



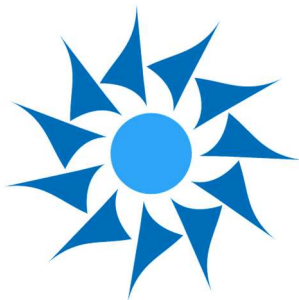
Pnu-Soal.ir

جزوه درسه

سیستم های عامل

تالیف: مهندس حمید عالمی

عضو هیئت علمی دانشگاه پیام نور مرکز آران و بیدگل



انجمن علمی دانشجویی
رشته مهندسی کامپیوتر



کانون علمی پژوهش
بسیج دانشجویی

Pnu-Soal.ir

پیشگفتار

سیستم عامل عنوان یکی از دروس تخصصی رشته مهندسی کامپیوتر (نرم افزار) و مهندسی فناوری اطلاعات می باشد.

آنچه در این مجموعه مشاهده می کنید مطالبی است که تحت عنوان جزوه سیستم عامل در دانشگاه پیام نور آران و بیدگل تدریس می شود که با تلاش و کوشش انجمن علمی مهندسی کامپیوتر دانشگاه تدوین شده است.

لازم به ذکر است که این جزوه جایگزینی برای کتاب نیست بلکه با ارائه توضیحات بیشتر درمورد مطالب کتاب سعی در آسان سازی مفاهیم آن را دارد. امید است دانشجویان با بهره گیری از آن در کنار کتاب، مطالب را به خوبی درک کنند. همچنین برای آشنایی دانشجویان با سوالات امتحانی این درس در آخر هر فصل نیز سوالات دو ترم اخیر آزمون های پیام نور ارائه گردیده است.

از آنجایی که جزوه پیش رو اولین ویرایش جزوه سیستم عامل می باشد بدیهی است خالی از اشکال نیست. بنابراین از دانشجویان عزیز تقاضا می کنم هرگونه اشکال و انتقاد خود را در جهت هرچه بهتر و پربارتر شدن این جزوه از طریق ایمیل h_alemi_arani@yahoo.com با اینجانب در میان بگذارند.

در آخر لازم است از زحمات دانشجوی گرامی، آقای مجید بصیرتی که بنده را در نگارش و ویراستاری این جزوه یاری نمودند تشکر و قدردانی نمایم.

حمید عالمی آرانی

مهرماه ۹۰

فهرست مطالب

فصل اول ۷

- سیستم عامل چیست؟ ۸
- انواع برنامه های یک سیستم ۸
- انواع ارتباط با دستگاههای جانبی ۸
- دستیابی غیر مستقیم ۹
- چندبرنامگی ۹
- انواع وقفه ها : ۱۱
- حافظه نهان (پنهان) ۱۲
- سوالات تستی ۱۴
- سوالات تشریحی ۱۷
- پاسخنامه سوالات تستی ۱۸

فصل دوم ۱۹

- فرآیند چیست؟ ۲۰
- سوالات تستی ۲۴
- سوالات تشریحی ۲۵
- پاسخنامه سوالات تستی ۲۶

فصل سوم ۲۷

- زمان بندی تک پردازنده ای ۲۸
- الگوریتم های زمان بندی کوتاه مدت ۳۰
- سوالات تستی ۳۸
- سوالات تشریحی ۴۰
- پاسخنامه سوالات تستی ۴۱

فصل چهارم ۴۳

- ناحیه بحرانی ۴۴
- راه حل های دو فرآیندی ۴۵
- راه حل های چندفرآیندی ۵۱
- نمونه سوال ۶۰
- سوالات تستی ۶۱
- سوالات تشریحی ۶۴
- پاسخنامه سوالات تستی ۶۵

فصل پنجم ۶۷

- راههای مقابله با بن بست ۶۹
- سوالات تستی ۷۶
- سوالات تشریحی ۷۸
- پاسخنامه سوالات تستی ۷۹

فصل ششم ۸۱

- انواع پراکندگی ۸۲
- مجموعه مقیم ۸۳
- سوالات تستی ۸۸
- پاسخنامه سوالات تستی ۹۰

فصل هفتم ۹۱

- سوالات تستی ۹۶
- سوالات تشریحی ۹۷
- پاسخنامه سوالات تستی ۹۸

فصل هشتم ۹۹

- سوالات تستی ۱۰۲

فصل اول

Pnu-Soal.ir

سیستم عامل چیست؟

سیستم عامل یک نرم افزار بسیار پیچیده است که به عنوان واسطه بین کاربر و سخت افزار مورد استفاده قرار می گیرد. از این نظر سیستم عامل دو هدف اصلی را دنبال می کند:

۱- آماده کردن سخت افزار برای استفاده آسان کاربر

۲- استفاده بهینه از سخت افزار و افزایش کارایی سیستم

از یک دید دیگر می توان سیستم عامل را به عنوان مدیر منابع یک سیستم کامپیوتری معرفی کرد. از این دید منابع یک کامپیوتر به سه دسته تقسیم می شود:

۱- واحد پردازش مرکزی^۱

۲- حافظه اصلی

۳- منابع سخت افزاری و نرم افزاری

انواع برنامه های یک سیستم

می توان برنامه های یک سیستم را به دو دسته تقسیم نمود:

☑ برنامه های I/O limited

برنامه هایی هستند که بخش زیادی از زمان اجراشان را در ارتباط با دستگاه های ورودی-خروجی هستند و نیاز بیشتری به I/O دارند.

☑ برنامه های CPU limited

برنامه هایی که حجم محاسباتی بالایی دارند و برای اجرا شدن به مدت زمان زیادی CPU را نیاز دارند.

سیستم عامل به منظور افزایش کارایی سعی می کند تعادلی در اجرای این دو دسته برنامه ایجاد کند.

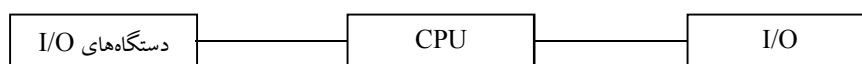
انواع ارتباط با دستگاه های جانبی

از نظر ارتباط سیستم عامل با دستگاه های جانبی، می توان سیستم ها را به دو دسته تقسیم کرد:

☑ سیستم های Online یا ارتباط مستقیم

سیستم هایی هستند که پردازنده به طور مستقیم با دستگاه های ورودی-خروجی در ارتباط است. در این سیستم ها، به دلیل پایین بودن سرعت دستگاه های جانبی، کارایی به شدت پایین می آید.

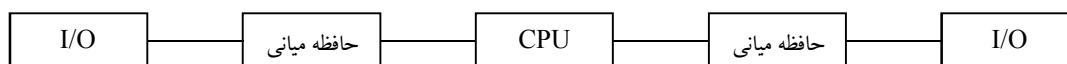
^۱ CPU: Central Processing Unit



شکل ۱-۱

✓ سیستم‌های Offline یا ارتباط غیرمستقیم

سیستم‌هایی که از حافظه میانی، بین دستگاه‌های ورودی-خروجی و CPU استفاده می‌کنند. این حافظه میانی می‌تواند باعث افزایش کارایی شود.



شکل ۲-۱

دستیابی غیر مستقیم

۱- بافردهی^۱

۲- Spooling

✓ Spooling

در Spooling از دیسک‌های مغناطیسی که دارای سرعت بیشتری هستند و حجم بالایی از برنامه‌ها را می‌توان در آن ذخیره کرد، استفاده می‌شود.

پردازش یک برنامه عمل ورودی-خروجی را برای برنامه‌ای دیگر انجام می‌دهد.

چندبرنامگی^۲

با استفاده از Spooling می‌توان چندکار را به طور همزمان اجرا کرد. در اینجا منظور از اجرای همزمان چندکار اجرای در بین هم برنامه‌ها خواهد بود. به عبارت دیگر چندبرنامگی به معنی اجرای لایه‌ای هم دستورات چند برنامه است.

در سیستم‌های چندبرنامگی دو معیار برای تعویض CPU بین فرآیندها وجود دارد :

✓ نیاز به عمل I/O

در این روش CPU به هر برنامه تخصیص داده می‌شود و مادامی که برنامه به عمل ورودی-خروجی نیاز ندارد CPU را در اختیار خواهد داشت، به محض نیاز به عمل I/O، CPU از برنامه گرفته شده و به برنامه دیگر داده می‌شود.

^۱ Buffering

^۲ Multi Programming

✓ تعریف یک بازه زمانی

در این روش یک بازه زمانی تعریف می‌شود و CPU به اندازه این بازه زمانی در اختیار هر برنامه خواهد بود. این دسته از سیستم‌ها را سیستم‌های اشتراک زمانی^۱ می‌نامند. در این دسته از سیستم‌ها دو دلیل می‌تواند باعث تعویض CPU از یک برنامه به برنامه دیگر قبل از اتمام بازه زمانی شود:

۱- پایان اجرای برنامه

۲- صدور وقفه برای اجرای عملیات ورودی-خروجی

تعیین این بازه زمانی در سیستم‌های اشتراک زمانی بر عهده سیستم‌عامل است و ممکن است این بازه زمانی در یک دوره کاری سیستم، چند مرتبه تغییر کند.

✓ سیستم‌های توزیع شده^۲

سیستم‌هایی هستند که در آنها از چند CPU استفاده می‌شود و اجرای برنامه‌ها بین این پردازنده‌ها توزیع می‌شود.

✓ سیستم‌های بلادرنگ^۳

سیستم‌هایی هستند که در آنها هر کار باید حداکثر در یک مهلت مشخصی، سرویس دهی شود. انجام آن کار پس از پایان مهلت آن مفید نخواهد بود. در واقع این سیستم‌ها باید زمان انجام کار را تضمین نمایند.

✓ روش‌های کار با دستگاه‌های ورودی-خروجی

برای کار با دستگاه‌های ورودی-خروجی، سیستم‌عامل باید روشی را برای هماهنگ کردن این بخش با پردازنده در نظر بگیرد، تا کارایی سیستم افزایش یابد. این هماهنگی معمولاً به دو روش انجام می‌شود:

(روش همه پرسى (انتظار مشغول)^۴ :

در این روش کار بدین صورت انجام می‌گیرد که هرگاه پردازنده نیاز به ورودی یا خروجی داشته باشد تقاضای خود را توسط سیستم‌عامل برای دستگاه موردنظر مطرح می‌کند. اگر دستگاه مشغول باشد پاسخ منفی می‌دهد در این صورت CPU مجدداً درخواست خود را مطرح می‌کند و این کار را آنقدر تکرار می‌کند تا پاسخ مثبت دریافت کند. اگر از آن دستگاه بیش از یکی در سیستم باشد درخواست CPU به نوبت برای همه مطرح می‌شود از این رو این روش را همه پرسى می‌گویند. در این سیستم‌ها چون CPU منتظر دستگاه می‌ماند کارایی به شدت پایین می‌آید.

^۱ Time Sharing

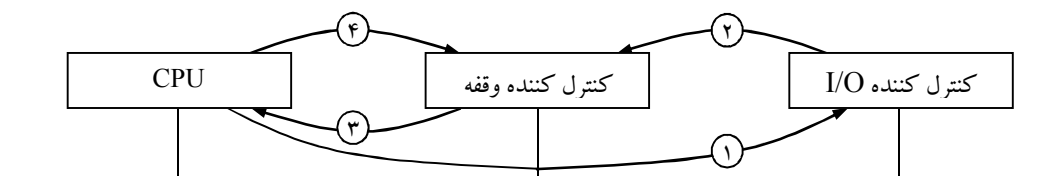
^۲ Distributed

^۳ Real Time

^۴ Pooling (Busy Waiting)

استفاده از وقفه ها^۱:

هدف از به کارگیری وقفه‌ها، استفاده بهینه از وقت پردازنده است. در سیستمی که از وقفه استفاده می‌کند چنانچه پردازنده نیاز به ورودی-خروجی داشته باشد آن را به دستگاه اعلام می‌کند و تا زمان دریافت پاسخ دستگاه (وقفه) به اجرای برنامه دیگری مشغول می‌شود و این باعث افزایش کارایی خواهد شد. در هنگام بروز یک وقفه، سیستم از حالت اجرای عادی برنامه خارج می‌شود و روال وقفه را اجرا می‌کند، پس از آن دوباره به روند اجرای عادی برنامه برمی‌گردد. روند به کارگیری وقفه‌ها، در یک سیستم به صورت زیر خواهد بود:



شکل ۱-۳

در هریک از مراحل به صورت زیر عمل می‌شود:

مرحله ۱: اگر در حین اجرای یک برنامه، که CPU به آن تعلق دارد نیاز به یک دستگاه ورودی و یا خروجی باشد CPU درخواستی را برای استفاده از آن دستگاه به کنترل کننده آن ارسال می‌کند.

مرحله ۲: در هر قطعه سخت افزاری، مولفه ای به نام کنترل کننده دستگاه جانبی وجود دارد، این قطعه در زمانی که آن دستگاه بیکار شد پیغامی را جهت اعلام آمادگی به کنترل کننده وقفه صادر می‌کند.

مرحله ۳: کنترل کننده وقفه، مولفه ای است که در داخل یا خارج از CPU قرار دارد و با اعلام آمادگی دستگاه ورودی-خروجی پیغامی را جهت کار CPU با آن دستگاه به CPU ارسال می‌دارد که این پیغام همان وقفه است.

مرحله ۴: پس از پاسخ دادن CPU به وقفه، کنترل کننده وقفه اجرای عادی برنامه را از سر می‌گیرد.

انواع وقفه ها:

به طور کلی می‌توان وقفه‌ها را به سه دسته تقسیم کرد:

☑ وقفه های خارجی

این نوع از وقفه‌ها عموماً از یک دستگاه جانبی صادر می‌شوند. دلایل اصلی صدور این وقفه‌ها عبارتند از:

۱- پایان کار دستگاه جانبی

۲- اتمام زمان تایمر دستگاه جانبی

^۱ Interrupt

۳- بروز خرابی در دستگاه جانبی

این وقفه‌ها را وقفه‌های سخت افزاری نیز می‌نامند.

☑ وقفه‌های داخلی^۱

این وقفه‌ها از دستورات خود برنامه ناشی می‌شوند. دستوراتی مانند تقسیم بر صفر و یا دستوراتی که کد آنها معتبر نیست باعث بروز این وقفه‌ها می‌شوند. با اجرای دوباره برنامه این وقفه‌ها مجدداً تکرار می‌شوند.

☑ وقفه‌های نرم افزاری

این وقفه‌ها دستوراتی از برنامه هستند که مانند فراخوانی یک برنامه فرعی لیستی از دستورات را اجرا می‌کنند. این وقفه‌ها غالباً زمانی رخ می‌دهند که برنامه کاربر نیاز به استفاده از حالت ناظر و امکانات آن داشته باشد.

حافظه نهان (پنهان)

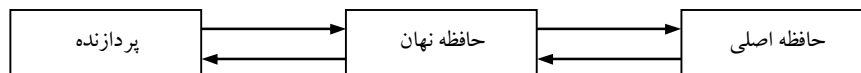
در یک سیستم کامپیوتری برای افزایش کارایی از حافظه‌های چندسطحی استفاده می‌کنند به گونه‌ای که سطوح نزدیک تر به پردازنده، دارای ظرفیت کمتر اما در عوض سرعت بیشتری هستند. یکی از این موارد استفاده از حافظه نهان است بدین صورت که هرگاه دستورالعمل یا داده‌ای در حافظه اصلی مورد استفاده قرار گیرد یکی از آن در حافظه نهان ایجاد می‌شود، دلیل آن این است که بر اساس اصل محلی‌گرایی گفته می‌شود هرگاه داده‌هایی مورد استفاده قرار گیرند به زودی در آینده نیز لازم خواهند بود، قرار دادن این داده‌ها و دستورالعمل‌ها در حافظه پنهان موجب افزایش سرعت دستیابی می‌شود، بنابراین زمانیکه داده یا دستورالعملی نیاز باشد ابتدا به حافظه پنهان مراجعه می‌شود و در صورت وجود آن استفاده می‌شود، در غیر این صورت به حافظه اصلی مراجعه شده و داده یا دستورالعمل مورد نظر مورد استفاده قرار می‌گیرد.

از آن جایی که ظرفیت حافظه نهان به نسبت حافظه اصلی بسیار کمتر است بنابراین نخواهیم توانست همه داده‌هایی را که در حافظه اصلی هستند به حافظه نهان ببریم پس ممکن است گاهی به داده‌هایی نیاز داشته باشیم که در حافظه نهان نیستند. اگر چنانچه حافظه نهان پر شده باشد و نیاز به خالی کردن بخشی از آن و جایگزینی آن با داده مورد نظر داشته باشیم از الگوریتم‌های جایگزینی حافظه استفاده می‌شود که دقیقاً همان الگوریتم‌های جایگزینی صفحه در حافظه اصلی می‌باشد.

اگر در مراجعه به حافظه نهان داده مورد نظر در آنجا وجود داشته باشد گوییم اصابت یا برخورد^۲ رخ می‌دهد که نرخ آن را با Hit ratio نشان می‌دهند. با در نظر گرفتن این مقدار چنانچه متوسط زمان دستیابی به حافظه نهان را

^۱ Trap^۲ Hit

T_C و متوسط زمان دستیابی به حافظه اصلی را T_M در نظر بگیریم آنگاه متوسط زمان دستیابی پردازنده به یک دستورالعمل یعنی T_S از رابطه زیر به دست خواهد آمد:



شکل ۴-۱

$$T_S = \text{hit ratio} \times T_C + (1 - \text{hit ratio})(T_C + T_M)$$

$$T_S = T_C + (1 - \text{hit ratio})T_M$$

در رابطه با حافظه نهان چنانچه با مراجعه به آن، داده مورد نظر در آن جا نباشد عدم اصابت^۱ رخ می‌دهد که به نرخ آن Miss ratio گفته می‌شود. بدیهی است که:

$$\text{hit ratio} = 1 - \text{miss ratio}$$

^۱ Miss

سوالات تستی

☑ نیم سال اول ۸۹-۹۰

۱- مثالهای زیر به ترتیب جز کدام دسته از وقفه ها قرار دارند؟

- تقسیم بر صفر -خطای توازن حافظه -مراجعه به آدرسی خارج از فضای مجاز کاربر
- الف) ورودی خروجی، نقص سخت افزار، برنامه ب) برنامه، ورودی خروجی، نقص سخت افزار
- ج) ورودی خروجی، برنامه، نقص سخت افزار د) برنامه، نقص سخت افزار، برنامه

۲- کدام گزینه صحیح است؟

- الف) ورودی/خروجی مبتنی بر وقفه، نیازمند دخالت فعال پردازنده برای انتقال داده ها بین حافظه و مولفه ورودی/خروجی است.
- ب) ورودی/خروجی برنامه سازی شده از ورودی/خروجی مبتنی بر وقفه کارآمدتر است.
- ج) حافظه پنهان توسط سیستم عامل قابل رویت است، اما توسط برنامه نویس قابل رویت نیست.
- د) نرخ انتقال ورودی/خروجی در DMA محدود به سرعتی است که پردازنده (CPU) میتواند یک دستگاه را بررسی کرده و خدمت دهد.

