

جزوه درسی سیستم عامل

(مؤسسه آموزش عالی پارسه)

« گاری از تیم فنی سایت کارشناسی

« آیا میدانستید اکثر رتبه های تگ رقی و دو رقی گنگور گاردانی به کارشناسی هر سال از جمله
نقر اول، دوم و سوم سال گذشته از اعضای سایت کارشناسی و تالار گفتمان سایت بوده اند ؟

« سوالات و ایهامات خود را در تالار گفتمان سایت مطرح نمایید

www.karshenasi.com

www.forum.karshenasi.com

«مبحث: الگوریتم های زمانبندی»

الگوریتم های زمانبند بلند مدت: این الگوریتم ها مربوط به انتخاب *job* می باشند.

(۱) *FIFO*: اولین کاری که به جدول *ISPT* مراجعه کرده ابتدا انتخاب شده و به پردازش تبدیل می گردد.

(۲) *(Shortest Job First) SJF*: در این الگوریتم ها ابتدا با استفاده از فرمول هایی زمان اجرای *job* های موجود در جدول *ISPT* محاسبه شده و سپس، کوتاهترین کار انتخاب می شود.

(۳) *Mixed*: در این الگوریتم سیستم عامل یک حالت تعادل بین کارهای *I/O-Limited* و *CPU-Limited* ایجاد کرده و سپس به صورت ترکیبی به این دو گروه پاسخ می دهد.

الگوریتم های زمانبند کوتاه مدت: با توجه به اینکه زمانبندی برای *CPU* به دو شکل انحصاری یا غیر قابل پس گرفتن (*non preemptive*) و غیر انحصاری یا قابل پس گرفتن (*preemptive*) می باشد در نتیجه الگوریتم های این دو بخش به صورت الگوریتم های انحصاری و الگوریتم های غیر انحصاری مطرح می شوند.

الگوریتم ها زمانبندی انحصاری: این الگوریتم ها به ترتیبی اولویت پردازشها را جهت اجرا مشخص می کنند و اگر پردازشی *CPU* را در اختیار بگیرد، *CPU* را پس نمی دهد مگر آنکه تمام شود یا نیاز به عملیات *I/O* داشته باشد.

(۱) الگوریتم *FCFS* یا *FIFO*: در این الگوریتم که ساده ترین روش می باشد پردازشها به همان ترتیبی که به صف آماده ی اجرا مراجعه می کنند به همان ترتیب به *CPU* ارسال می شوند.

در این الگوریتم نیاز به داشتن زمان اجرای فرآیند ها قبل از اجرا نیست - قابل پیاده سازی عملی است. میانگین زمان انتظار فرایندها زیاد است. این الگوریتم قحطی زدگی (*Starvation*) ندارد.

نکته: در این الگوریتم اگر یک پردازش *CPU Limited* با پردازشهای دیگری که زمان کمی می خواهند چند برنامگی شود، برنامه های کوچک می باید زمان زیادی را در صف انتظار *CPU* بمانند که این کار باعث اثر اسکورت (*Conroy effect*) می شود.

تست: اگر پنج پردازش مطابق جدول زیر موجود باشند و از روش *FCFS* استفاده شود میانگین زمان انتظار کدام گزینه است؟

پردازش	زمان ورود	زمان اجرا
P_1	۰	۳
P_2	۲	۴
P_3	۰	۱
P_4	۱	۵
P_5	۴	۲

(۱) ۴/۴ ۴/۵(۲) ۵/۵(۳) ۵/۴

(۲) الگوریتم *(Shortest Process Next) SPN*: نام دیگر این الگوریتم *SJF* یا *SPT* می باشد در این روش از میان پردازش های موجود در صف *Ready* پردازش دارای کوتاهترین زمان اجرا انتخاب شده و به پردازنده ارسال می شود.

این الگوریتم به نفع پردازش های کوچک عمل می کند - غیر قابل پیاده سازی است و قحطی زدگی دارد و بعد از الگوریتم **SRTF** کمترین میانگین زمان انتظار فرایندها را دارد.

تست: پنج پردازش مطابق جدول زیر موجود هستند با الگوریتم **SPN** میانگین زمان انتظار فرایندها کدام است؟

پردازش	زمان ورود	زمان اجرا
P_1	۰	۴
P_2	۱	۱
P_3	۰	۲
P_4	۱	۱
P_5	۴	۳

۲/۲(۴)

۵(۳)

۳(۲)

۲(۱)

۳) **زمانبندی اولویتی (Priority scheduling)**: در این الگوریتم با اختصاص عدد، اولویت فرایندها مشخص می شود. مثلاً در کامپیوتر چند کاربره دانشگاه، اولویت ریاست از همه بیشتر است، سپس اساتید و بعد دانشجویان، لذا کارهای اساتید باید زودتر از دانشجویان اجرا شود. اولویتها هم می تواند توسط اپراتور به صورت خارجی برای سیستم تعریف شود و هم خود سیستم عامل به صورت داخلی بنابر فاکتورهایی اولویتها را مشخص سازد.

۴) **زمانبندی شانسی (Lottery scheduling)**: در این الگوریتم سیستم عامل به هر پردازش تعدادی عدد می دهد. تعداد اعداد نسبت داده شده به هر پردازش، وابسته به اولویت آن است یعنی هر چه اولویت پردازش بیشتر باشد، اعداد بیشتری به آن داده می شود. سپس به طور اتفاقی یک عدد تولید شده و CPU در اختیار پردازشی که آن عدد را در اختیار دارد، قرار می گیرد. هر چقدر اولویت برنامه ها بیشتر باشد شانس آنها برای در اختیار داشتن عدد اتفاقی تولید شده و در نتیجه در اختیار گرفتن CPU بیشتر است.

۵) **زمانبندی بالاترین نسبت پاسخ (HRRN) : Highest Response Ratio Next**، برخی از مشکلات

الگوریتم **SPN** را برطرف می کند. در این زمانبندی اولویت ها به صورت دینامیک بوده و به شکل زیر محاسبه می شود:

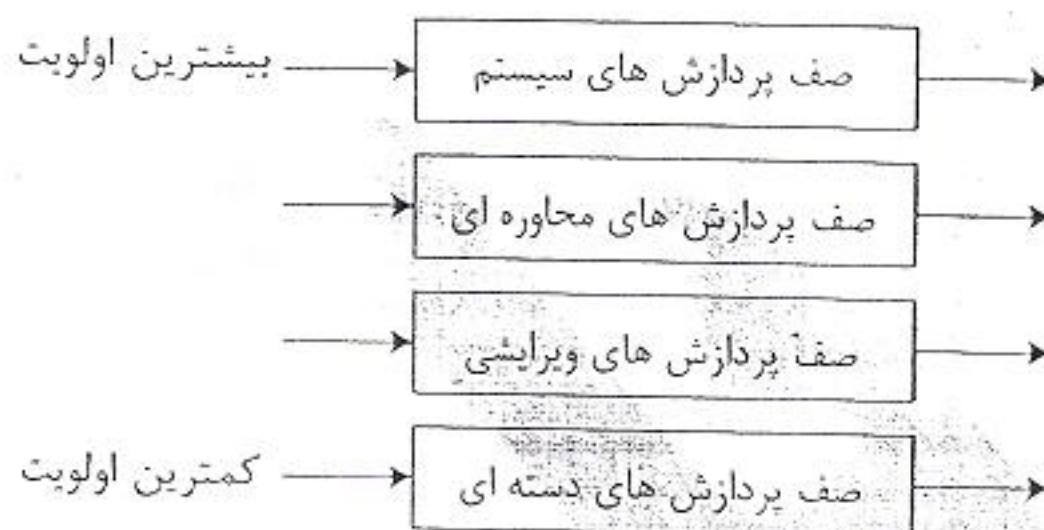
$$\text{Priority} = \frac{WT + CBT}{CBT} = 1 + \frac{WT}{CBT}$$

به دلیل آنکه در مخرج کسر زمان سرویس (**CBT**) موجود است پس کارهای کوتاهتر اولویت بیشتری دارند و زودتر اجرا می شوند ولی از طرف دیگر چون در صورت کسر، زمان انتظار داریم، کارهای طولانی نیز که مدت زیادی در صف انتظار بوده اند اولویت بیشتری کسب کرده و بالاخره در یک زمان معین اجرا می شوند. این الگوریتم غیر قابل پیاده سازی بوده ولی قحطی زدگی ندارد.

۶) **زمانبندی LPT**: در این زمانبندی که **Longest Processing Time** نام دارد، هرگاه که CPU آزاد

می گردد، از بین پردازشهای باقی مانده در صف آماده اجرا، طولانی ترین پردازش را جهت اجرا انتخاب می کند. این الگوریتم به نفع پردازشهای بزرگتر می باشد و احتمال قحطی زدگی فرایندها را دارد.

(۷) زمانبندی صف های چندگانه *Multiple queues*: به این الگوریتم، روش صف های چند سطحی یا *Multi Level Queues* نیز گفته می شود. در این الگوریتم، صف آماده اجرا به صف های جداگانه مختلفی تجزیه می شود و هر پردازش وارد یک صف می گردد. اولویت صف ها با هم فرق دارد. در این الگوریتم احتمال قحطی زدگی وجود دارد، یک نمونه از صف های این سیستم می تواند به صورت زیر باشد:

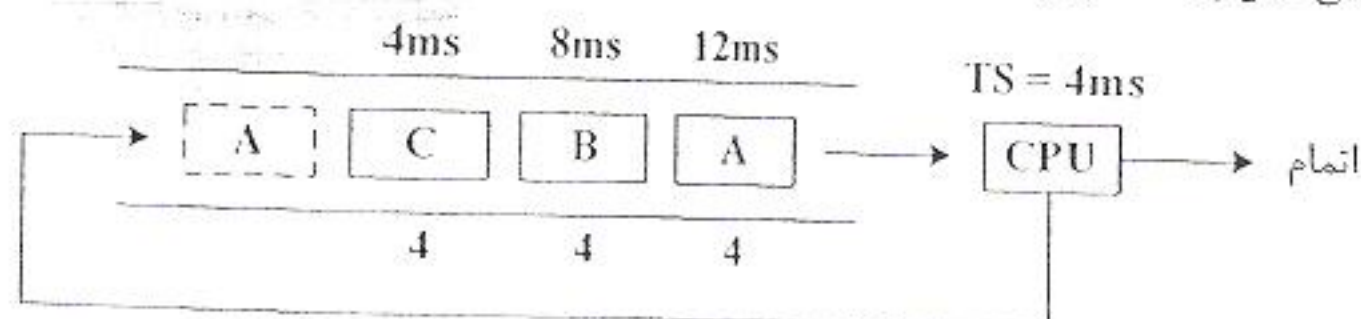


الگوریتم های زمانبندی غیر انحصاری

(۱) زمانبندی نوبت گردشی (*Round Robin*): این الگوریتم شکل غیر انحصاری الگوریتم *FCFS* می باشد. در این روش به همان ترتیبی که پردازشها به صف آماده اجرا مراجعه می کنند، یک واحد کوچک زمانی به نام کوانتم زمانی یا برش زمانی به آنها داده می شود. اگر فرایند بیش از این زمان را نیاز داشته باشد یا نیاز به عملیات *I/O* داشته باشد، *CPU* پس گرفته شده و فرایند به انتهای صف می رود.

نکات:

- (۱) این الگوریتم قحطی زدگی ندارد.
- (۲) عادلانه ترین الگوریتم زمانبندی است.
- (۳) در سیستم های اشتراک زمانی و *Interactive* استفاده می شود.
- (۴) به صورت عملی قابل پیاده سازی است.



انتظار برای I/O یا پایان یافتن برش زمانی

تست: در یک سیستم مبتنی بر الگوریتم *RR* اگر زمان تعویض متن ۲ میلی ثانیه و کوانتم زمانی ۱۴ میلی ثانیه باشد و در صورتی که تعداد زیادی فرایند طولانی *CPU-bounded* در حال اجرا باشند آنگاه درصد بهره وری از *CPU* کدام است؟

- ۸۵ (۱) ۱۲/۵ (۲) ۱۵ (۳) ۸۷/۵ (۴)

تست: اگر n فرآیند در صف آماده اجرا موجود باشند و از الگوریتم RR استفاده شود و کوانتم زمانی را با C نمایش دهیم، حداکثر زمان انتظار فرآیندها برای گرفتن اولین چرخه اجرایی CPU کدام گزینه است؟

$$(1) nc \quad (2) \frac{n}{2}c \quad (3) (n+1)c \quad (4) (n-1)c$$

(۲) زمانبندی کوتاه‌ترین زمان باقی‌مانده (SRT): به این الگوریتم زمانبندی، $SRPT$ یا $SRTF$ نیز می‌گویند. ($Shortest Remaining Time First$) در این روش برنامه‌ای که احتیاج به کمترین زمان جهت تکمیل دارد ابتدا اجرا می‌شود. در هنگام انتخاب یک برنامه کارهایی که تازه به صف آماده‌وارد می‌شوند هم در نظر گرفته می‌شوند. در این حالت ممکن است CPU از یک برنامه در حال اجرا توسط برنامه جدیدی که نیاز به زمان کمتری جهت تکمیل دارد گرفته شود. این روش احتمال قحطی زدگی را برای فرآیندها دارد.

