است، پس کنترل برنامه از حلقه خارج نشده و داخل ناحیه بحرانی فرآیند P_1 قرار نمی‌گیرد، این حلقه مدام تکرار می‌شود و فرآیند P_1 در یک حلقه انتظار مشغول پشت ناحیه بحرانی خود می‌ماند و می‌چرخد تا مادامی که کوانتوم آن تمام شود. این پدیده به انتظار مشغول (Busy Waiting) موسوم است.

همانطور که در (گام ۵) گفتیم قرار شد که فرآیند دوم به ابتدای برنامه برگردد و مجددا تصمیم بگیرد وارد ناحیه بحرانی خودش شود، اگر موفق شود که مجددا وارد ناحیه بحرانی خودش شود، آنگاه در این حالت شرط پیشرفت برقرار است، خب نشد، فرآیند دوم نتوانست مجددا وارد ناحیه بحرانی خودش بشود. بنابراین شرط پیشرفت برقرار نیست.

توجه: در الگوریتم آسانسور پس از پیاده‌شدن مسافر از آسانسور(خروج از ناحیه باقی‌مانده)، آسانسور به طبقه بعدی می‌رود. یعنی مسافر نمی‌تواند در همان طبقه یکبار پیاده شود و در همان لحظه مجدد سوار آسانسور شود (ورود به ناحیه بحرانی). پس از پیاده شدن مسافر از آسانسور نوبت به مسافر طبقه بعدی می‌رسد. و همین موضوع عامل نقض شرط پیشرفت در الگوریتم آسانسور است.

قانون چهارم ارسطو (آزمون گرسنگی)

(گام ۱) فرآیند اول را داخل ناحیه بحرانی خودش قرار بده، **(گام ۲)** سپس فرآیند دوم را پشت ناحیه بحرانی خودش قرار بده، **(گام ۳)** در ادامه فرآیند اول را داخل ناحیه باقی‌مانده خودش قرار بده، **(گام ۴)** در نهایت همان فرآیند اول به ابتدای برنامه برگردد و مجددا تصمیم بگیرد وارد ناحیه بحرانی خودش شود، اگر موفق شود که مجددا وارد ناحیه بحرانی خودش شود، آنگاه در این حالت فرآیند دوم دچار گرسنگی شده است. و شرط انتظار محدود نقض شده است.

P_0:
```
 {
(1) while (turn ! = 0); /*loop*/
/ * critical section */
(2) turn = 1;
/ * reminder section */
}
```

P_1:
```
 {
(1) while (turn ! = 1); /*loop*/
/ * critical section */
(2) turn = 0;
/ * reminder section */
}
```

(گام ۱): فرآیند اول را داخل ناحیه بحرانی خودش قرار بده، یعنی:
فرض کنید فرآیند P_0 قصد دارد وارد ناحیه بحرانی خودش شود، به صورت زیر:

P_0:
(1) while (turn ! = 0);

توجه: هم اکنون turn = 0 است.

شرط حلقه FALSE است، پس کنترل برنامه از حلقه خارج شده و داخل ناحیه بحرانی فرآیند P_0 قرار می‌گیرد، به صورت زیر:

P_0:
/*critical section*/

(گام ۲): فرآیند دوم را پشت ناحیه بحرانی خودش قرار بده، یعنی:
در ادامه پردازنده را از فرآیند P_0 بگیرید و به فرآیند P_1 بدهید.
فرض کنید فرآیند P_1 نیز قصد دارد وارد ناحیه بحرانی خودش شود، به صورت زیر:

P_1:
(1) while (turn ! = 1);

توجه: هم اکنون turn = 0 است.

شرط حلقه TRUE است، پس کنترل برنامه از حلقه خارج نشده و داخل ناحیه بحرانی فرآیند P_1 قرار نمی‌گیرد، این حلقه مدام تکرار می‌شود و فرآیند P_1 در یک حلقه انتظار مشغول پشت ناحیه بحرانی خود می‌ماند و می‌چرخد تا مادامی که کوانتوم آن تمام شود. این پدیده به انتظار مشغول (Busy Waiting) موسوم است.

همانطور که در (گام ۲) گفتیم قرار شد که فرآیند دوم را پشت ناحیه بحرانی خودش قرار بدهیم، خب قرار دادیم. حال در ادامه آزمون چهارم وارد (گام ۳) می‌شویم. هم اکنون پردازنده پشت ناحیه بحرانی فرآیند P_1 در یک حلقه انتظار، دچار انتظار مشغول است.

(گام ۳): فرآیند اول را داخل ناحیه باقی‌مانده خودش قرار بده، یعنی:
در ادامه پردازنده را از فرآیند P_1 بگیرید و به فرآیند P_0 بدهید.
فرض کنید فرآیند P_0 از بخش خروج از ناحیه بحرانی خودش عبور کند، به صورت زیر:

P_0:
/*critical section*/
(2) turn = 1;

حال در ادامه فرآیند P_0 پس از عبور از بخش خروج از ناحیه بحرانی خودش در ناحیه باقی‌مانده خودش قرار می‌گیرد، به صورت زیر:

P_0:
/*remainder section*/

همانطور که در (گام ۳) گفتیم قرار شد که فرآیند اول را داخل ناحیه باقی‌مانده خودش قرار بدهیم، خب قرار دادیم. حال در ادامه آزمون چهارم وارد (گام ۴) می‌شویم. هم اکنون پردازنده داخل ناحیه باقی‌مانده فرآیند P_0 مشغول حرکت است.

(گام ۴): فرآیند اول به ابتدای برنامه برگردد و مجددا تصمیم بگیرد وارد ناحیه بحرانی خودش شود، اگر موفق شود که مجددا وارد ناحیه بحرانی خودش شود، آنگاه در این حالت فرآیند دوم دچار گرسنگی شده است. یعنی:

فرض کنید فرآیند P_0 قصد دارد مجددا وارد ناحیه بحرانی خودش شود، به صورت زیر:

P_0:

(1) while (turn ! = 0);

توجه: هم اکنون turn = 1 است.

شرط حلقه TRUE است، بنابراین نمی‌توانیم فرآیند اول را داخل ناحیه بحرانی خودش قرار بدهیم. همانطور که در (گام ۴) گفتیم قرار شد که فرآیند اول به ابتدای برنامه برگردد و مجددا تصمیم بگیرد وارد ناحیه بحرانی خودش شود، اگر موفق شود که مجددا وارد ناحیه بحرانی خودش شود، آنگاه در این حالت فرآیند دوم دچار گرسنگی شده است، خب نشد، فرآیند اول نتوانست مجددا وارد ناحیه بحرانی خودش بشود. بنابراین گرسنگی رخ نداده است.

توجه: در الگوریتم آسانسور گرسنگی وجود ندارد. زیرا فرآیندها به صورت آسانسوری و نوبتی وارد ناحیه بحرانی می‌شوند و نه تصادفی، این آسیاب به نوبت است. بنابراین گرسنگی رخ نمی‌دهد.

قانون دوم ارسطو(آزمون بن‌بست)

جهت بررسی بن‌بست از همان **قانون دوم** استفاده می‌شود. در واقع روال بررسی همان قانون دوم است، اما نتیجه قانون متفاوت است.

فرآیند اول و دوم را به طور همروند در سیستم تک پردازنده‌ای و یا موازی در سیستم چند پردازنده‌ای حرکت بدید اگر هر دو باهم نتوانستند وارد ناحیه بحرانی شوند و هردو باهم پشت ناحیه بحرانی خودشان مسدود شدند، آنگاه در این حالت بن بست رخ داده است و شرط انتظار محدود نقض شده است. به عبارت دیگر هرگاه دو فرآیند متقاضی ورود به ناحیه بحرانی به طور همزمان تا ابد منتظر ورود به ناحیه بحرانی باشند، در این شرایط هر دو فرآیند مسدود و به خواب رفته‌اند که در این حالت «**بن‌بست**» رخ داده است.

همانطور که در **آزمون دوم** گفتیم قرار شد که فرآیند اول و دوم را به طور همروند در سیستم تک پردازنده‌ای و یا موازی در سیستم چند پردازنده‌ای حرکت بدهیم اگر هر دو باهم نتوانستند وارد ناحیه بحرانی شوند و هردو باهم پشت ناحیه بحرانی خودشان مسدود شدند، آنگاه در این حالت بن بست رخ داده است و شرط انتظار محدود نقض شده است.

توجه: در الگوریتم آسانسور بن‌بست وجود ندارد. زیرا فرآیندها به صورت آسانسوری و نوبتی وارد ناحیه بحرانی می‌شوند و نه تصادفی و موازی و هم‌روند، این آسیاب به نوبت است. بنابراین بن‌بست رخ نمی‌دهد.

توجه: برای برقرار بودن شرط انتظار محدود باید **قانون دوم ارسطو** (آزمون بن‌بست) و **قانون چهارم ارسطو**(آزمون گرسنگی) هر دو باهم برقرار باشند. بنابراین شرط انتظار محدود در سوال مطرح شده برقرار است.
توجه: پُر واضح است که گزینه اول پاسخ سوال است، زیرا فقط شرط پیشرفت برقرار نیست.

تست‌های فصل پنجم: مدیریت حافظه مجازی

۱۰۴- در سیستم صفحه‌بندی سلسله مراتبی دو سطحی، اگر برای ترجمه شماره صفحه به شماره قاب، مراجعه به جدول صفحه در حافظه اصلی، در صورت شکست در جدول TLB نیاز باشد و تاخیر دستیابی به حافظه اصلی 150 ns و نرخ شکست (miss rate) در جدول ترجمه پیش‌رو (TLB) برابر 2 درصد باشد، متوسط زمان دستیابی به یک داده با آدرس مجازی کدام مورد است؟ (تاخیر دسترسی به TLB ناچیز فرض شود.) **(مهندسی کامپیوتر - دولتی ۱۴۰۱)**

۱) 156 ۲) 6 ۳) 150 ۴) 60

عنوان کتاب: سیستم عامل

مولف: استاد خلیلی‌فر

ناشر: انتشارات راهیان ارشد و دکتری

آدرس سایت گروه بابان: khalilifar.ir

پاسخ‌های فصل پنجم: مدیریت حافظه مجازی

۱۰۴- گزینه (۱) صحیح است.

مفروضات صورت سوال به صورت زیر است:

زمان دسترسی به **TLB** برابر 0 ns

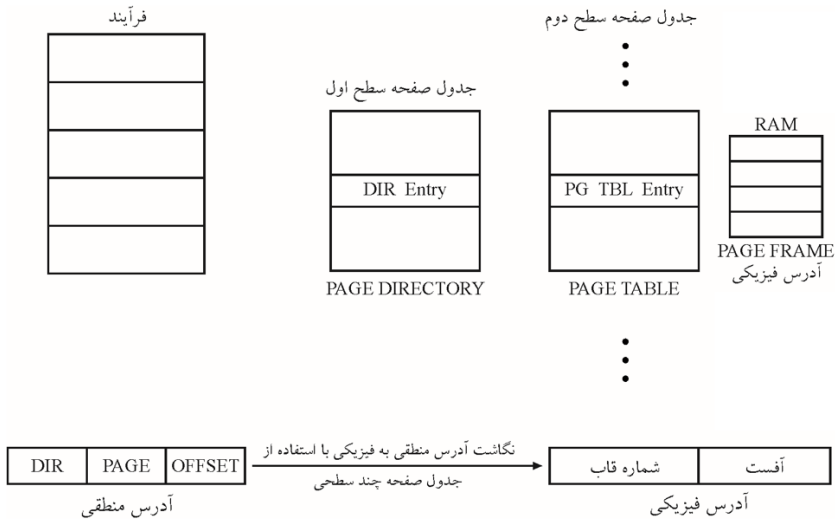
زمان دسترسی به حافظه برابر 150 ns

نسبت اصابت **TLB (miss ratio)** برابر **0.02**

نسبت اصابت **TLB (hit ratio)** برابر **0.98**

آدرس منطقی به صورت زیر است:

مطابق فرض مساله، برای نگاشت آدرس منطقی به فیزیکی از جداول صفحه چند سطحی (دوسطحی) استفاده شده است.



به طور کلی میانگین زمان دسترسی سیستم به مقصد نهایی برای هر آدرس سیستم از رابطه زیر محاسبه می‌گردد:

$$T_{Access} = T_{Translation} + T_{Destination}$$

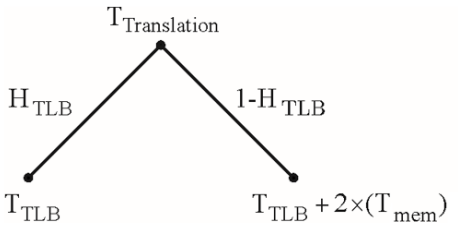
که پارامترهای آن شامل $T_{Translation}$ یعنی **بخش ترجمه** و $T_{Destination}$ یعنی **بخش مقصد** می‌باشد.

$\mathbf{T_{Translation}}$: میانگین زمان ترجمه

توجه: مطابق فرض سوال در **بخش ترجمه** یعنی $T_{Translation}$، دو عنصر T_{TLB} و T_{Mem} برای ترجمه وجود دارد، بنابراین باید میانگین آنها محاسبه گردد.

برای محاسبه $T_{Translation}$ میانگین T_{TLB} و T_{Mem} را محاسبه می‌کنیم که همان $T_{Translation}$ است. به صورت زیر:

برای بدست آوردن $T_{Translation}$ درخت زیر را در نظر بگیرید:



توجه: وجود ضریب 2 در پشت T_{Mem} به این دلیل است که برای تولید آدرس فیزیکی، به دو آدرس DIR در جدول صفحه جزئی سطح اول، یعنی PAGE DIRECTORY و آدرس PAGE در جدول صفحه جزئی سطح دوم، یعنی PAGE TABLE نیاز داریم. بنابراین به 2 بار T_{Mem} و مراجعه به RAM نیاز است. یعنی یک T_{Mem} برای مراجعه به جدول صفحه جزئی سطح اول و یک T_{Mem} دیگر برای مراجعه به جدول صفحه جزئی سطح دوم.

توجه: در سیستم جداول صفحه چندسطحی درایه‌های موجودی که در TLB قرار دارند، شامل اطلاعات مرتبط همه سطوح است، نمی‌شود سطحی باشد و سطحی نباشد، در سؤال مذکور یا تعدادی از درایه‌های مرتبط دو جدول PAGE DIRECTORY و PAGE TABLE به طور همزمان داخل TLB هستند و یا به طور همزمان داخل TLB نیستند.

رابطه درخت فوق به صورت زیر خواهد بود:

$$T_{Translation} = [H_{TLB} \times T_{TLB}] + [(1 - H_{TLB}) \times [T_{TLB} + 2 \times (T_{Mem})]]$$

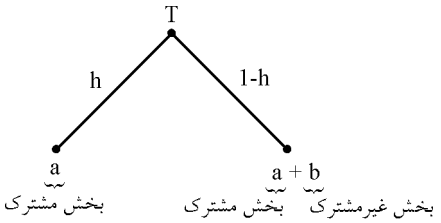
پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Translation}$ خواهیم داشت:

$$T_{Translation} = T_{TLB} + (1 - H_{TLB}) \times 2 \times (T_{Mem})$$

$$T_{Translation} = 0 + (0.02) \times 2 \times (150) = 6 \text{ ns}$$

توجه: دقت کنید که در صورت سوال نرخ شکست (miss rate) داده شده است و نه نرخ برخورد (hit rate) بنابراین مقدار $(1 - H_{TLB})$ برابر همان مقدار miss rate و برابر 0.02 است.