۳- پردازنده ای را در نظر بگیرید که به دو سطح از حافظه دسترسی دارد سطح یک شامل ۱۰۰۰ کلمه و زمان دستیابی $0.1 \mu s$ و سطح دو شامل ۱۰۰۰۰۰ کلمه و زمان دسترسی $1 \mu s$ است. فرض کنید پردازنده به حافظه سطح ۱ دسترسی مستقیم دارد ولی برای دسترسی به هر کلمه از حافظه سطح ۲، ابتدا آن کلمه باید به حافظه سطح ۱ انتقال یابد همچنین از مدت زمانی که پردازنده برای تعیین سطح یک کلمه از حافظه نیاز دارد، صرف نظر می-کنیم. اگر ۸۵٪ از دسترسی ها به حافظه در سطح یک یافت شود در این صورت متوسط زمان دسترسی به یک کلمه چند است؟

- الف) $1.45 \mu s$ ب) $1.05 \mu s$ ج) $0.25 \mu s$ د) $0.15 \mu s$

۴- هدف اصلی سیستم های چند برنامه ای دسته ای و سیستمهای اشتراک زمانی به ترتیب کدام است؟

- الف) حداقل زمان پاسخ - حداکثر استفاده از پردازنده
- ب) حداقل زمان پاسخ - تمایل به کارهای اشتراکی
- ج) حداکثر استفاده از پردازنده - حداقل زمان پاسخ
- د. حداکثر استفاده از پردازنده - کاهش سخت افزارهای لازم

۵- کدام گروه از پارامترهای زیرفاز امتیازات معماری چندپردازی متقارن نسبت به معماری تک پردازنده ای می باشد؟

- الف) کارآیی، دسترسی پذیری، توزیع پذیری، استقلال حافظه ای
- ب) کارآیی، دسترسی پذیری، رشد، مقیاس پذیری
- ج) کارآیی، استقلال حافظه ای، مقیاس پذیری، توزیع پذیری
- د) استقلال حافظه ای، مقیاس پذیری، رشد، توزیع پذیری

۶- کدام یک از عبارات زیر، درمورد سیستم عامل W2K (ویندوز ۲۰۰۰ مایکروسافت) صحیح می باشد؟

- الف) W2K یک سیستم تک کاربره است.
- ب) W2K نرم افزار کاربردی و نرم افزار سیستم عامل باهم پیوسته هستند.
- ج) W2K دارای یک ریز هسته محض است.
- د) W2K تنها روی ماشینهای Intel اجرا می گردد.

☑ نیم سال دوم ۸۹-۹۰

۱- محتوای ثبات دستورالعمل (IR) چیست؟

- الف) اطلاعات وضعیت
- ب) علاوه بر بیت وضعیت، شامل بیت حالت کاربر/ سرپرست نیز می باشد
- ج) آدرس دستورالعملی که باید واکنشی شود
- د) آدرس آخرین دستورالعمل واکنشی شده

۲- حداقل اطلاعات مورد نیاز برای از سرگیری برنامه جاری (از نقطه بروز وقفه) که باید ذخیره گردد کدام است؟

- الف) PSW, IR
- ب) PC, IR
- ج) PC, PSW
- د) PSW

۳- کدام گزینه صحیح است؟

- الف) DMA به کنترل گذرگاه نیاز ندارد.
- ب) روشهای DMA و برنامه سازی شده نیاز به دخالت فعال پردازنده ندارد.
- ج) ورودی خروجی برنامه سازه شده کارآمدتر از روش مبتنی بر وقفه است.
- د) ورودی خروجی مبتنی بر وقفه به دخالت فعال پردازنده نیاز دارد.

۴- کدام گزینه از محورهای اصلی در ایجاد و توسعه سیستم کامپیوتری نمی باشد؟

- الف) چند برنامه‌گی
 ب) سیستم های دسته ای
 ج) اشتراک زمانی
 د) سیستم های تراکنش بلادرنگ

۵- کدام موارد زیر از مسئولیت های سیستم عامل در مدیریت حافظه می باشد؟

- ۱- حافظه دراز مدت
 ۲- جداسازی فرآیندها
 ۳- حافظه کوتاه مدت
 ۴- حفاظت و کنترل دسترسی
- الف) ۱
 ب) ۱ و ۲ و ۳
 ج) ۱ و ۲ و ۳ و ۴
 د) ۱ و ۲ و ۴

سوالات تشریحی

✓ نیم سال اول ۸۹-۹۰

۱- مراحل پردازش وقفه ها را در قالب یک فلوجارت رسم کنید. (۱ نمره)

پاسخنامه سوالات تستی

نیم سال اول ۸۹-۹۰

د	۱
الف	۲
ج	۳
ج	۴
ب	۵
الف	۶

نیم سال دوم ۸۹-۹۰

د	۱
ج	۲
د	۳
ب	۴
د	۵

فصل دوم

فرآیندها و مدیریت آنها

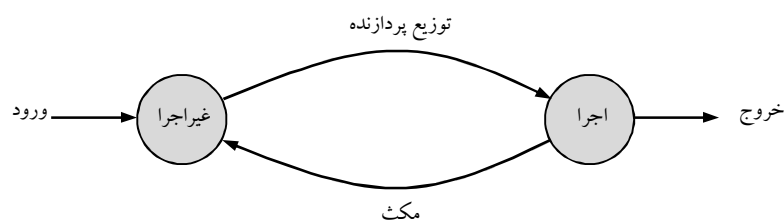
Pnu-Soal.ir

فرآیند چیست؟

فرآیند، برنامه ای خواهد بود که قصد اجرا شدن در سیستم را دارد؛ به عبارت دیگر هرگاه یک برنامه برای اجرا شدن، خودش را به سیستم عامل معرفی می کند تا مورد توجه سیستم عامل قرار گیرد به آن فرآیند گفته می شود. در اکثر منابع به فرآیند، یک نهاد فعال و به برنامه یک نهاد غیر فعال گفته می شود.

✓ مدل دو حالت

برای فرآیندها و حالت های آنها مدل های مختلفی تعریف می شود. ساده ترین مدلی که برای یک فرآیند تعریف می شود یک مدل دو حالت شامل حالت های اجرا و غیر اجرا خواهد بود منظور از یک فرآیند در حالت اجرا، فرآیندی است که در حافظه اصلی قرار دارد و CPU به آن تخصیص داده شده است. منظور از حالت غیر اجرا، آن است که فرآیند در حالت اجرا نیست.



شکل ۱-۲

✓ ضعف مدل دو حالت

مدل دو حالت دارای دو ضعف عمده است :

اولاً اینکه حالت غیر اجرا مشخص نمی کند که محل دقیق قرار گرفتن فرآیند کجاست.

ثانیاً اینکه عمل مکث که باعث می شود فرآیند از حالت اجرا به حالت غیر اجرا برده شود مشخص نمی کند که دلیل این انتقال چیست (دلیل این انتقال می تواند نیاز فرآیند به I/O و یا پایان بازه زمانی باشد)، بنابراین مدل کامل تر ۵ حالتی که دارای حالت های زیر است معرفی می شود:

✓ حالت جدید

حالتی است که در آن یک فرآیند ایجاد می شود و در واقع همان زمانی است که برنامه مفهوم فرآیند به خود می گیرد. در این زمان، برای هر فرآیند یک شناسنامه به نام **بلوک کنترل فرآیند**^۱ در نظر گرفته می شود.

^۱ PCB: Process Control Block

✓ حالت آماده

فرآیند مورد نظر در حافظه‌ی اصلی است و آماده‌ی اجرا شدن است و تنها به CPU نیاز دارد.

✓ حالت مسدود (توقف یا بلوکه)

در این حالت فرآیند تا بروز حادثه‌ای نمی‌تواند اجرا شود یا به عبارت دیگر مسدود شده است. این حادثه می‌تواند اتمام یک عمل ورودی/خروجی باشد. در این حالت فرآیند در حافظه‌ی اصلی قرار دارد.

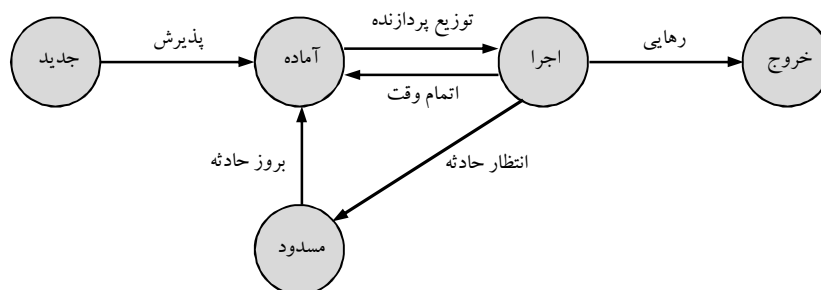
✓ حالت اجرا

این حالت دقیقاً همان حالت اجرای مدل دو حالتی است

✓ حالت خروج

این حالت معرف حالتی است که اجرای فرآیند به هر دلیلی پایان یافته باشد.

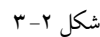
مدل ۵ حالتی به صورت زیر است :



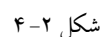
شکل ۲-۲

هرچند مدل ۵ حالتی یک مدل تقریباً کامل و عمومی برای فرآیندهاست اما می‌توان به آن ایرادهایی نیز گرفت. به عنوان مثال در این مدل می‌گوییم فرآیندی که نیاز به I/O دارد به حالت مسدود برده شود، اگر در حین انتظار این فرآیند برای دریافت حادثه به دلیل کمبود جا در حافظه‌ی اصلی بخواهیم این فرآیند را از حافظه‌ی اصلی خارج کنیم، مدل پنج حالتی جوابگوی این تغییر حالت نخواهد بود. از سوی دیگر گاه ممکن است به دلیل کمبود حافظه لازم باشد فرآیندی را از حافظه‌ی اصلی خارج کنیم که نیاز به هیچ حادثه‌ای ندارد (آماده‌ی اجراست). در این مورد نیز مدل پنج حالتی تغییر حالت لازم را در خود ندارد.

برای رفع این نیازها، دو مدل ۶ و ۷ حالتی در نظر گرفته می‌شود. در مدل ۶ حالتی، با در نظر گرفتن یک حالت به نام تعلیق مشکل اول را حل می‌کنند.



حالت مسدود و معلق حالتی است که فرآیند در حافظه‌ی ثانویه قرار دارد و منتظر حادثه‌ای است. حالت آماده و معلق حالتی است که فرآیند حادثه‌ی مورد نظر را دریافت کرده (منتظر هیچ حادثه‌ای نیست) و در حافظه‌ی ثانویه قرار دارد و در صورت وجود جا در حافظه‌ی اصلی و قرار گرفتن در آن آماده‌ی اجرا می‌شود.



در مورد فرآیندها، از دو اصطلاح تغییر حالت فرآیند و تعویض فرآیند صحبت می‌شود. منظور از تغییر حالت یک فرآیند، انتقال آن از یک حالت به حالت دیگر در هریک از مدل‌های ۵، ۶ یا ۷ حالت است. لزوماً هر تغییر حالتی در فرآیندها موجب تغییر مکان فیزیکی آن نخواهد شد.

منظور از تعویض فرآیند، گرفتن CPU از آن فرآیند و تخصیص به فرآیند دیگر است که به آن تعویض متن^۱ گفته می‌شود. تعویض فرآیند غالباً به هنگام بروز وقفه یا اتمام بازه زمانی و یا نیاز یک فرآیند به I/O رخ می‌دهد.

^۱ Context Switch

سوالات تستی

✓ نیم سال اول ۸۹-۹۰

۱- اگر فرآیندی چیزی را درخواست کند که بخاطرش باید منتظر بماند در حالت..... گذاشته می شود.

الف) مسدود ب) آماده ج) معلق د) خروج

✓ نیم سال دوم ۸۹-۹۰

۱- کدام حالات فرآیند، امکان رفتن به حالت مسدود وجود دارد؟

۱. آماده

۲. اجرا

۳. جدید

۴. معلق

الف) ۱ و ۲ ب) ۱ ج) ۱ و ۲ و ۴ د) ۲ و ۴

۲- کدامیک از موارد زیر در بلوک کنترل فرآیند ذخیره نمی شود؟

الف) برنامه کاربر ب) مالکیت و استفاده از منابع

ج) اطلاعات زمانبندی د) مدیریت حافظه

۳- کدام موارد از گزینه های زیر جزء دلایل پایان یک فرآیند نیست؟

الف) دستورالعمل نامعتبر ب) ورود فرآیند با اولویت بالاتر

ج) پایان یافتن پدر د) استفاده نامناسب از داده

سوالات تشریحی

✓ نیم سال اول ۸۹-۹۰

۱- مدل ۵حالته برای فرآیندها را رسم کنید. (۰/۷۵ نمره)

پاسخنامه سوالات تستی

[illegible]

فصل سوم

Pnu-Soal.ir

زمان بندی تک پردازنده ای

به طور کلی در هر سیستم تک پردازنده، چهار دسته زمان بندی تعریف می شود:

✓ زمان بندی کوتاه مدت (توزیع کننده)

در حین اجرای فرآیندها به چند دلیل ممکن است بخواهیم اجرای یک فرآیند را متوقف نموده و فرآیند جدیدی را اجرا کنیم. انتخاب یک فرآیند از میان فرآیندهای درون صف آماده، جهت تخصیص CPU به آن توسط توابع زمان بندی کوتاه مدت صورت می گیرد.

✓ زمان بندی میان مدت (بخشی از عملیات مبادله)

این زمان بندی در مورد افزودن فرآیندها به فرآیندهای در حالت آماده تصمیم گیری می کند.

✓ زمان بندی بلند مدت (کنترل کننده درجه چند برنامگی)

این زمان بندی درباره پذیرش فرآیندها و زمان ورود آنها به حالت ایجاد (پذیرش) تصمیم گیری می کند.

✓ زمان بندی ورودی-خروجی

این نوع زمان بندی در مورد ترتیب پاسخ دهی به ورودی-خروجی ها مخصوصاً ترتیب پاسخ دهی به درخواست های مختلف از دیسک تصمیم گیری می کند.

✓ زمان بندی کوتاه مدت

وظیفه زمان بندی کوتاه مدت انتخاب یک فرآیند از میان فرآیندهای آماده و تخصیص CPU به آن است. به این منظور از الگوریتم هایی تحت عنوان الگوریتم های زمان بندی کوتاه مدت استفاده می شود. برای مقایسه کارایی این الگوریتم ها یک سری پارامتر تعریف می شود که عبارتند از:

✓ بهره وری CPU¹

به میزان استفاده مفید از زمان CPU نسبت به کل زمان کار آن بهره وری CPU گویند که این مقدار به صورت درصد مطرح می شود.

✓ برون داد² (توان عملیاتی)

به تعداد فرآیند اجرا شده در واحد زمان، برون داد می گویند. فرآیندهای کوتاه مدت باعث بالا رفتن برون داد، و فرآیندهای بلند مدت یا طولانی باعث کاهش برون داد می شوند.

¹ CPU Utilization

² Throughput

۱- زمان کل : (زمان برگشت - زمان گردش کار)^۱

فاصله بین زمان ورود یک فرآیند به حالت آماده تا خروج نهایی آن فرآیند را زمان گردش کار یا زمان کل می نامند.

۲- زمان انتظار^۲

به مدت زمانی که یک فرآیند در حالت آماده برای در اختیار گرفتن cpu انتظار می کشد زمان انتظار می گویند. ممکن است در بعضی از روشها یک فرآیند چندین مرتبه انتظار بکشد که مجموع همه ی این زمانهای انتظار زمان انتظار آن فرآیند خواهد بود.

۳- زمان پاسخ^۳

به مدت زمان بین ورود یک فرآیند به حالت آماده تا دریافت اولین پاسخ آن زمان پاسخ می گویند. در مورد زمان دریافت اولین پاسخ، نظرات مختلفی وجود دارد. در بعضی از منابع، شروع دریافت اولین بار CPU را زمان پاسخ در نظر می گیرند و در بعضی از منابع، پایان اولین دریافت CPU را زمان دریافت پاسخ می دانند. در مورد پارامترهای زمانی، این پارامترها به صورت متوسط محاسبه می شوند. یعنی به طور مثال متوسط زمان انتظار چند فرآیند، از تقسیم مجموع زمانهای انتظار، بر تعداد فرآیندهای اجرا شده بدست می آید. نکته دیگر اینکه در مورد دو پارامتر اول هرچه مقدار آنها بیشتر باشد الگوریتم کارا تر خواهد بود و در مورد سه پارامتر زمانی، هرچه مقدار آنها کمتر باشد کارایی بیشتر خواهد بود. الگوریتم های زمانبندی کوتاه مدت به دو دسته تقسیم می شوند :

الگوریتم های انحصاری - انقطاع ناپذیر و یا بدون قبضه کردن^۴

این دسته الگوریتم هایی هستند که در حین اجرای یک فرآیند سیستم عامل نمی تواند CPU را از فرآیند پس بگیرد مگر آنکه فرآیند به I/O نیاز داشته باشد، بنابراین می توان گفت در این دسته یک فرآیند اگر به I/O نیاز نداشته باشد به طور پیوسته انجام می گیرد.

الگوریتم های غیر انحصاری - انقطاع پذیر و یا با قبضه کردن^۵

این دسته الگوریتم هایی هستند که در آنها سیستم عامل اجازه دارد در حین اجرای یک فرآیند CPU را از آن پس گرفته (قبضه کردن) و به فرآیند دیگر تخصیص دهد پس گرفتن CPU از یک فرآیند و تخصیص آن به

^۱ Turn Around Time

^۲ Waiting Time

^۳ Response Time

^۴ Non Preemption

^۵ Preemption

فرآیند دیگر تعویض متن نامیده می‌شود. به دلیل همین زمان‌های تعویض متن غالباً بهره وری CPU کمتر از ۱۰۰٪ می‌شود.