تست: چهار پردازش مطابق جدول زیر حضور دارند با الگوریتم $SRTF$ میانگین زمان انتظار کدام گزینه است؟

پردازش	زمان ورود	زمان اجرا
P1	۰	۵
P2	۱	۱
P3	۴	۱
P4	۶	۲

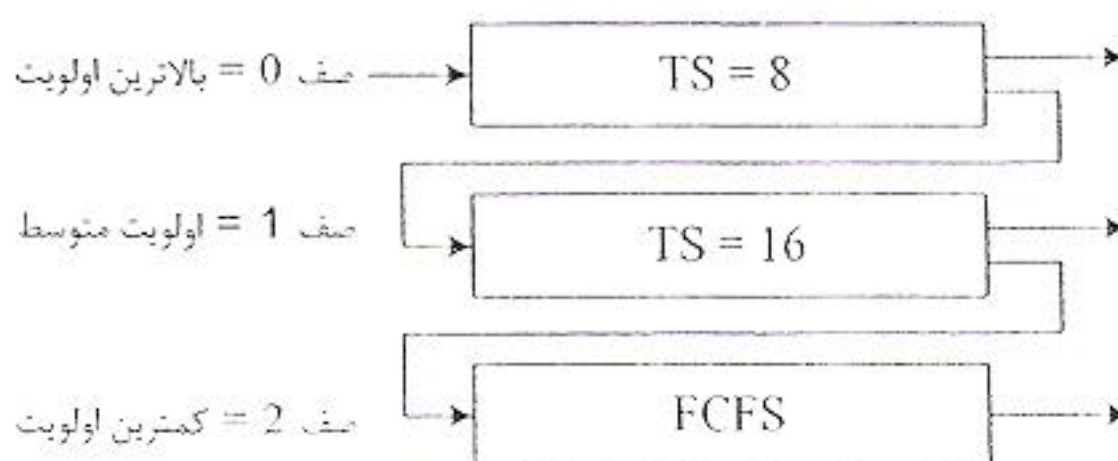
(۱) ۰/۲۵

(۲) ۰/۷۵

(۳) ۱

(۴) ۲

(۳) زمانبندی صفهای چندگانه با فیدبک $MLFQ$ یا MFQ : در الگوریتم MLQ امکان حرکت فرآیندها بین صفها موجود نبود ولی در این الگوریتم این امکان موجود است. در این الگوریتم پردازشی که زیاد وقت CPU را بگیرد به صف با اولویت پایین‌تر فرستاده می‌شود. از طرف دیگر فرآیندی که در صف با اولویت پایین، مدت زیادی منتظر باقی‌مانده به صف با اولویت بیشتری فرستاده می‌شود. در این روش زمانبندی هر صف معمولاً RR می‌باشد و این الگوریتم در سیستم عامل‌های $Main Frame$ استفاده شده است اما احتمال قحطی زدگی دارد.



تست : چهار فرایند A, B, C, D با مشخصات زیر مفروض است. میانگین زمان پاسخدهی آنها در الگوریتم های زمانبندی به ترتیب ورود ($FCFS$) و با گردش نوبتی (RR) از راست به چپ کدام است ؟ (فرض کنید تکه زمانی معادل یک واحد زمان است و در مورد سیاست با گردش نوبتی ، فرایندی که وارد سیستم می شود در همان ابتدای ورودش اجرای آن آغاز می گردد) (ارشد مهندسی کامپیوتر - ۸۰)

زمان اجرا	زمان ورود	پردازش
۳	۰	A
۳	۱	B
۳	۴	C
۲	۶	D

(۱) ۴/۵ و ۶

(۲) ۶/۷۵ و ۶

(۳) ۴/۵ و ۲/۷۵

(۴) ۶/۷۵ و ۲/۷۵

تست : چهار فرایند P_1, P_2, P_3, P_4 مطابق جدول زیر موجود هستند به فرض اینکه زمان کوانتم 2MS و زمان تعویض متن ناچیز باشد . آنگاه متوسط زمان پاسخ و انتظار بر حسب ms با استفاده از الگوریتم زمانبندی نوبت چرخشی (RR) به ترتیب کدام است ؟ (آزاد ۸۳)

زمان اجرا	زمان ورود	فرایند
6	2	P_1
3	4	P_2
5	6	P_3
2	12	P_4

(۱) ۸/۷۵ و ۴/۷۵

(۲) ۸/۵ و ۴/۵

(۳) ۸/۲۵ و ۴/۲۵

(۴) ۸ و ۴

تست : در جدول فرایندهای زیر ، میانگین زمان انتظار فرایندها براساس الگوریتم های $SRT-FCFS$ کدام است ؟

(مسابقات آموزشکننده های فنی ۸۴)

زمان اجرا	زمان ورود	فرایند
۱	۶	P_1
۴	۲۰	P_2
۶	۷	P_3
۷	۳	P_4

(۱) ۱۳ و ۵

(۲) ۵ و ۳

(۳) ۴ و ۱۲

(۴) ۱۲/۷۵ و ۴/۲۵

تست: سیستمی از روش زمانبندی نوبت چرخشی RR با کوانتم $3ms$ و زمان تعویض متن $1ms$ استفاده می نماید. اگر چهار فرایند مطابق جدول زیر زمانبندی شوند درصد بهره وری از CPU برای اجرای کامل چهار فرایند کدام است؟ (آزاد ۱۵)

فرایند	زمان ورود	زمان اجرا
P_1	۸	۰
P_2	۶	۲
P_3	۴	۴
P_4	۴	۸

(۱) ۶۹

(۲) ۲۶

(۳) ۳۱

(۴) ۷۴

تست: چهار فرایند مطابق جدول زیر مفروض هستند. میانگین زمان انتظار برای اجرای فرایند و با استفاده از روش زمانبندی $FIFO$ به صورت غیر قابل پس گرفتن کدام است؟ (آزاد ۱۴)

فرایند	زمان ورود	زمان اجرا
P_1	۱	۲
P_2	۳	۳
P_3	۵	۲
P_4	۶	۱

(۱) ۱

(۲) ۰/۵

(۳) ۰/۷۵

(۴) ۱/۲۵

« مبحث: مدیریت حافظه »

بخشی از هر سیستم عامل که سلسله مراتب حافظه را اداره می کند مدیر حافظه (*Memory Manager*) معرفی می شود. این بخش باید بداند که کدام قسمت حافظه خالی و کدام یک استفاده شده است. همچنین اگر چند برنامه همزمان می خواهند با هم اجرا شوند بایستی همگی به حافظه آورده شوند و این مدیر حافظه است که از تداخل آنها در حافظه جلوگیری می کند. از طرف دیگر اگر مقدار *RAM* برای اجرای برنامه کافی نباشد، سیستم عامل می تواند از دیسک به عنوان یک حافظه مجازی استفاده کند.

تعریف آدرس: محل قرارگیری یک متغیر را آدرس آن متغیر گویند.

انواع آدرس: (۱) آدرس منطقی (مجازی) (۲) آدرس فیزیکی (واقعی)

آدرس منطقی: به آدرسی که محل یک متغیر در برنامه (فایل اجرایی) را مشخص می کند آدرس منطقی گفته می شود. این آدرس نسبت به شروع آن برنامه بیان می شود.

آدرس فیزیکی: به آدرسی که محل قرارگیری یک متغیر در حافظه اصلی (هنگام اجرا) را مشخص می کند، آدرس فیزیکی می گویند.

تکنیک های مدیریت حافظه:

(۱) تکنیک های تخصیص همجوار (پیوسته):

- تخصیص تک قسمتی (یکجا)
- تخصیص چند قسمتی
 - (۱) پارتیشن های ثابت (ایستا)
 - (۲) پارتیشن های پویا (دینامیک)

(۲) تکنیک های تخصیص غیر همجوار:

- مدیریت حافظه به روش صفحه بندی
- مدیریت حافظه به روش قطعه بندی

تخصیص تک قسمتی (یکجا): در این تکنیک یک قسمت از حافظه برای سیستم عامل در نظر گرفته می شود و بقیه آن کلاً در اختیار یک فرایند قرار می گیرد.

تخصیص چند قسمتی: در این تکنیک یک قسمت از حافظه برای سیستم عامل و بقیه آن به صورت همزمان در اختیار چندین فرایند قرار می گیرد.

مدیریت حافظه با پارتیشن های ثابت (ایستا):

در این روش یک قسمت از حافظه برای سیستم عامل و مابقی آن در ابتدای کار (پار شدن سیستم عامل) به قطعاتی با اندازه های ثابت (به صورت منطقی) تقسیم بندی می شود. به هر قسمت، یک *Partition* گویند. هر فرایند در یک *Partition* قرار می گیرد، فرایند شکسته نمی شود و نمی توان از یک *Partition* به صورت اشتراکی استفاده کرد.

مدیریت حافظه با پارتیشن های پویا (دینامیک) :

در این روش یک قسمت از حافظه برای سیستم عامل و مابقی به صورت یک *partition* خالی در نظر گرفته می شود. با ورود هر فرایند، یک *Partition* دقیقاً به اندازه همان فرایند ساخته شده و به آن اختصاص داده می شود.

الگوریتم های جستجو برای یافتن قطعه خالی جهت ساختن *Partition*:

هنگامی که بتوان برای یک فرایند در چندین محل مختلف *Partition* ساخت باید بر اساس یک الگوریتم یکی از آن قطعات خالی را برای ساخت *Partition* انتخاب نمود.

(۱) *First Fit* (اولین مناسب): در این روش سیستم عامل با دریافت یک تقاضای حافظه هنگام بار کردن یک برنامه، در لیست فضاهای خالی حافظه اولین بخش آزاد را که به اندازه اعلام شده، گنجایش داشته باشد تخصیص می دهد. شروع جستجو برای یافتن فضای آزاد مناسب همواره از ابتدای لیست صورت می گیرد. بدین ترتیب تراکم فضای اشغال شده در اول حافظه بیشتر خواهد بود.

(۲) *Next Fit* (مناسب بندی): این روش مانند *First Fit* است با این تفاوت که جستجو از محلی در لیست آغاز می شود که آخرین بار، تخصیص از آن محل صورت گرفته است. بدین ترتیب یکنواختی توزیع برنامه ها در سطح حافظه نسبت به روش قبلی بیشتر خواهد شد.

(۳) *Best Fit* (بهترین مناسب): در این روش کل لیست فضاهای آزاد، جستجو شده و کوچکترین حفره که به اندازه کافی بزرگ است به پردازش تخصیص داده می شود. این روش باعث می شود که کوچکترین حفره بر اثر تخصیص باقی بماند. با این روش فضاهای بزرگتر برای تقاضاهای بیشتر حفظ می شوند. از آنجا که تمام لیست بلاک های آزاد باید بررسی شود، این تکنیک زمانبر است.

(۴) *Worst Fit* (بدترین مناسب): در این روش کل لیست فضاهای آزاد جستجو شده و بزرگترین حفره موجود به پردازش تخصیص داده می شود. منطق این روش این است که از حفره های باقی مانده بتوان برای پردازش های دیگر استفاده کرد. ایراد این تکنیک این است که امکان دارد، تقاضاهایی که ناحیه بزرگی می خواهند، دیگر نتوانند برآورده شوند چرا که بلاک های بزرگ زودتر تخصیص یافته و کوچک می شوند.

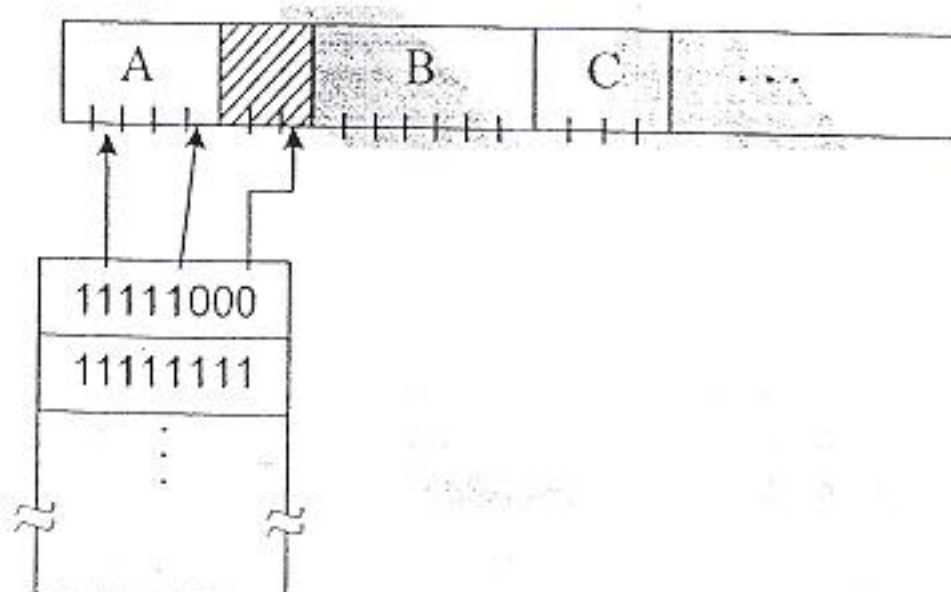
(۵) *Quick Fit*: در این روش برای هر دسته از پروسس هایی با اندازه های متداول، یک لیست جداگانه تهیه می شود. مثلاً این الگوریتم دارای جدولی با n خانه است که خانه اول اشاره گری به ابتدای لیستی شامل حفره های چهار کیلو بایتی دارد، خانه دوم به لیست حفره های ۸ کیلو بایتی و خانه سوم به لیست حفره های ۱۲ کیلو بایتی اشاره می کند و الی آخر.