توجه: جهت ساده‌سازی رابطه درخت فوق، همواره می‌توان از **اتحاد درخت احتمال** به صورت زیر، استفاده نمود:



$$T = \text{بخش مشترک} + (1-h) \times \text{بخش غیرمشترک}$$

$$T = a + (1-h) \times b$$

پس از ساده‌سازی درخت مطرح شده براساس **اتحاد درخت احتمال**، رابطه زیر را برای محاسبه T_{Mem} خواهیم داشت:

$$T_{Translation} = T_{TLB} + (1 - H_{TLB}) \times 2 \times (T_{Mem})$$

$\mathbf{T_{Destination}}$: میانگین زمان دسترسی به مقصد موردنظر

توجه: مطابق فرض سوال در **بخش مقصد** یعنی $T_{Destination}$، یک عنصر T_{Mem} برای دسترسی به مقصد مورد نظر وجود دارد.

پس $T_{Destination}$ به صورت زیر محاسبه می‌گردد:

$$T_{Destination} = T_{Mem} = 150 \text{ ns}$$

و در نهایت براساس رابطه کل زمان دسترسی سیستم خواهیم داشت:

$$T_{Access} = T_{Translation} + T_{Destination}$$

$$T_{Access} = 6 + 150 = 156 \text{ ns}$$

تست‌های فصل هفتم: مدیریت بن بست

۱۰۵- در سیستمی با پنج فرآیند و دو منبع مطابق جداول زیر، حداقل x + y چقدر باشد تا سیستم در حالت امن باشد؟ **(مهندسی کامپیوتر - دولتی ۱۴۰۱)**

Available	
R_1	R_2
x	y

MAX	R_1	R_2
P_1	5	2
P_2	3	9
P_3	4	5
P_4	1	4
P_5	8	5

Allocation	R_1	R_2
P_1	1	2
P_2	2	5
P_3	2	0
P_4	1	1
P_5	0	0

۱) 7 ۲) 6 ۳) 5 ۴) 4

پاسخ‌های فصل هفتم: مدیریت بن‌بست

۱۰۵- گزینه (۴) صحیح است.

مطابق صورت سوال، بردار Available به صورت زیر است:

Available	R_1	R_2
	x	y

براساس رابطه زیر داریم:

Need = MAX – Allocation

فرآیند	Need	
	R_1	R_2
P_1	4	0
P_2	1	4
P_3	2	5
P_4	0	3
P_5	8	5

=

فرآیند	MAX	
	R_1	R_2
P_1	5	2
P_2	3	9
P_3	4	5
P_4	1	4
P_5	8	5

–

فرآیند	Allocation	
	R_1	R_2
P_1	1	2
P_2	2	5
P_3	2	0
P_4	1	1
P_5	0	0

Available	x	y

خواسته سوال محاسبه حداقل مقدار x+y است، بنابراین ابتدا گزینه چهارم را بررسی می‌کنیم که حداقل مقدار ممکن ($x + y = 4$) در بین گزینه‌ها است، که حالت‌های ممکن آن به صورت زیر است:

x	y	x + y	امن/ناامن
1	3	4	امن
3	1	4	ناامن
0	4	4	امن
4	0	4	ناامن

ابتدا حالت (1,3) را بررسی می‌کنیم:

$$(1,3) \xrightarrow[+(1,1)]{P_4} (2,4) \xrightarrow[+(2,5)]{P_2} (4,9) \xrightarrow[+(2,0)]{P_3} (6,9) \xrightarrow[+(1,2)]{P_1} (7,11) \longrightarrow \times$$

$$(1,3) \xrightarrow[+(1,1)]{P_4} (2,4) \xrightarrow[+(2,5)]{P_2} (4,9) \xrightarrow[+(1,2)]{P_1} (5,11) \xrightarrow[+(2,0)]{P_3} (7,11) \longrightarrow \times$$

مشاهده می‌شود که با بردار Available با مقادیر (1,3)، نمی‌توان نیاز فرآیند P_5 را باتوجه به ماتریس Need مرتفع ساخت، بنابراین یک توالی امن کامل برای تمامی فرآیندها یافت نمی‌شود.

اما به واسطه فقط یک فرآیند باقی‌مانده، سیستم در یک وضعیت ناامن قرار نمی‌گیرد و احتمال وقوع بن‌بست هم وجود ندارد. فرآیندها یا به صورت (1) **غیرهمروند و تنها** اجرا می‌شوند مانند اجرای فرآیند P_5 که برای ایجاد بن‌بست معنا ندارد، **چون یک فرآیند به تنهایی بن‌بست با خودش ایجاد نمی‌کند** و یا به صورت (2) **همروند** اجرا می‌شوند مانند اجرای فرآیندهای P_1 تا P_4 که اجرا شدند و تمام هم شدند.

حالت (3,1) را بررسی می‌کنیم:

$$(3,1) \longrightarrow \times$$

مشاهده می‌شود که با بردار Available با مقادیر (3,1)، نمی‌توان نیاز هیچ فرآیندی را باتوجه به ماتریس Need مرتفع ساخت، بنابراین توالی امن یافت نمی‌شود و سیستم در یک وضعیت ناامن قرار دارد و **احتمال وقوع بن‌بست** وجود دارد. در واقع در شرایط ناامن وقوع بن‌بست و عدم بن‌بست به رفتار فرآیندها در آینده، بستگی دارد. یعنی فرآیندها رفتار نگهداری و درخواست را در پیش گیرند و یا رفتار رهاسازی و درخواست را.

حالت (0,4) را بررسی می‌کنیم:

$$(0,4) \xrightarrow[+(1,1)]{P_4} (1,5) \xrightarrow[+(2,5)]{P_2} (3,10) \xrightarrow[+(2,0)]{P_3} (5,10) \xrightarrow[+(1,2)]{P_1} (6,12) \longrightarrow \times$$

مشاهده می‌شود که با بردار Available با مقادیر (0,4)، همان شرایط (1,3) ایجاد شد.

حالت (4,0) را بررسی می‌کنیم:

$$(4,0) \xrightarrow[+(1,2)]{P_1} (5,2) \longrightarrow \times$$

مشاهده می‌شود که با بردار Available با مقادیر (4,0)، مشابه همان شرایط (3,1) ایجاد شد.

توجه: سازمان سنجش آموزش کشور، در کلید اولیه خود، گزینه سوم را به عنوان پاسخ اعلام کرده بود. اما در کلید نهایی نظر خود را عوض کرد و گزینه چهارم را به عنوان پاسخ اعلام کرد.

در ادامه گزینه سوم ($x + y = 5$) هم بررسی می‌کنیم که حالت‌های ممکن آن به صورت زیر است:

x	y	x + y	امن /ناامن
4	1	5	امن
1	4	5	امن
2	3	5	امن

حالت (4,1) را بررسی می‌کنیم:

$$(4,1) \xrightarrow[+(1,2)]{P_1} (5,3) \xrightarrow[+(1,1)]{P_4} (6,4) \xrightarrow[+(2,5)]{P_2} (8,9) \xrightarrow[+(2,0)]{P_3} (10,9) \xrightarrow[+(0,0)]{P_5} (10,9)$$

$$(4,1) \xrightarrow[+(1,2)]{P_1} (5,3) \xrightarrow[+(1,1)]{P_4} (6,4) \xrightarrow[+(2,5)]{P_2} (8,9) \xrightarrow[+(0,0)]{P_5} (8,9) \xrightarrow[+(2,0)]{P_3} (10,9)$$

به این ترتیب و با توجه به ماتریس‌های فوق، دنباله‌ی امنِ $< P_1, P_4, P_2, P_3, P_5 >$ (از چپ به راست) یا $< P_1, P_4, P_2, P_5, P_3 >$ (از چپ به راست) برای این سیستم موجود است. بنابراین سیستم در وضعیت امن (مطمئن) قرار دارد. و البته در حالت حداقلی $x + y$ همان مقدار 4 در گزینه چهارم پاسخ سوال است.

حالت (1,4) را بررسی می‌کنیم که البته نیاز هم نیست چون کشف یک دنباله امن کافی است:

$$(1,4) \xrightarrow[+(2,5)]{P_2} (3,9) \xrightarrow[+(2,0)]{P_3} (5,9) \xrightarrow[+(1,2)]{P_1} (6,11) \xrightarrow[+(1,1)]{P_4} (7,12) \xrightarrow[+(0,0)]{P_5} (7,12)$$

حالت (2,3) را بررسی می‌کنیم:

$$(2,3) \xrightarrow[+(1,1)]{P_4} (3,4) \xrightarrow[+(2,5)]{P_2} (5,9) \xrightarrow[+(2,0)]{P_3} (7,9) \xrightarrow[+(1,2)]{P_1} (8,11) \xrightarrow[+(0,0)]{P_5} (8,11)$$

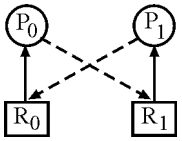
$$(2,3) \xrightarrow[+(1,1)]{P_4} (3,4) \xrightarrow[+(2,5)]{P_2} (5,9) \xrightarrow[+(1,2)]{P_1} (6,11) \xrightarrow[+(2,0)]{P_3} (8,11) \xrightarrow[+(0,0)]{P_5} (8,11)$$

توجه: 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 است.

توجه: در حالت ناامن نیز ممکن است، فرآیندها در موقعیت رهاسازی منابع در اختیار خود قرار بگیرند و سیستم به بن‌بست نرسد. به بیان دیگر در حالت ناامن وقوع بن‌بست قطعی نیست. اما اگر در حالت ناامن فرآیندها علاوه بر نگهداری منابع تحت تملک خود، منابع دیگری را مورد درخواست قرار دهند، در این حالت فرآیندهایی که نیاز آن‌ها برطرف نمی‌شود یک به یک در صف انتظار قرار می‌گیرند و همچون سرنوشت‌های به هم گره خورده، تا ابد منتظر یکدیگر باقی می‌مانند، یعنی بن‌بست رخ داده است.

مثال: در یک سیستم تعداد منبع اولیه R_0 برابر یک و تعداد منبع اولیه R_1 برابر یک است. اگر فرآیند P_0 یک منبع R_0 را در تملک داشته باشد و برای اتمام، نیاز به یک منبع R_1 نیز داشته باشد و همچنین فرآیند P_1 یک منبع R_1 را در تملک داشته باشد و برای اتمام، نیاز به یک منبع R_0 نیز داشته باشد، وضعیت سیستم را در این شرایط بررسی کنید:

حل: ماتریس و گراف منابع و فرآیندها به صورت زیر است:



فرآیند	Need	
	R_0	R_1
P_0	0	1
P_1	1	0

+

فرآیند	Allocation	
	R_0	R_1
P_0	1	0
P_1	0	1

=

فرآیند	MAX	
	R_0	R_1
P_0	1	1
P_1	1	1

با توجه به ماتریس تخصیص منابع (Allocation) و منابع اولیه، بردار Available را بدست می‌آوریم:

R_0 منبع موجود $= 1-(1) = 0$

R_1 منبع موجود $= 1-(1) = 0$

بنابراین بردار Available به صورت زیر است:

Available	0	0

مشاهده می‌شود که با بردار Available موجود، نمی‌توان نیاز هیچ فرآیندی را با توجه به ماتریس Need مرتفع ساخت، بنابراین توالی امن یافت نمی‌شود و سیستم در یک وضعیت ناامن قرار دارد و احتمال وقوع بن‌بست وجود دارد.

توجه: در این شرایط ناامن، اگر فرآیند P_0 یا P_1 و یا هر دو، در موقعیت رهاسازی منابع در اختیار خود قرار بگیرند، یعنی رهاسازی کنند و سپس درخواست منابع باقی‌مانده خود را صادر کنند، بن‌بست رخ نمی‌دهد.

توجه: در این شرایط ناامن، اگر فرآیند P_0 در عین حال اینکه منبع R_0 را تحت تملک و نگهداری خود دارد، منبع R_1 را نیز درخواست کند، از آن‌جا که منبع R_1 در اختیار فرآیند P_1 می‌باشد، فرآیند P_0 در صف انتظار، می‌خوابد (نگهداری و انتظار) حال اگر در ادامه ماجرا، فرآیند P_1 نیز همین رویه را در پیش گیرد، یعنی فرآیند P_1 نیز در عین حال اینکه منبع R_1 را تحت تملک و نگهداری خود دارد، منبع R_0 را نیز درخواست کند، از آن‌جا که منبع R_0 در اختیار فرآیند P_0 می‌باشد، فرآیند P_1 نیز در صف انتظار، می‌خوابد (نگهداری و انتظار). این سرنوشت‌های به هم گره خورده، در ادامه بن‌بست را به ارمغان می‌آورد.

توجه: از دو توجه فوق این نتیجه استنباط می‌گردد که در شرایط ناامن، **احتمال وقوع بن‌بست** وجود دارد. در واقع در شرایط ناامن وقوع بن‌بست و عدم بن‌بست به رفتار فرآیندها در آینده، بستگی دارد. یعنی فرآیندها رفتار نگهداری و درخواست را در پیش گیرند و یا رفتار رهاسازی و درخواست را.

توجه: سازمان سنجش آموزش کشور، در کلید اولیه خود، گزینه سوم را به عنوان پاسخ اعلام کرده بود. اما در کلید نهایی نظر خود را عوض کرد و گزینه چهارم را به عنوان پاسخ اعلام کرد.

تست‌های فصل اول: مفاهیم اولیه

۱۰۶- در خصوص اجرای دستورالعمل در کامپیوترهای مطابق الگوریتم فون نیومن که داخل یک حلقه بی‌انتها دستورالعمل‌ها واکشی شده و اجرا می‌گردد و با توجه به بحث بهره‌وری CPU در هنگام وجود سیستم عامل و برنامه‌های کاربر، کدام مورد درست‌تر است؟

(مهندسی کامپیوتر - دولتی ۱۴۰۱)

۱) بهره‌وری CPU تحت هر شرایطی 100 درصد است؛ زیرا همواره الگوریتم فون نیومن اجرا می‌شود که شامل اجرای فرآیندها با سیستم عامل است.
۲) چون طبق الگوریتم فون نیومن CPU مدام درگیر خواهد بود، در مواقعی که برنامه‌ای برای اجرا وجود ندارد و سیستم عامل کاری ندارد، CPU به وضعیت بیکار (Halt) می‌رود.
۳) بهره‌وری CPU را نباید با اجرای سیستم عامل به صورت همزمان لحاظ کرد، چون سیستم عامل سربار ناچیزی دارد.
۴) بهره‌وری CPU نباید شامل اجرای سیستم عامل گردد، لذا همیشه بهروری کمتر از 100 درصد است.

عنوان کتاب: سیستم عامل

مولف: استاد خلیلی‌فر

ناشر: انتشارات راهیان ارشد و دکتری

آدرس سایت انتشارات بابان: BABAN.IR

پاسخ تست‌های فصل اول: مفاهیم اولیه

۱۰۶- گزینه (۲) صحیح است.

بهره‌وری پردازنده (CPU Utilization or CPU Efficiency)

یک الگوریتم زمانبندی باید بهره‌وری پردازنده را به حداکثر برساند، به این معنی که حتی الامکان در تمام زمان‌ها پردازنده مشغول باشد تا زمان‌های هدر رفته پردازنده به حداقل برسد. بهره‌وری همان درصد استفاده **مفید** از CPU است. زمان‌های تعویض متن زمان‌های **غیرمفید** محسوب می‌شوند. رابطه بهره‌وری به صورت زیر محاسبه می‌شود:

$$\text{Utilization}_{CPU} = \text{بهره‌وری پردازنده} = \frac{\text{مفید}}{\text{مفید + غیرمفید}}$$

توجه: پردازش کد فرآیند (برنامه کاربر) توسط CPU **زمان مفید** و پردازش کد سیستم عامل توسط CPU جهت عمل تعویض متن **زمان غیرمفید** محسوب می‌شود.

توجه: سوئیچ‌کردن CPU بین فرآیندهای مختلف، تعویض متن (Context Switch) نام دارد. این عمل نیازمند ذخیره‌کردن حالت فرآیند قبلی و بارگذاری حالت فرآیند فعلی است. از آن‌جا که از نظر اجرایی، کار مفیدی صورت نمی‌گیرد، این عمل یک زمان سربار(غیرمفید) به حساب می‌آید.