الگوریتم‌های زمان بندی کوتاه مدت

✓ الگوریتم خدمت به ترتیب ورود^۱

این الگوریتم یک الگوریتم انحصاری است که در آن هر فرآیندی که زودتر وارد حالت آماده شود زودتر اجرا خواهد شد.

از مزایای این الگوریتم می‌توان به سادگی پیاده سازی آن اشاره کرد. اما از معایب آن زیاد بودن میانگین زمان پاسخ و نیز غیر قابل استفاده بودن در سیستم‌های اشتراک زمانی نام برد.

مثال ۳ - ۱ سیستمی شامل ۴ فرآیند هر یک با زمان‌های ورود و اجرای زیر را در نظر بگیرید اگر در این سیستم از الگوریتم FCFS^۲ برای اجرای فرآیندها استفاده شود، پس از تعیین ترتیب اجرای فرآیندها، پارامترهای بهره وری CPU، برونداد، میانگین زمان گردش کار، میانگین زمان انتظار و میانگین زمان پاسخ را مشخص کنید. (زمان‌ها بر حسب میلی ثانیه هستند).

فرآیند	زمان ورود	زمان اجرا
P ₁	0	8
P ₂	1	4
P ₃	2	9
P ₄	3	5

اگر در اجرای مثال، زمان تعویض متن داده نشود، آن را برابر صفر در نظر می‌گیریم. همچنین اگر در اجرای یک مثال، زمان ورود فرآیندها داده نشود، زمان ورود همه فرآیندها را برابر صفر در نظر می‌گیریم و ترتیب ورود آنها را به همان ترتیبی که در جدول ذکر شده در نظر می‌گیریم.

$$CPU \text{ بهره‌وری} = \frac{\text{زمان مفید}}{\text{زمان کل کار}} = \frac{26}{26} \times 100\% = 100\%$$

$$\text{متوسط زمان گردش کار} = \frac{(8 - 0) + (12 - 1) + (21 - 2) + (26 - 3)}{4} = \frac{61}{4} = 15.25$$

$$\text{متوسط زمان انتظار} = \frac{(0 - 0) + (8 - 1) + (12 - 2) + (21 - 3)}{4} = \frac{35}{4} = 8.75$$

^۱ FIFO : First In First Out

^۲ First Come First Service

$$\text{متوسط زمان پاسخ} = \frac{(0 - 0) + (8 - 1) + (12 - 2) + (21 - 3)}{4} = \frac{35}{4} = 8.75$$

$$\text{متوسط زمان پاسخ} = \frac{(8 - 0) + (12 - 1) + (21 - 2) + (26 - 3)}{4} = \frac{61}{4} = 15.25$$

☑ الگوریتم نوبت گردشی^۱

در این الگوریتم که نسخه غیرانحصاری FIFO است یک بازه زمانی^۲ (برش زمانی، کوانتوم زمانی) در نظر گرفته می شود و فرآیندها به ترتیب ورود و به اندازه بازه زمانی، CPU را در اختیار می گیرند. اگر فرآیندی در بین بازه زمانی خاتمه یابد از سیستم خارج می شود و CPU را برای اجرای فرآیندهای دیگر آزاد می کند. اما اگر با پایان بازه زمانی اجرای یک فرآیند تمام نشود آن فرآیند CPU را آزاد کرده و در انتهای صف آماده قرار می گیرد تا در لحظات بعد دوباره CPU به آن داده شود.

الگوریتم نوبت گردشی، برای کم کردن زمان پاسخ ارائه شده است.

در این روش، چنانچه یک فرآیند در انتهای بازه زمانی CPU را آزاد کرده و بخواهد در انتهای صف قرار گیرد و درست در همین لحظه فرآیند جدیدی بخواهد وارد حالت آماده شود برای کمتر شدن میانگین زمان پاسخ فرآیند جدید را در صف، جلوتر قرار می دهیم.

از مزایای این روش می توان به انقطاع پذیری آن و نیز زمان پاسخ نسبتا مناسب آن اشاره کرد.

این روش جزء منصفانه ترین روش هاست.

از مشکلات این روش تنظیم برش زمانی است، چرا که اگر برش زمانی بسیار کوچک باشد تعداد تعویض متن ها زیاده شده در نتیجه کارایی کاهش می یابد و اگر برش زمانی بسیار بزرگ باشد این روش شبیه FIFO عمل می کند.

مثال ۳ - ۲ با در نظر گرفتن سیستم مثال ۳ - ۱ و با استفاده از الگوریتم نوبت گردشی (RR) با دو برش زمانی:

الف) ۱ میلی ثانیه

ب) ۲ میلی ثانیه

ترتیب اجرای فرآیندها را مشخص نموده و متوسط زمانهای گردش کار، انتظار و پاسخ را محاسبه نمایید.

^۱ RR : Round Robin

^۲ Time Slice

فرآیند	زمان ورود	زمان اجرا
P ₁	0	8
P ₂	1	4
P ₃	2	9
P ₄	3	5

(الف)

$$\text{متوسط زمان گردش کار} = \frac{(23 - 0) + (13 - 1) + (26 - 2) + (20 - 3)}{4} = \frac{76}{4} = 19 \text{ ms}$$

$$\begin{aligned} \text{متوسط زمان انتظار} &= \frac{(0 + 1 + 3 + 3 + 3 + 2 + 2 + 1) + (0 + 2 + 3 + 3)}{4} \\ &+ \frac{(1 + 3 + 3 + 3 + 2 + 2 + 1) + (2 + 3 + 3 + 2 + 2)}{4} = \frac{50}{4} = 12.5 \end{aligned}$$

$$\text{میانگین زمان پاسخ}^1 = \frac{(0 - 0) + (1 - 1) + (3 - 2) + (5 - 3)}{4} = \frac{3}{4} = 0.75$$

$$\text{میانگین زمان پاسخ}^2 = \frac{(1 - 0) + (2 - 1) + (4 - 2) + (6 - 3)}{4} = \frac{7}{4} = 1.75$$

(ب)

$$\text{متوسط زمان گردش کار} = \frac{(22 - 0) + (12 - 1) + (26 - 2) + (23 - 3)}{4} = \frac{77}{4} = 19.25 \text{ ms}$$

$$\text{متوسط زمان انتظار} = \frac{(0 + 4 + 6 + 4) + (1 + 6) + (2 + 6 + 4 + 3) + (5 + 6 + 4)}{4} = \frac{51}{4} = 12.75$$

$$\text{میانگین زمان پاسخ} = \frac{(0 - 0) + (2 - 1) + (4 - 2) + (8 - 3)}{4} = \frac{8}{4} = 2$$

$$\text{میانگین زمان پاسخ} = \frac{(2 - 0) + (4 - 1) + (6 - 2) + (10 - 3)}{4} = \frac{16}{4} = 4$$

^۱ زمانی که ابتدای اولین بازه، زمان دریافت پاسخ می باشد. در ادامه، تمامی مثال ها دارای دو زمان انتظار است که اولین زمان، همین زمان می باشد.

^۲ زمانی که انتهای اولین بازه، زمان دریافت پاسخ می باشد. در ادامه، تمامی مثال ها دارای دو زمان انتظار است که دومین زمان، همین زمان می باشد.

✓ الگوریتم کوتاهترین فرآیند^۱

در این روش، هر فرآیندی که زمان اجرای کمتری داشته باشد اولویت بالاتری برای اجرا خواهد داشت. این روش به صورت انحصاری انجام می گیرد. در این روش برای انتخاب یک فرآیند باید با توجه به زمان اجراها در مورد فرآیندهایی که در صف آماده هست تصمیم بگیریم. از مزایای این روش می توان به کم بودن میانگین زمان انتظار آن و از معایبش می توان به غیرقابل استفاده بودن در سیستم های اشتراک زمانی و نیز غیرقابل اجرا بودن در هنگامی که زمان اجرای فرآیندها را به طور کامل محاسبه نکرده باشیم، اشاره کرد.

گرسنگی^۲: در بعضی از الگوریتم ها که براساس اولویت کار می کنند چنانچه به دلیل ورود مداوم فرآیندهای با اولویت بالا، یک فرآیند با اولویت پایین به مدت زیادی به تعویق بیفتد اصطلاحاً می گوییم آن فرآیند دچار گرسنگی شده است.

روش SPN دارای مشکل گرسنگی خواهد بود، زیرا اگر یک فرآیند با زمان اجرای طولانی در سیستم باشد و مدام فرآیندهای کوتاهتر وارد شوند فرآیند طولانی دچار گرسنگی می شود.

مثال ۳-۳ با در نظر گرفتن سیستم مثال ۳-۱ و با استفاده از الگوریتم SPN

فرآیند	زمان ورود	زمان اجرا
P ₁	0	8
P ₂	1	4
P ₃	2	9
P ₄	3	5

$$\text{متوسط زمان گردش کار} = \frac{(8-0) + (12-1) + (26-2) + (17-3)}{4} = \frac{57}{4} = 14.25 \text{ ms}$$

$$\text{متوسط زمان انتظار} = \frac{(0-0) + (8-1) + (17-2) + (12-3)}{4} = \frac{31}{4} = 7.75 \text{ ms}$$

$$\text{میانگین زمان پاسخ} = \frac{(0-0) + (8-1) + (17-2) + (12-3)}{4} = \frac{31}{4} = 7.75 \text{ ms}$$

$$\text{میانگین زمان پاسخ} = \frac{(8-0) + (12-1) + (26-2) + (17-3)}{4} = 14.25 \text{ ms}$$

^۱ SPN : Shortest Process Next

^۲ Starvation

✓ الگوریتم کوتاهترین زمان باقیمانده^۱

این الگوریتم نسخه غیرانحصاری الگوریتم SPN است، یعنی در این روش نیز هر فرآیندی زمان اجرای کوتاهتری داشته باشد اولویت بالاتری برای اجرا دارد. فرآیندها در این روش به صورت غیر انحصاری انجام می شوند. بدین معنی که چنانچه در حین اجرای یک فرآیند، فرآیند جدیدی وارد سیستم شود زمان اجرای فرآیند تازه وارد با زمان باقیمانده از فرآیند در حال اجرا مقایسه می شود و هرکدام زمان کمتری داشت اجرا می شود. اگر زمان اجرای فرآیند تازه وارد با زمان باقیمانده فرآیند در حال اجرا برابر باشند فرآیند در حال اجرا ادامه می یابد، زیرا این کار باعث کم شدن تعداد تعویض متن ها و در نتیجه افزایش کارایی می شود.

از مزایای این روش می توان به کمترین میانگین زمان انتظار که توسط آن به دست می آید اشاره کرد و از معایبش می توان به نیاز به دانستن زمان اجرای فرآیندها قبل از اجرای آنها و هم چنین گرسنگی اشاره کرد.

مثال ۳ - ۴ با در نظر گرفتن سیستم مثال ۳ - ۱ و با استفاده از الگوریتم SRT

فرآیند	زمان ورود	زمان اجرا
P ₁	0	8
P ₂	1	4
P ₃	2	9
P ₄	3	5

$$\text{متوسط زمان گردش کار} = \frac{(17 - 0) + (5 - 1) + (26 - 2) + (10 - 3)}{4} = \frac{52}{4} = 13 \text{ ms}$$

$$\text{متوسط زمان انتظار} = \frac{(0 - 9) + 0 + 15 + 2}{4} = \frac{26}{4} = 6.5$$

$$\text{میانگین زمان پاسخ} = \frac{(0 - 0) + (1 - 1) + (17 - 2) + (5 - 3)}{4} = \frac{17}{4} = 4.25$$

$$\text{میانگین زمان پاسخ} = \frac{(1 - 0) + (5 - 1) + (26 - 2) + (10 - 3)}{4} = 9$$

✓ الگوریتم بالاترین نسبت پاسخ^۲

یکی از روش های اجرای فرآیندها روشی است که به طریقی به هر فرآیند یک عدد اولویت نسبت داده می شود که این روش به روش اولویت معروف است. روش اولویت می تواند هم به صورت انحصاری و هم به صورت

^۱SRT : Shortest Remaining Time

^۲HRRN : Highest Response Ratio Next

غیرانحصاری باشد. یکی از روشهای اولویت معروف الگوریتم HRRN است. در این روش عدد اولویت فرآیندها از رابطه نسبت پاسخ که به صورت زیر تعریف می شود به دست می آید:

$$\text{نسبت پاسخ} = \frac{\text{زمان اجرا} + \text{زمان انتظار}}{\text{زمان اجرا}}$$

الگوریتم HRRN یک روش انحصاری است که در آن اولویت بالاتر با فرآیندی است که نسبت پاسخ بیشتری دارد.

مثال ۳-۵ با در نظر گرفتن سیستم مثال ۳-۱ و با استفاده از الگوریتم HRRN

فرآیند	زمان ورود	زمان اجرا
P ₁	0	8
P ₂	1	4
P ₃	2	9
P ₄	3	5

$$\left\{ \begin{array}{l} P_1 \text{ نسبت پاسخ} = \frac{7+4}{4} = 2.75 \\ P_2 \text{ نسبت پاسخ} = \frac{6+9}{9} = 1.6 \\ P_3 \text{ نسبت پاسخ} = \frac{5+5}{5} = 2 \end{array} \right.$$

$$\left\{ \begin{array}{l} P_2 \text{ نسبت پاسخ} = \frac{10+9}{9} = 2.1 \\ P_3 \text{ نسبت پاسخ} = \frac{9+5}{5} = 2.8 \end{array} \right.$$

$$\text{متوسط زمان گردش کار} = \frac{(8-0) + (12-1) + (26-2) + (17-3)}{4} = \frac{57}{4} = 14.25 \text{ ms}$$

$$\text{متوسط زمان انتظار} = \frac{(0-0) + (8-1) + (17-2) + (12-3)}{4} = \frac{31}{4} = 7.75 \text{ ms}$$

$$\text{میانگین زمان پاسخ} = \frac{(0-0) + (8-1) + (17-2) + (12-3)}{4} = \frac{31}{4} = 7.75 \text{ ms}$$

$$\text{میانگین زمان پاسخ} = \frac{(8-0) + (12-1) + (26-2) + (17-3)}{4} = 14.25 \text{ ms}$$

✓ الگوریتم بازخورد^۱

در این روش می توان به جای در نظر گرفتن یک صف زمان بندی، چند صف در سطوح مختلف در نظر گرفت که هر صف دارای اولویت خاصی است. معمولاً صف ها با شروع از صفر شماره گذاری می شوند و به طور پیش فرض صف های سطوح بالاتر (صف صفر) اولویت بیشتری دارند و هرچه به سمت سطوح پایین تر حرکت می کنیم از اولویت صف ها کاسته می شود. بنابراین اگر دو فرآیند در دو صف داشته باشیم انتخاب آنها بر اساس اولویت صفشان خواهد بود. در این روش تعداد صف ها می تواند محدود یا نامحدود باشد. هر فرآیند هرگاه در یک صف قرار گرفت، بر اساس الگوریتم آن صف اجرا می شود و چنانچه هنوز تمام نشده باشد به انتهای صف بعدی می رود. اگر تعداد صف ها محدود باشد و الگوریتم صف آخر غیرانحصاری باشد صف آخر به صورت بازگشتی استفاده می شود، یعنی اگر فرآیندی از آن انتخاب شود و در یک نوبت اجرائش تمام نشود دوباره در انتهای همان صف قرار می گیرد. معمول ترین الگوریتمی که برای صف ها استفاده می شود الگوریتم نوبت گردشی (RR) است و معمولاً در این حالت صف های پایین تر (با شماره اعداد بیشتر) برش زمانی بیشتری نسبت به صف های بالاتر دارند. برش زمانی صف ها نیز یا به صورت دقیق داده می شود و یا از طریق رابطه ای با توجه به شماره صف ها مشخص می شود (مانند شماره صف^۲).

مثال ۳ - ۶ با در نظر گرفتن فرآیندهای موجود در سیستم مثال قبل و با یک الگوریتم بازخورد که دارای سه صف بدین صورت که صف اول الگوریتم RR با برش زمانی ۱ میلی ثانیه، صف دوم الگوریتم RR با برش زمانی ۲ میلی ثانیه و صف سوم الگوریتم FIFO باشد، ترتیب اجرای فرآیندها را مشخص و متوسط زمان های گردش کار، انتظار و پاسخ را محاسبه کنید.

صف ۰	P ₀	P ₁	P ₂	P ₃	$t_s = 1 \text{ ms}$
صف ۱	P ₀	P ₁	P ₂	P ₃	$t_s = 2 \text{ ms}$
صف ۲	P ₀	P ₁	P ₂	P ₃	FIFO

$$\text{متوسط زمان گردش کار} = \frac{(17 - 0) + (18 - 1) + (24 - 2) + (26 - 3)}{4} = \frac{79}{4} = 19.75 \text{ ms}$$

$$\text{متوسط زمان انتظار} = \frac{(0 + 3 + 6) + (0 + 4 + 9) + (0 + 5 + 7) + (0 + 6 + 12)}{4} = \frac{53}{4} = 13.25$$

$$\text{میانگین زمان پاسخ} = \frac{(0 - 0) + (1 - 1) + (2 - 2) + (3 - 3)}{4} = \frac{0}{4} = 0$$

^۱ FB : Feed Back

$$\text{میانگین زمان پاسخ} = \frac{(1 - 0) + (2 - 1) + (3 - 2) + (4 - 3)}{4} = \frac{4}{4} = 1 \text{ ms}$$

سوالات تستی

✓ نیم سال اول ۸۹-۹۰

۱- هر کدام از جملات زیر مربوط به کدام نوع زمانبندی است، گزینه صحیح تر را انتخاب کنید.

- به توزیع کننده نیز مشهور است.

- بخشی از عملیات مبادله است.

- درجه چند برنامه‌گی را مدیریت می‌کند.