(۶) الگوریتم *(Buddy)*: در این روش همه بخشهای آزاد به قطعاتی که همگی توان ۲ هستند، تقسیم می شوند (مثل بلوک های آزاد 1k و 2k و 4k و 8k و ...). و برای هر گروه لیست جداگانه در نظر گرفته می شود. بدین ترتیب جهت تخصیص یک بلاک تنها باید بلاک مورد نظر را از لیست مناسب خارج کرد. پس از تخصیص، اگر فضای باقی مانده آن بلاک، توانی از ۲ باشد در لیست مربوطه اش قرار می گیرد و در غیر این صورت به چندین بخش که اندازه هر کدام توانی از ۲ می باشد تقسیم می شود.

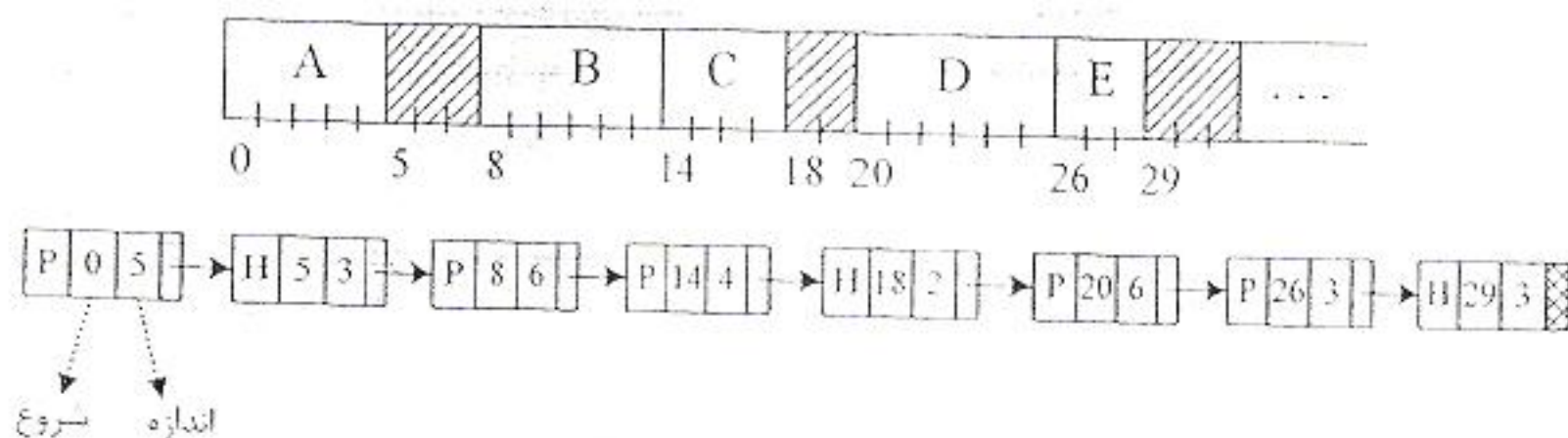
مدیریت فضاهای آزاد:

مدیر حافظه در هر لحظه باید بداند کدام قسمت حافظه آزاد است و کدام قسمت استفاده شده است. برای این منظور از دو روش زیر استفاده می کند:

(۱) روش نگاشت بیتی (*Bit Maps*): در این روش حافظه به چندین واحد تخصیص، تقسیم می شود. اندازه این واحدها می تواند به کوچکی چندین کلمه یا به بزرگی چندین کیلو بایت باشد. متناظر با هر واحد تخصیص، یک بیت در نگاشت بیتی وجود دارد. اگر واحدی استفاده شده باشد بیت متناظر آن "۱" شده و اگر آزاد باشد بیت متناظر آن "۰" می شود.



(۲) روش لیست پیوندی: در این روش یک لیست پیوندی از قطعه های آزاد یا تخصیص یافته حافظه تشکیل می دهیم. به عبارت دیگر هر گره یا یک پروسس و یا یک حفره. در شکل زیر H در اول گره نمایانگر حفره (*Hole*) و حرف P نمایانگر پروسس (*Process*) می باشد.

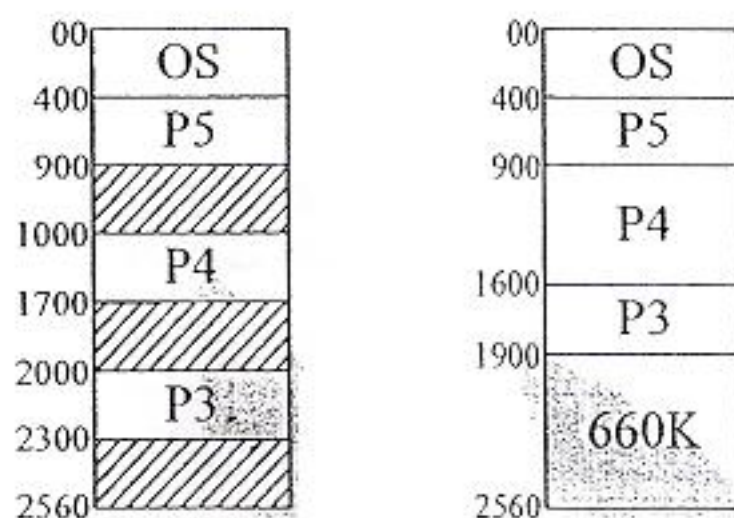


مشکلات مدیریت حافظه:

مشکلات پراکنندگی (پارگی) داخلی و خارجی (*External And Internal Fragmentation*):

پارگی خارجی به این معنا می باشد که مجموع فضاهای آزاد، مقدار زیادی است ولی از آنجا که حفره ها پراکنده اند و همجوار نیستند نمی توان پردازش را در حافظه بار کرد. برای برطرف کردن مشکل پارگی خارجی، فشرده سازی

(*Compaction*) ارائه می شود، در فشرده سازی حافظه، حفره های کوچک با یکدیگر انجام شده و حفره یک پارچه بزرگی را ایجاد می کنند:



مشکل پارگی داخلی را می توان به این صورت توضیح داد اگر حفره ای با ۱۸۴۶۴ بایت داشته باشیم و پردازش بعدی ۱۸۴۶۲ بایت درخواست کند، در اثر این تخصیص حفره ای به اندازه ۲ بایت باقی می ماند که سربار حاصل از تعقیب این حفره قابل توجه تر از خود حفره می باشد. یعنی در عمل در بلوکهای تخصیص یافته، فضای کوچک بلا استفاده باقی می ماند که به آنها پارگی داخلی گفته می شود.

مدیریت حافظه به روش قطعه بندی: *Segmentation*:

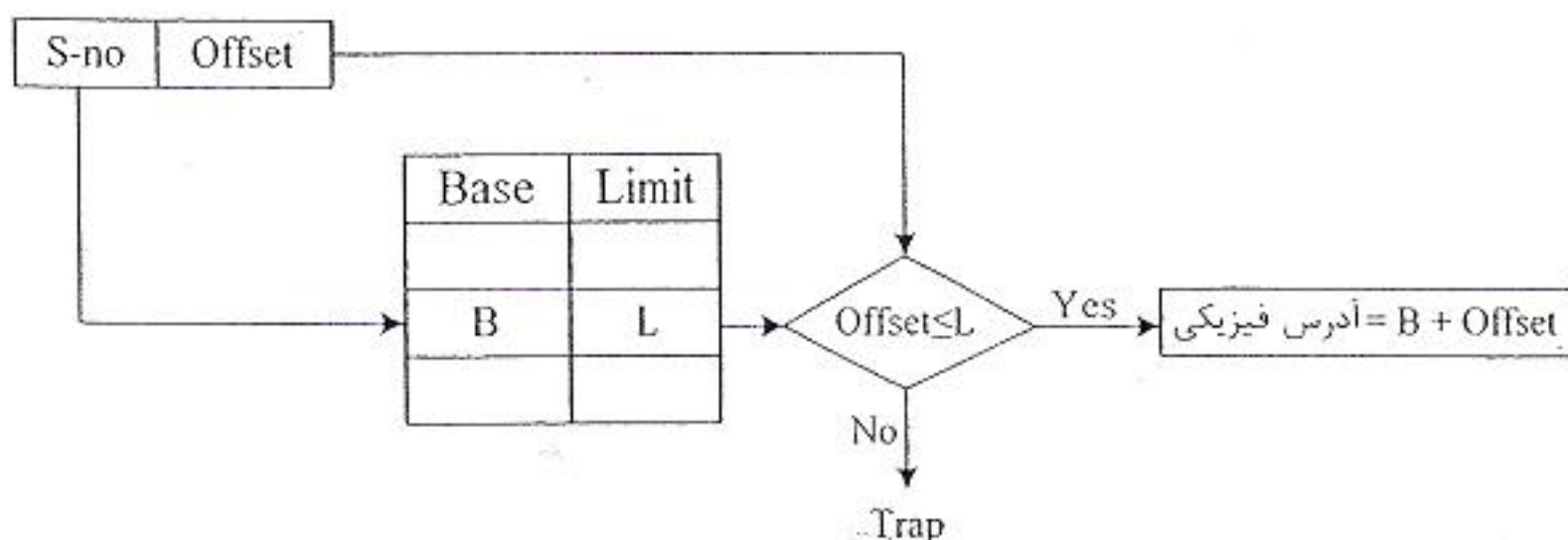
روش تخصیص: در مدیریت حافظه به روش قطعه بندی، فرایند به تعدادی بخش به نام *Segment* تقسیم می شود. محتویات *Segment* ها به لحاظ منطقی به هم مرتبط هستند، اندازه *Segment* ها می تواند متفاوت باشد. حافظه اصلی به صورت یک *Partition* خالی بوده، با ورود هر فرایند برای هر *Segment* آن یک *Partition* دقیقاً به اندازه همان *Segment* ساخته می شود. فضای اختصاص یافته به *Segment* ها می تواند یکپارچه نباشد. اما نمی توان یک *Segment* را به چند بخش تقسیم کرد.

روش نگاشت

جدول قطعه (*Segment Table*): هر فرایند دارای جدول قطعه منحصر به خود است. جدول قطعه هر فرایند مشخص می کند که ذخیره سازی هر قطعه فرایند از کجای حافظه آغاز شده و طول هر قطعه چقدر است. بنابراین آرایه ای از زوج مقدارهای *Base-Limit* می باشد.

تبدیل آدرس منطقی به فیزیکی

آدرس منطقی به صورت شماره قطعه و آفست قطعه بیان می گردد. با استفاده از جدول قطعه (شماره قطعه به عنوان اندیسی برای جدول قطعه به کار می رود) محل شروع ذخیره سازی قطعه و همچنین طول قطعه به دست می آید. چنانچه آفست قطعه از طول قطعه بزرگتر نباشد، مجموع آفست قطعه و محل شروع قطعه، آدرس فیزیکی را تشکیل می دهد.



تست: در یک سیستم حافظه قطعه بندی شده، جدول قطعه مقابل مفروض است. در این سیستم چه مقدار از آدرس های منطقی زیر، فاقد آدرس فیزیکی هستند؟

Segment	Base	Limit
0	1100	500
1	2500	1000
2	200	600
3	4000	1200

- (a) 0, 300 ۴ (۱)
 (b) 2, 800 ۳ (۲)
 (c) 1, 600 ۲ (۳)
 (d) 3, 1100 ۱ (۴)
 (e) 1, 1111

معایب:

۱- تکه تکه شدن خارجی (External Fragmentation)

۲- اتلاف حافظه توسط جدول قطعه

۳- کاهش سرعت دسترسی

نکته: در سیستم های قطعه بندی، یک رجیستر مبنای جدول قطعه به نام *STBR* به ابتدای جدول قطعه اشاره می کند. هنگامی که در تعیین متن، *CPU* می خواهد به سراغ پردازش دیگری برود این ثابت به جدول قطعه پردازش اشاره خواهد کرد.

نکته: در جدول قطعه، علاوه بر ستونهای *Base* و *Limit* یک سری بیت های کنترلی و حفاظتی نیز وجود دارد.