توجه: اگر سیستم تک برنامه باشد، آنگاه انتظار برای عملیات IO هم **زمان غیرمفید** محسوب می‌شود.

توجه: عمل تعویض متن توسط بخشی از سیستم عامل به‌نام Dispatcher صورت می‌گیرد. در واقع Dispatcher ماژولی است که کنترلCPU را به فرآیندی که توسط زمانبند کوتاه‌مدت انتخاب شده است، اعطا می‌کند.

توجه: هر برنامه برای اینکه بتواند به یک برنامه در حال اجرا (فرآیند) تبدیل شود باید ابتدا در حافظه اصلی قرار گیرد، در واقع با قرار گرفتن برنامه در حافظه اصلی دستورالعمل‌های آن در دسترس CPU قرار می‌گیرد. اجرای یک دستورالعمل در سه مرحله صورت می‌گیرد.

۱- واکشی (Fetch): در این مرحله دستورالعملی در خانه‌ای از حافظه که توسط رجیستر PC (یا IP) به آن اشاره شده به رجیستر IR در پردازنده منتقل می‌شود.

۲- رمزگشایی (Decode): در این مرحله دستورالعمل موجود در IR تفسیر و آماده اجرا می‌شود.

۳- اجرا (Execute): در این مرحله دستورالعمل‌های رمزگشایی شده به اجرا درآمده و تأثیر آن‌ها بر روی رجیسترها اعمال می‌شود.

سه مرحله فوق برای تمام دستورالعمل‌های برنامه‌های اجرایی، انجام می‌شوند.

توجه: مطابق فرض سوال «**سیستم عامل**» و «**برنامه کاربر**» هر دو وجود دارد. و در محاسبه بهره‌وری جدا از هم نیستن.

صورت گزینه‌ها به صورت زیر است:

۱) **بهره‌وری CPU تحت هر شرایطی 100 درصد است؛ زیرا همواره الگوریتم فون نیومن اجرا می‌شود که شامل اجرای فرآیندها با سیستم عامل است.**

گزینه اول پاسخ سوال نیست، زیرا پردازش کد سیستم عامل توسط CPU جهت عمل تعویض متن **زمان غیرمفید** محسوب می‌شود.

۲) **چون طبق الگوریتم فون نیومن CPU مدام درگیر خواهد بود، در مواقعی که برنامه‌ای برای اجرا وجود ندارد و سیستم عامل کاری ندارد، CPU به وضعیت بیکار (Halt) می‌رود.**

گزینه دوم پاسخ سوال است، زیرا در وضعیت Halt فرآیند کاربر درحال اجرا نیست و سیستم عامل هم منتظر دریافت دستورات جدید است. و اصلا فرآیندی اجرا نمی‌شود که میزان بهره‌وری آن قرار باشد محاسبه شود. اگر زمان مفیدی باشد آنگاه زمان غیرمفید آن هم محاسبه می‌شود که نتیجه آن محاسبه بهره‌وری است. در صورتی که برنامه‌ای برای اجرا وجود نداشته باشد، پردازنده یک برنامه ساده که شامل یک حلقه انتظار مشغول است را اجرا می‌کند و در واقع در یک حلقه انتظار (وضعیت Halt)، منتظر ظهور کار جدید می‌ماند.

۳) **بهره‌وری CPU را نباید با اجرای سیستم عامل به صورت همزمان لحاظ کرد، چون سیستم عامل سربار ناچیزی دارد.**

گزینه سوم پاسخ سوال نیست، زیرا مطابق فرض سوال «**سیستم عامل**» و «**برنامه کاربر**» هر دو وجود دارد و در محاسبه بهره‌وری جدا از هم نیستن. پردازش کد فرآیند (برنامه کاربر) توسط CPU **زمان مفید** و پردازش کد سیستم عامل توسط CPU جهت عمل تعویض متن **زمان غیرمفید** محسوب می‌شود. عمل تعویض متن توسط سیستم عامل در واقعیت سربار دارد و درصورت سوال ناچیز فرض نشده است.

۴) **بهره‌وری CPU نباید شامل اجرای سیستم عامل گردد، لذا همیشه بهروری کمتر از 100 درصد است.**

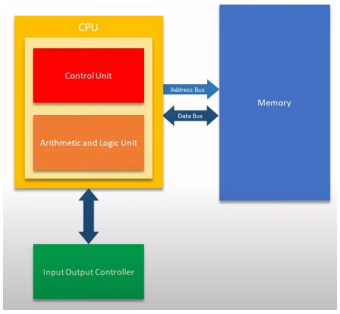
گزینه چهارم پاسخ سوال نیست، زیرا مطابق فرض سوال «**سیستم عامل**» و «**برنامه کاربر**» هر دو وجود دارد و در محاسبه بهره‌وری جدا از هم نیستن. پردازش کد فرآیند (برنامه کاربر) توسط CPU **زمان مفید** و پردازش کد سیستم عامل توسط CPU جهت عمل تعویض متن

زمان غیرمفید محسوب می‌شود.

به طور کلی معماری کامپیوترها به دو طبقه زیر تقسیم می‌شود:

معماری Von-Neumann

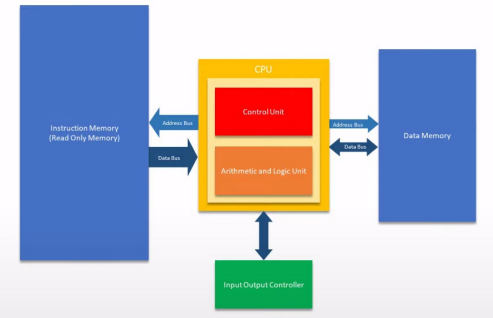
این ساختار طراحی به خاطر جان فون نویمان (دانشمند علوم رایانه‌ای) نامگذاری شده‌است.
در Von-Neumann گذرگاه آدرس Address Bus به طور مشترک برای Code و Data استفاده می‌شود.



توجه: در Von-Neumann حافظه Code و Data باهم مشترک هستند.
توجه: در Von-Neumann نوع حافظه Code و Data فقط RAM است.
توجه: بیشتر در سیستم‌های general purpose کاربرد دارد.

معماری Harvard

در Harvard گذرگاه آدرس Address Bus برای Code Memory و Data Memory از هم جدا هستند.



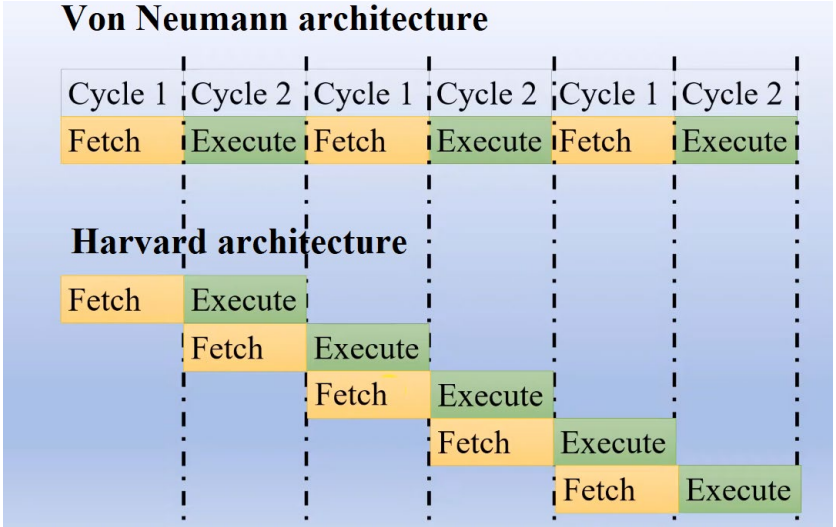
توجه: در Harvard حافظه Code و Data از هم جدا هستند.

توجه: در Harvard نوع حافظه Code از جنس ROM و نوع حافظه Data از جنس RAM است.
توجه: بیشتر در سیستم‌های embedded کاربرد دارد.

جدول زیر تفاوت بین معماری Von-Neumann و Harvard را بیان کرده‌است:

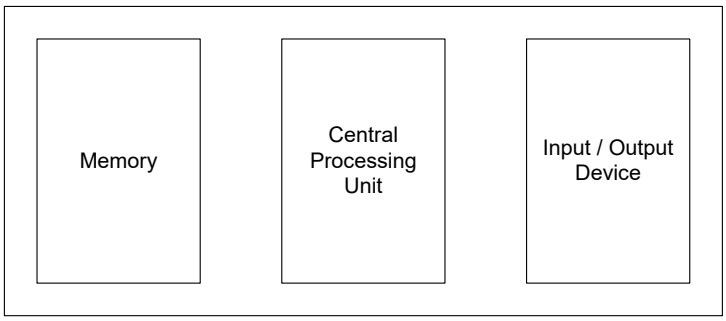
Parameters	Von Neumann	Harvard
Memory	❖ Data and Program {Code} are stored in same memory	❖ Data and Program {Code} are stored in different memory
Memory Type	❖ It has only RAM for Data & Code	❖ It has RAM for Data and ROM for Code
Buses	❖ Common bus for Address & Data/Code	❖ Separate Bus Address & Data/Code
Program Execution	❖ Code is executed serially and takes more cycles	❖ Code is executed in parallel with data so it takes less cycles.
Data/Code Transfer	❖ Data or Code in one cycle	❖ Data and Code in One cycle
Control Signals	❖ Less	❖ More
Space	❖ It needs less Space	❖ It needs more space
Cost	❖ Less	❖ Costly

توجه: مطابق شکل زیر واضح است که در Von-Neumann در هر Cycle فقط یک Fetch یا Execute انجام می‌شود. و به تبع Fetch یا Execute به صورت سریالی (ترتیبی) اجرا می‌شود. اما در Harvard در هر Cycle یک Fetch و Execute باهم انجام می‌شود. و به تبع Fetch از Code memory و Execute در Data Memory به صورت موازی اجرا می‌شود. چون در Harvard خط گذرگاه آدرس Address Bus برای Code Memory و Data Memory از هم جدا هستند. که منجر به انجام دستورات بیشتری در واحد زمان می‌شود. در واقع



معماری سیستم کامپیوتری

یک سیستم کامپیوتری از سه بخش اصلی تشکیل شده است، که عبارتند از: حافظه (Memory) ، دستگاه‌های ورودی و خروجی (IO Devices) و پردازشگر مرکزی (Central Processing Unit: CPU). هر کدام از این سه بخش وظایف مشخصی را در سیستم برعهده دارند که در ادامه به شرح هرکدام خواهیم پرداخت.



اجزای کلی یک سیستم کامپیوتری

۱. **حافظه (Memory):** برای ذخیره‌سازی اطلاعات به کار می‌رود. اطلاعاتی که در یک سیستم کامپیوتری مورد استفاده قرار می‌گیرد به دو بخش قابل تقسیم است: اطلاعات ثابت و اطلاعات متغیر.

اطلاعات ثابت: در هر سیستم کامپیوتری برنامه‌هایی وجود دارد که عمل سیستم را مشخص نموده، چگونگی انجام کارها و مراحل مختلف آنرا معلوم می‌سازد. این برنامه‌ها از ابتدای شروع به کار سیستم مورد استفاده قرار می‌گیرند و عملکرد مرحله به مرحله وظایف سیستم را مشخص می‌سازند. این نوع اطلاعات نوعی از برنامه‌های سیستمی هم هستند. به عنوان مثال یک ماشین حساب، از لحظه شروع به کار برنامه‌ای را اجرا می‌کند که پس از آماده‌سازی اولیه ماشین حساب، این امکان را به کاربر می‌دهد که بتواند از طریق صفحه کلید اعداد مورد نظر را به همراه عملگر دلخواه وارد نموده، ضمن نمایش کلیدهای فشرده شده روی صفحه نمایش، پس از فشردن کلید = یا Enter نتیجه محاسبات را روی صفحه نمایش نشان دهد و آماده گرفتن داده‌های جدید شود. اصول این عملیات در مورد هر سیستم کامپیوتری دیگری نیز صادق است. نوع دیگر اطلاعات ثابت، شامل مقادیری است که سیستم کامپیوتری در حین انجام عملیات، از آنها استفاده می‌کند. به عنوان مثال در ماشین حساب برای محاسبه سینوس یک زاویه یا محاسبه لگاریتم یک عدد، نیازمند جداولی هستیم که مقادیر سینوس زوایای مختلف و یا لگاریتم اعداد را درون خود داشته باشد. مسلماً لازم است این گونه اطلاعات درون یک حافظه قرار می‌گیرد که پس از خاموش شدن سیستم از بین نرود. به

عبارت دیگر این نوع اطلاعات در حافظه‌ای به صورت ثابت جای خواهد گرفت. دقت داشته باشید کاربرد این دو نوع اطلاعات با یکدیگر متفاوت است اما جاییکه برای ذخیره آنها استفاده می‌شود حافظه‌ای خواهد بود که اطلاعات موجود در آن تنها خوانده می‌شود و نیازی به نوشتن مجدد آن نیست، ضمن اینکه نباید با خاموش شدن سیستم یا به هر طریق دیگر اطلاعات آن عوض شده و یا از بین برود. چنین حافظه‌هایی را حافظه‌های فقط خواندنی (Read Only Memory: ROM) می‌نامند.

اطلاعات متغیر: در یک سیستم کامپیوتری برخی داده‌ها را از ورودی سیستم دریافت می‌نماییم و لازم است تا پس از دریافت، آنها را ذخیره نماییم، مانند کلیدهای فشرده شده در یک ماشین حساب که اعداد و یا عملیات مرتبط به آن کلیدها در حافظه به صورت موقت ذخیره می‌شود. ضمن اینکه در سیستم‌های کامپیوتری برای ذخیره‌سازی داده‌های میان محاسباتی (داده‌هایی که در حین محاسبات به وجود می‌آیند و برای محاسبه نتیجه کامل، باید به صورت موقت ذخیره شوند) از حافظه‌های موقت استفاده می‌کنیم. این داده‌های موقت باید در حافظه‌هایی ذخیره شوند که قابلیت نوشتن و خوانده شدن را دارا باشند. علاوه بر این دو، برخی از برنامه‌ها وجود دارند که به برنامه‌های کاربردی معروفند و برخی سیستم‌های کامپیوتری اجازه می‌دهند تا کاربر، این برنامه‌ها را به سیستم وارد نماید، و پس از ذخیره شدن در حافظه‌ای به صورت موقت، امکان اجرای آنها به وجود آید. نمونه بارز این برنامه‌ها را در ماشین حساب‌های مهندسی و یا کامپیوترهای شخصی مثل برنامه فتوشاپ می‌توان دید. حافظه‌هایی که برای این منظور استفاده می‌شوند، حافظه‌های خواندنی و نوشتنی (Read / Write Memory: RWM) گویند. گرچه امروزه در تمامی کاربردها این حافظه‌ها را به نام RAM می‌شناسیم.

۲. **دستگاه‌های ورودی و خروجی (Input / Output Devices)**: در سیستم‌های کامپیوتری، دریچه ارتباطی سیستم با دنیای خارج از سیستم، دستگاه‌های ورودی و خروجی هستند. به عنون مثال صفحه کلید در ماشین حساب و کامپیوترهای شخصی و یا صفحه نمایش و مانیتور.

۳. **پردازشگر مرکزی (Central Processing Unit - CPU)**: وظیفه اصلی انجام عملیات محاسباتی و پردازشی، دادن فرامین کنترلی به واحدهای حافظه و IO و هماهنگی میان واحدهای مختلف یک سیستم کامپیوتری توسط این واحد انجام شده و یا فرمان انجام آن صادر می‌شود. به عنوان مثال در یک ماشین حساب، خوانده شدن صفحه کلید، انجام محاسبات ریاضی و منطقی مورد نیاز، نمایش اطلاعات برروی صفحه نمایش تماماً بر عهده

CPU است. آنچه مهم است که به آن توجه نماییم این است که CPU تنها بخشی است که می‌تواند برنامه‌های داده شده را اجرا نماید. تمام کارهایی که در مثال ماشین حساب، باید CPU انجام دهد را طراح سیستم، توسط برنامه‌ای در اختیار CPU قرار داده است و وظیفه اصلی این واحد را می‌توان به صورت دقیق‌تر، خواندن این برنامه و انجام تک تک دستورات آن در نظر گرفت.