الف) کوتاه مدت، میان مدت، بلند مدت ب) کوتاه مدت، میان مدت، ورودی خروجی

ج) کوتاه مدت، بلند مدت، میان مدت د) کوتاه مدت، بلند مدت، ورودی خروجی

۲- در کدام یک از الگوریتمهای زمان بندی امکان گرسنگی وجود ندارد؟

الف) SPN ب) SRT ج) HRRN د) FB

۳- کدام گزینه صحیح است؟

الف) سیاست FCFS به نفع فرآیندهای در تنگنای ورودی خروجی در مقابل فرآیندهای در تنگنای پردازنده است.

ب) نوبت گردشی مجازی (VRR) به منظور بها دادن بیشتر به فرآیندهای در تنگنای ورودی خروجی پیشنهاد گردید.

ج) در الگوریتم RR بهتر این است که برهه زمانی کمتر از زمان لازم برای یک محاوره متداول باشد.

د) FCFS به نفع فرآیندهای کوتاه است تا فرآیندهای طولانی.

۴- پنج فرآیند A, B, C, D, E با مشخصات زیر را در نظر بگیرید. اگر از سیاست RR با برهه زمانی ۱ استفاده شود میانگین زمان کل برابر خواهد بود با:

فرآیند	A	B	C	D	E
زمان ورود	۰	۲	۴	۶	۸
زمان خدمت	۳	۶	۴	۵	۲

الف) ۱۰.۸ ب) ۱۰.۴ ج) ۱۰ د) ۱۰.۶

✓ نیم سال دوم ۸۹-۹۰

۱- در کدامیک از روش های زمانبندی زیر، امکان گرسنگی برای فرآیندها وجود ندارد؟

الف) SRT ب) HRRN ج) SPN د) FB

۲- کدام الگوریتم زمانبندی برای کارهای طولانی مناسب تر است؟

الف) بالاترین نسبت پاسخ (HRRN) ب) کوتاهترین زمان باقیمانده (SRT)
ج) بازخورد (FB) د) کوتاهترین فرآیند (SPN)

سوالات تشریحی

✓ نیم سال اول ۸۹-۹۰

۱- مجموعه فرآیندهای زیر را در نظر بگیرید. الگوریتم های زمانبندی FCFS و SRT را روی آنها اجرا کنید و میانگین زمان انتظار را برای هر کدام محاسبه نمایید. (۱/۵ نمره)

فرآیند	ورود	پردازش
A	0	3
B	1	5
C	3	2
D	9	5
E	12	5

✓ نیم سال دوم ۸۹-۹۰

۱- اطلاعات پنج فرآیند با واحد زمانی ثانیه درجدول زیر آورده شده است. نمودار زمانبندی با روشهای SRT و RR با برهه زمانی برابر با ۲ واحد زمانی را ترسیم نموده و کل زمانی که فرآیند در سیستم میگذراند، محاسبه نمایید (۱/۲۵ نمره)

نام برنامه	A	B	C	D	E
زمان ورود	0	2	3	5	6
زمان اجرا	3	5	2	3	3

پاسخنامه سوالات تستی

نیم سال اول ۸۹-۹۰	
الف	۱
ج	۲
ب	۳
الف	۴
نیم سال دوم ۸۹-۹۰	
ب	۱
الف	۲

Pnu-Soal.ir

فصل چهارم^۱

انحصار متقابل و همگام سازی

Pnu-Soal.ir

^۱ این فصل، فصل پنجم کتاب می باشد.

در یک سیستم تک پردازنده، چندبرنامگی که به معنای اجرای در بین هم چندین برنامه است ممکن است به فرآیندها صدمه بزند، یعنی تعویض متوالی فرآیندهای در حال اجرا ممکن است شرایطی را پیش آورد که هریک از فرآیندها به خروجی واقعی خود نرسند. در یک سیستم چندبرنامگی که چند فرآیند لایه هم اجرا می‌شوند ممکن است هر ترتیبی برای اجرای خطوط این فرآیندها پیش آید. به طور مثال فرض کنید دو فرآیند زیر در یک سیستم وجود داشته باشند:

P_0	$A=10$	P_1
---		---
---		---
---		---
1 READ A		3 READ A
2 $A=A+6$		4 $A=A*5$
---		---
---		---
---		---

در هر دو فرآیند فوق، از یک متغیر مشترک A استفاده شده است. اگر مقدار اولیه این متغیر ۱۰ باشد، فرآیند P_0 انتظار دارد، مقدار متغیر در بخش باقیمانده، ۱۶ باشد و فرآیند P_1 انتظار دارد که A را ۵۰ ببیند. ممکن است شرایطی در اجرای لایه هم دستورات این دو فرآیند پیش آید که یکی از دو فرآیند یا هردوی آنها مقدار این متغیر را به اندازه ای که انتظارش را نمی‌کشیدند در بخش باقیمانده ببینند، به طور مثال اگر فرآیند P_0 پس از اجرای دستورات ۱ و ۲، CPU را تحویل دهد (بازه زمانی تمام شود. و سپس P_1 ، CPU را در اختیار گرفته و تا پایان دستور شماره ۴ انجام شود آنگاه مقدار متغیر A برابر ۸۰ خواهد بود که برای ادامه هردو فرآیند نامعتبر است. مشکل به وجود آمده ناشی از تغییری است که به طور مشترک بین دو فرآیند استفاده شده و سیستم عامل اجازه داده که این دو فرآیند به هر ترتیبی که می‌خواهند اجرا شوند.

ناحیه بحرانی^۱

اگر چند فرآیند دارای بخشی از کد که از متغیر مشترک استفاده می‌کنند داشته باشیم ممکن است اجرای در بین هم این کدها متغیر مشترک را دارای مقدار نامعتبر کند. به این کد مشترک اصطلاحاً بخش بحرانی می‌گویند که اگر تعویض متن فرآیندها در این بخش صورت گیرد احتمال بروز مشکل وجود دارد. سیستم عامل باید تضمین کند هرگاه یک فرآیند در بخش بحرانی خود به سر می‌برد هیچ فرآیند دیگری حق ورود به آن ناحیه را نداشته باشد. در واقع سیستم عامل باید ورود به ناحیه بحرانی هر فرآیند را منحصر به فرد کند یا به اصطلاح در ورود به ناحیه بحرانی فرآیندها انحصار متقابل ایجاد کند.

در روش‌های نرم افزاری حل مشکل ناحیه بحرانی معمولاً ناحیه بحرانی در

فرآیند

Entry Section
C-S
Exit Section

^۱ CS: Critical Section

میان دو بخش به نام‌های بخش ورود و بخش خروج و به صورت زیر قرار می‌گیرد. در بخش ورود ابتدا بررسی می‌شود که فرآیند جاری اجازه ورود به ناحیه بحرانی را دارد یا خیر و اگر اجازه ورود داشته باشد ورود این فرآیند به بخش بحرانی را به فرآیندهای دیگر اعلام می‌کند. در بخش خروج هنگام خروج یک فرآیند از ناحیه بحرانی این خروج به فرآیندهای دیگر اعلام می‌شود.

کدهایی که در بخش ورود و بخش خروج نوشته می‌شوند یا به عبارت دیگر روش‌های حل مشکل ناحیه بحرانی باید دارای یک سری ویژگی‌هایی باشند که سه تا از مهم‌ترین آنها عبارتند از:

✓ انحصار متقابل

منظور از انحصار متقابل آن است که هیچگاه نباید دو فرآیند با هم وارد ناحیه بحرانی شوند. به عبارت دیگر اگر فرآیندی در ناحیه بحرانی باشد به فرآیند دیگر اجازه ورود داده نشود.

✓ پیشرفت یا پیشروی

این شرط بیان می‌کند که اگر فرآیندها بخواهند وارد ناحیه بحرانی شوند اولاً الگوریتم به کار رفته در ورود فرآیندها به ناحیه بحرانی ترتیب خاصی را اتخاذ نکند و همچنین شرایطی را برای پیشرفت فرآیندهایی که تاکنون بخشی از آنها اجرا شده فراهم آورد.

✓ انتظار نامحدود

الگوریتم‌های ارائه شده باید به گونه ای باشند که در نهایت به فرآیندها اجازه ورود داده شود. اگر هیچ فرآیندی نتواند وارد ناحیه بحرانی شود آنگاه انتظار نامحدود یا بن بست خواهیم داشت که از معایب آن الگوریتم خواهد بود.

روش‌های حل مشکل ناحیه بحرانی

برای حل مشکل ناحیه بحرانی، روش‌های سخت افزاری و نرم افزاری متعددی وجود دارند. بعضی از روش‌های نرم افزاری تنها برای حل مشکل دو فرآیند به کار می‌روند و بعضی از آنها قادرند مشکل ناحیه بحرانی را برای هر تعداد از فرآیند نیز برطرف سازند.

راه حل های دو فرآیندی

✓ الگوریتم اول (تلاش اول)

در این روش از یک متغیر مشترک به نام turn استفاده می‌شود که هر فرآیندی که turn برابر شماره آن باشد می‌تواند وارد ناحیه بحرانی شود. این الگوریتم برای دو فرآیند P_i و P_j به صورت زیر خواهد بود:

P_i	P_j
---	---
---	---
---	---
While (turn $\neq i$);	While (turn $\neq j$);
C-S	C-S
turn = j	turn = i
---	---
---	---
---	---

مقدار اولیه turn در این روش می تواند i یا j باشد.

این الگوریتم شرط اول یعنی انحصار متقابل را دارد؛ زیرا در این روش از یک متغیر turn استفاده شده که در هر لحظه تنها می تواند یک مقدار داشته باشد و فرآیندی که turn برابر شماره آن باشد اجازه ورود دارد و این متغیر تنها زمانی تغییر می کند که فرآیند مورد نظر از ناحیه بحرانی اش خارج شود.

در این الگوریتم شرط سوم یعنی انتظار محدود نیز برقرار است زیرا مقدار اولیه turn هرچه باشد فرآیند مربوط به آن ابتدا وارد ناحیه بحرانی می شود و آن فرآیند با خروج از ناحیه بحرانی متغیر turn را به نام دیگری تغییر می دهد تا فرآیند دوم نیز بتواند وارد شود.

این الگوریتم شرط پیشرفت را ندارد زیرا در اجرای فرآیندها ترتیب خاصی قائل می شود، یعنی فرآیندها مجبورند به صورت یکی در میان اجرا شوند. چنانچه در سیستمی نرخ اجرای دو فرآیند با هم متفاوت باشد این الگوریتم مفید نخواهد بود

از مشکلات دیگر این روش آن است که اگر یکی از فرآیندها با شکست مواجه شود فرآیند دیگر نخواهد توانست اجرا شود.

☑ الگوریتم دوم (تلاش دوم)

در الگوریتم اول مشکلات به وجود آمده ناشی از آن است که تنها از یک متغیر در همگام سازی استفاده شده و این متغیر اطلاعات کافی در مورد حالت هر فرآیند در مورد ناحیه بحرانی را ارائه نمی دهد. در الگوریتم دوم از دو متغیر برای هریک از فرآیندها استفاده می شود یعنی یک آرایه دو عنصری به نام فلگ برای فرآیندها تعریف می شود. این الگوریتم برای دو فرآیند P_i و P_j به صورت زیر است:

P_0	P_1
---	---
---	---
---	---
1: While (Flag[j]);	2: While (Flag[i]);
5: Flag[i] = true	3: Flag[j] = true
6: C-S	4: C-S
Flag[i] = false	Flag[j] = false
---	---
---	---
---	---

مقدار اولیه متغیر Flag برای هر دو فرآیند false خواهد بود.

این الگوریتم شرط انحصار متقابل را ندارد زیرا اگر به فرض فرآیند P_i شروع به اجرا کند، در دستور ۱ چون Flag[j] را false می‌بیند از حلقه عبور می‌کند. سپس اگر تعویض متن صورت گیرد و CPU به P_j داده شود این فرآیند نیز در خط ۲، Flag[i] را false دیده و از حلقه عبور می‌کند. سپس خط ۳ و خط ۴ (ناحیه بحرانی) اجرا می‌شود. اگر در این حین تعویض متن صورت گیرد و CPU به P_i داده شود از آنجایی که این فرآیند قبلاً از حلقه عبور کرده اجرایش از خط ۵ ادامه می‌یابد و وارد خط ۶ می‌شود، یعنی هر دو فرآیند در ناحیه بحرانی اند.

این الگوریتم شرط پیشرفت را دارد زیرا ابتکار عمل در مورد تغییر متغیر مورد استفاده در خود فرآیندها خواهد بود و این باعث می‌شود که هر فرآیند بتواند چندین مرتبه پشت سرهم اجرا شود (فرآیندها را با هر ترتیبی می‌توانیم اجرا کنیم).

این الگوریتم شرط انتظار محدود را دارد، زیرا مقدار اولیه هر دو فلگ برابر false است و هر فرآیندی که بخواهد وارد ناحیه بحرانی شود با false دیدن فلگ فرآیند دیگر، این اجازه را خواهد داشت.

☑ الگوریتم سوم (تلاش سوم)

مشکل الگوریتم دوم این بود که ابتدا ورود به ناحیه بحرانی را بررسی می‌کرد (شرط حلقه while) و سپس ورود به این ناحیه را اعلام می‌کرد. چنانچه فرآیندی پس از این بررسی فرصت اعلام کردن را نداشته باشد باعث خواهد شد که فرآیند دوم نیز وارد ناحیه بحرانی شود. برای رفع این مشکل می‌توان در بخش ورود، ابتدا اعلام ورود کرد و سپس شرایط را بررسی نمود. این امر در الگوریتم سوم و به صورت زیر پیاده سازی شده است:

P_i	P_j
---	---
---	---
---	---
1: Flag[i] = true	2: Flag[j] = true
While (Flag[j]);	While (Flag[i]);
C-S	C-S
Flag[i] = false	Flag[j] = false
---	---
---	---
---	---

در این الگوریتم نیز مقدار اولیه متغیرهای Flag، false خواهد بود. این الگوریتم شرط اول یعنی انحصار متقابل را دارد، زیرا هر فرآیندی که بخواهد وارد ناحیه بحرانی شود ابتدا فلگ خود را true می‌کند. پس اگر این فرآیند، اجازه ورود گرفته باشد فلگ آن true خواهد بود. در مورد این فرآیند چه دقیقاً بعد از عبور از while و چه در میان ناحیه بحرانی اگر تعویض متن صورت گیرد و CPU به فرآیند دیگر داده شود به دلیل true بودن فلگ فرآیند اول، فرآیند دوم اجازه ورود به ناحیه بحرانی را نخواهد داشت.

این الگوریتم به همان دلایلی که در الگوریتم دوم گفته شد شرط پیشرفت را دارد.

این الگوریتم شرط سوم یعنی انتظار محدود را ندارد، زیرا در صورت وقوع شرایطی برای آن، بن بست رخ می‌دهد. به طور مثال اگر فرآیند P_i تا پایان دستور ۱ انجام شود Flag[i] برابر true خواهد شد. سپس اگر تعویض متن صورت گیرد و فرآیند P_j اجرا شود در پایان اجرای دستور ۲، Flag[j] نیز true می‌شود و از این به بعد نه P_j و نه P_i امکان عبور از حلقه while را نخواهند داشت. پس تحت این شرایط، انتظار دو فرآیند برای ورود به ناحیه بحرانی، انتظاری است نامحدود.

☑ الگوریتم چهارم (تلاش چهارم)

مشکل بن بست موجود در الگوریتم سوم، از حلقه‌های بدون دستور while ناشی می‌شود، زیرا اگر شرایطی پیش آید که هر دو فرآیند وارد حلقه‌های while شوند از آنجایی که این دو حلقه بدون دستورند دیگر خروج از آن امکان پذیر نخواهد بود. برای رفع این مشکل در الگوریتم چهارم حلقه‌های بدون دستور while الگوریتم سوم به شکل زیر تغییر می‌کنند:

P_i	P_j
---	---
---	---
1: Flag[i] = true	2: Flag[j] = true
3: While (Flag[j]) {	4: While (Flag[i]) {
5: Flag[i] = false	6: Flag[j] = false
Delay	Delay
7: Flag[i] = true	8: Flag[j] = true
}	}
C-S	C-S
Flag[i] = false	Flag[j] = false
---	---
---	---

این الگوریتم شرط انحصار متقابل را دارد، زیرا هر فرآیندی که در ناحیه بحرانی باشد حتما قبل از فلگ آن true شده و اگر تعویض متن صورت گیرد و فرآیند دوم بخواهد وارد ناحیه بحرانی شود به خاطر true بودن فلگ فرآیند اول این اجازه به فرآیند دوم داده نمی‌شود. تا زمانی که فلگ فرآیند اول false شود و این false شدن تنها زمانی انجام می‌گیرد که فرآیند اول از ناحیه بحرانی اش خارج شود.

این الگوریتم به همان دلایلی که در الگوریتم دوم گفته شد شرط پیشرفت را دارد.

در مورد این الگوریتم می‌توان ادعا کرد که شرط سوم یعنی انتظار محدود را دارد، زیرا حتی اگر دو فلگ نیز true شوند باز هم در حلقه‌های while شرایطی وجود دارد که این دو فلگ تغییر کنند، از این رو می‌توان گفت که ممکن است در یک بار false شدن یکی از این فلگ‌ها فرآیند دیگری وارد ناحیه بحرانی شود، بنابراین، این الگوریتم دچار بن بست نمی‌شود.

در این الگوریتم شرایطی وجود دارد که تحت آن ممکن است در ورود به ناحیه بحرانی دو فرآیند مشکل پیش آید، به طور مثال اگر P_i تا پایان دستور ۱ اجرا شده و تعویض متن صورت گیرد و P_j تا پایان دستور ۲ انجام شود هر دو فلگ true می‌شود. حال اگر ترتیب دستورات ۳ تا ۸ در دو فرآیند به ترتیب شماره‌هایشان اجرا شوند باز شرایطی داریم که در پایان حلقه‌ها هر دو فلگ true می‌شوند یعنی باز هم دو فرآیند نمی‌توانند وارد ناحیه بحرانی شوند این شرایط بسیار خاص به اجرای یکی در میان دستورات و نیز به مقدار delay بستگی دارد اما از آنجایی که در آینده احتمال خروج از این شرایط وجود دارد به آن بن باز می‌گویند.

یکی از مشکلات این الگوریتم تنظیم میزان delay خواهد بود.