Readable ← *R* -

Writable ← *W* -

Accessed ← *A* -

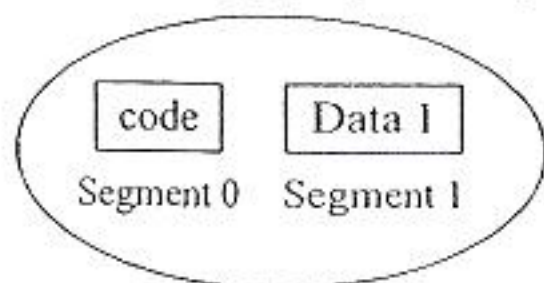
Obey ← *O* -

Segment Present ← *P* -

Privilege Level ← *PL* -

نکته: یکی دیگر از مزایای سیستم قطعه بندی، امکان به اشتراک گذاشتن بعضی از قطعات می باشد. مثلاً *Server* یک دانشگاه که دانشجویان درس برنامه نویسی همزمان به آن متصل می شوند. در این حال دانشجویان همزمان باید مثلاً

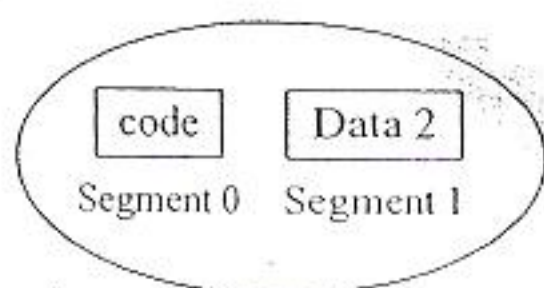
کامپایلر C را اجرا کنند. از آنجایی که بخش کد برنامه فقط خواننده و اجرا می شود یعنی تغییر نمی یابد می توان برای تمام دانشجویان فقط یک نسخه از کد برنامه را در حافظه قرار داد ولی بخش دیتای هر کاربر را به صورت مجزا به او اختصاص داد.



حافظه منطقی پردازش P1

	Limit	Base
0	25286	43062
1	4425	68348

جدول قطعه P1



حافظه منطقی پردازش P2

	Limit	Base
0	25286	43062
1	8850	90003

جدول قطعه P2

نکته: ۳ روش موجود است که به کمک آنها دو یا چند پردازش قطعاتی را به صورت اشتراکی استفاده کنند:

(۱) روش همه مستقیم (۲) یکی مستقیم، بقیه غیر مستقیم (۳) همه غیر مستقیم

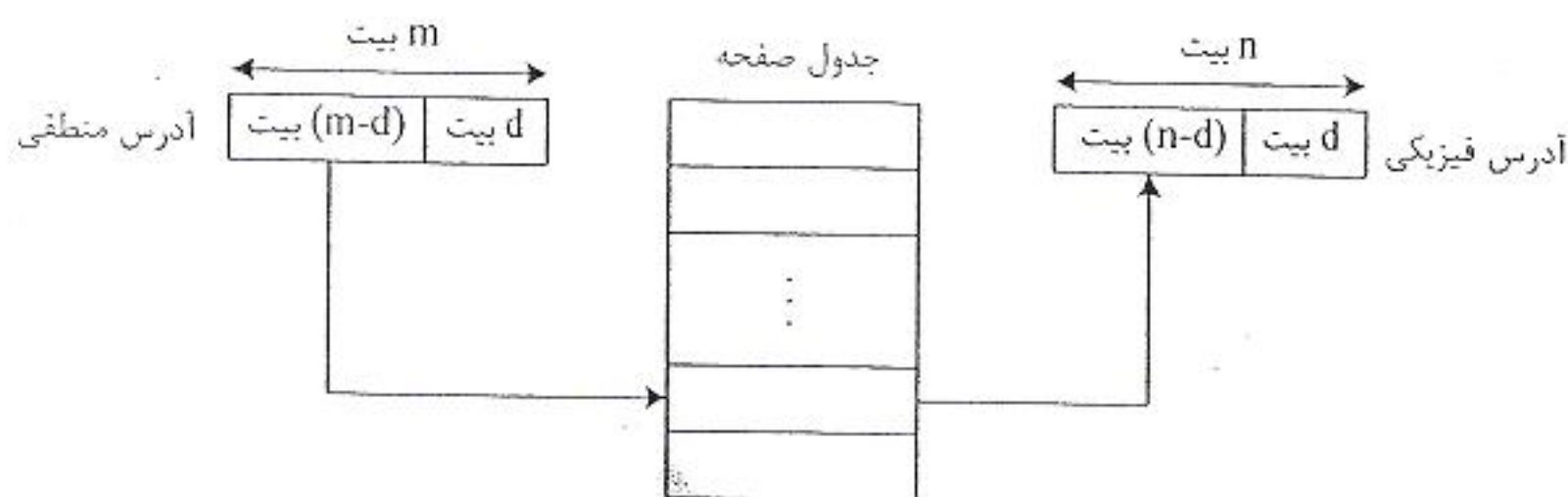
مدیریت حافظه به روش صفحه بندی Page Memory Management

روش تخصیص: در روش مدیریت حافظه به صورت صفحه بندی ابتدا حافظه اصلی به قطعاتی با اندازه های ثابت و یکسان به نام بلاک یا قاب صفحه، به صورت منطقی شکسته می شود (تقسیم می شود) فرایندی که می خواهد وارد حافظه شود به قطعاتی هم اندازه با بلاک ها به نام *Page* یا صفحه شکسته می شود. حال می توان هر صفحه برنامه را در هر قاب آزاد دلخواه قرار داد.

روش نگاشت

جدول صفحه (*Page Table*): هر فرایند دارای جدول صفحه منحصر به خود می باشد. جدول صفحه یک فرایند مشخص می کند هر *Page* در چه بلاکی از حافظه قرار گرفته است. تعداد خانه های جدول صفحه برابر تعداد *Page* های فرایند است.

تبدیل آدرس منطقی به فیزیکی: آدرس منطقی به صورت شماره صفحه و آفست بیان می گردد. چون اندازه صفحه و بلاک با هم برابر است در نتیجه آفست صفحه برابر آفست بلاک خواهد بود. از شماره صفحه به عنوان اندیس جدول صفحه استفاده شده شماره بلاکی که صفحه در آن قرار گرفته است، به دست می آید.



2^d = اندازه هر قاب = اندازه هر صفحه

2^m = اندازه برنامه = اندازه حافظه منطقی

2^{m-d} = تعداد صفحه های برنامه = تعداد سطرهای (مدخل های) جدول صفحه

2^{n-d} = تعداد فریم های حافظه

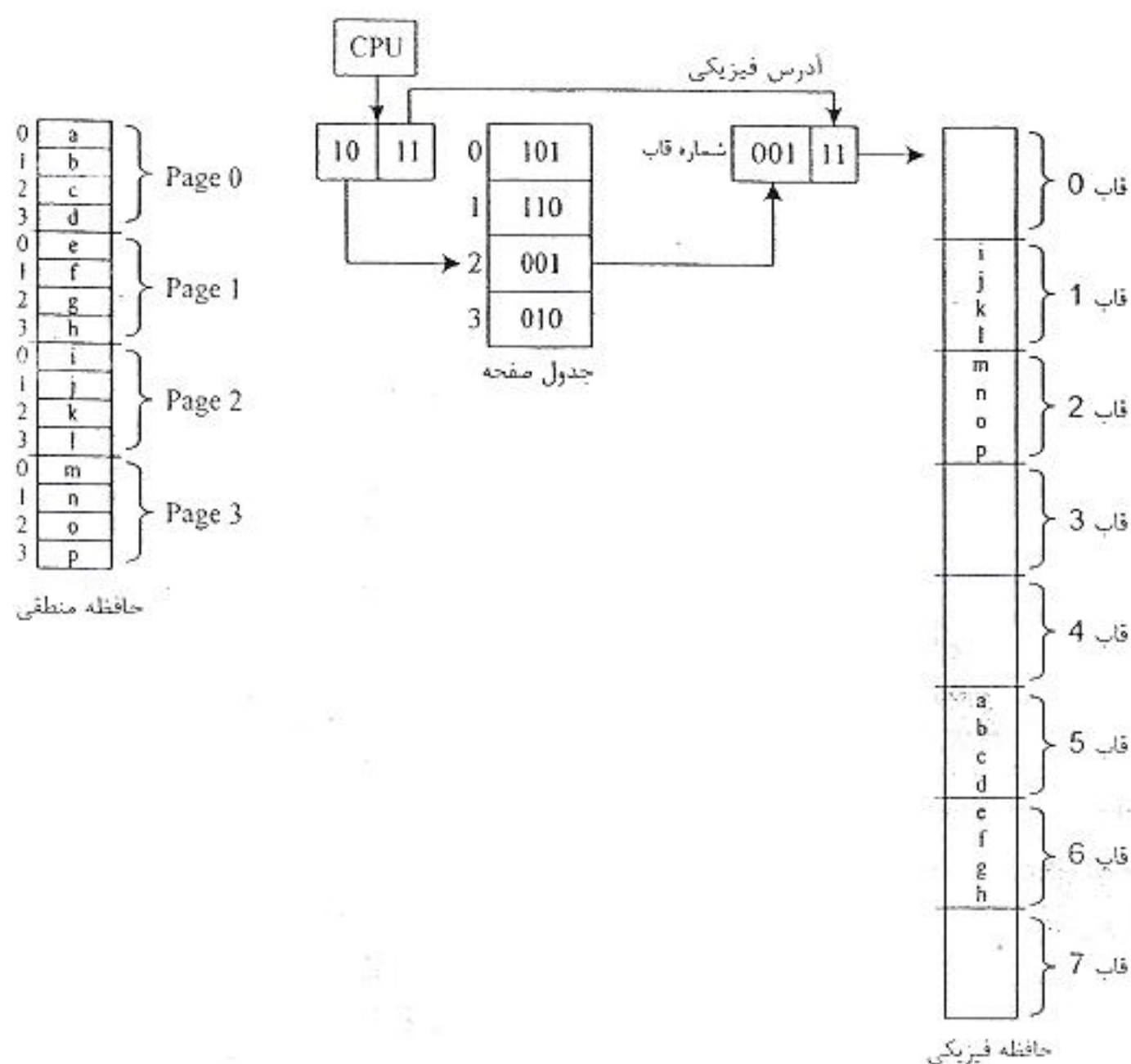
2^n = اندازه حافظه اصلی

سؤال) اگر برنامه ای از ۴ صفحه منطقی تشکیل شده باشد و به صورت زیر در قابهای حافظه فیزیکی قرار گرفته باشند جدول صفحه به چه صورت است؟

شماره قاب	
0	
1	Page 0
2	
3	Page 2
4	Page 1
5	
6	
7	Page 3

حافظه فیزیکی

نکته: اندازه صفحه همانند اندازه قاب توسط سخت افزار معین می گردد و این اندازه ها همواره توان های ۲ بوده و پس ۵۱۲ بایت تا ۱۶ مگابایت تغییر می کند. در مثال شکل زیر فرض بر این است که تعداد صفحات $2^2 = 4$ عدد و تعداد قاب ها $2^3 = 8$ عدد باشد.



معایب روش صفحه بندی:

(1) اتلاف حافظه

الف) تکه تکه شدن داخلی: چنانچه اندازه فرایند مضرب صحیحی از اندازه بلاک نباشد مقداری از فضای آخرین بلاک اختصاص داده شده به فرایند، خالی می ماند به این پدیده تکه تکه شدن داخلی می گویند که مقدار متوسط این اتلاف حافظه $\frac{P}{2}$ است.