اکنون که با کارهای کلی تک‌تک بخش‌های یک سیستم کامپیوتری آشنا شدیم، می‌توانیم چگونگی ارتباط این قسمت‌ها را به صورت دقیق‌تری بررسی نماییم.

برای ارتباط میان این اجزا از خطوط ارتباطی که به آنها باس (Bus) می‌گوییم استفاده می‌شود. وظیفه هر یک از این خطوط ارتباطی به صورت زیر است:

۱. **باس داده (Data Bus)**: برای آنکه امکان تبادل اطلاعات میان حافظه و پردازشگر مرکزی و یا دستگاه‌های ورودی و خروجی و پردازشگر مرکزی وجود داشته باشد، باید خطوطی را میان پردازشگر، حافظه و دستگاه‌های ورودی و خروجی متصل نمود تا از طریق آنها پردازشگر بتواند اطلاعات مورد نیاز خود را از حافظه یا دستگاه ورودی بخواند و یا در صورت نیاز، برخی داده‌ها را درون حافظه و یا دستگاه خروجی بنویسد. این خطوط را باس داده (Data Bus) می‌نامند.

باس داده دارای مشخصاتی است که در زیر به آن اشاره می‌شود:

الف‌ـ باس داده موازی است. یعنی اطلاعاتی که از طریق باس داده، ارسال یا دریافت می‌شود به صورت موازی است. در واقع برای هر بیت از اطلاعات، یک خط وجود دارد و همزمان می‌توان تعدادی بیت را ارسال یا دریافت کرد. علت اصلی این مسئله، سرعت است. اگر باس داده به صورت موازی استفاده نشود، اطلاعات به صورت بیت به بیت و ترتیبی ارسال یا دریافت می‌شود وبه تبع، زمان بسیار بیشتری صرف خواهد شد. که در آن صورت، کارایی و سرعت سیستم کامپیوتری کاهش می‌یابد. البته در بعضی از سیستم‌ها برای خواندن برخی از دستگاه‌های ورودی و خروجی ممکن است از روش سریال استفاده شود، اما به هرحال باس اصلی داده در این سیستم‌ها موازی خواهد بود.

ب‌ـ باس داده دارای وضعیت سوم (High Impedance) است. یعنی در برخی از مواقع که لازم است تا پردازنده خود را از باس داده به صورت موقت جدا کند. روش کلی کار به این صورت است که هر خط باس داده دارای سه وضعیت است، حالت منطقی 0 ، حالت منطقی

1 و حالت امپدانس زیاد یا High Impedance. اگر خط در وضعیت High Impedance قرار بگیرد، اتصال بخش‌های داخلی پردازنده با خطوط خارجی بوسیله یک مقاومت بسیار زیاد صورت می‌پذیرد، از آنجا که مقاومت مذکور در حد چندین مگا اهم است، لذا عملاً اتصال الکتریکی میان بخش‌های داخلی و خطوط خارجی قطع می‌شود و گویی هیچ اتصالی میان بخش‌های داخلی پردازنده با باس داده وجود ندارد. به این حالت وضعیت سوم گویند و باس داده را سه وضعیتی (Three-state) می‌نامند.

ج- باس داده دوطرفه (Bidirectional) است. از آنجا که باس داده هم برای خواندن اطلاعات از حافظه و IO و هم برای نوشتن اطلاعات درون حافظه و IO استفاده می‌شود، بنابراین باس داده باید بتواند به صورت دو منظوره استفاده شود یا به صورتی دیگر باید دوطرفه Bidirectional باشد.

۲. **باس آدرس (Address Bus)**: حافظه‌ای که توسط پردازنده استفاده می‌شود دارای سلول‌های متعددی است که در هرکدام داده مناسبی ذخیره می‌شود. بنابراین برای دسترسی به هریک از اطلاعات مذکور لازم است تا سلول مورد نظر انتخاب شود تا اطلاعات از سلول مشخص شده خوانده و یا درون آن نوشته شود. برای انتخاب کردن یک سلول حافظه از آدرس استفاده می‌شود. آدرس یک عدد باینری است که مشخص کننده مکانی از حافظه است که در آن داده مورد نظر وجود دارد و یا قرار است در آن نوشته شود و لازم است در هنگام استفاده از حافظه، این آدرس برای انتخاب سلول مورد نظر از پردازنده به حافظه ارسال شود. از طرف دیگر دستگاه‌های ورودی و خروجی که در سیستم وجود دارند، نیز متعدد بوده و باید مشخص شود که کدامیک از آنها مورد توجه پردازنده است، برای این منظور به هر یک از دستگاه‌های ورودی و خروجی آدرسی مشخص اختصاص داده می‌شود و پردازنده با ارسال آدرس مناسب، یکی از آنها را انتخاب می‌کند. با توجه به این مسائل، خطوطی را برای ارسال عدد باینری آدرس در نظر می‌گیرند که به آنها باس آدرس Address Bus می‌گویند.

باس آدرس دارای مشخصاتی است که در زیر به آن اشاره می‌شود:

الف- باس آدرس موازی است. علت موازی بودن باس آدرس نیز همانند باس داده است.

ب- باس آدرس دارای وضعیت سوم است.

ج- باس آدرس خروجی است. نکته‌ای که لازم است در اینجا به آن اشاره نماییم، این است که ورودی یا خروجی بودن خطوط نسبت به پردازنده سنجیده می‌شود. یعنی اگر می‌گوییم که باس آدرس خروجی است به این معناست که اطلاعات آدرس (عدد باینری آدرس) از طرف پردازنده صادر می‌شود. علت خروجی بودن باس آدرس این است که پردازنده وظیفه اصلی فرمان دادن و کنترل کردن حافظه و دستگاه‌های ورودی و خروجی را برعهده دارد، لذا مشخص کردن آدرس سلول حافظه و یا شماره دستگاه ورودی و خروجی نیز بر عهده پردازنده است و نتیجه اینکه آدرس از پردازنده به حافظه یا IO منتقل می‌گردد.

۳. **باس کنترل (Control Bus)**: علاوه بر دو باس آدرس و داده، در سیستم کامپیوتری خطوطی نیاز است تا بتواند ارتباط کنترلی میان پردازنده و حافظه و یا پردازنده و دستگاه‌های ورودی و خروجی را برقرار کند. تصور کنید که قصد داریم اطلاعاتی را درون حافظه بنویسیم، بنابراین باید مشخص شود که در چه سلولی از حافظه باید اطلاعات قرار بگیرد، این کار از طریق آدرس مشخص شده از طریق باس آدرس صورت می‌گیرد و سپس اطلاعات از طریق باس داده به حافظه ارسال خواهد شد. اما حافظه چگونه متوجه می‌شود که با آمدن آدرس نوشتن، باید اطلاعات روی باس داده را دریافت کند و درون سلول مشخص شده بنویسد (عمل نوشتن درون حافظه) و یا اینکه با آمدن آدرس خواندن، باید اطلاعات درون سلول تعیین شده را بر روی باس داده قرار دهد (عمل خواندن از حافظه). بنابراین در هنگام استفاده از حافظه، **فرمان خواندن یا نوشتن** را نیز به حافظه انتقال می‌دهیم. این موضوع برای دستگاه‌های ورودی و خروجی نیز کاربرد دارد. بدین منظور در کنار باس داده و آدرس، خطوط دیگری نیز استفاده می‌شود که به خطوط کنترلی یا باس کنترلی موسومند.

دسته‌بندی خطوط کنترلی

خطوط کنترلی به دو نوع ورودی و خروجی تقسیم می‌شود:

الف- خطوط کنترل خروجی: این خطوط از طرف پردازنده صادر شده و به قسمت‌های دیگر سیستم کامپیوتری منتقل می‌شوند. این خطوط خروجی به دو منظور در یک پردازنده استفاده می‌شوند که عبارتند از:

الف-۱- صدور فرمان: همانگونه که در بخش قبل عنوان شد، خطوط کنترلی را نیاز داریم که وظیفه آنها ارسال فرامین کنترلی پردازنده به حافظه یا دستگاه‌های ورودی و خروجی باشد. در واقع انتخاب حافظه یا IO ، دستور خواندن و یا نوشتن و دیگر فرامین مورد نیاز یک سیستم کامپیوتری، توسط این خطوط به بخش‌های مختلف سیستم منتقل می‌شود.

الف_۲_اعلام وضعیت: در برخی موارد باید پردازنده وضعیت خود را اعلام نماید. به عنوان مثال در مقابل درخواستی از دستگاه ورودی و خروجی، پردازنده لازم است پذیرش درخواست را به دستگاه اطلاع دهد. در این موارد نیز خطوط کنترلی مشخصی در پردازنده‌ها تعبیه می‌شود تا این نیاز را برآورده نماید.

ب_خطوط کنترلی ورودی: این خطوط برای برآورده کردن دو مقصود مختلف استفاده می‌شود.

ب_۱_ دریافت درخواست IO: برای ارتباط مؤثر میان پردازنده و دستگاه‌های ورودی و خروجی، طراحان امکان درخواست دستگاه‌های ورودی و خروجی را از پردازنده، توسط خطوط کنترلی مشخصی ایجاد می‌نمایند.

ب_۲_ دریافت وضعیت: گاهی برای ارتباط صحیح میان پردازنده و دستگاه ورودی و خروجی نیاز است پردازنده را از وضعیت دستگاه آگاه نماییم تا ارتباط میان پردازنده و IO زمانی برقرار شود که دستگاه آمادگی آنرا داشته باشد. بنابراین از خطوط کنترلی خاصی برای انتقال وضعیت IO به پردازنده استفاده می‌کنیم. پردازنده با بررسی مناسب این خطوط، ارتباط خود را با IO تنظیم می‌کند. کاربرد دیگری که برای این خطوط وجود دارد هماهنگی سرعت دریافت یا ارسال اطلاعات میان پردازنده و حافظه یا پردازنده و IO است. بدین طریق پردازنده سرعت دریافت و یا ارسال داده را با توجه به وضعیت حافظه وIO تنظیم می‌کند.

تست‌های فصل اول: مفاهیم اولیه

۱۰۷- در چه صورتی یک فرآیند فرزند که Zombie شده است، تبدیل به یک فرآیند Orphan (یتیم) می‌شود؟

(مهندسی کامپیوتر - دولتی ۱۴۰۱)

۱) در صورتی که فرآیند پدر، دستور (terminate) را برای فرآیند فرزند اجرا نکرده باشد.
۲) در صورتی که فرآیند پدر برای فرآیند فرزند، دستور (wait) را اجرا نکرده باشد.
۳) چنین حالتی هیچ‌گاه در سیستم عامل رخ نمی‌دهد.
۴) درصورتی که فرآیند فرزند دچار بن بست شود.

عنوان کتاب: سیستم عامل

مولف: استاد خلیلی‌فر

ناشر: انتشارات راهیان ارشد و دکتری

آدرس سایت انتشارات بابان: BABAN.IR

پاسخ تست‌های فصل اول: مفاهیم اولیه

۱۰۷- گزینه (۲) صحیح است.

فرآیند فرزند توسط دستور fork متولد می‌شود. یک فرآیند فرزند، زمانی خاتمه می‌یابد که کامل اجرا شود. انجام خاتمه نهایی فرآیند فرزند توسط فرآیند پدر انجام می‌شود. فرآیند والد جهت خاتمه نهایی فرآیند فرزند باید از هویت فرزند آگاه باشد. تا بداند کدام فرزند را و در چه وضعیتی خاتمه دهد.

دستوراتی که توسط فرآیند پدر اجرا می‌شود، به صورت زیر است:

```
pid t pid;
int status;
pid = wait(&status);
```

توجه: فرآیند پدر با استفاده از فراخوان سیستمی ()wait منتظر تکمیل فرآیند فرزند می‌ماند، در واقع فرآیند پدر منتظر می‌ماند تا کار فرآیند فرزند تمام شود. هنگامی که فرآیند فرزند تکمیل شد، فرآیند پدر از جاییکه فراخوان سیستمی ()wait را فراخوانی کرده است، شروع به ادامه کار می‌کند.

توجه: فراخوان سیستمی ()wait مقدار Process id فرآیند فرزندی که پایان یافته است را در خروجی بر می‌گرداند.

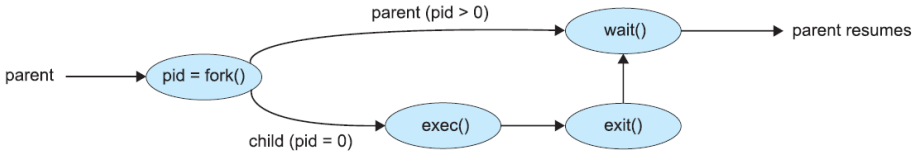
توجه: فرآیند فرزند از طریق فراخوان سیستمی ()wait می‌تواند با فرآیند پدرش ارتباط برقرار کند، به عبارت دیگر مقدار Process id فرآیند فرزند توسط فراخوان سیستمی ()wait به فرآیند پدرش پاس داده می‌شود.

قطعه کد فوق داخل فرآیند پدر قرار دارد. فرآیند والد توسط تابع wait منتظر خاتمه فرآیند فرزند می‌ماند.

بعد از اینکه فرآیند فرزند کار خودش را تمام کرد و منابع مورد نیازش را هم آزاد کرد هنوز نمی‌تواند به طور کامل از سیستم خارج بشود. چون پدری دارد که شماره شناسایی فرزند داخل PCB پدر هنوز هست همچنین شماره فرآیند فرزند همچنان داخل جدول فرآیندها قرار دارد. اینجا جایی هست که فرآیند فرزند Zombie شده است یعنی روح ندارد و فقط جسم و اطلاعات هویتی آن مانده است همه فرآیندهای فرزند به طور موقت بعد اتمام Zombie می‌شوند.

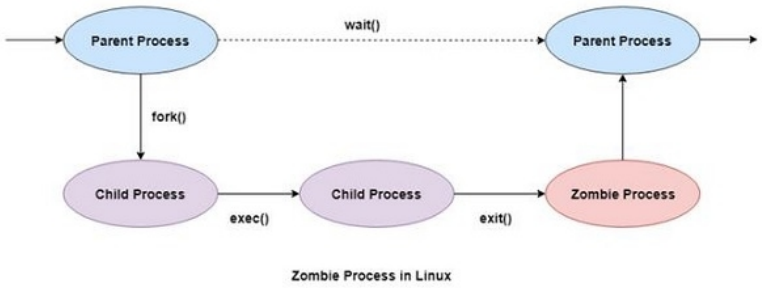
فرزند zombie شده باید اطلاعات هویتی (status و pid) خودش را به پدر ارسال کند. فرزند zombie شده با status که یک اشاره‌گر هست تمام شدن خودش را به والد اعلام می‌کند، پس خاتمه نهایی هر فرزندی توسط پدر اعلام انجام می‌شود. توسط نام تابع pid = wait(&status) مقدار pid فرآیند فرزند به پدر منتقل می‌شود تا فرآیند پدر بداند کدام فرزندش تمام شده است تا

PCB خود را تنظیم کند و در نهایت پدر؛ فرآیند فرزند خود را به طور کامل خاتمه می‌دهد. شکل زیر گویای مطلب است:



اگر پدر نباشد که کارهای خروج فرزند را انجام بدهد به فرآیند zombie شده فرآیند orphan یا یتیم می‌گویند. در این شرایط هر یک مدت یکبار پدر پدرها ریشه ریشه‌ها یعنی init برای فرآیندهای zombie پدری می‌کند و آنها را خاتمه می‌دهد.