❑ الگوریتم Dekker

در الگوریتم Dekker به منظور تکامل الگوریتم‌های قبلی از هر دو متغیر بهنگام سازی استفاده می‌شود یعنی یک متغیر turn مشترک بین دو فرآیند خواهیم داشت و هر فرآیند یک متغیر اختصاصی فلگ دارد. مقدار اولیه دو فلگ false بوده و مقدار اولیه turn برابر شماره یکی از فرآیندهاست، در واقع با در نظر گرفتن این دو متغیر می‌-

توانیم اطمینان بیشتری حاصل کنیم که یک فرآیند در منطقه بحرانی است یا خیر. الگوریتم Dekker برای فرآیند P_i به صورت زیر است:

```

Pi
While (true)
{
    Flag[i] = true
    While (Flag[j])
    {
        If (turn == j)
        {
            Flag[i] = false
            While (turn == j);
            Flag[i] = true
        }
    }
    C-S
    Turn = j
    Flag[i] = false
}

```

الگوریتم Dekker هر سه شرط انحصار متقابل، پیشرفت و انتظار محدود را دارا می‌باشد، هر چند اثبات وجود این شرایط در این الگوریتم مشکل خواهد بود در مورد دو شرط اول می‌توانیم استدلالی مشابه آنچه در الگوریتم چهارم وجود داشت ذکر کنیم. الگوریتم Dekker حتی احتمال بسیار کم بن بست (بن باز) در الگوریتم چهارم را ندارد، زیرا در بخشی از الگوریتم چهارم که این مشکل را به وجود می‌آورد بخش معادل آن در الگوریتم Dekker از یک متغیر مشترک turn استفاده کرده که این متغیر در یک لحظه تنها می‌تواند یک مقدار داشته باشد. از سویی دیگر با استفاده از همین متغیر، بخش delay الگوریتم چهارم حذف شده است.

✓ الگوریتم پترسن^۱

<pre> P_i While (true) { Flag[i] = true Turn = j While (Flag[j] and turn == j) C-S Flag[i] = false } </pre>	<pre> P_j While (true) { Flag[j] = true Turn = i While (Flag[i] and turn == i) C-S Flag[j] = false } </pre>
---	---

الگوریتمی است که هر سه شرط انحصار متقابل، پیشرفت و انتظار محدود را دارا می‌باشد. از آنجایی که هر فرآیندی که قصد ورود به ناحیه بحرانی را دارد مقدار فلگ آن true می‌شود و نیز turn به نام آن است. هرگاه این فرآیند وارد ناحیه بحرانی شود حتماً فرآیند دیگر در حلقه while متوقف می‌شود، از این رو شرط انحصار متقابل برقرار است.

^۱ Peterson

این الگوریتم با استدلال‌های قبلی شرط پیشرفت را نیز دارد.

این الگوریتم شرط انتظار محدود را دارد یعنی هیچ وقت دچار بن بست نمی‌شود، زیرا بن بست زمانی اتفاق می‌افتد که شرایطی پیش آید که دو فرآیند در حلقه while گرفتار شوند. در این الگوریتم هرچند ممکن است هر دو فلگ، true شوند اما از آنجایی که turn بیشتر از یک مقدار نخواهد داشت پس یقیناً شرط کلی یکی از حلقه‌ها false می‌شود و فرآیند مربوط به آن وارد ناحیه بحرانی می‌شود یعنی بن بست نداریم.

راه حل‌های چندفرآیندی

✓ سمافورها^۱

سمافورها یکی از ابزارهای مورد استفاده برای حل مشکل ناحیه بحرانی هستند که می‌توان آنها را برای چندین فرآیند نیز به کار برد. در واقع سمافور یک نوع داده است و یک مقدار عددی صحیح در آن قرار می‌گیرد. یک متغیر از نوع سمافور به جز مقداردهی اولیه در معرض دو تابع اتمیک استاندارد به نام‌های wait () و signal () قرار می‌گیرد.

منظور از اتمیک بودن این دو تابع این است که این دو تابع به صورت وقفه ناپذیر اجرا می‌شوند. تابع wait () را با P () و تابع signal () را با V () نیز نشان می‌دهند.

دو تابع wait () و signal () دارای روش‌های مختلف پیاده سازی هستند اما ساده ترین شکل بیان این دو تابع به صورت زیر می‌باشد. اگر S یک متغیر از نوع سمافور باشد آنگاه :

```
Wait ( S ) {                               Signal ( S ) {
    While ( S <= 0 );                        S = S+1;
    S=S-1;                                  }
}
```

معمولاً یک متغیر از نوع سمافور دارای مقدار اولیه صحیح و مثبت خواهد بود. هنگامی که تابع wait روی آن اجرا می‌شود در صورت مثبت بودن، یکی از آن کم شده و از این تابع گذر می‌کنیم اما هنگامی که مقدار متغیر صفر باشد این تابع در انتظار مشغول به سر می‌برد. تابع signal تنها یک واحد به متغیر از نوع سمافور اضافه می‌کند.

✓ کاربرد سمافورها

دو استفاده عمده از سمافورها می‌شود:

^۱ Semaphore

✓ حل مشکل ناحیه بحرانی :

برای حل مشکل ناحیه بحرانی می توان از سمافورها استفاده کرد بدین شکل که ناحیه بحرانی چند فرآیند را بین یک تابع wait در بخش ورود و یک تابع signal در بخش خروج قرار دهیم. اغلب در این موارد مقدار اولیه متغیر سمافور برابر یک در نظر گرفته می شود. به طور مثال می توان مشکل ناحیه بحرانی دو فرآیند P_i و P_j را به صورت زیر حل نمود:

Semaphore S=1

P_i	P_j
---	...
---	---
Wait (S) ;	Wait (S) ;
C-S	C-S
Signal (S);	Signal (S);

✓ استفاده از سمافورها در هماهنگ سازی فرآیندها

از سمافورها می توان برای ایجاد هماهنگی در اجرای بخش های مختلف دو یا چند فرآیند نیز استفاده کرد، به طور مثال فرض کنیم در دو فرآیند P_0 و P_1 زیر بخواهیم دستور X در P_0 قبل از دستور Y در P_1 اجرا شود:

Semaphore A=0

P_0	P_1
---	---
---	---
X---	Wait (A) ;
Signal (A);	Y---
---	---
---	---

✓ انواع سمافورها

متغیرهای از نوع سمافور به دو دسته تقسیم می شوند:

✓ سمافورهای دودویی یا باینری:

متغیرهای از نوع سمافورهای دودویی متغیرهایی هستند که تنها می توانند دارای دو مقدار باشند. غالباً برای متغیرهای سمافور دودویی دو مقدار صفر و یک را در نظر می گیرند. بر روی متغیرهای سمافور دودویی دو تابع waitB() و signalB() قابل اجرا می باشند. تابع waitB() اگر متغیر سمافور دارای مقدار یک باشد یک واحد از آن کم می کند و از آن عبور می کند اما اگر مقدار متغیر صفر باشد هیچ تغییری حاصل نکرده و در آن متوقف می شود. تابع signalB() اگر مقدار متغیر سمافور صفر باشد یکی به آن می افزاید و از آن تابع عبور می کند. اما

اگر مقدار متغیر یک باشد هیچ تغییری در آن حاصل نکرده و از تابع عبور می‌کند. این دو تابع به دو شکل زیر تعریف می‌شوند:

```
WaitB ( S ) {
    While ( S <= 0 );
    S=S-1;
}

SignalB ( S ) {
    If ( S==0 )
        S = S+1;
}
```

✓ سمافورهای چندتایی (شمارشی، چرخشی)

یک متغیر از نوع سمافور شمارشی می‌تواند هر مقدار صحیح غیرمنفی را در خود داشته باشد. دو تابع قابل اجرا روی این سمافورها همان دو تابع wait () و signal () استاندارد هستند.

✓ مساله تولید و مصرف کننده با بافر نامحدود و استفاده از سمافور باینری

✓ تعریف مساله تولید کننده و مصرف کننده

فرض کنید یک بافر که دارای تعدادی خانه است موجود باشد و دو فرآیند به نام‌های تولید کننده و مصرف کننده قادر به استفاده از این بافر باشند. فرآیند تولید کننده در نظر دارد تا آیتم‌هایی را تولید و درخانه‌های بافر قرار دهد (در هرخانه تنها یک آیتم قرار می‌گیرد)، فرآیند مصرف کننده قصد مصرف این آیتم‌ها را دارد. می‌خواهیم با استفاده از سمافورها انحصار متقابل در دستیابی به بافر و نیز محدودیت‌های مساله را پیاده سازی کنیم.

Semaphore: S=1 delay=0
Intn=0

Producer		Consumer	
	While (true) {	5	waitB (delay)
	Produce an item		while (true) {
10,1	waitB (S)	19,15,6	waitB (S)
	Add item to Buffer		Remove item from buffer
11,2	n++	20,16,7	n--
12,3	if (n==1)	17,8	signalB (S)
	signalB (delay)		Consume the item
13,4	signalB (S)	18,14,9	if (n==0)
	}		waitB (delay)
			}

در این دو فرآیند از متغیر سمافور S برای حل مشکل ناحیه بحرانی (ایجاد انحصار متقابل) استفاده شد که مقدار اولیه آن برابر ۱ است. متغیر سمافور delay نیز برای ایجاد هماهنگی بین تولید کننده و مصرف کننده به کار گرفته شد و مقدار اولیه آن صفر است. همچنین متغیر صحیح n نشان دهنده تعداد آیتم‌های درون بافر است.

در دو فرآیند مطرح شده برای تولید کننده و مصرف کننده بعضی از ترتیب خاص اجرای دستورات می‌تواند مشکل ایجاد کند. به طور مثال اگر فرض کنیم دستورات فرآیند تولید کننده از ۱ تا ۴ انجام شوند و تعویض متن صورت گیرد (n=1, delay=1, S=1) و پس از آن CPU به مصرف کننده تعلق گیرد و دنباله دستورات از ۵ تا

۹ در مصرف کننده اجرا شود مقدار متغیرها به صورت ($n=0, \text{delay}=0, S=1$) خواهد بود. بنابراین مصرف کننده در دستور ۹ متوقف می‌شود تا تعویض متن صورت گیرد. پس از آن اگر دنباله دستورات از ۱۰ تا ۱۳ در تولید کننده اجرا شود مقادیر متغیرها به صورت ($n=1, \text{delay}=1, S=1$) خواهد بود. اگر چنانچه با تعویض متن، CPU دوباره به مصرف کننده برسد، مصرف کننده اجرای دستورات را از موقعیت ۱۴ دنبال می‌کند. در این زمان چون n برابر یک است بدون اجرای $\text{waitB}(\text{delay})$ فرآیند مصرف کننده اجازه یک بار دیگر اجرا شدن را خواهد داشت. در این صورت اگر دنباله دستورات از ۱۵ تا ۱۸ انجام شوند مقادیر متغیرها به صورت ($S=1, \text{delay}=1, n=0$) خواهد بود. این مقدار delay تا قبل از اجرای دستور ۱۸ است. پس از اجرای دستور ۱۸ چون $n=0$ است و $\text{delay}=1$ ، یک واحد از delay کم شده و به مصرف کننده اجازه اجرای مجدد داده می‌شود. در این صورت در این اجرای مجدد، مصرف کننده قصد برداشتن آیتمی را دارد که در بافر وجود ندارد یا به عبارت دیگر در دستور ۲۰ مقدار $n=-1$ می‌شود که یک مقدار غیر واقعی است.

اشکال به وجود آمده ناشی از دستور شماره ۹ است که در آن بر اساس متغیر n در مورد متغیر delay تصمیم گیری می‌شود یعنی بین دو دستور ۹ و ۱۴ که یک بار سراغ تولید کننده می‌رویم، مصرف کننده از تغییری که در n حاصل می‌شود آگاه نیست. برای رفع این مشکل باید تغییری که در این دستور، تصمیم گیری بر اساس آن انجام می‌گیرد یک متغیر محلی باشد. برای رفع این عیب فرآیند مصرف کننده به شکل زیر تغییر می‌کند:

Consumer

```

Int m
waitB ( delay )
while ( true ){
    waitB ( S )
    Remove item from buffer
    n--
    m = n
    signalB ( S )
    Consume the item
    If ( m==0 )
        waitB ( delay )
}

```

☑ مسئله تولید کننده و مصرف کننده با بافر نامحدود و استفاده از سمافور چندتایی

در این مدل به دلیل داشتن سمافور چندتایی می‌توانیم از متغیر سمافور برای نگه داری تعداد آیتم‌های بافر استفاده کنیم. برنامه این دو فرآیند به صورت زیر خواهد بود:

Semaphore: $S=1$ $n=0$

Producer

```
While ( true ) {
    Produce an item
    Wait ( S )
    Add item to buffer
    Signal ( S )
    Signal ( n )
}
```

Consumer

```
While ( true ) {
    Wait ( n )
    Wait ( S )
    Remove item from buffer
    Signal ( S )
    Consume the item
}
```

در این دو فرآیند، متغیر سمافور S برای ایجاد انحصار متقابل است که مقدار اولیه آن ۱ می‌باشد و متغیر سمافور n برای ایجاد هماهنگی دو فرآیند است و مقدار اولیه آن صفر خواهد بود.

مسئله تولیدکننده و مصرف کننده با بافر محدود و استفاده از سمافور چندتایی

در این مدل، علاوه بر اینکه مصرف کننده نمی‌تواند آیتمی از بافر خالی بردارد تولیدکننده نیز نخواهد توانست آیتمی در بافر پر قرار دهد. بنابراین دو فرآیند تولیدکننده و مصرف کننده را با استفاده از سه متغیر سمافور و به شکل زیر پیاده سازی می‌کنیم:

Semaphore: $S=1$, $n=0$, $e=|buffer|$

Producer

```
While ( true ){
    Produce an item
    Wait ( e )
    Wait ( S )
    Add item to buffer
    Signal ( S )
    Signal ( n )
}
```

Consumer

```
While ( true ) {
    Wait ( n )
    Wait ( S )
    Remove item from buffer
    Signal ( S )
    Signal ( e )
    Consume the item
}
```

☑ مسئله خوانندگان و نویسندگان

فایلی شامل یک یا چند رکورد به طور همزمان بین چند فرآیند خواننده و نویسنده به اشتراک گذاشته می‌شود. فرآیندهای خواننده تنها قصد خواندن از فایل را دارند و فرآیندهای نویسنده می‌خواهند بعضی از رکوردها را تغییر دهند. می‌خواهیم با استفاده از سمافورها انحصار متقابل در استفاده از فایل‌ها را برای یک خواننده و یک نویسنده و نیز برای دو نویسنده ایجاد کنیم. ایجاد انحصار متقابل برای دو یا چند خواننده لازم نیست زیرا دو یا چند خواننده می‌توانند به طور همزمان یک فایل را بخوانند.

پیاده سازی دو فرآیند خواننده و نویسنده به صورت زیر خواهد بود:

Semaphore: wsem=1 X=1

IntreadCount=0

Reader

```

While ( true ) {
    Wait ( X )
    readCount ++
    if (readCount==1)
        wait ( wsem )
    signal ( X )
    Reading...
    Wait ( X )
    readCount --
    if ( readCount==0 )
        signal ( wsem )
    signal ( X )
}

```

Writer

```

While ( true ) {
    Wait ( wsem )
    Writing...
    Signal ( wsem )
}

```

در این دو فرآیند از دو متغیر سمافور استفاده شده که هر دو برای ایجاد انحصار متقابل (حل مشکل ناحیه بحرانی) به کار می‌روند و مقدار اولیه هر دو یک است. سمافور wsem برای انحصار متقابل در دستیابی به فایل و سمافور X برای انحصار متقابل در استفاده از read count به کار می‌روند.

در برنامه نوشته شده، خوانندگان نسبت به نویسندگان از اولویت بالاتری برخوردارند. این اولویت به این معناست که اگر یک خواننده مشغول خواندن باشد و یک نویسنده درخواست نوشتن دهد متوقف می‌شود. اگر در زمانهای بعدی خواننده‌های دیگری درخواست خواندن بدهند مادامی که خواننده اول مشغول خواندن است، خواننده‌های دیگر نیز می‌توانند از فایل بخوانند. بنابراین چنین به نظر می‌رسد که این خواننده‌های دیگر که پس از نویسنده درخواستشان را مطرح کرده اند اولویت بالاتری از آن نویسنده دارند. به منظور دادن اولویت بالاتر به نویسندگان این دو فرآیند به شکل زیر پیاده سازی می‌شوند:

Semaphore wsem=1 , X=1 , rsem=1, Y=1, Z=1

IntreadCount=0 writeCount=0

Reader

```
While ( true ) {
    Wait ( Z )
    Wait ( rsem )
    Wait ( X )
    readCount ++
    if ( readCount==1 )
        wait ( wsem )
    signal ( X )
    signal ( rsem )
    signal ( Z )
    Reading ...
    Wait ( X )
    readCount --
    if ( readCount==0)
        signal ( wsem )
    signal ( X )
}
```

Writer

```
While ( true ) {
    Wait ( Y )
    writeCount ++
    if ( writeCount==1)
        wait ( rsem )
    signal ( Y )
    wait ( wsem )
    Writing ...
    Signal ( wsem )
    Wait ( Y )
    writeCount --
    if ( writeCount == 0)
        signal ( rsem )
    signal ( Y )
}
```

در این دو برنامه از چند متغیر سمافور استفاده شده که وظایف هریک به شکل زیر است:

متغیر سمافور wsem برای ایجاد انحصار متقابل، در دستیابی به فایل به کار می‌رود و مقدار اولیه آن یک خواهد بود.

متغیر سمافور X در ایجاد انحصار متقابل برای تغییر و استفاده متغیر مشترک read count بین خواننده‌ها به کار می‌رود و مقدار اولیه آن یک خواهد بود.

متغیر سمافور rsem برای اولویت دادن به نویسندگان به کار می‌رود و مقدار اولیه آن برابر یک خواهد بود.

متغیر سمافور Y برای ایجاد انحصار متقابل در دستیابی نویسندگان به write count به کار می‌رود و مقدار اولیه آن یک خواهد بود.

متغیر سمافور Z برای ایجاد یک صف جدا برای خواننده‌ها به کار می‌رود و مقدار اولیه آن یک خواهد بود. (صف rsem اولویت بالاتری از صف Z برای اجرا شدن خواهد داشت)

دو متغیر صحیح read count و write count به ترتیب تعداد خواننده‌ها و تعداد نویسندگان را نشان می‌دهند و مقدار اولیه شان صفر است.