ب) فضای مورد نیاز جدول صفحه: برای ذخیره سازی جدول مقداری حافظه اشغال می شود که میزان این اتلاف به ازای هر فرایند $\frac{S}{P} \times e$ متوسط سایز فرایندها، P اندازه صفحه، e اندازه هر مدخل جدول صفحه)

ج) کل اتلاف حافظه: $\frac{P}{2} + \frac{S}{P} \times e$

(2) کاهش سرعت دسترسی

تست: اگر بلوک های خالی حافظه به ترتیب به صورت 39k و 45k و 7k و 9k و 13k باشد و *Head* یا نقطه دسترسی به بلوک های خالی حافظه، بلوک 39k باشد و درخواست های جدیدی توسط فرایندها برای چهار بلوک خالی به ترتیب از راست به چپ و اندازه 18k و 22k و 11k و 13k صادر شود آن گاه وضعیت بلوک های خالی حافظه پس از تخصیص توسط سیستم عامل با استفاده از الگوریتم *NEXT FIT* کدام است؟

(۲) 2K, 9K, 7K, 12K, 21K

(۱) 13K, 9K, 7K, 10K, 10K

(۴) 3K, 9K, 7K, 12K, 8K

(۳) 9K, 7K, 12K, 21K

تست: در زیر فقط بلوکهای خالی حافظه به ترتیب از چپ به راست نشان داده شده است:

Head → 62, 82, 35, 70, 53

اگر فرایندهای جدیدی به ترتیب از چپ به راست با اندازه های ۱۲ و ۵۰ و ۴۷ و ۲۳ بخواهند وارد حافظه شوند و سیستم از الگوریتم *Worst Fit* استفاده نماید، وضعیت بلوک های خالی حافظه بعد از تخصیص کدام است؟

(۲) Head → 50, 49, 35, 23, 53

(۱) Head → 15, 82, 23, 37, 3

(۴) Head → 17, 35, 35, 20, 53

(۳) Head → 12, 49, 35, 23, 41

تست: رجیستر پایه (*Base Register*) برای رفع کدام نیاز مدیریت حافظه در پردازنده قرار داده شده است؟

(۴) جابه جایی

(۳) اشتراک

(۲) تکه تکه شدن حافظه

(۱) حفاظت

تست: در یک سیستم که مدیریت حافظه با استفاده از مبادله انجام می شود، حافظه اصلی شامل فضاهای خالی با

اندازه های به ترتیب زیر (از چپ به راست) است:

10k, 4k, 20k, 18k, 7k, 9k, 12k, 15k

برای درخواست تکه هایی از حافظه به طور متوالی و به مقدارهای 8k, 10k, 12k (از راست به چپ) با استفاده از

روش *First Fit* کدام یک فضاهای خالی فوق الذکر استغال می شوند؟

(۲) 18k, 10k, 20k

(۱) 20k, 10k, 20k

(۴) 9k, 18k, 20k

(۳) 10k, 18k, 20k

تست: در سیستمی بلوکهای خالی حافظه عبارتند از: $m_1 = 40k$, $m_2 = 20k$, $m_3 = 70k$, $m_4 = 12k$ چهار پردازش به ترتیب از p_1 تا p_4 با اندازه های زیر وارد حافظه می شوند: $p_1 = 19k$, $p_2 = 20k$, $p_3 = 6k$, $p_4 = 13k$ اگر الگوریتم تخصیص *Best Fit* باشد، هر پردازش در کدام فضای خالی قرار می گیرد؟

- (۱) p_1 در m_2 و p_2 در m_1 و p_3 در m_4 و p_4 در m_1
- (۲) p_1 در m_1 و p_2 در m_2 و p_3 در m_4 و p_4 در m_3
- (۳) p_1 در m_2 و p_2 در m_1 و p_3 در m_4 و p_4 در m_3
- (۴) p_1 در m_1 و p_2 در m_2 و p_3 در m_3 و p_4 در m_4

تست: سیستمی 256k حافظه اصلی خالی دارد و از الگوریتم *Buddy* جهت تخصیص استفاده می کند. پردازش های زیر به ترتیب از چپ به راست و با اندازه های مشخص شده وارد سیستم می شوند:

$$p_1 = 6k, p_2 = 53k, p_3 = 37k, p_4 = 23k$$

اندازه بلوک های باقی مانده حافظه کدام است؟

- (۱) 8K, 8K, 32K, 8K, 32K
- (۲) 8K, 16K, 32K
- (۳) 4K, 8K, 128K
- (۴) 8K, 16K, 64K

تست: سیستمی برای نگهداری فضای حافظه از لیست پیوندی و همچنین از الگوریتم تخصیص حافظه *Next FIT* استفاده می کند اگر فضای خالی حافظه از *Head* به ترتیب از راست به چپ برابر با 8k, 7k, 12k, 3k باشد آنگاه در صورت ورود فرایندهای جدید (از راست به چپ) با اندازه های 4KB, 5KB, 1KB, 9KB, 4KB تعداد فضاهای خالی حافظه (اعضای لیست پیوندی که برچپ H دارند) کدام است؟

- (۱) ۲
- (۲) ۱
- (۳) ۴
- (۴) ۳

«مبحث: حافظه مجازی»

حافظه مجازی: حافظه مجازی تکنیکی است که اجازه می دهد پردازش ها بدون آنکه لازم باشند به طور کامل در حافظه اصلی قرار گیرند، به بیان دیگر می توان حافظه مجازی را به اینگونه تعریف کرد: حافظه مجازی یعنی به کارگیری حافظه جانبی در کنار حافظه اصلی به منظور اینکه فقط آن قسمتی از هر برنامه که برای اجرا مورد نیاز است وارد حافظه اصلی شده و مابقی آن در حافظه مجازی (روی دیسک) باقی بماند. در حین اجرا چنانچه قسمتی از برنامه نیاز باشد که در حافظه اصلی موجود نباشد باید ابتدا از حافظه مجازی به حافظه اصلی آورده شود.

مزایای حافظه مجازی:

- (۱) عدم محدودیت اندازه فرایند به اندازه حافظه اصلی
- (۲) از آنجایی که فقط بخشی از هر فرایند به حافظه اصلی آورده می شود، می توان تعداد بیشتری فرایند به حافظه اصلی آورد که باعث افزایش درجه چند برنامگی و افزایش کارایی سیستم می گردد.
- (۳) عدم انجام I/O بیهوده (قبل از نیاز)

روش صفحه بندی نیازی (Demand Paging):

تکنیک صفحه بندی نیازی ترکیبی از تکنیک صفحه بندی و مبادله می باشد. در تکنیک مبادله کل پردازش بین دیسک و حافظه جابه جا می شود ولی در صفحه بندی نیازی فقط صفحانی از پردازش که واقعاً مورد نیاز می باشند به حافظه آورده می شوند.

این سیستم نیاز به امکانات سخت افزاری ویژه ای دارد تا مشخص شود کدام صفحه در حافظه و کدام صفحه روی دیسک قرار دارد. معمولاً برای اینکار از بیت *Valid-Invalid* استفاده می گردد.

0	A
1	B
2	C
3	D
4	E
5	F

Logical Memory

0	4	v
1		i
2	6	v
3		i
4		i
5	9	v
6		i
7		i

Page Table

0	
1	
2	
3	
4	A
5	
6	C
7	
8	
9	F
10	
	⋮
n	

Physical Memory

تعریف نقص صفحه یا فقدان صفحه (*Page Fault*):

چنانچه فرآیند صفحه ای را بخواهد (به صفحه ای مراجعه کند) که در حافظه اصلی موجود نباشد نقص صفحه یا *Fault* رخ داده است.

رشته مراجعه: به دنباله ی شماره صفحه هایی که فرآیند در طی اجرائش به آن ها مراجعه کرده رشته مراجعه گویند.
جایگزین صفحه (*Page Replacement*):

به هر فرآیند تعداد محدودی بلاک داده می شود تا زمانی که فرآیند بلاک های اختصاص داده به خود را پر نکرده باشد سیستم عامل صفحه های مورد نیاز فرآیند را در حافظه اصلی بار (*Load*) می کند. اما هنگامی که بلاک های در اختیار یک فرآیند توسط صفحه هایش اشغال شد، چنانچه فرآیند بخواهد صفحه دیگری را به حافظه اصلی بیاورد ابتدا باید یکی از صفحه های موجود در حافظه را از حافظه خارج کرده و صفحه جدید را با آن جایگزین کند.

الگوریتم های جایگزین صفحه:

(1) *FIFO*

(2) بهینه یا *OPT*

(3) *LRU*

(4) *Aging* یا سالمندی

(5) دومین شانس *Second chance*

(6) الگوریتم صفحه ساعت (*clock page*)

(7) *NRU*

(8) *LFU*

(9) *MFU*

(1) الگوریتم *FIFO*: این الگوریتم ساده ترین روش جایگزینی صفحه است در این روش صفحه ای برای جایگزینی انتخاب می شود یا به بیان دیگر صفحه ای که طولانی ترین مدت در حافظه بوده است.
مثال: رشته مراجعه زیر را از چپ به راست در نظر بگیرید:

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

با الگوریتم *FIFO* در دو حالت با تعداد قابهای سه و چهار تعداد خطاها را بدست آورید؟

نکته: گاهی اگر تعداد قابها را زیاد کنیم، برخلاف انتظار تعداد نقص صفحه ها نیز افزایش یافته است، این مساله به نام ناهنجاری بلیدی (*Belady's anomaly*) معروف است. یعنی در بعضی از الگوریتم های جایگزینی افزایش تعداد قابها به جای کاهش، سبب افزایش تعداد نقص صفحه ها می گردد.

نست: پردازش ای به 5 صفحه A, B, C, D, E به ترتیب از چپ به راست مراجعه دارد:

استفاده قرار گیرد. تعداد خطای صفحه در طی این رشته مراجعات در حالت ۴ حالت ۴ قاب صفحه به فرض خالی بودن حافظه اصلی در شروع کار، چقدر است؟

(۱) ۶ (۲) ۸ (۳) ۱۰ (۴) ۱۲

(۲) الگوریتم بهینه یا *OPT (Optimal)*: این الگوریتم در هنگام نقص صفحه، صفحه ای را برای جایگزین انتخاب می کند که تا اولین مراجعه بعدی به آن در آینده، بیشترین فاصله باقی مانده است. این الگوریتم به دلیل نیاز به دانستن مراجعات بعدی فرآیند قابل پیاده سازی نبوده و فقط به عنوان یک معیار یا ملاک است. به بیان دیگر این الگوریتم صفحه ای را جایگزین می کند که برای بیشترین پریود زمانی در آینده مورد استفاده نخواهد بود.

(۳) الگوریتم *LRU (Least Recently Used)*: در این الگوریتم صفحه ای جایگزین می شود که طولانی ترین پریود زمانی مورد استفاده قرار نگرفته است. الگوریتم *LRU* همانند احتمال مشکل ناهنجاری بلیدی را ندارد. تست: چنانچه رشته مراجعه ی یک فرآیند به صفحه هایش به صورت زیر باشد و از الگوریتم جایگزینی صفحه *LRU* استفاده شود، با داشتن ۴ قاب صفحه و با فرض این که در شروع کار حافظه خالی است، درصد خطای صفحه چقدر است؟
 $1, 2, 3, 2, 1, 4, 5, 2, 1, 3 \rightarrow$

(۱) ۳۰٪ (۲) ۴۰٪ (۳) ۵۰٪ (۴) ۶۰٪

(۴) الگوریتم سالمندی (*Aging*): این الگوریتم در واقع *LRU* را شبیه سازی می کند. یک شمارنده (غالباً ۸ بیتی) را برای هر صفحه، در جدولی نگه می داریم. در فاصله های مرتب (مثلاً هر ۱۰۰ میلی ثانیه) یک وقفه تایمر، کنترل سیستم را به سیستم عامل انتقال می دهد. در این حال شمارنده صفحات یک بیت به سمت راست شیفت داده می شوند. بیت سمت راستی از بین رفته و بیت سمت چپی با بیت *R* مربوط به آن صفحه پر می شود. مثلاً اگر شمارنده مربوط به صفحه *A* برابر 11101001 باشد و بیت *R* آن صفحه نیز ۱ باشد آنگاه پس از وقفه تایمر محتوای شمارنده مذکور برابر 11101000 می گردد. یا مثلاً اگر شمارنده صفحه ای در حال حاضر 10110000 و بیت آن (*R*) برابر صفر باشد، پس از وقفه تایمر محتوای آن شمارنده برابر 01011000 خواهد شد در واقع این شیفت رجیسترهای ۸ بیتی تاریخچه استفاده از هر صفحه را برای هشت پریود اخیر نگه می دارد.

نکته: در برخی از الگوریتم های جایگزینی صفحه از دو بیت وضعیت *R* و *M* که به صورت سخت افزاری در اکثر سیستم ها وجود دارند استفاده می کنیم. این دو بیت در همه سطرهای جدول صفحه و به ازای هر صفحه وجود دارند. هرگاه به صفحه ای رجوع شود (جهت خواندن یا نوشتن) بیت *R* یا بیت مراجعه (*Referenced*) آن صفحه "۱" می شود. بیت *M* یا بیت تغییر (*Modified*) وقتی "۱" می شود که محتوای صفحه تغییر کند.