نموداری که ایجاد و خاتمه یک فرآیند زامبی را نشان می‌دهد به شرح زیر است:



Zombie Process in Linux

توجه: فرآیندهای زامبی از هیچ منبع سیستمی استفاده نمی‌کنند اما شناسه فرآیند خود را حفظ می کنند.

تست‌های فصل پنجم: مدیریت حافظه مجازی

۹۴- دستور ()Fork موجب ایجاد یک فرآیند جدید فرزند با خط اجرای یک گام بعدتر و با خروجی تابع صفر برای فرزند می‌شود. مقدار تابع فراخوانی شده در فرآیند پدر غیرصفرخواهد بود. در تکه برنامه زیر، پس از اجرای کامل، چندبار پیام «hello» چاپ می‌شود؟ (فرض کنید بهینه‌سازی کامپایلر خاموش است و همه بخش‌ها در یک شرط if اجرا می‌شوند.) **(مهندسی IT - دولتی ۱۴۰۱)**

```
⋮
if ( fork () & & fork () )
    fork ();
if ( fork () || fork () )
    fork ();
Print ("hello");
⋮
```

۱) 35
۲) 64
۳) 25
۴) 49

عنوان کتاب: سیستم عامل

مولف: استاد خلیلی‌فر

ناشر: انتشارات راهیان ارشد و دکتری

آدرس سایت انتشارات بابان: BABAN.IR

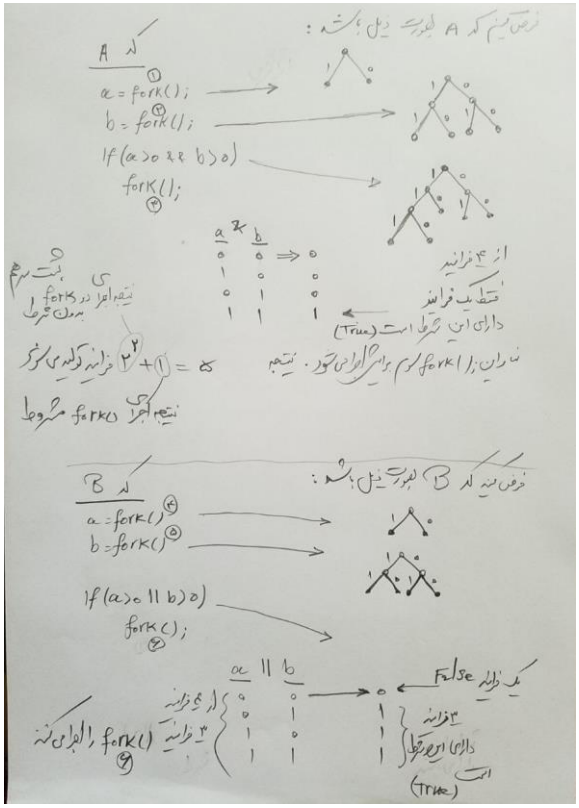
پاسخ تست‌های فصل پنجم: مدیریت حافظه مجازی

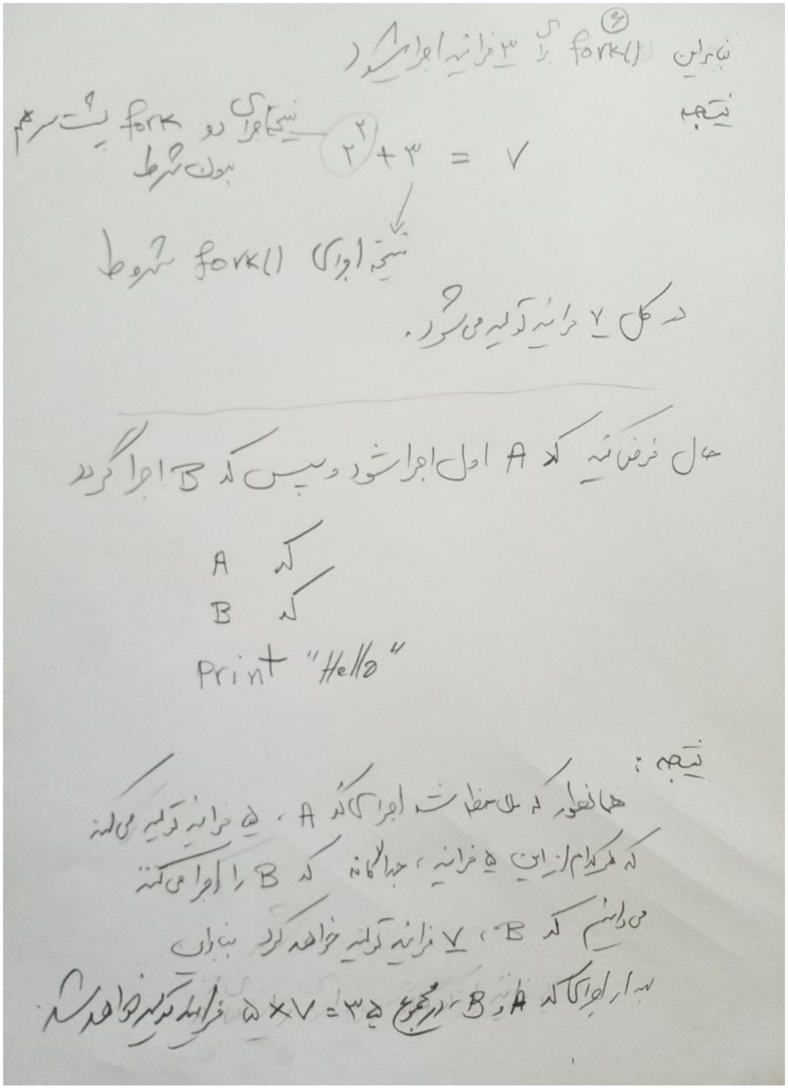
۹۴- گزینه (۱) صحیح است.

جهت مشاهده حل به صفحه آپارات زیر مراجعه نمایید:

https://www.aparat.com/khalilifar.ir

https://www.aparat.com/v/HgbLA





بنابراین fork() ⑥ ها ۳ فرآیند اجرا می‌کند

نتیجه

$2^2 + 3 = 7$

نتیجه اجرای دو fork پشت سرهم بدون شرط ← 2^2

۳ ← نتیجه اجرای fork() شروط

در کل ۷ فرآیند تولید می‌شود.

حال فرض کنید که A اول اجرا شود و سپس کد B اجرا گردد

کد A

کد B

Print "Hello"

نتیجه:

همانطور که ملاحظه شد اجرای کد A، ۵ فرآیند تولید می‌کند که هرکدام از این ۵ فرآیند، جداگانه کد B را اجرا می‌کنند

می‌دانیم کد B، ۷ فرآیند تولید خواهد کرد بنابراین

با اجرای کد A و B، در مجموع ۳۵ = ۷ × ۵ فرآیند تولید خواهد شد

تست‌های فصل چهارم: مدیریت حافظه اصلی

۹۵- کدام مورد جزو مزایای استفاده از فایل‌های DLL یا تکنیک Shared Library محسوب نمی‌شود؟ **(مهندسی *IT* - دولتی ۱۴۰۱)**

۱) کاهش اندازه فایل اجرایی
۲) ایجاد اشتراک بین چند فرآیند
۳) امنیت بالاتر
۴) صرفه‌جویی در مصرف حافظه

عنوان کتاب: سیستم عامل

مولف: استاد خلیلی‌فر

ناشر: انتشارات راهیان ارشد و دکتری

آدرس سایت گروه بابان: BABAN.IR

پاسخ تست‌های فصل چهارم: مدیریت حافظه اصلی

۹۵- گزینه (۳) صحیح است.

مطابق اصول مهندسی نرم‌افزار و ساخت برنامه‌های کامپیوتری، پیمانه‌ای کردن (مولفه‌سازی) نرم‌افزار منجر به خوانایی بالاتر برنامه و کنترل بهتر برنامه در حال ساخت می‌گردد. پیمانه‌ای‌کردن همچنین این خاصیت را فراهم می‌آورد تا در صورتی که پیمانه‌هایی با رعایت **اصول طراحی معماری ایده‌آل** ایجاد گردند، این پیمانه‌ها هم در نرم‌افزار فعلی دارای خاصیت **نگهداری آسان** باشند و هم در نرم‌افزارهای آتی خاصیت **قابلیت استفاده مجدد** را دارا باشند تا ساخت پروژه‌های نرم‌افزاری آتی با صرفه جویی در هزینه‌ها مقرون به صرفه گردد. زمان و هزینه‌ی فرآیند تولید نرم‌افزار با استفاده از قطعات آماده (قابلیت استفاده مجدد) کاهش چشمگیری دارد. در واقع یک قطعه با قابلیت استفاده‌ی مجدد یک بار ساخته می‌شود و بارها و بارها استفاده می‌شود و این یعنی سودآوری!

مؤلفه یک قطعه‌ی آماده، کاربردی و پیاده‌سازی شده است که دارای واسط لازم جهت اتصال با سایر قطعات و استقرار در بخشی از یک سیستم عملیاتی می‌باشد.

در برنامه‌نویسی به مؤلفه، پیمانه نیز گفته می‌شود. در حیطه‌ی مهندسی نرم‌افزار، **واحد مولفه**، یک قطعه عملیاتی مبتنی بر تابع است که بر دو نوع می‌باشد:

۱) تابع کاربردی: مانند تابع جمع که برنامه‌نویس آن را می‌نویسد. اگر تابع کاربردی، شرایط قابل حمل بودن (مانند تعریف متغیر درون تابع به صورت‌محلی و صریح و عدم استفاده از متغیرهای سراسری) را داشته باشد می‌تواند به عنوان یک قطعه‌ی آماده‌ی قابل استفاده‌ی مجدد در پروژه‌های بعدی مورد استفاده مجدد قرار گیرد.

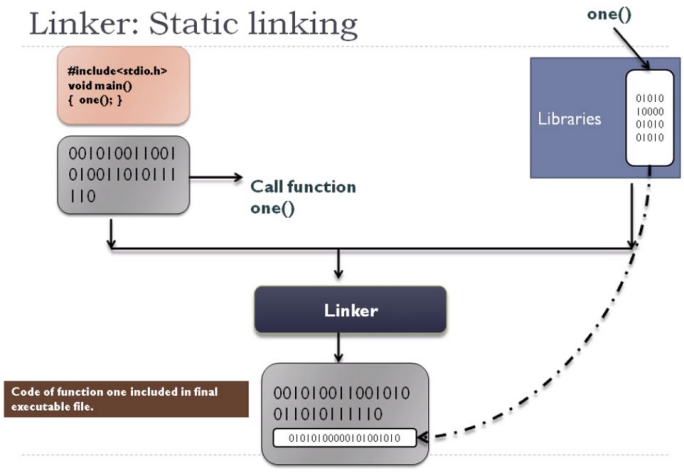
۲) تابع سیستمی: مانند تابع سیستمی که کامپایلر یا سیستم عامل تعاریف آن را فراهم می‌کند و می‌تواند به عنوان یک قطعه‌ی آماده و قابل استفاده‌ی مجدد، در برنامه‌ها مورد استفاده مجدد قرار گیرد. این توابع سیستمی به دو صورت **کتابخانه ایستا** (Static Library) برای مثال با پسوند Lib. (Library) در سیستم عامل ویندوز و **کتابخانه پویا** (Dynamic Library) برای مثال با پسوند DLL. (Dynamic link library) در سیستم عامل ویندوز وجود دارد.

یک کتابخانه مجموعه‌ای از کدهای قابل استفاده مجدد است. کتابخانه‌ها، برنامه‌های مستقلی هستند که یک برنامه‌نویس یا یک برنامه دیگر می‌تواند بارها از آن‌ها استفاده کند. فایل‌های DLL نوعی از این قبیل کتابخانه‌ها هستند. فایل DLL شامل کارکردهایی است که دیگر برنامه‌ها در هنگام نیاز از آن استفاده می‌کنند.

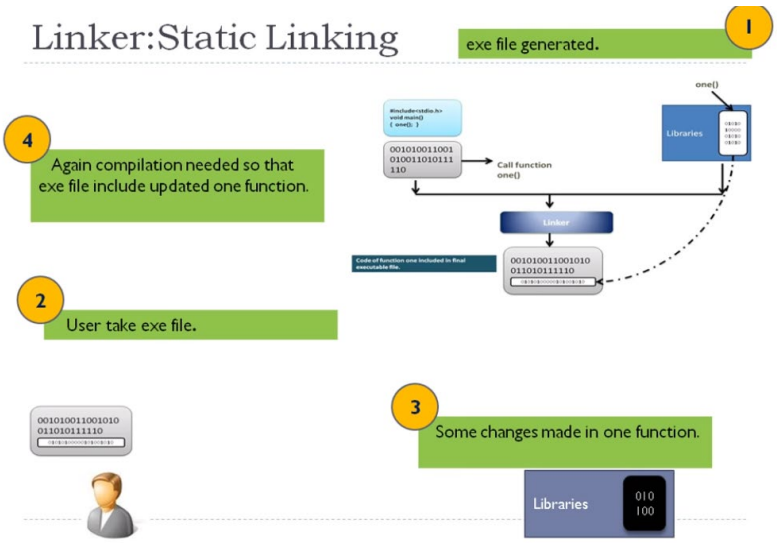
برای مثال وقتی از واژه‌پرداز word استفاده می‌کنید، گاهی می‌خواهید صفحه‌ای را چاپ کنید، اما برنامه word نمی‌داند چگونه عمل چاپ را به انجام برساند. در این صورت برنامه باید دستورالعمل‌های این کار را از یک کتابخانه فراخوانی کند.

این مورد، همان جایی است که کتابخانه‌ها به کار می‌آیند. این کتابخانه‌ها، در هر زمان که واژه‌پرداز لازم می‌بیند، تمام کدهایی که برنامه برای عمل چاپ گرفتن نیازمند آن‌ها است، فراهم می‌آورد. این تعریف از کتابخانه از چارچوب برنامه‌نویسی پیمانه‌ای (ماژولار) می‌آید. برنامه‌نویسی ماژولار، مفهومی در توسعه نرم‌افزار است که در آن یک برنامه به زیربرنامه‌های مستقلی تقسیم می‌شود که هر کدام قابلیت اجرای جداگانه دارند.