✓ ناظر^۱

هرچند سمافورها یکی از بهترین ابزارهای ایجاد انحصار متقابل در اجرای نواحی بحرانی هستند اما دارای دو عیب کلی زیر هستند:

اولا اینکه ممکن است توابع wait و signal مربوط به سمافور در سرتاسر چندین فرآیند پخش شوند و از این رو کنترل و مدیریت آنها مشکل باشد.

ثانیا متغیرهای از نوع سمافور، متغیرهایی هستند که در دسترس تمامی فرآیندهای دارای ناحیه بحرانی خواهند بود و یک اشتباه کوچک در به کاربردن تابع wait یا signal یکی از فرآیندها می‌تواند فرآیندهای دیگر را دچار مشکل کند.

ناظر، ساختار دیگری است که برای ایجاد انحصار متقابل در اجرای ناحیه بحرانی فرآیندها استفاده می‌شود. ناظرها همان توانایی سمافورها را دارند با این تفاوت که ساختار آنها و کنترل آنها بسیار ساده تر است. ساختار ناظر به عنوان یک تابع سیستم‌عامل است و در اغلب زبانهای برنامه نویسی امکان استفاده از آن وجود دارد. ناظر یک مولفه نرم افزاری است که دارای چند تابع و متغیر محلی است. ویژگیهای اصلی ناظر عبارتند از:

- ۱- متغیرهای محلی ناظر، تنها برای خود ناظر قابل دسترسی هستند.
- ۲- هر فرآیند با احضار ناظر در صف ورود به آن قرار می‌گیرد.
- ۳- در هر زمان تنها یک فرآیند می‌تواند در ناظر در حال اجرا باشد، در این زمان فرآیند متقاضی دیگر در صف ورود، منتظر می‌ماند. این ویژگی بیان می‌کند که ناظر می‌تواند انحصار متقابل را برقرار سازد.
- متغیرهایی که در داخل ناظر استفاده می‌شوند متغیرهای شرطی (کنترل کننده شرایط) هستند. بر روی این متغیرها، دو تابع قابل اعمال است. به طور مثال اگر C یک متغیر شرطی ناظر باشد این دو تابع عبارتند از:

۱- Cwati (C)

فرآیندی که این عملیات را فراخوانی می‌کند روی شرط C به تعویق می‌افتد، درواقع اجرای این تابع باعث می‌شود که فرآیند اجراکننده ناظر را ترک کرده و آن را برای فرآیند دیگر مهیا سازد.

۲- Csignal (C)

یک فرآیند، با اجرای این تابع، اجرای فرآیند دیگر را که بر روی عمل Cwait معلق مانده بود از سر می‌گیرد. اگر چندین فرآیند در صف معلق باشند یکی از آنها انتخاب می‌شود. اما اگر هیچ فرآیندی در صف معلق نباشد هیچ کاری صورت نخواهد گرفت.

^۱ Monitor

در مورد ناظرها، اگر هریک از دو تابع $Cwati$ یا $Csignal$ انجام شوند و هیچ وظیفه ای بر روی متغیر شرطی مورد نظر منتظر نباشد علامت صادرشده گم می شود.

نمونه سوال

۱- دو فرآیند P_1 و P_2 زیر به صورت همروند (اجرای موازی) اجرا می شوند و امکان اجرای آنها به صورت در بین هم وجود دارد. در صورتیکه مقدار اولیه متغیر a ، صفر باشد بعد از اجرای کامل دو فرآیند کدام یک از گزینه های زیر نادرست است؟

کد P_1	کد P_2
$a=1$	$b=a$ $c=a$

(الف) مقادیر a و b هر کدام یک می باشد و مقدار c ، صفر است.

(ب) هریک از مقادیر a و b و c یک می باشند.

(ج) مقادیر a و c هر کدام یک می باشد و مقدار b صفر است.

(د) مقادیر b و c صفر می باشند و مقدار a یک است.

۲- اگر مقدار اولیه متغیر x برابر صفر باشد در صورتیکه i متغیر مشترک بین دو فرآیند P_0 و P_1 در یک سیستم تک پردازنده باشد و این دو فرآیند به صورت همروند اجرا شوند حداکثر مقدار x پس از اجرای دو فرآیند چه مقدار می تواند باشد؟

P_0	P_1
For ($i=0$; $i<3$; $i++$) $x++$;	For ($i=0$; $i<3$; $i++$) $x--$;
(د) ۴	(الف) ۶ (ب) صفر (ج) ۳

۳- سه فرآیند P_1 ، P_2 ، P_3 در حالت اجرا هستند و طبق جدول زیر (به ترتیب از چپ به راست) دستورات wait و signal را روی سمافور s که مقدار اولیه آن ۱ است اجرا می کنند. در صورتی که دو فرآیند متوقف باشند و دستور signal صادر شود فرآیندی که شماره بزرگتری دارد برای اجرا اولویت پیدا می کند. حالت این سه فرآیند پس از اجرای دستورات زیر چیست؟

فرآیند	P_1	P_2	P_3	P_2	P_1	P_3	P_2	P_2	P_3	P_1
فرمان	W (s)	W (s)	S (s)	W (s)	S (s)	S (s)	W (s)	W (s)	S (s)	W (s)

(الف) همه فرآیندها در حالت اجرا هستند.

(ب) P_1 و P_2 در حالت اجرا هستند و P_3 در حالت مسدود است.

(ج) P_1 و P_3 در حالت اجرا هستند و P_2 در حالت مسدود است.

(د) P_2 و P_3 در حالت اجرا هستند و P_1 در حالت مسدود است.

سوالات تستی

☑ نیم سال اول ۸۹-۹۰

۱- فرآیندهایی را در نظر بگیرید که از اسامی (شناسه فرآیند. همدیگر مطلع نیستند ولی در دسترسی به بعضی اشیاء مانند بافر (میانگیر) ورودی/خروجی مشترکند. در حقیقت این فرآیندها با چه روشی باهم محاوره می کنند؟

- الف) اطلاع غیر مستقیم فرآیندها از یکدیگر ب) اطلاع مستقیم فرآیندها از یکدیگر
ج) بی اطلاعی فرآیندها از یکدیگر د) رقابت فرآیندها با یکدیگر

۲- آیا کد زیر برای مساله تولید کننده و مصرف کننده قابل قبول است؟ چرا؟

```
Semaphore n=0, s=1;
void consumer( ) {
    void producer( ) {
        while(1) {
            while(1) {
                produce( );
                wait(s);wait(n);wait(s);
                append( );
                take( );
                signal(s);
                signal(n);
                signal(s);
            }
        }
        consume( );
    }
}
```

الف) خیر. چون مصرف کننده وارد بخش بحرانی خود نمی شود.

ب) بلی. بن بست رخ می دهد.

ج) بلی. انحصار متقابل برقرار است.

د) بلی. گرسنگی و بن بست رخ نمی دهد.

۳- کدام یک جزء مزایای استفاده از دستورالعمل ویژه ماشین برای اعمال انحصار متقابل نمی‌باشد؟

الف) برای هر تعداد از فرآیندها، که از حافظه مشترک استفاده میکنند قابل بکارگیری است.

ب) امکان گرسنگی وجود نخواهد داشت.

ج) ساده است و بنابراین واریسی آن آسان است.

د) از آن برای حمایت از بخشهای بحرانی متعدد میتوان استفاده نمود.

۴- کدامیک از موارد زیر در رابطه با ناظرها و راهنماها صحیح میباشد؟

الف) خود ساختار راهنما، انحصار متقابل را اعمال میکند.

ب) درمورد استفاده از ناظر، مسئولیت انحصار متقابل و همچنین همگام سازی به عهده برنامه ساز است.

ج) در هر دو مسئولیت انحصار متقابل و همچنین همگام سازی بر عهده برنامه ساز است.

د) درمورد استفاده از ناظر، برنامه ساز باید اولیه های $Cwait$ و $Csignal$ را طور مناسب در ناظر قرار دهد.

✓ نیم سال دوم ۸۹-۹۰

۱- دو فرآیند زیر به صورت همروند اجرا میشوند. تمامی دستورات این فرآیندها یکپارچه هستند و مقدار اولیه

و متغیرهای x و y صفر می باشد و مقدار سمافور دودویی $mutex$ یک است. بعد از اجرای کامل دو

فرآیند، متغیرهای x و y کدام گزینه نمیتواند باشد؟

P1	P2
$x=1$	$x=x+2$
$wait(mutex)$	$wait(mutex)$
$y=y+x$	$y=y-1$
$x=2$	$x=x-y$
$signal(mutex)$	$signal(mutex)$

ب) $x=2, y=3$

الف) $x=0, y=2$

د) $x=2, y=0$

ج) $x=1, y=2$

۲- جهت ایجاد شرایط انحصار متقابل به کمک سمافورها، کدام روش درست است؟

الف) استفاده از دو سمافور با مقدار اولیه صفر و اجرای دستور $wait$ برای سمافور دوم و دستور $signal$ برای

سمافور اول.

ب) استفاده از یک سمافور با مقدار اولیه یک و اجرای دستور $wait$ قبل و $signal$ بعد از ناحیه بحرانی.

ج) استفاده از یک سمافور با مقدار اولیه صفر و اجرای دستور $wait$ قبل و $signal$ بعد از ناحیه بحرانی.

د) استفاده از یک سمافور با مقدار اولیه دلخواه و اجرای دستور $wait$ بعد از $signal$ و قبل از ناحیه بحرانی.

۳- در یک سیستم تک پردازنده ای اشتراک زمانی، سه پردازش زیر مفروض است. در صورت اجرای همزمان آنها، کدام خروجی اصلا رخ نمی دهد؟

P1	P2	P2
while(1) { Wait(A); Printf("C"); Signal(C) }	while(1) { Wait(B); Printf("A"); wait(B); Printf("B") Signal(A); }	while(1) { Wait(C); Printf("B"); signal(B); }
BCCA (د)	BCBA (ج)	CCBC (ب) CBBA (الف)

۴- کدامیک از گزینه های زیر برای حل مساله تولید کننده مناسب است؟

الف) الگوریتم Peterson	ب) روش TSL (دستوالعمل های ویژه ماشین)
ج) استفاده از راهنما	د) الگوریتم Dekker

سوالات تشریحی

✓ نیم سال اول ۸۹-۹۰

- ۱- با استفاده از پیامها، راه حلی برای مسئله تولید کنندگان و مصرف کنندگان با میانگیر محدود بنویسید (۱/۲۵ نمره)

✓ نیم سال دوم ۸۹-۹۰

- ۱- روش ناظر با علامت در حل مساله همزمانی را شرح دهید (۱/۲۵ نمره)
- ۲- هدف الگوریتم Peterson چیست؟ قطعه کد آنرا نوشته و تشریح نمایید (۱/۲۵ نمره)

پاسخنامه سوالات تستی

نیم سال اول ۸۹-۹۰	
الف	۱
ب	۲
ب	۳
د	۴
نیم سال دوم ۸۹-۹۰	
ج	۱
ب	۲
ب	۳
ج	۴

Pnu-Soal.ir

فصل پنجم^۱

همزمانی فرایندها

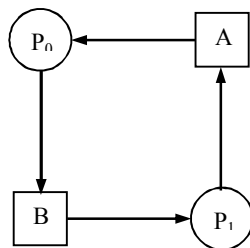
Pnu-Soal.ir

^۱ این فصل، فصل ششم کتاب می باشد.

در یک سیستم چند برنامگی امکان رقابت فرایندها برای در اختیار گرفتن منابع وجود دارد. اگر حالتی پیش آید که چند فرآیند منابعی را در اختیار داشته باشند و منتظر به خدمت گرفتن منابعی باشند که در اختیار فرآیندهای دیگر است، ممکن است حالتی به نام بن بست رخ دهد.

برای تشریح این حالت می‌توانیم مثال زیر را در نظر بگیریم:

مثال ۵ - ۱ اگر فرآیندی مانند P_0 منبع A را در اختیار داشته باشد و درخواست منبع B را مطرح کند و از سوی دیگر فرآیند منبع B را در اختیار داشته و منبع A را تقاضا کند بین این نگهداری منابع و تقاضای منابع اصطلاحاً انتظار مدور پیش می‌آید که این می‌تواند باعث وقوع بن بست شود.



رفتار یک فرآیند در مورد یک منبع می‌تواند شامل ۳ مورد زیر باشد:

- ۱- فرآیند منبع را تقاضا کند.
- ۲- فرآیند منبع را در اختیار گیرد.
- ۳- فرآیند پس از اتمام کارش با منبع آن را آزاد کند.

✓ شرایط وقوع بن بست

برای وقوع بن بست در یک سیستم چند شرط وجود دارد، که جزء شرایط لازم برای وقوع بن بست محسوب می‌شوند، این شرایط عبارتند از:

✓ انحصار متقابل^۱

منابعی در سیستم وجود دارند که به طور همزمان نمی‌توانند در خدمت چند فرآیند باشند. یا به عبارتی قابلیت استفاده‌ی مشترک را ندارند. یعنی تا کار یک فرآیند را به طور کامل اجرا نکنند نمی‌توانند در اختیار فرآیند دیگر قرار گیرند. این منابع، دارای ویژگی انحصار متقابل خواهند بود.

✓ نگهداری و انتظار^۲

این شرط، مطرح می‌کند که برای وقوع بن بست، لازم است منابعی که در اختیار فرایندها هستند و فرآیندهای دیگری انتظار آنها را می‌کشند تا تکمیل اجرای فرایندها آزاد نشوند به عبارتی اگر فرآیندی منبعی را در اختیار

^۱ Mutual Exclusion

^۲ Hold and wait

دارد و منتظر منبع دیگری است اجازه ی آزاد شدن منبع در اختیارش را تا قبل از گرفتن منبع مورد تقاضایش صادر نکند.

☑ انقطاع ناپذیری^۱ (بدون قبضه کردن)

منابعی که در بن بست گرفتار می شوند، باید به گونه ای باشند که نتوان آنها را در حین استفاده توسط یک فرآیند، از آن فرآیند گرفته و به فرآیند دیگر داد، به این شرط اصطلاحاً انقطاع ناپذیری یا به اصطلاح، بدون قبضه کردن می گویند.

☑ انتظار مدور^۲ (چرخشی)

این شرط بیان می کند که اگر بخواهد بن بست رخ دهد باید بین نگهداری منابع، توسط فرآیندها و انتظار فرآیندها برای منابع دیگر، یک حلقه یا دور ایجاد شود. اگر چنین حلقه ای وجود نداشته باشد به هیچ عنوان بن بست رخ نمی دهد.

برای وقوع بن بست، لازم است که این ۴ شرط، همزمان با هم رخ دهد. اما حتی اگر یکی از آنها هم نقض شود، بن بست رخ نمی دهد.

راههای مقابله با بن بست

سه راه مقابله با بن بست وجود دارد که عبارتند از:

- ۱- پیشگیری از بن بست
- ۲- اجتناب از بن بست
- ۳- برخورد با بن بست

☑ پیشگیری از بن بست

برای پیشگیری از وقوع بن بست، لازم است که یکی از شرایط چهارگانه ی وقوع بن بست را نقض کنیم، هرچند ممکن است نقض بعضی از این شرایط، در عمل، غیرممکن باشد، اما برای هر یک از این شرایط و نقض آنها می توان به شکل زیر عمل نمود:

نقض شرط انحصار متقابل

بعضی از منابع در سیستم وجود دارند که به هیچ عنوان نمی توان آنها را به طور مشترک توسط چند فرآیند استفاده نمود. در مورد این منابع نمی توان شرط انحصار متقابل را نقض کرد اما بعضی از منابع، مانند فایل های فقط خواندنی می توانند به طور مشترک توسط چند فرآیند استفاده شوند، بنابراین به هنگام استفاده آنها توسط

¹ Non preemption

² Circular waiting

یک فرآیند، آنها را از دسترس فرآیندهای دیگر، خارج نمی کنیم. بنابراین، این شرط بستگی به خود منابع خواهد داشت.

نقض شرط نگهداری و انتظار

برای نقض این شرط باید مانع از ایجاد حالتی بشویم که در آن یک فرایند منبعی را در اختیار گیرد و تقاضای منبع دیگری را مطرح کند برای این کار می توان از دو روش زیر استفاده کرد:

روش اول: در این روش قبل از شروع اجرای هر فرآیند، کل منابع مورد نیاز آن فرآیند را در اختیارش قرار می دهیم، با این کار دیگر هیچ فرآیندی انتظار به خدمت گرفتن هیچ منبعی را نخواهد داشت.

روش دوم: در این روش به یک فرآیند، زمانی اجازه ی درخواست منبع را می دهیم که هیچ منبع دیگری در اختیارش نباشد، بنابراین هیچ فرآیندی در زمان مطرح کردن تقاضا، هیچ منبع دیگری را نگهداری نخواهد کرد.

نقض هر دوی این شرایط، تقریباً غیر ممکن است. در مورد روش اول هم محاسبه ی کل منابع مورد نیاز یک فرآیند قبل از اجرای آن غیر ممکن است و هم اینکه به علت محدود بودن تعداد منابع، تخصیص همه ی منابع مورد نیاز فرآیندها قبل از اجرای آنها امکان پذیر نخواهد بود. در مورد روش دوم نیز بسیار اتفاق خواهد افتاد که نتیجه ی کار یک فرایند با یک منبع به در اختیار گرفتن منبع دیگری بستگی داشته باشد.

نقض شرط انقطاع ناپذیری

برای نقض این شرط می توان هنگامی که یک فرآیند، تقاضای یک منبع را مطرح می کند و امکان تخصیص آن وجود ندارد فرآیند را بررسی کرد. در صورتی که این فرآیند منابعی را در اختیار دارد که مورد تقاضای فرآیندهای دیگر است آنها را از این فرآیند قبضه نموده و به فرآیندهای دیگر تخصیص داد.

مشکل این روش زمانی خواهد بود که یک منبع نتواند به طور منقطع توسط یک فرآیند استفاده شود. در اینصورت اگر این منبع از فرآیند مورد نظر قبضه شود برای اجرای دوباره فرآیند باید یکبار دیگر آن را از ابتدا اجرا کنیم.