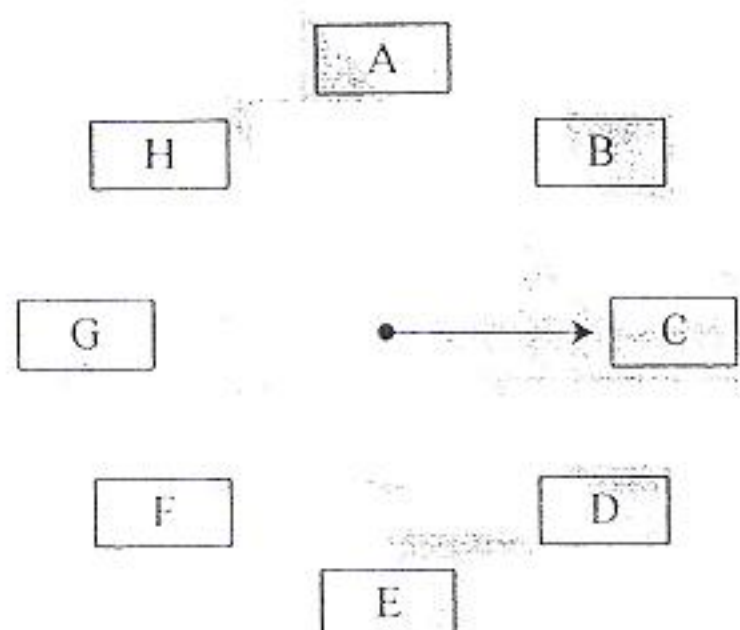
(۵) الگوریتم دومین شانس (*second chance*): این الگوریتم باید تغییر کوچک در الگوریتم *FIFO* موجب می گردد صفحاتی که زیاد استفاده شده اند، از حافظه خارج نسوند. در این روش از اول صف شروع کرده و بیت *R* قدیمی ترین صفحه بررسی می شود اگر این بیت 0 باشد، یعنی صفحه هم قدیمی و هم استفاده ای از آن نشده، لذا بلافاصله این صفحه یا صفحه جدید جایگزین می گردد. ولی اگر بیت *R* آن 1 باشد، آن بیت صفر شده، صفحه به انتهای لیست انتقال یافند و زمان

ورودش به زمان فعلی مقدار دهی می شود (مانند اینکه آن صفحه تازه وارد حافظه شده باشد). در واقع بدین ترتیب به آن صفحه شانس دومی برای باقی ماندن در حافظه داده شده است.

الگوریتم دومین شانس مانند *FIFO* مشکل ناهنجاری بلیدی را دارد.

۶) الگوریتم صفحه ساعت (*Clock page*): جابه جایی صفحات در لیست مربوط به الگوریتم دومین شانس، باعث

می گردد که این روش زیاد مناسب نباشد. یک پیاده سازی دیگر آن است که صفحات را در یک لیست حلقوی مشابه یک ساعت نگهداری کنیم، به گونه ای که عقربه ساعت به قدیمی ترین صفحه اشاره می کند:



هنگام رخ دادن نقص صفحه، بیت R صفحه ای که عقربه آن اشاره می کند بررسی می شود. اگر بیت 0 باشد صفحه مذکور خارج می شود و صفحه جدید در همان نقطه جایگزین آن شده و عقربه نیز یک خانه جلو می رود. اگر بیت R برابر 1 باشد، آنگاه این بیت 0 شده و عقربه به خانه بعدی می رود. این روند همچنان ادامه می یابد تا زمانی که صفحه ای با بیت R مساوی صفر جهت خروج پیدا شود.

۷) الگوریتم *NRU (Not Recently Used)*: در این الگوریتم از هر دو بیت R و M استفاده می شود هنگامی که پردازش شروع می شود، سیستم عامل این دو بیت را برای تمام صفحات آن پردازش صفر می کند. بعد از آن بیت R به صورت متناوب مثلاً در هر ۲۰ میلی ثانیه وقفه تایمر، صفر می شود تا صفحه هایی که اخیراً به آنها دسترسی نشده است از سایر صفحات تشخیص داده شوند. هنگامی که وقفه نقص صفحه رخ می دهد سیستم عامل بر اساس مقادیر (R, M) صفحات را به چهار دسته زیر تقسیم می کند:

کلاس ۰: $(R, M = 0, 0)$ یعنی صفحاتی که اخیراً استفاده نشده اند و تغییر هم نکرده اند. این صفحات بهترین انتخاب برای جایگزینی می باشد.

کلاس ۱: $(R, M = 0, 1)$ یعنی صفحاتی که اخیراً استفاده نشده اند ولی تغییر یافته اند. این صفحات برای جایگزینی کاملاً خوب نیستند، چرا که قابل جایگزینی لازم است بر روی دیسک نوشته شوند.

کلاس ۲: $(R, M = 1, 0)$ یعنی صفحاتی که اخیراً استفاده شده اند اما تغییر نکرده اند. این صفحات احتمالاً دوباره

به زودی مورد نیاز خواهند بود.

کلاس ۳: $(R, M=1, 1)$ یعنی صفحاتی که اخیراً استفاده شده اند و تغییر نیز کرده اند. این صفحات را حتی الامکان نباید خارج کرد.

تست: سیستم عاملی از الگوریتم NRU استفاده می کند با توجه به جدول زیر در صورت رخداد نقص صفحه، صفحه جدید با کدام صفحه درون حافظه فیزیکی جایگزین می شود؟ (آزاد ۸۴)

R	M	شماره صفحه
0	1	1
1	0	2
1	1	3

(۱) ۲

(۲) ۱

(۳) ۳

(۴) اطلاعات داده شده کافی نمی باشد.

شکل دیگری از الگوریتم NRU به این صورت است که فقط از بیت R استفاده شود. هنگام دسترسی به صفحه مقدارش "۱" می گردد. الگوریتم به صورت $FIFO$ و چرخه ای صفحات را بررسی می کند. مثال زیر این حالت را نشان می دهد:

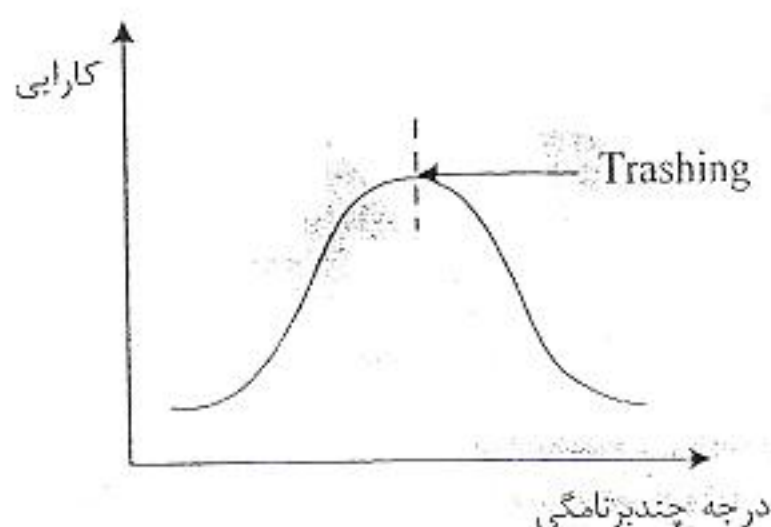
	1	2	3	4	5	1	2	6	7	1	2	3	7	8
قاب ۱	1R	1R	1R	1R	1R	1R	1R	6R	6R	6R	6R	6R	6R	8R
قاب ۲		2R	2R	2R	2R	2R	2R	2	7R	7R	7R	7R	7R	7
قاب ۳			3R	3R	3R	3R	3R	3	3	1R	1R	1R	1R	1
قاب ۴				4R	4R	4R	4R	4	4	4	2R	2R	2R	2
قاب ۵					5R	5R	5R	5	5	5	5	3R	3R	3
نقص صفحه	*	*	*	*	*			*	*	*	*	*	*	*

۸) الگوریتم LFU (Least Frequently Used): در این روش یک شمارنده نرم افزاری به هر صفحه نسبت داده می شود. مقدار همه شمارنده ها در ابتدا صفر است. در هر وقفه ساعت، سیستم عامل بیت R هر صفحه را (چه صفر و چه ۱) با مقدار شمارنده مربوط به آن صفحه جمع می کند. بدین ترتیب هر شمارنده نمایانگر آن است که صفحه مربوط چندبار دستیابی شده است. هنگامی که نقص رخ می دهد، صفحه ای که شمارنده آن کمترین مقدار را دارد جهت خروج از حافظه انتخاب می گردد.

۹) الگوریتم MFU (Most Frequently Used): در این الگوریتم برعکس LFU صفحه ای که شمارنده آن بیشترین مقدار را دارد جهت جایگزینی انتخاب می شود. منطق این روش آن است که صفحه با کوچکترین شمارش، احتمالاً نازده وارد سیستم شده و هنوز در حال استفاده می باشد.

مفهوم کوبیدگی (Thrashing):

با افزایش درجه چند برنامگی تعداد قاب صفحه هایی که به هر فرآیند اختصاص می یابد، کاهش پیدا می کند بنابراین تعداد نقص صفحه فرآیند ها افزایش یافته و عمده زمان سیستم صرف مبادله صفحه ها می گردد. در نتیجه پردازش صورت نگرفته و کارایی کاهش می یابد به این پدیده کوبیدگی یا Thrashing گویند.



اندازه صفحه: اگر فقط فاکتورهای اندازه جدول صفحه و پدیده تکه تکه شدن داخلی را در نظر بگیریم می توانیم به یک فرمول مشخص برای اندازه صفحه برسیم:

میانگین اندازه پردازش ها را S بایت و اندازه صفحه را P بایت می گیریم. همچنین فرض کنید که هر خانه جدول صفحه c بایت باشد. آنگاه تعداد صفحات مورد نیاز هر پردازش $\frac{S}{P}$ و جدول صفحه آن $\frac{Se}{P}$ بایت خواهد بود. فضای به هدر رفته بر اثر پدیده تکه تکه شدن داخلی نیز به طور متوسط $\frac{P}{2}$ بایت است. بنابراین کل فضای به هدر رفته بر اثر جدول صفحه و تکه تکه شدن داخلی برابر $H = \frac{Se}{P} + \frac{P}{2}$ می باشد. جهت یافتن حداقل تابع H بر حسب P ، از تابع H نسبت به P مشتق می گیریم:

$$H'_P = -\frac{Se}{P^2} + \frac{1}{2} = 0 \Rightarrow P^2 = 2Se \Rightarrow P = \sqrt{2Se}$$

« برخی از تست های مهم آزمونهای اخیر »

تست : سیستمی از تکنیک حافظه مجازی استفاده می کند. اگر اندازه فرآیندی 2KB و اندازه حافظه فیزیکی 3KB و اندازه هر صفحه 256 بایت باشد و اگر این فرآیند فقط چهار صفحه فیزیکی در اختیار داشته باشد و در ابتدا هیچ صفحه ای از فرآیند در حافظه فیزیکی نباشد آنگاه با استفاده از سیاست جایگزینی صفحه *FIFO* تعداد نقص صفحه کدام است در صورتیکه فرآیند از چپ به راست به آدرسهای موجود در حافظه مجازیش رجوع نماید. (آزاد ۸۳)

540 , 1700 , 275 , 280 , 783 , 1652 , 1520 , 1700 , 540 , 805 , 1400 , 1222 , 304 , 805

(۴) ۹

(۳) ۸

(۲) ۶

(۱) ۵

تست : سیستمی از تکنیک صفحه بندی حافظه مجازی استفاده می کند. حافظه فیزیکی سیستم 356KB است. اگر حداکثر برنامه قابل اجرا بتواند 2MB فضای حافظه را آدرس دهی نماید و تعداد درایه های جدول صفحه ۲۵۶ باشد آنگاه اندازه صفحه چقدر است ؟ (آزاد ۸۳)

(۲) 8KB

(۱) 16KB

(۴) اطلاعات داده شده برای بدست آوردن اندازه صفحه کافی نیست.

(۳) 4KB

تست : سیستمی از تکنیک صفحه بندی حافظه مجازی استفاده می کند. اگر اندازه صفحات ۱۰۲۴ بایت باشد و چهار فرآیند P1 , P2 , P3 , P4 با اندازه های 2048 , 1064 , 1800 , 2412 در حافظه فیزیکی RAM باشد و حجم حافظه فیزیکی 10KB باشد آنگاه متوسط مقدار تکه تکه شدن داخلی بر حسب بایت برای هر فرآیند کدام است ؟ (آزاد ۸۳)

(۴) ۴۷۳

(۳) ۷۲۹

(۲) ۸۰۷

(۱) ۴۲۶

تست : اگر حداکثر ۶۴ صفحه ۸ کیلو بیتی را در حافظه فیزیکی RAM بتوان قرار داد و حداکثر برنامه قابل اجرا 2MB باشد آنگاه تعداد سطرهای جدول صفحه کدام است ؟ (آزاد ۸۴)

(۴) ۶۴

(۳) ۵۱۲

(۲) ۲۵۶

(۱) ۱۲۸

تست : در صورتی که تعداد سطرهای یک جدول صفحه ۵۱۲ باشد ، اندازه حافظه فیزیکی RAM برابر 64MB و حداکثر برنامه قابل اجرا در کامپیوتر 2GB باشد آنگاه شماره قاب صفحه چند بیتی است ؟ (علمی کاربردی آزاد ۸۴)

(۴) ۸

(۳) ۴

(۲) ۷

(۱) ۵

تست : محیط یک سیستم عامل که از روش حافظه مجازی (Virtual Memory) برای مدیریت حافظه استفاده می کند ، چنانچه اندازه هر پروسس 64KB و برای هر پروسس در Page Table هست بایت اطلاعات ذخیره شده باشد ، اندازه Page بهینه در این سیستم چقدر است ؟

(۴) ۵۱۲ بایت

(۳) ۲۰۴۸ بایت

(۲) ۱۰۲۴ بایت

(۱) ۷۲۴ بایت

تست: سیستمی از روش جایگزینی صفحه *NRU* (*Not Recently Used*) استفاده می کند. اگر چهار صفحه P_1, P_2, P_3, P_4 مطابق جدول زیر در *RAM* باشند آنگاه در صورت وقوع نقص صفحه، صفحه جدید با کدام صفحه جایگزین می شود؟ (آزاد ۸۵)

صفحه	R بیت ارجاع	M بیت تغییر	زمان بارگذاری در حافظه (بالس ساعت)
P_1	1	1	48
P_2	1	1	92
P_3	1	0	66
P_4	0	1	85

 P_3 (۱) P_2 (۲) P_4 (۳) P_1 (۴)

تست: سیستمی از روش جایگزینی صفحه *LFU* (*Least Frequently Used*) استفاده می کند. اگر از سمت چپ به راست به شماره صفحات مجازی در برنامه ای ارجاع شود و تعداد قاب صفحه ۳ عدد و در ابتدا هر سه خالی باشند آنگاه جایگزینی صفحه چند بار رخ می دهد؟ (آزاد ۸۵)

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

۷(۴)

۵(۳)

۳(۲)

۴(۱)

تست: پیش صفحه بندی (*Prepaging*) در صفحه بندی به چه معناست؟ (آزاد ۸۵)

(۱) قبل از اجرای یک فرآیند، صفحات مورد نیاز اولیه فرآیند در *RAM* آورده می شود.