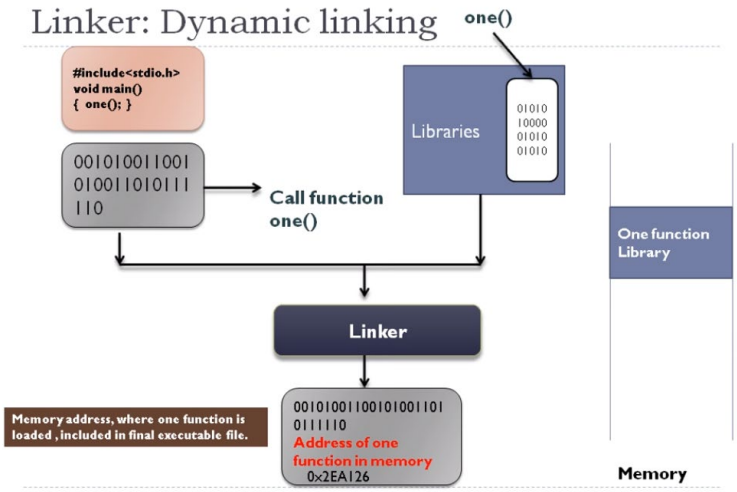
توجه: در Static linking قطعه کد تابع مورد نیاز از **کتابخانه ایستا** (Static Library) داخل فایل اجرایی exe. برنامه اصلی قرار می‌گیرد. بنابراین قطعه کد تابع مورد نیاز به ازای هر برنامه مستقل حافظه مصرف می‌کند. شکل زیر گویای مطلب است:



توجه: در Static linking قطعه کد تابع مورد نیاز از **کتابخانه ایستا** (Static Library) داخل فایل اجرایی exe. برنامه اصلی (1) قرار می‌گیرد. سپس فایل اجرایی در اختیار کاربر نهایی قرار می‌گیرد (2) ، حال اگر قطعه کد مورد نیاز از کتابخانه ایستا تغییر کند و به روز شود (3) ، آنگاه نیاز است برنامه اصلی و قطعه کد مورد نیاز به روز شده از کتابخانه ایستا مجدد کامپایل شود و نسخه exe. مجدد ایجاد شود (4). شکل زیر گویای مطلب است:

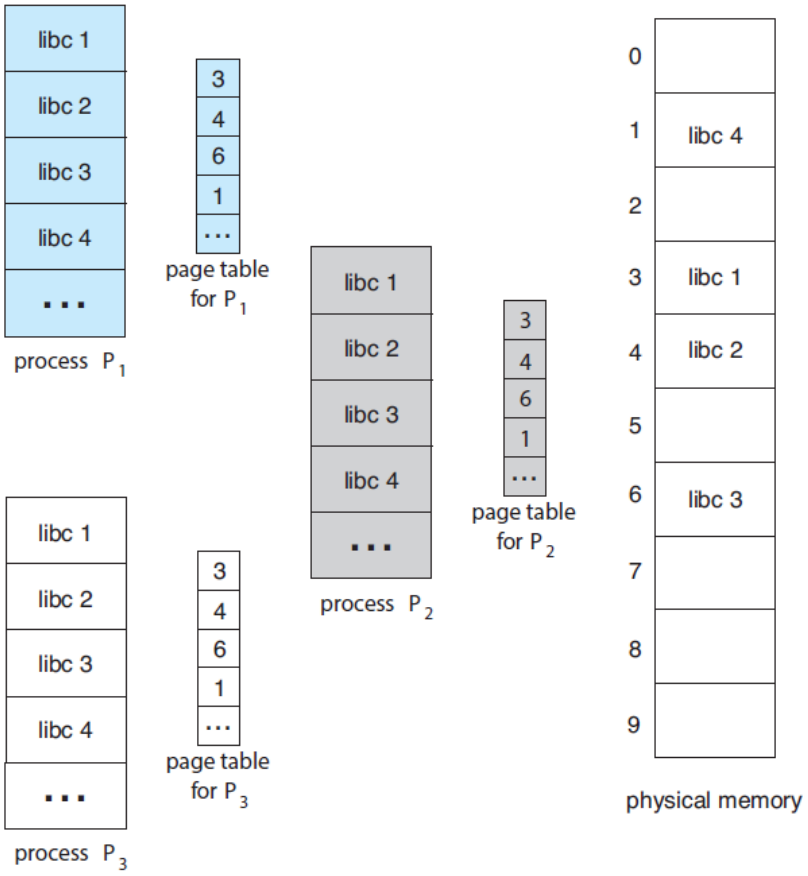


توجه: در Dynamic linking قطعه کد تابع مورد نیاز از **کتابخانه پویا** (Dynamic Library) داخل فایل اجرایی exe. برنامه اصلی فقط **آدرس‌دهی** می‌شود و خود قطعه کد داخل فایل اصلی قرار نمی‌گیرد. دقت کنید که طول فایل اجرایی نهایی (فایل اصلی بعلاوه تابع کتابخانه) کوچک نمی‌شود، بلکه به قطعه کد تابع مورد نیاز از کتابخانه پویا آدرس‌دهی می‌شود. همچنین تغییرات به روز شده در تابع کتابخانه توسط آدرس‌دهی به راحتی قابل دسترسی است. عدم استفاده مجدد قطعه کد تابع مورد نیاز از کتابخانه پویا در یک برنامه نهایی و فقط اکتفا کردن به آدرس‌دهی منجر به صرفه‌جویی در مصرف حافظه می‌شود. شکل زیر گویای مطلب است:

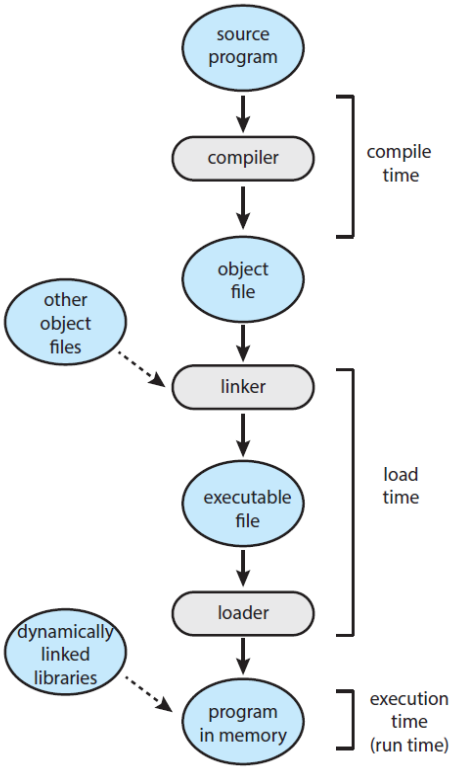


توجه: هدف **کتابخانه پویا** (Dynamic Library) بـرای مثـال بـا پسـوند DLL. (Dynamic link library) در سیستم عامل ویندوز به عنوان **کتابخانه مشترک** یا **Shared Library** تامین و افزایش امنیت نیست. تامین امنیت در سطوح مختلف بر عهده سخت‌افزار و سیستم عامل و در محیط‌هـای پایگاه داده بر عهده DBMS است.

توجه: ایجاد اشتراک قطعه کد تابع مورد نیاز از کتابخانه پویا بین چند فرآیند منجر به صرفه‌جویی در مصرف حافظه می‌شود که از مزیت‌های **کتابخانه پویا** (Dynamic Library) محسوب می‌شود. شکل زیر گویای مطلب است:



توجه: در یک نمای کلی مراحل ساخت یک برنامه اجرایی نهایی به صورت زیر است:


source program
compiler
compile time
object file
other object files
linker
load time
executable file
loader
dynamically linked libraries
program in memory
execution time (run time)

تست‌های فصل دوم: مدیریت فرآیندها و زمان‌بندی پردازنده

۹۶- زمان ورود و مدت زمان پردازش برای چهار فرآیند به صورت زیر است. برای زمان‌بندی پردازنده از الگوریتم نوبتی گردشی (round robin) با کوانتوم زمانی 4 میلی‌ثانیه استفاده می‌شود. (زمان تعویض فرآیند را یک میلی‌ثانیه در نظر بگیرید.) متوسط زمان انتظار این فرآیندها چند میلی‌ثانیه است؟ (همه زمان‌ها بر حسب میلی‌ثانیه هستند.) **(مهندسی *IT* - دولتی ۱۴۰۱)**

فرآیند	زمان ورود	مدت زمان لازم برای پردازش
P_1	0	8
P_2	4	6
P_3	3	4
P_4	6	4

۱) 17
۲) 21
۳) 8.8
۴) 19.75

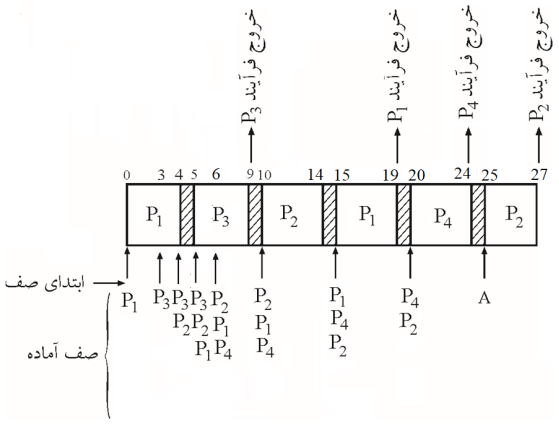
پاسخ تست‌های فصل دوم: مدیریت فرآیندها و زمان‌بندی پردازنده

۹۶- گزینه () صحیح است.

با توجه به مفروضات مطرح شده در صورت سؤال داریم:

فرآیند	زمان ورود	زمان اجرا	زمان انتظار+	زمان بازگشت =
P_1	0	8		
P_2	4	6		
P_3	3	4		
P_4	6	4		

با توجه به برش زمانی quantum = 4 و زمان سربار تعـویض مـتن فرآینـدها context switch = 1 نمودار گانت زیر را داریم:



زمان ورود فرآیند- زمان خروج فرآیند= زمان بازگشت فرآیند

P_1 زمان بازگشت = 19–0=19

P_2 زمان بازگشت =27–4=23

P_3 زمان بازگشت =9–3=6

P_4 زمان بازگشت =24–6=18

میانگین زمان بازگشت = $\frac{19+23+6+18}{4}=\frac{66}{4}=16.5$

زمان اجرای فرآیند - زمان بازگشت فرآیند= زمان انتظار فرآیند

P_1 زمان انتظار =19–8=11

P_2 زمان انتظار =23–6=17

P_3 زمان انتظار =6–4=2

P_4 زمان انتظار =18–4=14

میانگین زمان انتظار = $\frac{11+17+2+14}{4}=\frac{44}{4}=11$

میانگین زمان اجرا = $\frac{8+6+4+4}{4}=\frac{22}{4}=5.5$

میانگین زمان انتظار + میانگین زمان اجرا= میانگین زمان بازگشت

16.5 = 5.5+11

با توجه به اطلاعات به دست آمده، جدول قبل، به شکل زیر تکمیل می‌گردد:

فرآیند	زمان ورود	زمان اجرا	زمان انتظار+	زمان بازگشت =
P_1	0	8	11	19
P_2	4	6	17	23
P_3	3	4	2	6
P_4	6	4	14	18

میانگین زمان بازگشت = میانگین زمان انتظار + میانگین زمان اجرا

16.5 11 5.5

توجه: سازمان سنجش آموزش کشور، ابتدا در کلید اولیه خود گزینه سوم را به عنوان پاسخ اعـلام نمود، اما در کلید نهایی سوال حذف شد، که کار درستی بوده است.

تست‌های فصل دوم: مدیریت فرآیندها و زمان‌بندی پردازنده

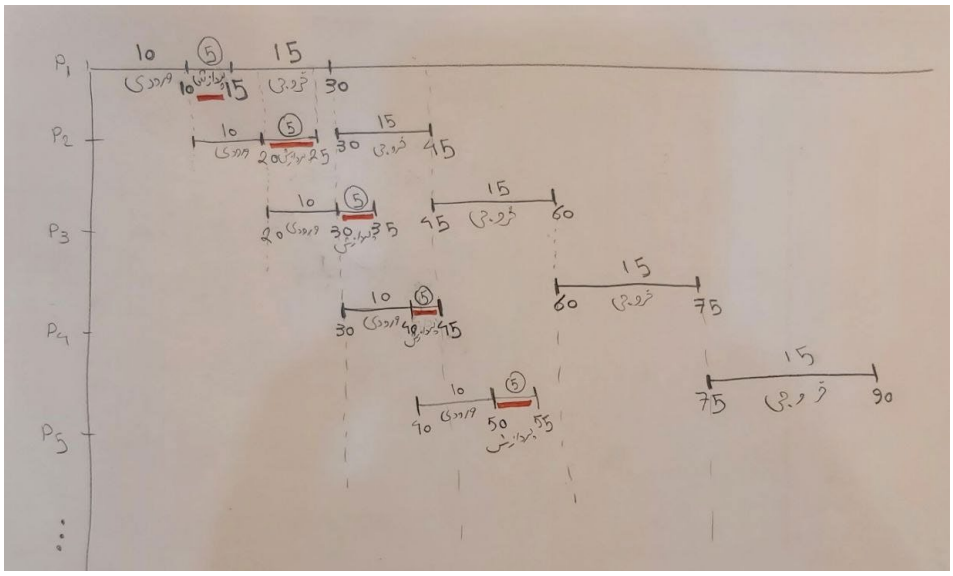
۹۷- سیستمی که از چندبرنامگی حمایت می‌کند و فقط یک دستگاه ورودی، یک CPU تک هسته‌ای و یک دستگاه خروجی دارد و در هر لحظه هر کدام فقط یک کار را می‌توانند انجام دهند در نظر بگیرید. اگر برنامه‌های روی این سیستم برای ورودی، 10 نانوثانیه، برای پردازش روی CPU، 5 نانوثانیه و برای دستگاه خروجی، 15 نانوثانیه زمان نیاز داشته باشند، بهره‌وری CPU برای تعداد برنامه به اندازه کافی بزرگ، چند درصد خواهد بود؟ (برای منابع موجود در این سیستم صف وجود ندارد.) **(مهندسی *IT* – دولتی ۱۴۰۱)**

۱) 50 ۲) 25 ۳) 100 ۴) 33

پاسخ تست‌های فصل دوم: مدیریت فرآیندها و زمان‌بندی پردازنده

۹۷- گزینه (۴) صحیح است.

مطابق فرض سوال، یک برنامه نیاز به 10 نانوثانیه کار ورودی، 5 نانوثانیه کار پردازش و 15 نانوثانیه کار خروجی دارد، نمودار گانت برای n فرآیند به صورت اجرای چند برنامگی به صورت زیر است:



تعداد فرآیندها	زمان مفید CPU	زمان پایان فرآیندها
1	5	30
2	10	45
3	15	60
4	20	75
5	25	90
⋮	⋮	⋮
n	$5\times n$	$30+(n-1)\times 15$

$$\text{بهره‌وری} = \frac{5\times n}{30+(n-1)\times 15} = \frac{5\times n}{15+15n}$$

$$\lim_{n\to\infty} \frac{5n}{15+15n} = \frac{5}{15} = \frac{1}{3} = 33\%$$

تست‌های فصل چهارم: مدیریت حافظه اصلی

۹۸- سیستمی با آدرس مجازی (virtual address) 24-بیتی از ترجمه آدرس سلسله مراتبی 2 سطحی و اندازه صفحه 2 کیلوبایت بهره می‌برد. اگر اندازه هر مدخل جدول صفحه برابر 8 بایت (شامل اطلاعات ترجمه و دیگر اطلاعات کنترلی لازم) باشد، اندازه بیتی بخش میدان‌های جابه‌جایی (offset) ، اندیس به جدول داخلی ترجمه و اندیس به جدول بیرونی ترجمه به ترتیب از راست به چپ کدام است؟ (بهترین گزینه را انتخاب کنید.) **(مهندسی *IT* – دولتی ۱۴۰۱)**

۱) 4, 10, 10 ۲) 5, 8, 11 ۳) 5, 11, 8 ۴) 2, 11, 11

عنوان کتاب: سیستم عامل

مولف: استاد خلیلی‌فر

ناشر: انتشارات راهیان ارشد و دکتری

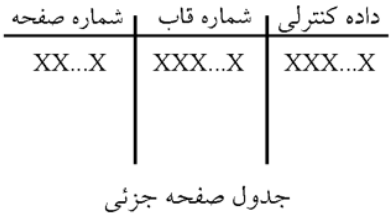
آدرس سایت گروه بابان: BABAN.IR

پاسخ تست‌های فصل چهارم: مدیریت حافظه اصلی

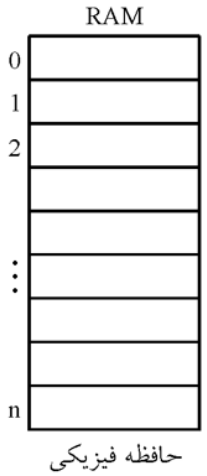
۹۸- گزینه (۲) صحیح است.

در اینجا برای جدول صفحه جزئی محدودیتی به اندازه یک قاب داریم.

بنابراین اندازه جدول صفحه جزئی برابر اندازه قاب می‌باشد. بنابراین برای محاسبه تعداد سطرهای جدول صفحه جزئی، کافی است، اندازه قاب که برابر اندازه جدول صفحه جزئی است بر اندازه عرض جدول صفحه جزئی تقسیم گردد. به شکل زیر توجه کنید:



جدول صفحه جزئی



حافظه فیزیکی

آدرس منطقی(مجازی) آدرس فیزیکی

توجه: عرض جدول صفحه همواره برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب است، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد.