نقض شرط انتظار مدور

برای نقض این شرط، باید به شکلی دور را در نگهداری و انتظار فرآیندها نسبت به منابع از بین ببریم. به طور مثال می توانیم فرآیندها و منابع را شماره گذاری کنیم و به هر فرآیند تنها اجازه به خدمت گرفتن منابع با شماره بزرگتر یا مساوی آن را بدهیم. به طور مثال اگر دو فرآیند P_1 و P_2 و دو منبع R_1 و R_2 در سیستم باشند P_1 اجازه ی درخواست R_1 و R_2 را دارد اما P_2 تنها اجازه ی درخواست R_2 را خواهد داشت بنابراین از ایجاد دور جلوگیری می شود.

✓ اجتناب از بن بست

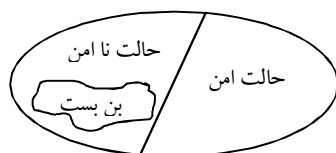
در این روش، برای جلوگیری از وقوع بن بست، سیستم عامل فرآیندها را بررسی می کند و چنانچه ترتیبی برای اجرای آنها بدون بن بست وجود داشته باشد آن را انتخاب می کند. در زمان اجرای فرآیندها و کار آنها با منابع، دو حالت می تواند در سیستم رخ دهد:

حالت امن

حالتی است که در آن حداقل یک ترتیب اجرای امن برای فرآیندها وجود دارد. یک ترتیب اجرای امن ترتیبی از اجرای فرآیندهاست که به هیچ عنوان باعث بروز بن بست نخواهد شد.

حالت نا امن

اگر در مورد چند فرآیند به هیچ عنوان نتوان ترتیب اجرای امنی پیدا کرد، آنگاه سیستم در حالت نا امن خواهد بود. در حالت نا امن امکان وقوع بن بست وجود دارد (تحت هر شرایطی مثلاً از بین رفتن یکی از فرآیندها ممکن است بن بست رخ دهد). در مورد حالت امن، حالت نا امن و بن بست می توان نمودار زیر را رسم نمود:



✓ الگوریتم بانکداران^۱

برای بررسی یک سیستم جهت یافتن یک ترتیب اجرای امن، از الگوریتمی به نام Banker یا بانکداران استفاده می شود. این الگوریتم با توجه به فرآیندها و منابع موجود در سیستم از داده های زیر استفاده می کند:

ماتریس (Max) claim :

یک بردار دو بعدی است که هر عضو آن مشخص می کند که هر فرآیندی برای اجرا شدن حداکثر به چه تعداد از هر منبع نیاز دارد. این مقدار حداکثر، هم شامل تعدادی می شود که هم اکنون در اختیار فرآیند است و هم تعدادی که فرآیند در آینده آنها را درخواست خواهد کرد. یعنی اگر $claim[i][j] = x$ باشد، فرآیند P_i برای اجرا شدن حداکثر به x منبع نیاز دارد این ماتریس را حداکثر منابع مورد نیاز هم می نامند.

ماتریس Allocation

یک آرایه دو بعدی است که هر عضو آن بیان می کند که هم اکنون به چه تعداد از هر منبع در اختیار هر فرآیند است. به عبارت دیگر اگر $Allocation[i][j] = y$ باشد یعنی هم اکنون به تعداد y از منبع r_j در اختیار P_i خواهد بود. این ماتریس را ماتریس منابع تخصیص یافته نیز می نامند.

^۱ Banker

ماتریس Need

یک آرایه دو بعدی است که هر عضو آن بیان می کند هر فرآیند از این به بعد به چه تعداد از هر منبع برای کامل شدن نیاز دارد. یعنی اگر $Need[i][j] = k$ باشد فرایند P_i برای اجرای کاملش به k منبع r_j دیگر نیاز دارد. ماتریس Need از رابطه زیر محاسبه می شود:

$$Need = Claim - Allocation$$

بردار Resource

یک آرایه یک بعدی است که هر عضو آن بیان می کند که به طور کلی از هر منبع به چه تعداد در سیستم وجود دارد. به عبارت دیگر اگر $Resource[j] = q$ باشد یعنی به تعداد q از منبع r_j در سیستم وجود دارد. Resource را بردار منابع اولیه نیز می نامند.

بردار Available

یک آرایه یک بعدی است که هر عضو آن نشان می دهد هم اکنون از هر منبع به چه تعداد به صورت آزاد در سیستم موجود است. چنانچه $Available[j] = q'$ باشد یعنی هم اکنون به تعداد q' از منبع r_j به طور آزاد در سیستم خواهیم داشت. این بردار را بردار منابع موجود نیز می نامند.

برای اجرای الگوریتم Banker ابتدا ماتریس Need را محاسبه می کنیم. سپس بردار Available را بدست می آوریم. برای محاسبه Available کل منابع یکسان تخصیص یافته به فرآیندها را محاسبه می کنیم (جمع می کنیم) سپس آن را از مقدار کلی آن منبع در بردار Resource کم می کنیم تا تعداد آزاد آن منبع به دست آید.

برای اجرای الگوریتم Banker باید یکی از فرآیندهای ماتریس Need را که می تواند اجرا شوند انتخاب کنیم. فرآیندی از ماتریس Need می تواند انتخاب شود که همه نیازهایش کمتر یا مساوی با منابع بردار Available باشد. اگر چنین فرآیندی را پیدا کردیم آن را در ترتیب می نویسیم (در ترتیب اجرای امن)، سپس مقدار جدید Available را محاسبه می کنیم. بدین منظور، سطر مربوط به فرآیند انتخاب شده را در ماتریس Allocation یافته و به مقدار جاری Available اضافه می کنیم و این روند را به همین شکل ادامه می دهیم. اگر بتوانیم ترتیبی مطرح کنیم که در آن همه فرآیندها اجرا شوند آن ترتیب، ترتیب اجرای امن خواهد بود.

اگر در زمان انتخاب یک فرآیند از ماتریس Need، بیش از یک فرآیند برای انتخاب داشته باشیم به دلخواه یکی را انتخاب می کنیم. این انتخاب تاثیری در حالت امن یا ناامن بودن سیستم ندارد، بنابراین می توان گفت ترتیب اجرای امن می تواند منحصر به فرد نباشد.

اگر طی اجرای الگوریتم Banker یک یا بیش از یک فرآیند نتوانند اجرا شوند ترتیب اجرای امن نخواهیم داشت و سیستم در حالت ناامن خواهد بود که احتمال وقوع بن بست وجود دارد.

مثال ۵ - ۲ سیستمی شامل ۵ فرآیند و ۳ منبع A، B و C را در نظر بگیرید. ماتریس‌های Claim و Allocation و هم‌چنین بردار Resource به صورت زیر خواهند بود. آیا سیستم در حالت امن است؟ چرا؟

Claim			Allocation			Resource		
	A	B	C		A	B	C	
P ₀	7	5	3	P ₀	0	1	0	10
P ₁	3	2	2	P ₁	2	0	0	5
P ₂	9	0	2	P ₂	3	0	2	7
P ₃	2	2	2	P ₃	2	1	1	
P ₄	4	3	3	P ₄	0	0	2	
Need			Available					
	A	B	C		A	B	C	
P ₀	7	4	3		3	3	2	
P ₁	1	2	2		5	3	2	
P ₂	6	0	0		7	4	3	
P ₃	0	1	1		7	5	3	
P ₄	4	3	1		10	5	5	
					10	5	7	

سیستم در حالت امن خواهد بود زیرا حداقل یک ترتیب اجرای امن برای فرآیندها وجود دارد.

مثال ۵ - ۳ در سیستم مثال قبل فرض کنید فرآیند P₁ یک نمونه دیگر از منبع A و دو نمونه دیگر از منبع C درخواست کند. با این فرض سیستم در چه حالتی خواهد بود؟

اگر با توجه به شرایط موجود در مسئله، یک سری درخواستهای دیگر نیز مطرح شود، حتما باید ابتدا درخواستهای جدید مطرح شده را پاسخ دهیم سپس با استفاده از الگوریتم Banker حالت سیستم را بررسی کنیم. در این صورت یکی از سه مورد زیر رخ می‌دهد:

۱- اگر چنانچه با توجه به مقدار اولیه Available بتوانیم درخواستهای جدید را پاسخ دهیم (درخواست حداقل یکی از منابع از تعداد آزاد آن بیشتر باشد. بدون اجرای الگوریتم Banker می‌گوییم سیستم در حالت ناامن است.

۲- اگر با توجه به منابع موجود در Available بتوانیم درخواستهای جدید را پاسخ دهیم، پس از پاسخ دهی این درخواستها (کم کردن منابع تخصیص داده شده از Available و افزودن آنها به Allocation مورد نظر بدون تغییر Need) الگوریتم Banker را اجرا می‌کنیم که در صورت یافتن یک ترتیب اجرای امن، سیستم در حالت امن خواهد بود.

۳- در این مورد نیز شرایط مورد دوم برقرار است اما با اجرای الگوریتم Banker متوجه می‌شویم که ترتیب اجرای امن وجود ندارد و حالت سیستم ناامن است.

با توجه به شرایط موجود هیچ فرآیندی نمی تواند اجرا شود، پس ترتیب اجرای امن نداریم و سیستم در حالت ناامن است و احتمال وقوع بن بست وجود دارد.

✓ برخورد با بن بست

در این روش هیچ تلاشی برای پیشگیری یا اجتناب از بن بست قبل از وقوع آن صورت نمی گیرد، بلکه سیستم صبر میکند تا زمانی که بن بست رخ دهد، تا در صورت رخ دادن بن بست با آن برخورد کند. در این روش، سیستم ابتدا باید راهی جهت کشف بن بست داشته باشد. برای کشف بن بست می توان به دو روش زیر عمل کرد:

استفاده از wait for graph

در این روش برای فرآیندها و منابع در اختیار آنها و همچنین منابع مورد نیازشان، یک گراف تشکیل می شود. در این گراف هر فرآیند به صورت (P) و هر منبع به صورت (R) نشان داده می شوند. یالهای این گراف به دو شکل توصیف می شوند:

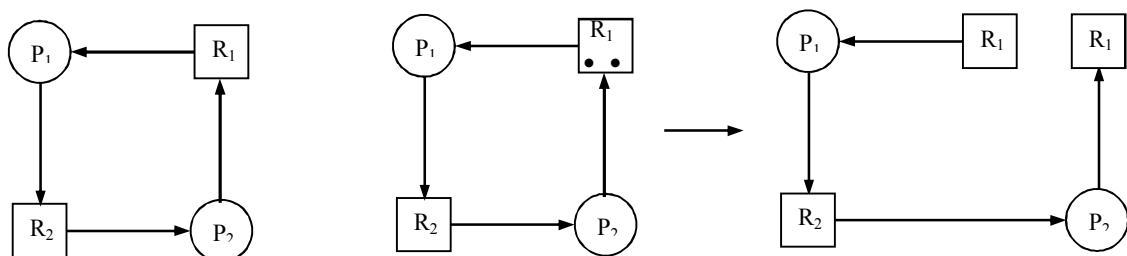
الف - $(R_1) \rightarrow (P_1)$ به معنی آن است که فرآیند P_1 منبع R_1 را در اختیار دارد.

ب - $(P_1) \leftarrow (R_1)$ به معنی آن است که فرآیند P_1 منبع R_1 را درخواست کرده است.

در گراف انتظار چنانچه از یک منبع بیش از یکی در سیستم باشد یا آن منبع در گراف تکرار می شود و یا اینکه تنها یکبار رسم می شود و با قرار گرفتن تعدادی نقطه در آن تعداد این منبع در سیستم مشخص می شود. به طور

مثال (R_1) نشان می دهد که دو واحد از منبع R_1 در سیستم وجود دارد.

با توجه به این نکات هرگاه در گراف انتظار دور ایجاد شود بن بست می دهد.



در این روش به شکل خاصی از الگوریتم Banker استفاده می شود، بدین صورت که پس از ساختن ماتریس Need فرآیندی را که تمامی منابع مورد نیازش صفر باشد را علامت می زنیم. همچنین فرآیندی را که تمامی منابع مربوط به آن در Allocation صفر باشد علامت می زنیم. این دو نوع فرآیند تاثیری در بن بست نخواهد داشت، زیرا فرآیندی که تمامی سطر مربوط به آن در Need صفر باشد به هیچ منبع جدیدی نیاز ندارد. بنابراین خود به خود به طور کامل اجرا می شود. همچنین فرآیندی که تمامی سطر آن در Allocation صفر باشد هیچ

منبعی را در اختیار ندارد یا به اصطلاح هیچ منبعی را نگهداری نکرده است. بنابراین شرط نگهداری و انتظار در مورد این دو نوع فرآیند برقرار نخواهد بود و این دو فرآیند درگیر بن بست نمی‌شوند. سپس در مورد بقیه فرآیندهای علامت نخورده، مطابق همان روندی که در الگوریتم Banker توصیف شد عمل می‌کنیم و هر فرآیندی را که شرایط اجرایش وجود دارد علامت می‌زنیم. اگر در پایان، هیچ فرآیند علامت نخورده‌ای وجود نداشته باشد در حال حاضر در سیستم بن بست نداریم اما اگر حداقل یک فرآیند علامت نخورده داشته باشیم اکنون در سیستم بن بست رخ داده است.

هر دو روش کشف بن بست، تنها قادرند وضعیت سیستم را به طور لحظه‌ای بررسی کنند یعنی اگر با اجرای هر کدام از روش‌ها به این نتیجه برسیم که هم اکنون بن بست در سیستم وجود ندارد هیچ تضمینی نخواهد بود که در آینده هم بن بست نداشته باشیم. بنابراین باید هریک از این دو روش به طور متناوب تکرار شوند تا در صورت بروز بن بست بتوانیم آن را کشف کنیم. پس از کشف بن بست باید به گونه‌ای آن را ترمیم کنیم.

سوالات تستی

☑ نیم سال اول ۸۹-۹۰

۱- فرض کنید منابع A, B دو منبع تجدید شدنی و انحصاری در سیستم هستند و فرآیندهای P, Q به شکل زیر تعریف شده اند آنگاه پس از اجرای کدام مجموعه دستورات زیر بن بست اجتناب ناپذیر خواهد بود:

Process P	Process Q
Get A	Get A
Get B	Get A
Release A	Release B
Release B	Release A

(الف) P منبع A را در اختیار گرفته و سپس منبع B را در اختیار بگیرد و.....

(ب) P منبع A را در اختیار گرفته و Q منبع B در اختیار بگیرد و.....

(ج) Q منبع B را در اختیار گرفته و سپس منبع A را در اختیار بگیرد و.....

(د) در هیچ حالتی، امکان بن بست در این سیستم وجود نخواهد داشت.

۲- فرض کنید برای پیشگیری از بروز شرایط "نگهداشتن و انتظار" در بروز بن بست فرآیندها را ملزم به درخواست یکباره تمام منابع مورد نیاز و مسدود کردن آن فرآیند تا موقعی که تمام منابع در اختیارش گذاشته شود، نموده ایم. این کار باعث بروز چه مشکلاتی می شود؟

(الف) ممکن است فرآیندی برای مدت طولانی در انتظار تخصیص کامل تمام منابع مورد درخواستش باقی بماند.

(ب) ممکن است فرآیند همه منابعی که در آینده نیاز دارد را از قبل نداد.

(ج) ممکن است منابعی که به یک فرآیند تخصیص داده شده است برای مدت قابل ملاحظه ای بی استفاده بماند.

(د) همه موارد فوق میتوانند پیش آیند.

۳- کدام گزینه جزء محدودیتهای اجتناب از بن بست نیست؟

(الف) عدم نیاز به قبضه کردن و عقب برگشتن فرآیند.

(ب) تعیین حداکثر منابع مورد نیاز از ابتدا.

(ج) ثابت بودن تعداد منابع تخصیصی.

(د) فرآیندی که منبعی در اختیار داشته باشد نمیتواند خارج گردد.

۴- وضعیت سیستمی را با ۴ فرآیند و ۳ منبع را در نظر بگیرید. اگر اطلاعات زیر در دسترس باشد مقادیر بردار A (Available) (منابع در دسترس) برابر است با:

$$Claim = \begin{bmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{bmatrix} \quad allocation = \begin{bmatrix} 1 & 0 & 0 \\ 5 & 1 & 1 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

Resource	R ₁	R ₂	R ₃	=
	9	2	6	

الف) A=(1, 0, 1) ب) A=(0, 2, 2) ج) A=(2, 0, 1) د) A=(1, 0, 2)

✓ نیم سال دوم ۸۹-۹۰

۱- اگر وضعیت پردازش ها و منابع یک سیستم به صورت زیر باشد، کدام گزینه درست است؟

	A	B	C	D		A	B	C	D		A	B	C	D
P0	0	0	1	2	P0	0	0	1	2		1	5	2	0
P1	1	7	5	0	P1	1	0	0	0		Available			
P2	2	3	5	6	P2	1	3	5	4					
P3	0	6	5	2	P3	0	6	3	2					
P4	0	6	5	6	P4	0	0	1	4					
Max					Allocation									

الف) سیستم در حالت امن قرار دارد.

ب) سیستم در حالت نا امن قرار دارد.

ج) سیستم در حالت بن بست قرار دارد.

د) نمی توان وضعیت سیستم را با این داده ها تعیین کرد.

سوالات تشریحی

✓ نیم سال دوم ۸۹-۹۰

۱- شرایطی که باعث بوجود آمدن بن بست می گردد شرح دهید. (۱ نمره)

پاسخنامه سوالات تستی

نیم سال اول ۸۹-۹۰	
ب	۱
د	۲
الف	۳
د	۴
نیم سال دوم ۸۹-۹۰	
الف	۱

Pnu-Soal.ir

فصل ششم^۱

مدیریت حافظه

Pnu-Soal.ir

^۱ این فصل، فصل هفتم کتاب می باشد.

به منظور پیاده سازی چندبرنامگی در یک سیستم لازم است حافظه اصلی را بخش بندی کنیم تا بتوانیم در یک لحظه بیش از یک فرآیند را در حالت آماده داشته باشیم. بخش بندی حافظه به دو صورت انجام می گیرد:

✓ بخش بندی ایستا

در این نوع بخش بندی پس از بخش بندی ابتدایی حافظه اصلی این بخش بندی تا پایان یک دوره کاری سیستم ثابت خواهد بود.

✓ بخش بندی پویا:

در حین اجرای فرآیندها بخش بندی حافظه قابل تغییر است.