(۲) هرگاه به صفحه ای از فرآیند، ارجاع شود که در *RAM* نباشد فقط آن صفحه به درون *RAM* آورده می شود.

(۳) قبل از اجرای یک فرآیند، کل صفحات آن درون *RAM* آورده می شود.

(۴) در صورت اجرای دستورات یک فرآیند به صورت پی در پی سیستم دچار نقص صفحه می شود.

« جزوه میحث بن بست (Dead Locks) »

Resource: هر موجودیتی که یک فرایند برای ادامه اجرا به آن نیاز داشته باشد، منبع نام دارد.

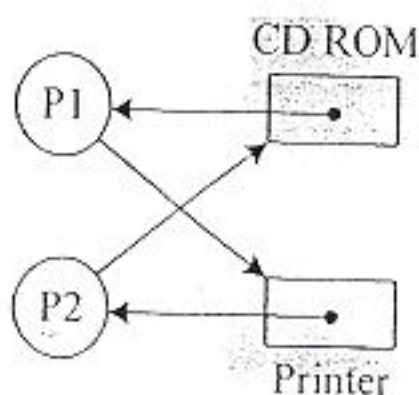
مراحل استفاده از منبع:

✓ **Request** (درخواست)

✓ بکارگیری منبع

✓ رهاسازی منبع

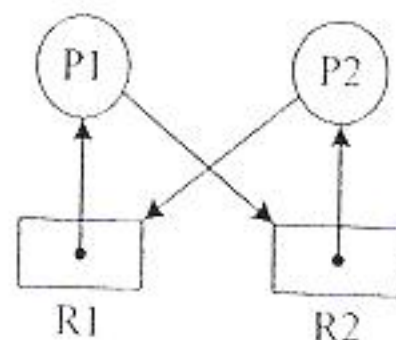
تعریف بن بست (**Dead Locks**): مجموعه ای از فرایندها در حالت بن بست قرار دارند، اگر هر فرایند منتظر آزاد شدن منبعی از فرایند دیگر از همین مجموعه باشد اگر زمان آزاد شدن به صورت نامحدود شود حتماً بن بست وجود خواهد داشت.



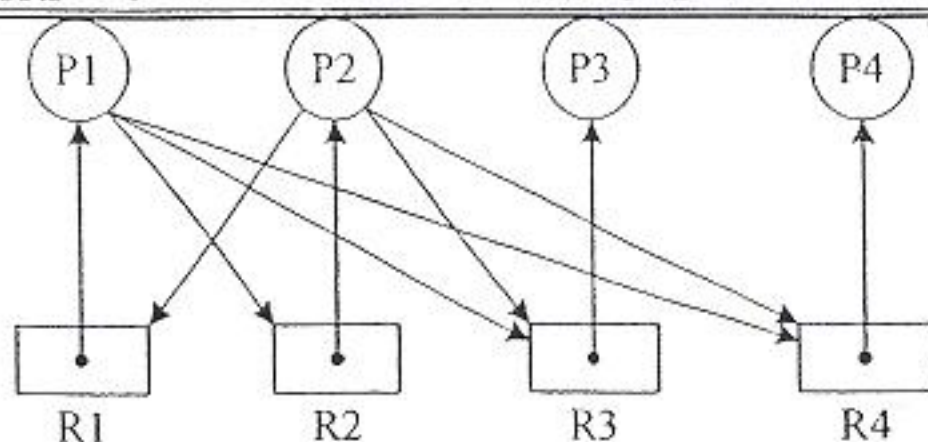
شرایط ۴ گانه کافمن برای وقوع بن بست:

هرگاه شرایط ۴ گانه کافمن در سیستمی برقرار باشد احتمال وقوع بن بست وجود دارد (وقوع بن بست حتمی نیست) اما هنگامی که بن بست رخ می دهد حتماً شرایط ۴ گانه کافمن برقرار بوده اند این شرایط به ترتیب زیر می باشند.

(۱) **انحصار متقابل (Mutual Exclusion)**: یعنی حداقل یک منبع باید غیر قابل اشتراک باشد. یعنی اگر منبع در اختیار پردازش بود و پردازش دیگری درخواست آن را کرد، باید پردازش درخواست کننده به تأخیر بیفتد تا منبع آزاد شود. به عبارت دیگر فقط یک پردازش در هر زمان می تواند از منبع استفاده کند.

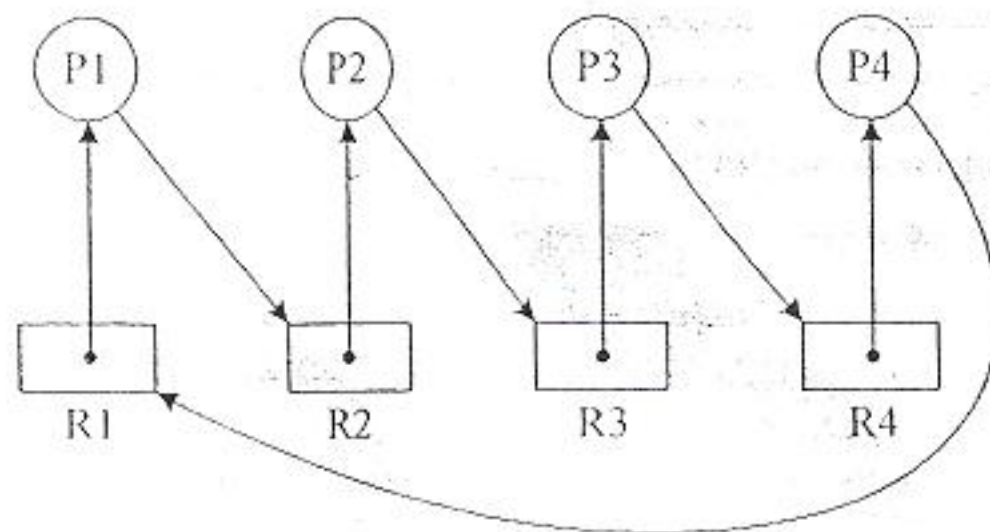


(۲) **گرفتن و منتظر ماندن (Hold and Wait)**: باید پردازشی وجود داشته باشد که حداقل یک منبع را گرفته و در حال انتظار برای کسب منابع اضافی تر (که در حال حاضر در اختیار پردازش های دیگر است) باشد.



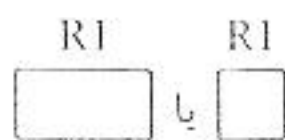
۳) **انحصاری بودن (No Preemption):** منبع را نمی توان به اختیار از پردازش گرفت و پردازش پس از اتمام کارش داوطلبانه آن را رها می کند.

۴) **انتظار چرخشی (Circular Wait):** بایستی مجموعه ای از پردازش ها $\{P_0, P_1, \dots, P_n\}$ وجود داشته باشد به طوری که P_0 منتظر منبعی از P_1 ، P_1 منتظر منبعی از P_2 و P_n منتظر منبعی از P_0 باشد.



گراف تخصیص منبع (Resource Allocation Graph):

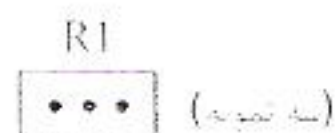
بن بست ها را می توان توسط گراف جهت داری به نام گراف تخصیص منبع نمایش داد، در این گراف گره ها از پردازش ها و منابع تشکیل شده اند. پردازش ها بصورت دایره و منابع به شکل مربع یا مستطیل ترسیم می شوند، تعداد نمونه های یک منبع را بصورت نقطه درون مستطیل ها نمایش می دهیم.



نمونه ای از R1 به پردازش P1 تخصیص یافته

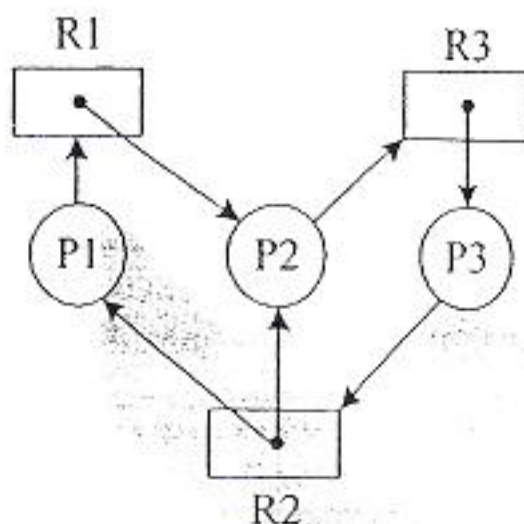


پردازش P1، نمونه ای از منبع R1 را درخواست کرده



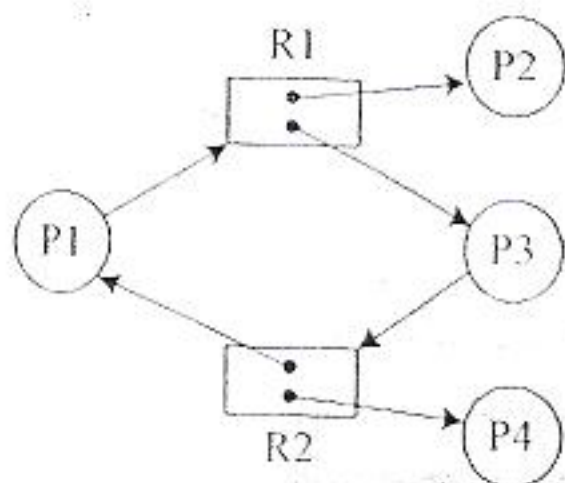
روش های کشف:

- (۱) اگر از هر منبع فقط یکی (نمونه) در سیستم وجود داشته باشد، وجود سیکل جهت دار در گراف منابع، شرط لازم و کافی برای وقوع بن بست است. پردازش های قرار گرفته در سیکل در حالت بن بست می باشد.



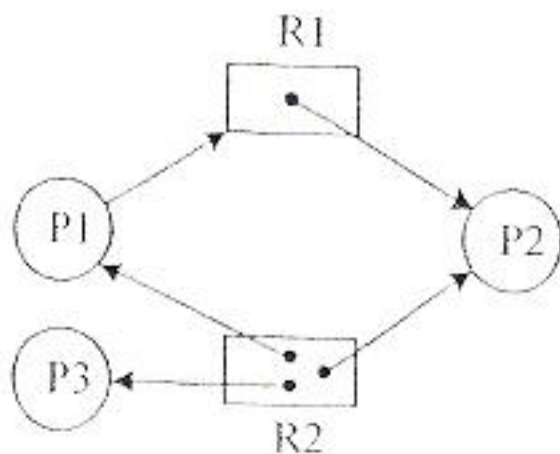
- (۲) اگر حداقل یکی از منابع بیش از یک نمونه داشته باشد، وجود سیکل جهت دار، شرط لازم برای وقوع بن بست بوده اما کافی نیست چنانچه سیکل، غیر قابل شکسته شدن باشد، آنگاه بن بست رخ داده است و پردازش های موجود در سیکل در بن بست هستند، در غیر این صورت بن بست رخ نداده است.
- نکته مهم: فرض بر این است که وقتی فرآیندی همه منابع مورد نیاز خود را در اختیار داشته باشد به سرعت پردازشش را انجام داده و کارش تمام شده و منابع در اختیارش را آزاد کرده و بیرون می رود.

تست: با توجه به گراف زیر کدام گزینه صحیح می باشد؟



- (۱) در این نمودار حلقه وجود ندارد
- (۲) در این نمودار حلقه و بن بست وجود ندارد.
- (۳) در این نمودار هم حلقه و هم بن بست وجود دارد.
- (۴) در این نمودار حلقه هست اما بن بست وجود ندارد.