بنابراین داریم:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب = عرض جدول صفحه جزئی

عرض جدول صفحه جزئی = 8 Byte

توجه: مطابق فرض صورت سوال اندازه هر مدخل جدول صفحه برابر 8 بایت (شامل اطلاعات ترجمه و دیگر اطلاعات کنترلی لازم)، بنابراین مجموع تعداد بیت‌های شماره قاب بعلاوه تعداد بیت‌های کنترلی برابر 8 بایت است.

$$\text{تعداد سطرهای جدول صفحه جزئی} = \frac{\text{اندازه قاب}}{\text{عرض جدول صفحه}} = \frac{2\,KB}{8\,B} = \frac{2\times 2^{10}B}{2^3 B} = \frac{2^{11}B}{2^3 B} = 2^8$$

$2^{24}B$ = اندازه فرآیند (فضای آدرس مجازی)

$2KB = 2^{11}B$ = اندازه صفحه = اندازه قاب

$$\text{تعداد صفحات فرآیند: } f = \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه}} = \frac{2^{24}B}{2^{11}B} = 2^{13}$$

تعداد سطرهای جدول صفحه جزئی: $r = 2^8 = 256$

حال اطلاعات کافی برای محاسبه تعداد سطوح جدول صفحه چند سطحی را در اختیار داریم:

روش تجزیه

تعداد صفحات فرآیند باید در اندازه $r\,(2^8)$ تجزیه گردد:

$$\text{تعداد صفحات فرآیند} = 2^{13} = 2^{\overset{\uparrow}{(5)}} \times 2^{\overset{\uparrow}{(8)}} \quad \begin{matrix} PT1 & PT2 \end{matrix}$$

توجه مهم: هیچگاه مقدار PT1 نمی‌تواند از مقدار PT2 بزرگتر باشد به عبارت دیگر مقدار PT1 همواره کوچکتر یا مساوی مقدار PT2 است.

توجه: تعداد عملوندها برابر 2 است، بنابراین تعداد سطوح جدول چند سطحی نیز برابر 2 است.

روش لگاریتم

$$\text{تعداد سطوح جدول چند سطحی} = d = \left\lceil \log_r^f \right\rceil = \left\lceil \log_{2^8}^{2^{13}} \right\rceil = 2$$

روش تقسیم متوالی

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{2^{13}}{2^8} = 2^5$$

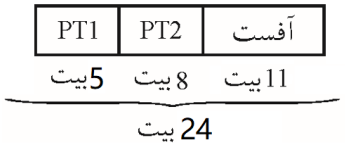
$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{2^5}{2^8} < 1$$

توجه: سطح اول، یک جدول به حساب می‌آید، که 2^5 سطر بیشتر نیاز ندارد.

توجه: تعداد تقسیم متوالی برابر 2 است، بنابراین تعداد سطوح جدول چند سطحی برابر 2 است.

تعداد بیت آفست = $\log_2$ اندازه صفحه = $\log_2^{2^{11}B}$ =11 بیت

بنابراین شکل آدرس منطقی (مجازی) به صورت زیر خواهد بود:



تست‌های فصل هفتم: مدیریت بن‌بست

۹۹- مشخصات 5 فرآیند به شرح زیر است: **(مهندسی *IT* - دولتی ۱۴۰۱)**

فرآیند	MAX		
	R_1	R_2	R_3
P_0	8	6	9
P_1	9	10	2
P_2	9	11	8
P_3	6	7	9
P_4	4	8	9

فرآیند	Allocation		
	R_1	R_2	R_3
P_0	5	4	6
P_1	7	5	2
P_2	6	1	5
P_3	2	4	7
P_4	4	5	3

R_1	R_2	R_3
4	5	6

Free Resources

در صورت اجابت کردن کدام درخواست، سیستم به حالت ناامن خواهد رفت؟

۱) P_1 درخواست 2 عدد R_1 بدهد.

۲) P_0 درخواست 3 عدد R_1 بدهد.

۳) P_2 درخواست 4 عدد R_2 بدهد.

۴) P_3 درخواست 2 عدد R_3 بدهد.

پاسخ تست‌های فصل هفتم: مدیریت بن‌بست

۹۹- گزینه (۳) صحیح است.

ابتدا وضعیت سیستم را قبل از اجابت درخواست فرآیند P_2 بررسی می‌کنیم:

با توجه به ماتریس تخصیص منابع (Allocation) و منابع در دسترس یعنی بردار Available، منابع اولیه را بدست می‌آوریم، برای یافتن منابع اولیه ابتدا برای هر منبع، منابع تخصیص یافته به هر فرآیند را با هم جمع کرده و سپس با منابع در دسترس یعنی بردار Available جمع می‌نماییم.

R_1 منبع اولیه $= 4 + (5 + 7 + 6 + 2 + 4) = 28$

R_2 منبع اولیه $= 5 + (4 + 5 + 1 + 4 + 5) = 24$

R_3 منبع اولیه $= 6 + (6 + 2 + 5 + 7 + 3) = 29$

بنابراین منابع اولیه به صورت زیر است:

منابع اولیه	28	24	29

براساس رابطه زیر داریم:

$$\text{Need} = \text{MAX} - \text{Allocation}$$

فرآیند	Need		
	R_1	R_2	R_3
P_0	3	2	3
P_1	2	5	0
P_2	3	10	3
P_3	4	3	2
P_4	0	3	6

=

فرآیند	MAX		
	R_1	R_2	R_3
P_0	8	6	9
P_1	9	10	2
P_2	9	11	8
P_3	6	7	9
P_4	4	8	9

−

فرآیند	Allocation		
	R_1	R_2	R_3
P_0	5	4	6
P_1	7	5	2
P_2	6	1	5
P_3	2	4	7
P_4	4	5	3

Available	4	5	6

$$(4,5,6) \xrightarrow[+(5,4,6)]{P_0} (9,9,12) \xrightarrow[+(7,5,2)]{P_1} (16,14,14) \xrightarrow[+(6,1,5)]{P_2} (22,15,19)$$

$$\xrightarrow[+(2,4,7)]{P_3} (24,19,26) \xrightarrow[+(4,5,3)]{P_4} (28,24,29)$$

به این ترتیب و با توجه به ماتریس‌های فوق، دنباله‌ی امن $< P_0, P_1, P_2, P_3, P_4 >$ (از چپ به راست) برای این سیستم موجود است. بنابراین قبل از اجابت درخواست‌های بعدی، سیستم در حالت امن قرار.

الگوریتم درخواست منبع بانکداران

هنگامی که فرآیند P_i، بردار Request[i] را به عنوان اعلام نیاز به منابع مطرح می‌کند، باید این درخواست مطابق مراحل زیر بررسی گردد:

۱) اگر $Request[i] \leq Need[i]$ است به مرحله بعدی برو، در غیر اینصورت خطای تقاضای بیش از نیاز اعلام می‌شود. در واقع در این مرحله بررسی می‌شود که اگر فرآیند درخواست بیش از آن چه که در ابتدا اعلام نیاز کرده است را دارد، با این درخواست مخالفت شود.

۲) اگر $Request[i] \leq Availabe[i]$ است به مرحله 3 برو، در غیر اینصورت منبع کافی جهت تخصیص به این فرآیند وجود نداشته و این فرآیند باید منتظر بماند.

۳) با فرض این که منابع مورد نیاز به این فرآیند اختصاص می‌یابد، در این صورت با اصلاح ماتریس‌ها و بردارها یک الگوریتم تشخیص وضعیت امن اجرا می‌شود. با اجرای این الگوریتم اگر سیستم در وضعیت امن باقی بماند، این منابع به فرآیند اختصاص می‌یابند، در غیراینصورت اگر با این تخصیص سیستم به وضعیت ناامن وارد شود، ماتریس‌ها و بردارها به حالت قبل بازگردانده شده و فرآیند تا آزاد شدن منابع بیشتر منتظر می‌ماند.

اصلاح ماتریس‌ها و بردارها در صورت تخصیص به صورت زیر انجام می‌شوند:

Available (new) = Available (old) – Request[i]

Need (new) [i] = Need (old) [i] – Request[i]

Allocation (new) [i] = Allocation (old) [i] + Request[i]

گزینه سوم پاسخ سوال است. زیرا: مطابق الگوریتم درخواست منبع بانکداران، ابتدا بررسی می‌شود که آیا درخواست فرآیند، کم‌تر از نیاز اعلام شده آن است یا خیر.

$Request[P_2] \leq Need[P_2]$

$[0,4,0] \leq [3,10,3]$

چون شرط اول برقرار است، به سراغ شرط دوم می‌رویم.

در شرط دوم بررسی می‌شود که آیا درخواست داده شده، کمتر از منابع آزاد سیستم است یا خیر.

$Request[P_2] \leq Available$

$[0,4,0] \leq [4,5,6]$

چون هر دو شرط برقرار است، سراغ مرحله سوم از الگوریتم می‌رویم.
در این مرحله، با فرض اینکه درخواست داده شده، اعمال شود، ماتریس‌ها به صورت زیر بروزرسانی می‌شوند:

فرآیند	Need (new)		
	R_1	R_2	R_3
P_0	3	2	3
P_1	2	5	0
P_2	3	⑥	3
P_3	4	3	2
P_4	0	3	6

=

فرآیند	MAX		
	R_1	R_2	R_3
P_0	8	6	9
P_1	9	10	2
P_2	9	11	8
P_3	6	7	9
P_4	4	8	9

−

فرآیند	Allocation (new)		
	R_1	R_2	R_3
P_0	5	4	6
P_1	7	5	2
P_2	6	⑤	5
P_3	2	4	7
P_4	4	5	3

با توجه به ماتریس تخصیص منابع **جدید** ((Allocation(new)) و منابع اولیه، بردار Available(new) را بدست می‌آوریم، برای یافتن بردار Available(new) ابتدا برای هر منبع، منابع تخصیص‌یافته به هر فرآیند را با هم جمع کرده و سپس از منابع اولیه کسر می‌نماییم.

R_1 منبع موجود $= 28 - (5+7+6+2+4) = 4$

R_2 منبع موجود $= 24 - (4+5+5+4+5) = 1$

R_3 منبع موجود $= 29 - (6+2+5+7+3) = 6$

بنابراین بردار Available(new) به صورت زیر است:

Available(new)	4	①	6

مطابق رابطه زیر نیز می‌توان Available (new) را محاسبه نمود:

Available (new) = Available (old) – Request [P_2]

Available (new) = [4,5,6] – [0,4,0] = [4,1,6]

$(4,1,6) \longrightarrow \times$

بعد از اجابت درخواست فرآیند P_2، سناریوی امن یافت نمی‌شود بنابراین سیستم در حالت ناامن قرار دارد، بنابراین گزینه سوم پاسخ سوال است.

گزینه اول پاسخ سوال نیست. زیرا:

$$\text{Request}[P_1] \le \text{Need}[P_1]$$
$$[2,0,0] \le [2,5,0]$$

چون شرط اول برقرار است، به سراغ شرط دوم می‌رویم.

$$\text{Request}[P_1] \le \text{Available}$$
$$[2,0,0] \le [4,5,6]$$

ماتریس‌ها به صورت زیر بروزرسانی می‌شوند:

فرآیند	Need (new)		
	R_1	R_2	R_3
P_0	3	2	3
P_1	⓪	5	0
P_2	3	10	3
P_3	4	3	2
P_4	0	3	6

=

فرآیند	MAX		
	R_1	R_2	R_3
P_0	8	6	9
P_1	9	10	2
P_2	9	11	8
P_3	6	7	9
P_4	4	8	9

−

فرآیند	Allocation (new)		
	R_1	R_2	R_3
P_0	5	4	6
P_1	⑨	5	2
P_2	6	1	5
P_3	2	4	7
P_4	4	5	3

R_1 منبع موجود $= 28 - (5+9+6+2+4) = 2$

R_2 منبع موجود $= 24 - (4+5+1+4+5) = 5$

R_3 منبع موجود $= 29 - (6+2+5+7+3) = 6$

بنابراین بردار Available(new) به صورت زیر است:

Available(new)	②	5	6

مطابق رابطه زیر نیز می‌توان Available (new) را محاسبه نمود:

Available (new) = Available (old) – Request [P_2]

Available (new) = [4,5,6] – [2,0,0] = [2,5,6]

$$(2,5,6) \xrightarrow[+(9,5,2)]{P_1} (11,10,8) \xrightarrow[+(5,4,6)]{P_0} (16,14,14) \xrightarrow[+(6,1,5)]{P_2} (22,15,19)$$

$$\xrightarrow[+(2,4,7)]{P_3} (24,19,26) \xrightarrow[+(4,5,3)]{P_4} (28,24,29)$$

به این ترتیب و با توجه به ماتریس‌های فوق، دنباله‌ی امنِ $< P_1, P_0, P_2, P_3, P_4 >$ (از چپ به راست) برای این سیستم موجود است. بنابراین بعد از اجابت درخواست فرآیند P_1، سیستم بازهم در حالت امن قرار.

گزینه دوم پاسخ سوال نیست. زیرا:

$$Request[P_0] \le Need[P_0]$$
$$[3,0,0] \le [3,2,3]$$

چون شرط اول برقرار است، به سراغ شرط دوم می‌رویم.

$$Request[P_0] \le Available$$
$$[3,0,0] \le [4,5,6]$$

ماتریس‌ها به صورت زیر بروزرسانی می‌شوند:

فرآیند	Need (new)		
	R_1	R_2	R_3
P_0	⓪	2	3
P_1	2	5	0
P_2	3	10	3
P_3	4	3	2
P_4	0	3	6

=

فرآیند	MAX		
	R_1	R_2	R_3
P_0	8	6	9
P_1	9	10	2
P_2	9	11	8
P_3	6	7	9
P_4	4	8	9

−

فرآیند	Allocation (new)		
	R_1	R_2	R_3
P_0	⑧	4	6
P_1	7	5	2
P_2	6	1	5
P_3	2	4	7
P_4	4	5	3

R_1 منبع موجود $= 28 - (8+7+6+2+4) = 1$

R_2 منبع موجود $= 24 - (4+5+1+4+5) = 5$

R_3 منبع موجود $= 29 - (6+2+5+7+3) = 6$

بنابراین بردار Available(new) به صورت زیر است:

Available(new)	①	5	6

مطابق رابطه زیر نیز می‌توان Available (new) را محاسبه نمود:

Available (new) = Available (old) – Request $[P_0]$

Available (new) = [4,5,6] – [3,0,0] = [1,5,6]

$$(1,5,6) \xrightarrow[+(8,4,6)]{P_0} (9,9,12) \xrightarrow[+(7,5,2)]{P_1} (16,14,14) \xrightarrow[+(6,1,5)]{P_2} (22,15,19)$$

$$\xrightarrow[+(2,4,7)]{P_3} (24,19,26) \xrightarrow[+(4,5,3)]{P_4} (28,24,29)$$

به این ترتیب و با توجه به ماتریس‌های فوق، دنباله‌ی امنِ $< P_0, P_1, P_2, P_3, P_4 >$ (از چپ به راست) برای این سیستم موجود است. بنابراین بعد از اجابت درخواست فرآیند P_0، سیستم بازهم در حالت امن قرار.