تست: با توجه به گراف زیر کدام گزینه صحیح می باشد؟



- (۱) سیستم در بن بست قرار دارد.
- (۲) سیستم در حالت امن است.
- (۳) سیستم در حالت نا امن است.
- (۴) وضعیت نا مشخص می باشد.

تست: فرض کنید منابع مورد نیاز و در اختیار هر پروسس را به صورت زیر نمایش دهیم:



با توجه به پردازش های زیر و وضعیت سیستم کدام گزینه است؟

P ₁	P ₂	P ₃	P ₄
R ₁ R ₄ R ₂ R ₅ R ₃	R ₆ R ₁₀ R ₇ R ₉	R ₄ R ₆ R ₈ R ₁₂ R ₉	R ₁₀ R ₂ R ₁₁ R ₉

(۱) بستگی به ترتیب برآورده کردن درخواست پروسس ها دارد

(۲) سیستم در شرایط امن است

(۳) ممکن است بن بست اتفاق بیفتد

(۴) بن بست اتفاق افتاده است

روش های ترمیم:

(۱) کشتن پردازش (Kill)

در این روش OS یکی از پردازش های موجود در سیکل را انتخاب کرده، آن را از سیستم بیرون انداخته و منابع آزاد شده اش را در اختیار فرایندهای موجود در سیکل قرار می دهد. این عمل تا شکسته شدن کامل سیکل ادامه دارد. معیارهای انتخاب قربانی:

- ۱- پردازشی که منابع بیشتری را در اختیار دارد.
- ۲- پردازشی که زمان کمتری از آغاز اجرای آن می گذرد.
- ۳- فرایندی که پاسخ گویی به درخواستش منجر به وقوع بن بست شده است.
- ۴- اولویت فرایندها

(۲) بازپس گیری منابع

در این روش، فرایند انتخاب شده از سیستم بیرون انداخته نمی شود و فقط منابعش را پس گرفته و به سایر فرایندهای موجود در سیکل می دهند.

راه های مدیریت بن بست (نحوه اداره بن بست)

- (۱) پیشگیری یا جلوگیری از بن بست (dead lock prevention): منظور این است که روشی را به کار ببریم که یکی از چهار شرط لازم برای وقوع بن بست پدید نیاید. بدین ترتیب هرگز بن بست رخ نخواهد داد.
- (۲) اجتناب از بن بست (dead lock avoidance): در این روش درخواستها و رهایی منابع آنی بررسی می شود و تصمیم گرفته می شود که آیا درخواست جاری اگر پاسخ داده شود، منجر به بن بست خواهد شد یا خیر؟ در صورتی که پاسخ به درخواست جاری در آینده منجر به بن بست شود، این درخواست به تعویق انداخته می شود.

۳) آشکار سازی و بازیافت (*detection and recovery*): در این سیستم نه الگوریتم های پیشگیری و نه الگوریتم های اجتناب به کار گرفته می شود و لذا ممکن است بن بست رخ دهد. در این روش سیستم، بررسی شده و در صورت بروز بن بست، تشخیص داده شده و الگوریتم دیگری سیستم را از بن بست خارج می کند.

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

نکته: یکی از روش های پیشگیری از بن بست در حالت انتظار چرخشی به این صورت می باشد که به هر نوع منبع، یک شماره یکتا داده می شود و سیستم کاری می کند که هر پردازش فقط بتواند منابع را در جهت صعودی شماره هایشان درخواست کند. مثلاً اگر شماره $CD-ROM = 1$ و شماره $HARD = 7$ و شماره $Printer = 15$ باشد آنگاه اگر پردازشی هارد را در اختیار داشته باشد نمی تواند $CD-ROM$ را نیز تقاضا کند و فقط می تواند پرینتر را تقاضا کند.

الگوریتم بانکدار (Banker):

در این الگوریتم که در روش اجتناب مطرح شده است هرگاه فرایندی برای منبعی تقاضا می کند سیستم عامل ابتدا فرض می کند که منبع را اختصاص داده است. با این فرض، شرایط جدید سیستم را شبیه سازی می کند. چنانچه در این شرایط بتوان ترتیبی از پاسخگویی به فرایندها، با منابع آزاد موجود یافت حالت وضعیت امن بوده و منبع را واقعاً به فرایند اختصاص می دهد. در غیر اینصورت از اختصاص منبع خودداری می کند.

اگر n تعداد فرایندها و m تعداد منابع باشد آنگاه:

۱- ماتریس کل منابع موجود (R): یک ماتریس خطی است که کل منابع موجود در سیستم را نشان می دهد.

$$R = (R_1, R_2, \dots, R_m)$$

۲- ماتریس منابع آزاد موجود (*Available*): یک ماتریس خطی است که منابع آزاد سیستم در حال حاضر را

نشان می دهد.

$$Available = (R_1, R_2, \dots, R_m)$$

۳- ماتریس کل منابع مورد نیاز (*Max*): یک ماتریس $n \times m$ است که در آن $Max(I, j)$ کل منابع مورد نیاز P_i

به R_j را نشان می دهد.

$$Max = \begin{bmatrix} P_1 & R_1 & R_2 & \dots & R_m \\ P_2 & & & & \\ \vdots & & & & \\ P_n & & & & \end{bmatrix}$$

۴- ماتریس منابع تخصیص یافته به فرایندها (*Allocation*): یک ماتریس $n \times m$ است که $Allocation(I, j)$

تعداد منبع R_j که به فرایند P_i تخصیص یافته را نشان می دهد.

$$\text{Allocation} = \begin{matrix} & R_1 & R_2 & \dots & R_m \\ \begin{matrix} P_1 \\ P_2 \\ \vdots \\ P_n \end{matrix} & \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \end{matrix}$$

۵- ماتریس باقیمانده منابع مورد نیاز فرایندها (*Need*): یک ماتریس $n \times m$ است که $Need(I, j)$ تعداد منابع R_j که برای فرآیند P_i نیاز است را نشان می دهد.

$$\text{Need} = \begin{matrix} & R_1 & R_2 & \dots & R_m \\ \begin{matrix} P_1 \\ P_2 \\ \vdots \\ P_n \end{matrix} & \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \end{matrix}$$

حال باید بررسی شود که با مقادیر *Available* به کدام فرآیند می توان پاسخ داد. به هر فرآیندی که پاسخ داده شود مقدار *Allocation* آن با *Available* جمع شده و *Available* جدید حاصل می شود. چنانچه به این طریق بتوان به همه فرایندها پاسخ داد وضعیت امن و در غیر اینصورت وضعیت ناامن می باشد.

$$\exists P_i \quad Need_i \leq Ava$$

پاسخگویی به فرآیند

$$Ava \leftarrow Ava + Alloc_i$$

« بیان دیگری از الگوریتم Banker »

با این الگوریتم می توان مشخص ساخت که آیا وضعیت جاری امن است یا نا امن .

- ۱- در ماتریس نیازها (*Need*) به دنبال سطری بگردید که کوچکتر از بردار منابع در دسترس (*Available*) باشد. اگر چنین سطری وجود ندارد سیستم در حالت نا امن است و الگوریتم تمام می شود.
 - ۲- اگر در ماتریس *Need* سطری را پیدا کردید که کوچکتر از بردار *Available* بود یعنی می توانید منابع در دسترس را پردازش بدهید تا تمام شود. لذا اعداد سطر مربوط به آن پردازش در ماتریس *Allocation* را به بردار *Available* اضافه کنید و این پردازش را به عنوان پردازش تمام شده علامت بزنید.
 - ۳- مراحل ۱ و ۲ را مرتباً تکرار کنید تا زمانی که تمام پروس ها به صورت خاتمه یافته علامتگذاری شوند که در این صورت وضعیت اولیه امن بوده است. یا اینکه از مرحله ۱ از الگوریتم خارج شوید که سیستم ناامن می باشد.
- نکته: اگر چندین پروسس، شرط مرحله ۲ را داشتند، اهمیتی ندارد که کدامیک را جهت تخصیص منابع انتخاب می کنید چرا که در هر حال آن پروسس پس از تخصیص منابع مورد نیاز و تمام شدن، کل منابع خود را آزاد می کند.

تست: در سیستمی ۱۴ نمونه از منبع R_1 و ۱۴ نمونه از R_2 موجود است. اگر وضعیت سیستم در زمان t به صورت زیر باشد، کدام گزینه درست است؟

	R_1	R_2
P_1	7	3
P_2	4	6
P_3	5	5
P_4	6	4

Max

(۲) سیستم در حالت ناامن است.

(۴) نمی توان تعیین کرد.

$$R_1 \text{ منابع موجود} = 14 - (4 + 2 + 3 + 3) = 2$$

$$\rightarrow \text{بردار Available} = (2, 2)$$

$$R_2 \text{ منابع موجود} = 14 - (3 + 3 + 3 + 3) = 2$$

	R_1	R_2
P_1	4	3
P_2	2	3
P_3	3	3
P_4	3	3

Allocation

(۱) سیستم در حالت امن قرار دارد

(۳) سیستم در حالت بن بست است.

$$Need = Max - Allocation \rightarrow Need =$$

	R_1	R_2
P_1	3	0
P_2	2	3
P_3	2	2
P_4	3	1

$$(2, 2) \xrightarrow{P_3} (5, 5) \xrightarrow{P_1} (9, 8) \xrightarrow{P_2} (11, 11) \xrightarrow{P_4} (14, 14)$$

پس سیستم در شرایط امن قرار دارد.

نکته: اگر در یک سیستم n پردازش و m منبع از یک نوع، موجود باشد و شرط زیر برقرار باشد هیچگاه بن بست رخ نمی دهد.

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

تست: اگر یک سیستم ۶ پردازش وجود داشته باشد و هر پردازش حداکثر ۳ تقاضا برای منبع کند، چه تعداد منبع

مسابه نیاز داریم تا بن بست رخ ندهد؟

(۱) حداقل ۱۲ منبع (۲) حداقل ۱۳ منبع (۳) حداقل ۱۴ منبع (۴) حداقل ۱۰ منبع

$$\sum_{i=1}^6 \text{Request}[i] = 6 \times 3 < m+6 \rightarrow 18 < m+6 \rightarrow m > 12$$

پس حداقل ۱۳ منبع لازم است.

تست: اگر در سیستمی n پردازش و m منبع از یک نوع، وجود داشته باشد و هر پردازش حداکثر به ۳ منبع نیاز داشته باشد، در کدام حالت ممکن است بن بست رخ دهد؟

(۱) $n=3, m=7$ (۲) $n=5, m=12$ (۳) $n=2, m=5$ (۴) $n=4, m=8$

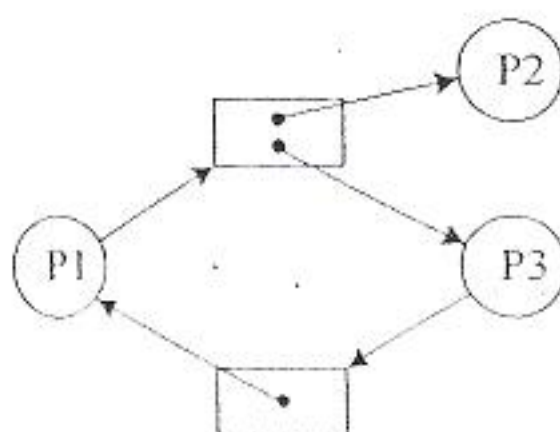
بن بست رخ نمی دهد $\sum_{i=1}^n \text{Request}[i] < m+n \rightarrow 3 \times 3 < 3+7 \rightarrow 9 < 10$

بن بست رخ نمی دهد $5 \times 3 < 5+12 \rightarrow 15 < 17$

بن بست رخ نمی دهد $2 \times 3 < 2+5 \rightarrow 6 < 7$

بن بست رخ می دهد $4 \times 3 \geq 4+8 \rightarrow 12 \geq 12$

تست: با توجه به مدل بن بست فرایند زیر کدام گزینه صحیح است؟



(۱) در این نمودار حلقه وجود دارد ولی بن بست وجود ندارد.

(۲) در این نمودار بن بست وجود دارد ولی حلقه وجود ندارد.

(۳) در این نمودار حلقه و بن بست وجود دارد.

(۴) در این نمودار حلقه و بن بست وجود ندارد.

تست: سیستمی دارای سه فرایند و ۲ نوع منبع به صورت زیر است:

حداکثر منابع مورد نیاز

	R_1	R_2
P_1	۴	۲
P_2	۳	۱
P_3	۲	۰

	R_1	R_2
P_1	۲	۱
P_2	۱	۱
P_3	۱	۰

$R_1 \quad R_2$
(6 4) = تعداد منابع اولیه

آنگاه تعداد منابع آزاد و باقی مانده کدام است؟

$R_1 \quad R_2$ (1 3) (۴) $R_1 \quad R_2$ (2 2) (۳) $R_1 \quad R_2$ (2 3) (۲) $R_1 \quad R_2$ (1 2) (۱)