گزینه چهارم پاسخ سوال نیست. زیرا:

$$Request[P_3] \le Need[P_3]$$
$$[0,0,2] \le [4,3,2]$$

چون شرط اول برقرار است، به سراغ شرط دوم می‌رویم.

$$Request[P_3] \le Available$$
$$[0,0,2] \le [4,5,6]$$

ماتریس‌ها به صورت زیر بروزرسانی می‌شوند:

فرآیند	Need (new)		
	R_1	R_2	R_3
P_0	3	2	3
P_1	2	5	0
P_2	3	10	3
P_3	4	3	⓪
P_4	0	3	6

=

فرآیند	MAX		
	R_1	R_2	R_3
P_0	8	6	9
P_1	9	10	2
P_2	9	11	8
P_3	6	7	9
P_4	4	8	9

−

فرآیند	Allocation (new)		
	R_1	R_2	R_3
P_0	5	4	6
P_1	7	5	2
P_2	6	1	5
P_3	2	4	⑨
P_4	4	5	3

R_1 منبع موجود $= 28 - (5+7+6+2+4) = 4$

R_2 منبع موجود $= 24 - (4+5+1+4+5) = 5$

R_3 منبع موجود $= 29 - (6+2+5+9+3) = 4$

بنابراین بردار Available(new) به صورت زیر است:

Available(new)	4	5	④

مطابق رابطه زیر نیز می‌توان Available (new) را محاسبه نمود:

Available (new) = Available (old) – Request [P_3]

Available (new) = [4,5,6] – [0,0,2] = [4,5,4]

$$(4,5,4) \xrightarrow[+(5,4,6)]{P_0} (9,9,10) \xrightarrow[+(7,5,2)]{P_1} (16,14,12) \xrightarrow[+(6,1,5)]{P_2} (22,15,17)$$

$$\xrightarrow[+(2,4,9)]{P_3} (24,19,26) \xrightarrow[+(4,5,3)]{P_4} (28,24,29)$$

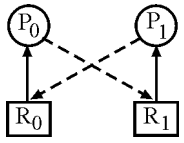
به این ترتیب و با توجه به ماتریس‌های فوق، دنباله‌ی امنِ $< P_0, P_1, P_2, P_3, P_4 >$ (از چپ به راست) برای این سیستم موجود است. بنابراین بعد از اجابتِ درخواست فرآیند P_3، سیستم بازهم در حالت امن قرار.

توجه: 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 است.

توجه: در حالت ناامن نیز ممکن است، فرآیندها در موقعیت رهاسازی منابع در اختیار خود قرار بگیرند و سیستم به بن‌بست نرسد. به بیان دیگر در حالت ناامن وقوع بن‌بست قطعی نیست. اما اگر در حالت ناامن فرآیندها علاوه بر نگهداری منابع تحت تملک خود، منابع دیگری را مورد درخواست قرار دهند، در این حالت فرآیندهایی که نیاز آن‌ها برطرف نمی‌شود یک به یک در صف انتظار قرار می‌گیرند و همچون سرنوشت‌های به هم گره خورده، تا ابد منتظر یکدیگر باقی می‌مانند، یعنی بن‌بست رخ داده است.

مثال: در یک سیستم تعداد منبع اولیه R_0 برابر یک و تعداد منبع اولیه R_1 برابر یک است. اگر فرآیند P_0 یک منبع R_0 را در تملک داشته باشد و برای اتمام، نیاز به یک منبع R_1 نیز داشته باشد و همچنین فرآیند P_1 یک منبع R_1 را در تملک داشته باشد و برای اتمام، نیاز به یک منبع R_0 نیز داشته باشد، وضعیت سیستم را در این شرایط بررسی کنید:

حل: ماتریس و گراف منابع و فرآیندها به صورت زیر است:



فرآیند	MAX	
	R_0	R_1
P_0	1	1
P_1	1	1

=

فرآیند	Allocation	
	R_0	R_1
P_0	1	0
P_1	0	1

+

فرآیند	Need	
	R_0	R_1
P_0	0	1
P_1	1	0

با توجه به ماتریس تخصیص منابع (Allocation) و منابع اولیه، بردار Available را بدست می‌آوریم:

R_0 منبع موجود $= 1-(1) = 0$

R_1 منبع موجود $= 1-(1) = 0$

بنابراین بردار Available به صورت زیر است:

Available	0	0

مشاهده می‌شود که با بردار Available موجود، نمی‌توان نیاز هیچ فرآیندی را با توجه به ماتریس Need مرتفع ساخت، بنابراین توالی امن یافت نمی‌شود و سیستم در یک وضعیت ناامن قرار دارد و احتمال وقوع بن‌بست وجود دارد.

توجه: در این شرایط ناامن، اگر فرآیند P_0 یا P_1 و یا هر دو، در موقعیت رهاسازی منابع در اختیار خود قرار بگیرند، یعنی رهاسازی کنند و سپس درخواست منابع باقی‌مانده خود را صادر کنند، بن‌بست رخ نمی‌دهد.

توجه: در این شرایط ناامن، اگر فرآیند P_0 در عین حال اینکه منبع R_0 را تحت تملک و نگهداری خود دارد، منبع R_1 را نیز درخواست کند، از آن‌جا که منبع R_1 در اختیار فرآیند P_1 می‌باشد، فرآیند P_0 در صف انتظار، می‌خوابد (نگهداری و انتظار) حال اگر در ادامه ماجرا، فرآیند P_1 نیز همین رویه را در پیش گیرد، یعنی فرآیند P_1 نیز در عین حال اینکه منبع R_1 را تحت تملک و نگهداری خود دارد، منبع R_0 را نیز درخواست کند، از آن‌جا که منبع R_0 در اختیار فرآیند P_0 می‌باشد، فرآیند P_1 نیز در صف انتظار، می‌خوابد (نگهداری و انتظار). این سرنوشت‌های به هم گره خورده، در ادامه بن‌بست را به ارمغان می‌آورد.

توجه: از دو توجه فوق این نتیجه استنباط می‌گردد که در شرایط ناامن، **احتمال وقوع بن‌بست** وجود دارد. در واقع در شرایط ناامن وقوع بن‌بست و عدم بن‌بست به رفتار فرآیندها در آینده، بستگی دارد. یعنی فرآیندها رفتار نگهداری و درخواست را در پیش گیرند و یا رفتار رهاسازی و درخواست را.

تست‌های فصل پنجم: مدیریت حافظه مجازی

۱۰۰- با توجه به تعاریف شاخص‌های زیر، صفحه‌آوری مبتنی بر درخواست کامل (Pure demand paiging) در قیاس با صفحه‌آوری مبتنی بر درخواست عادی (demand paging) به کدام شاخص اهمیت بیشتری داده است؟ **(مهندسی *IT* - دولتی ۱۴۰۱)**

- **شاخص بهره‌وری CPU**: میزان درصد استفاده از CPU در طول زمان.

- **شاخص پراکندگی ایستا**: هر فرآیند پس از بار شدن در حافظه (Load) تنها به کد و داده نیاز بیشتری دارد.

- **شاخص پراکندگی پویا**: هر فرآیند در هر بازه زمانی تعلق‌یافته برای اجرا تنها به بخش کوچکی از کد و بخش کوچکی از داده نیاز دارد.

۱) پراکندگی پویا
۲) پراکندگی ایستا و پویا
۳) پراکندگی ایستا
۴) بهره‌وری

عنوان کتاب: سیستم عامل

مولف: استاد خلیلی‌فر

ناشر: انتشارات راهیان ارشد و دکتری

آدرس سایت انتشارات بابان:BABAN.IR

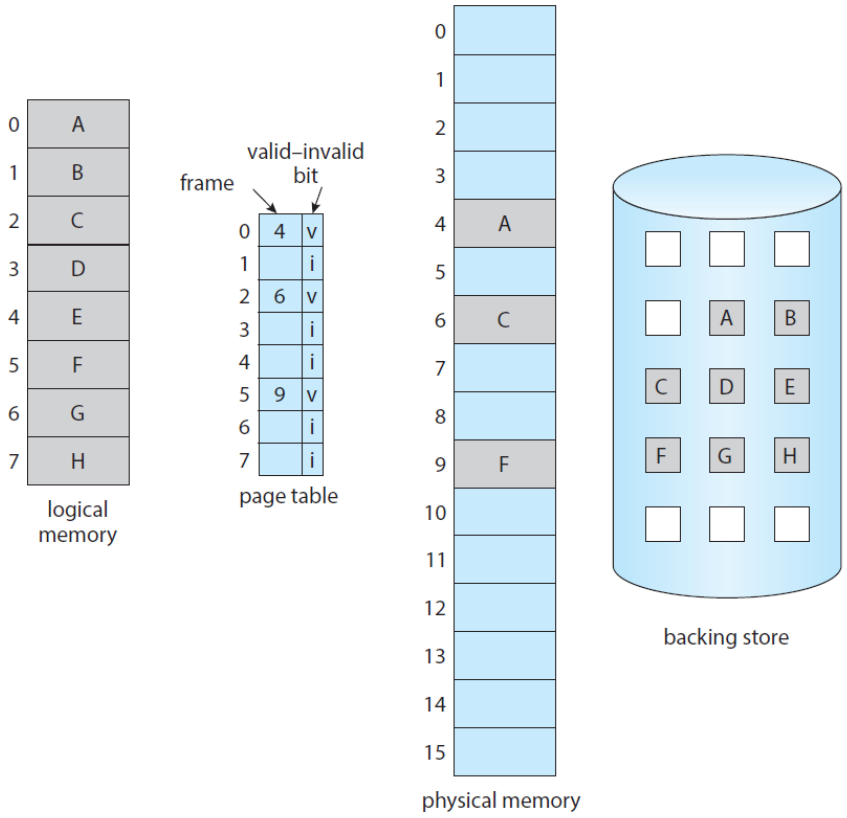
پاسخ تست‌های فصل پنجم: مدیریت حافظه مجازی

۱۰۰- گزینه (۱) صحیح است.

صفحه‌بندی بر حسب نیاز (Demand Paging)

یکی از بهترین روش‌ها جهت پیاده‌سازی ایده حافظه مجازی، استفاده از تکنیک صفحه‌بندی است. در این حالت تنها تعداد اندکی از صفحه‌های یک فرآیند به حافظه آورده و مابقی صفحات بر روی دیسک نگهداری می‌شوند. در این شیوه اگر به صفحات موجود بر روی دیسک نیاز پیدا کردیم، آن صفحات به جای صفحات قدیمی به حافظه آورده شده و در عوض صفحات قدیمی به دیسک منتقل می‌شوند.

در این روش باید مشخص شود کدام صفحات در حافظه و کدام صفحات بر روی دیسک قرار دارند. برای این منظور روش‌های مختلفی وجود دارد اما عموماً از یک بیت در جدول صفحه استفاده می‌کنند. این بیت که آن را **بیت اعتبار** می‌نامیم، مشخص می‌کند صفحه موردنظر در حافظه قرار دارد یا خیر. برای مثال اگر مقدار این بیت به ازای یک درایه در جدول صفحه یک بود، به این معناست که صفحه موردنظر معتبر (valid) است و در حافظه قرار دارد، اما اگر این بیت صفر بود به این معناست که صفحه مورد نظر نامعتبر (invalid) است و بر روی دیسک قرار دارد. شکل زیر گویای مطلب است:



Page table when some pages are not in main memory.

اگر فرآیندی به یکی از صفحاتش که در حافظه موجود نیست (بیت اعتبار آن با مقدار نـامعتبر پر شده است) نیاز داشته باشد، یک وقفه خطای صفحه (Page Fault) رخ می‌دهد که سیستم عامل باید برای این صفحه، یک قاب در حافظه تهیه کرده و آن را به حافظه منتقل کند.

آدرسی که توسط فرآیند مورد ارجاع قـرار مـی‌گیـرد، **آدرس مجـازی** نـام دارد و آدرس‌هـای حافظه اصلی را **آدرس حقیقی** گویند. با این تعریف محدوده آدرس‌های مجازی کـه یـک فرآینـد می‌تواند به آنها ارجاع کند، **فضای آدرس مجازی** نام دارد و محدوده آدرس‌های حقیقـی موجـود در یک سیستم را **فضای آدرس حقیقی** آن کامپیوتر گویند. هنگامی که فرآیندی در حال اجراست، آدرس‌های مجازی باید به آدرس‌های حقیقی تبدیل شوند.

هنگامی که یک صفحه با وقفه نقص صفحه مواجه شد، سیستم عامل باید هر چه سریع‌تر آن را به یکی از قاب‌های حافظه منتقل کند. در این حالت اگر هیچ قاب آزادی در حافظه موجود نباشـد، یکی از صفحات باید به دیسک منتقل شود تا یک قاب حافظه برای صفحه جدید آزاد گردد. با این

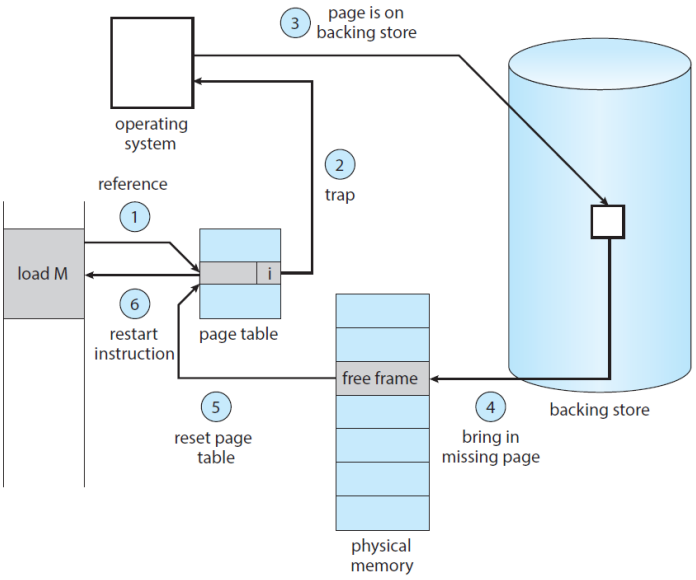
شرط، چگونگی انتخاب یک صفحه برای ترک حافظه بسیار مهم است و تأثیر مستقیمی بر کارایی و تعداد وقفه‌های نقص صفحه در آینده دارد.

صفحه‌بندی بر حسب نیاز خالص یا کامل (Pure Demand Paging)

در خالص‌ترین و کامل‌ترین حالت، می‌توان اجرای یک فرآیند را **بدون هیچ صفحه‌ای** در حافظه آغاز نمود و این یعنی Pure Demand Paging . هنگامی‌که سیستم عامل اشاره‌گر دستورالعمل را برابر با اولین دستورالعمل فرآیند قرار می‌دهد (که این دستورالعمل در صفحه‌ای است که در حافظه نیست) فرآیند بلافاصله برای آن صفحه، خطا تولید می‌کند. بعد از اینکه صفحه مربوطه به حافظه آورده شد، فرآیند به اجرای خودش ادامه می‌دهد، بنابراین هرگاه صفحه‌ای مورد نیاز باشد ولی در حافظه نباشد، تولید خطا ضروری است. وقتی تمام صفحات ضروری به حافظه آورده شوند، اجرای فرآیند می‌تواند بدون خطاهای بیشتر ادامه پیدا می‌کند. این طرح **صفحه‌بندی مبتنی بر تقاضای خالص یا کامل** یا **Pure Demand Paging** نامیده می‌شود. به طور خلاصه این طرح یعنی هیچ صفحه‌ای تا زمانی‌که مورد نیاز نباشد، به حافظه آورده نمی‌شود. تاکید روی هیچ صفحه در ابتدای کار است اما در روش **Demand Paging** تعداد اندکی از صفحه‌های یک فرآیند به حافظه آورده و مابقی صفحات بر روی دیسک نگهداری می‌شوند تا زمانی که نقص صفحه ایجاد شود.

مطابق مفروضات صورت سوال و تعریف **شاخص پراکندگی پویا**:
اینکه هر فرآیند در هر بازه زمانی تعلق‌یافته برای اجرا تنها به بخش کوچکی از کد و بخش کوچکی از داده نیاز دارد. مطابق تعریف **صفحه‌بندی مبتنی بر تقاضای خالص یا کامل** یا **Pure Demand Paging است** چون در هر تقاضا تنها بخش کوچکی از کد و داده نیاز است. بنابراین پرواضح است که گزینه اول پاسخ سوال است.

مراحل رسیدگی به خطای نقص صفحه به صورت زیر است:



Steps in handling a page fault.