از دید مقادیری که برای هر بخش بندی در نظر گرفته می شود حافظه به دو شکل کلی تقسیم بندی خواهد شد:

✓ روش صفحه بندی

در این روش ابتدا حافظه به چند بخش مساوی با هم که به هر یک، یک فریم یا قاب می گویند تقسیم می شود. سپس هر فرآیند نیز به چند بخش مساوی و هم اندازه با فریم ها تقسیم می شود که به هر یک از آنها یک صفحه یا page می گویند. اندازه هر صفحه با اندازه هر قاب حافظه برابر خواهد بود. هر صفحه برای اجرا شدن باید در یک قاب خالی از حافظه قرار گیرد یا اینکه جایگزین یک صفحه دیگر شود.

✓ روش قطعه بندی

در این روش ابتدا حافظه به چند بخش که لزوماً مساوی نیستند تقسیم می شود. از سوی دیگر فرآیندها نیز به چند بخش با اندازه های مختلف تقسیم خواهند شد. هر بخش از هر فرآیند برای اجرا شدن باید در یک بخش از حافظه با اندازه بزرگتر یا مساوی خود قرار گیرد. در این روش به هر بخش از فرآیند یک قطعه می گویند.

هر دو صفحه بندی و قطعه بندی، هم می توانند به طور ایستا و هم به صورت پویا پیاده سازی شوند. صفحه بندی و قطعه بندی می توانند به صورت صفحه بندی حافظه مجازی یا قطعه بندی حافظه مجازی نیز استفاده شوند که در این صورت برای اجرای یک صفحه یا یک قطعه خاص از فرآیند لازم نیست تمامی صفحات یا تمامی قطعات آن فرآیند به طور کامل در حافظه قرار گیرند.

انواع پراکندگی

در بخش بندی حافظه و استفاده از این بخش ها ممکن است مشکلی به نام پراکندگی^۱ یا تکه تکه شدن در حافظه رخ دهد. در واقع این مشکل زمانی رخ می دهد که چند حافظه خالی در میان داده ها در حافظه اصلی پراکنده شود یا به اصطلاح فضای خالی داخل حافظه یک فضای چند تکه یا تکه شده باشد.

^۱ Fragmentation

مشکل پراکندگی به دو گونه تقسیم می‌شود:

✓ پراکندگی داخلی

زمانی رخ می‌دهد که فضاهای خالی درون یک بخش بدون تغییر از نظر اندازه باشد.

✓ پراکندگی خارجی

به معنی وجود فضاهای خالی در داخل حافظه که در یک محدوده مشخصی نیستند خواهد بود.

تفاوت اصلی پراکندگی داخلی با پراکندگی خارجی آن است که فضاهای هدر رفته در پراکندگی داخلی به سادگی قابل استفاده نیستند، اما در پراکندگی خارجی می‌توان با مترکم سازی، فضاهای خالی را در کنار هم جمع کرد تا با ایجاد یک فضای خالی بزرگتر بتوانیم از آن برای فرآیندهای دیگر استفاده کنیم. به طور مثال، در صفحه بندی می‌تواند پراکندگی داخلی رخ دهد بدین صورت که اگر اندازه فرآیند، مضرب صحیحی از اندازه یک صفحه نباشد مقداری فضای خالی در آخرین صفحه از فرآیند خواهیم داشت.

مجموعه مقیم

در صفحه بندی یا قطعه بندی حافظه مجازی به مجموعه صفحات یا قطعاتی از یک فرآیند که در یک لحظه در حافظه اصلی باشد مجموعه مقیم گفته می‌شود.

✓ جایگذاری فرآیندها در حافظه اصلی

منظور از جایگذاری فرآیندها یا بخشی از آنها در حافظه، تخصیص یک فضای خالی به فرآیند مورد نظر می‌باشد. اگر بخش‌های حافظه دارای اندازه‌های مساوی باشند برای جایگذاری یک فرآیند در حافظه، انتخاب‌های متعددی نخواهیم داشت زیرا همه فضاهای خالی دارای اندازه‌های یکسان هستند و اینکه یک فرآیند در کدام یک از فضاهای خالی قرار گیرد چندان تفاوتی ایجاد نمی‌کند. اما اگر اندازه فضاهای خالی یکسان نباشد لازم است برای جایگذاری یک فرآیند در حافظه از میان فضاهای خالی یکی را انتخاب کنیم. آنچه در اینجا مهم است آن است که اولاً: هر فرآیند تنها می‌تواند در یک فضای خالی بزرگتر یا مساوی اندازه خود قرار گیرد و ثانياً: می‌خواهیم تخصیص فضا هم جوار باشد، یعنی اینکه برای جایگذاری یک فرآیند نمی‌توانیم آن فرآیند را به چند قسمت تقسیم نموده و هر قسمت را داخل یک بخش قرار دهیم. به منظور انتخاب یک فضا از میان فضاهای خالی و جایگذاری فرآیند در آن از الگوریتم‌های جایگذاری حافظه (الگوریتم‌های تخصیص فضای هم جوار) استفاده می‌شود. این الگوریتم‌ها عبارتند از:

✓ الگوریتم اولین جای مناسب (اولین برازش)^۱

در این روش، لیست فضاهای خالی از ابتدا بررسی می شود و با یافتن اولین فضای خالی مناسب، فرآیند در آن محل قرار می گیرد. اگر فرآیند مورد نظر در حافظه جا نشود (به طور هم جوار) این موضوع در پایان لیست مشخص می شود.

✓ الگوریتم جای مناسب بعدی^۲ (درپی برازش)

در این روش پس از جایگذاری اولین فرآیند (که حافظه از اول بررسی می شود. برای جایگذاری فرآیندهای دیگر فضاهای قبلی از محل تخصیص قبلی بررسی می شود. در این روش اگر فرآیندی در حافظه جا نشود این مطلب را پس از بررسی یک دور کامل لیست (ابتدا از تخصیص قبلی تا انتها و سپس از اول حافظه تا محل تخصیص قبلی) مشخص می شود.

✓ الگوریتم بهترین جای مناسب^۳ (بهترین برازش)

در این روش، فرآیند در کوچکترین فضای خالی ممکن قرار می گیرد. در این روش بهتر است برای سریع تر شدن کار، لیست فضاهای خالی به صورت صعودی مرتب شوند. در این روش نیز اگر فرآیندی در حافظه جا نشود این موضوع با بررسی کامل لیست مرتب شده مشخص می شود.

✓ الگوریتم بدترین جای مناسب^۴ (بدترین برازش)

در این روش فرآیند در صورت امکان در بزرگترین فضای خالی حافظه قرار می گیرد. برای سریع تر شدن کار، بهتر است لیست فضاهای خالی به صورت نزولی مرتب شوند و چنانچه فرآیندی در حافظه جا نشود با بررسی اولین عنصر لیست به آن پی خواهیم برد.

مثال ۶ - ۱ بلوکهای خالی حافظه به ترتیب از چپ به راست به صورت زیر هستند. اگر درخواستهای جدیدی برای چهار بلوک به اندازه های ۲۰k و ۳۰k و ۲۰k و ۳۵k مطرح شوند با استفاده از چهار روش جاگذاری وضعیت بلوکهای آزاد حافظه پس از این تخصیص چه خواهد بود؟

40 k 60 k 50 k 45 k 25 k 40 k = بلوکهای آزاد

روش First Fit:

^۱ First Fit

^۲ Next Fit

^۳ Best Fit

^۴ Worst Fit

= اولین درخواست	20	20	25	45	50	60	40
= دومین درخواست	30	20	25	15	50	60	40
= سومین درخواست	20	0	25	15	50	60	40
= چهارمین درخواست	35	0	25	15	15	60	40

روش Next Fit :

= اولین درخواست	20	20	25	45	50	60	40
= دومین درخواست	30	20	25	15	50	60	40
= سومین درخواست	20	20	25	15	30	60	40
= چهارمین درخواست	35	20	25	15	30	25	40

روش Best Fit :

= اولین درخواست	20	40	5	45	50	60	40
= دومین درخواست	30	10	5	45	50	60	40
= سومین درخواست	20	10	5	45	50	60	20
= چهارمین درخواست	35	10	5	10	50	60	20

روش Worst Fit :

= اولین درخواست	20	40	25	45	50	40	40
= دومین درخواست	30	40	25	45	20	40	40
= سومین درخواست	20	40	25	25	20	40	40
= چهارمین درخواست	35	5	25	25	20	40	40

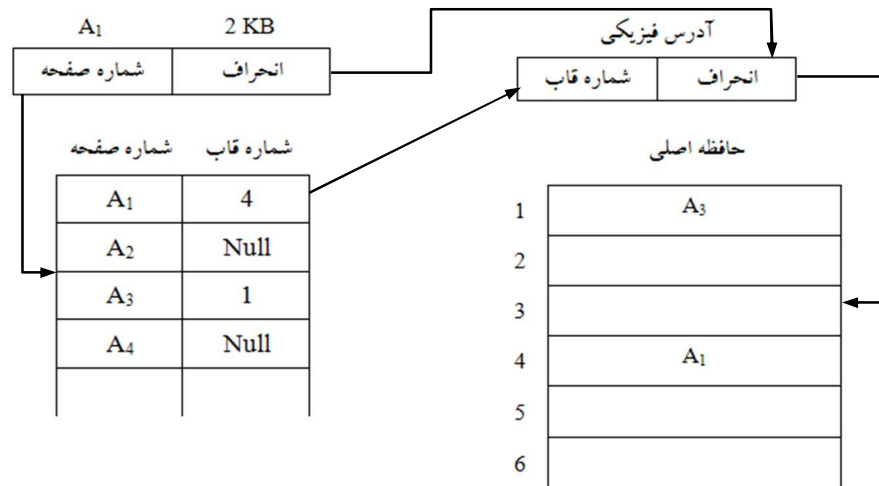
در سیستمی که از صفحه بندی استفاده می شود به منظور تسهیل در جست و جوی صفحات یک فرآیند، برای هر فرآیند یک جدول صفحه در نظر گرفته می شود. در جدول صفحه هر فرآیند، اطلاعاتی در مورد تمامی صفحات آن فرآیند چه در حافظه اصلی باشد چه نباشد نگهداری می شود.

جدول صفحه هر فرآیند شامل دو ستون شماره صفحه و شماره فریم یا قاب می باشد. چنانچه صفحه ای از فرآیند در حافظه اصلی قرار گیرد شماره قاب تخصیص داده شده به آن در جدول صفحه نوشته می شود.

در صفحه بندی از این جدول صفحه برای تبدیل آدرس منطقی به فیزیکی استفاده می شود. آدرس منطقی یک خط از برنامه یک صفحه از فرآیند دارای دو مولفه شماره صفحه و انحراف^۱ می باشد. شماره صفحه در واقع شماره صفحه ای از فرآیند است که خط برنامه مورد نظر در آن قرار دارد. انحراف به فاصله خط برنامه مورد نظر از ابتدای این صفحه اشاره می کند.

^۱ Offset

برای مراجعه به حافظه اصلی باید آدرس منطقی به آدرس فیزیکی تبدیل شود. مولفه‌های آدرس فیزیکی شماره فریم و انحراف هستند که منظور از شماره فریم شماره قابی از حافظه اصلی است که صفحه موردنظر در آنجا قرار دارد. از آنجایی که اندازه هر صفحه از فرآیند با اندازه هر قاب از حافظه اصلی برابر است بخش انحراف آدرس فیزیکی دقیقاً با بخش انحراف آدرس منطقی برابر خواهد بود. برای به دست آوردن شماره فریم آدرس فیزیکی کافیست شماره صفحه آدرس منطقی را به عنوان یک کلید جست و جو در جدول صفحه فرآیند در نظر بگیریم تا شماره فریم مربوط به آن مشخص شود. مدل این تبدیل آدرس به صورت زیر خواهد بود:



برای پیاده سازی تبدیل آدرس منطقی به آدرس فیزیکی با فرض اینکه در آدرس منطقی شماره صفحه با n بیت و انحراف با m بیت مشخص شوند مراحل زیر انجام می گیرد:

- ۱- استخراج n بیت سمت چپ (پرازش تر) آدرس منطقی به عنوان شماره صفحه
- ۲- اعمال شماره این صفحه به جدول صفحه برای به دست آوردن شماره فریم اگر شماره فریم را k در نظر بگیریم آنگاه:
- ۳- محاسبه آدرس فیزیکی شروع فریم مورد نظر از طریق رابطه $k \times 2^m$ (معمولاً شماره فریم‌های حافظه اصلی از صفر شروع می‌شود، بنابراین قبل از فریم شماره k به تعداد k فریم خواهیم داشت، همچنین اگر تعداد بیت‌های انحراف m باشد اندازه هر فریم یا هر صفحه برابر 2^m خواهد بود.
- ۴- پس از به دست آوردن آدرس فیزیکی شروع فریم این مقدار با بخش انحراف آدرس منطقی جمع می‌شود.

$$k \times 2^m + \text{تعداد فریم‌های قبلی} \times \text{اندازه یک فریم} = \text{آدرس ابتدای فریم مورد نظر}$$

در سیستم قطعه بندی نیز هر فرآیند، دارای یک جدول قطعه است. هر جدول قطعه دارای سه ستون شماره قطعه، طول قطعه و آدرس شروع قطعه (پایه) می‌باشد. آدرس منطقی در قطعه بندی شامل شماره قطعه و انحراف

خواهد بود. به منظور تبدیل آدرس منطقی به آدرس فیزیکی ابتدا از طریق شماره قطعه موجود در آدرس منطقی و اعمال آن به عنوان کلید جست و جو به جدول قطعه طول قطعه مورد نظر به دست می آید. سپس بخش انحراف آدرس منطقی با طول قطعه مورد نظر مقایسه می شود. چنانچه انحراف از طول قطعه بیشتر باشد آدرس منطقی معتبر نخواهد بود. اما در صورتی که انحراف کوچکتر یا مساوی طول قطعه مورد نظر باشد برای محاسبه آدرس فیزیکی پس از استخراج آدرس شروع قطعه (پایه) از جدول قطعه آن را با انحراف جمع می کنیم.

برای پیاده سازی چنانچه بخش شماره قطعه آدرس منطقی n بیت و انحراف آن m بیت باشد مراحل زیر صورت می گیرد:

- ۱- استخراج n بیت سمت چپ آدرس منطقی به عنوان شماره قطعه.
- ۲- اعمال شماره قطعه به عنوان کلید جست و جو به جدول قطعه برای به دست آوردن طول و آدرس شروع آن قطعه.
- ۳- مقایسه m بیت سمت راست آدرس منطقی (انحراف) با بخش طول قطعه مورد نظر چنانچه انحراف از طول قطعه بیشتر باشد آدرس معتبر نیست، در غیر این صورت :
- ۴- برای محاسبه آدرس فیزیکی، بخش پایه با انحراف جمع می شود.

سوالات تستی

✓ نیم سال اول ۸۹-۹۰

۱- کدام یک جزء نیازهایی که مدیریت حافظه باید پاسخگوی آنها باشد نیست؟

الف) جابه جایی ب) پیوند زدن ج) حفاظت د) اشتراک

۲- کدام گزینه صحیح می باشد؟

الف) صفحه بندی از دید برنامه ساز مخفی است ولی قطعه بندی مغمولا قابل رویت میباشد.

ب) نیازهای حفاظتی حافظه، باید توسط سیستم عامل برآورده شود نه پردازنده.

ج) روش بخش بندی حافظه با اندازه های ثابت، باعث تکه تکه شدن خارجی حافظه میشود.

د) در قطعه بندی، قطعه ها هم اندازه هستند، اما در صفحه بندی اینطور نیست.

۳- سیستمی ۵۱۲ کیلوبایت حافظه اصلی خالی دارد و از سیستم رفاقتی (Buddy) جهت تخصیص استفاده می کند. فرآیندهای زیر به ترتیب از چپ به راست و با اندازه های مشخص شده وارد سیستم می شوند. اندازه بلوکهای باقیمانده حافظه کدام است؟

P1=12 K

P2=50 K

P3=75 K

P4=110 K

ب) 16 K, 32 K, 64 K

الف) 8 K, 16 K, 256 K

د) 16 K, 16 K, 128 K

ج) 16 K, 32 K, 128 K

✓ نیم سال دوم ۸۹-۹۰

۱- در روش مدیریت حافظه اصلی با سیستم رفاقتی، کدامیک از موارد زیر درست نیست؟

الف) حافظه اصلی به واحد های کوچکتر با اندازه های ۴، ۲، ۱ و..... تقسیم میشود.

ب) پارگی داخلی (internal fragmentation) از مسائل عمده این الگوریتم می باشد.

ج) اختصاص حافظه به فرآیندها، با تقسیم بزرگترین بلوک موجود صورت می پذیرد.

د) بهترین حالت اختصاص حافظه زمانی است که اندازه فرآیندها توانی از دو باشد.

۲- در صفحه بندی حافظه، اگر فقط احتیاج به ناحیه بسیار کوچکی از حافظه باشد، چه مشکلی بروز میکند؟

ب) تکه تکه شدن خارجی

الف) تکه تکه شدن خارجی

د) روی هم گذاری

ج) مشکلی بوجود نمی آید

۳- تاثیر بزرگ شدن طول صفحه بر روی اندازه جدول و تکه تکه شدن داخلی چیست؟

الف) جدول صفحه بزرگ و تکه تکه شدن داخلی زیاد می شود.

ب) جدول صفحه کوچک و تکه تکه شدن داخلی کم می شود.

ج) جدول صفحه کوچک و تکه تکه شدن داخلی زیاد می شود.

د) جدول صفحه بزرگ و تکه تکه شدن داخلی کم می شود.

۴- در مدیریت حافظه به صورت قطعه بندی و صفحه بندی کدام گزینه صحیح است؟

الف) اندازه صفحه و قطعه توسط سیستم عامل معین می شود.

ب) اندازه صفحه توسط سخت افزار و یا سیستم عامل و اندازه قطعه توسط برنامه نویس تعیین می شود.

ج) اندازه صفحه و قطعه توسط برنامه تعیین می شود.

د) اندازه قطعه توسط سیستم عامل و اندازه صفحه توسط برنامه نویس تعیین می شود.

پاسخنامه سوالات تستی

نیم سال اول ۸۹-۹۰	
ب	۱
الف	۲
ج	۳
نیم سال دوم ۸۹-۹۰	
ج	۱
ب	۲
ج	۳
ب	۴

فصل هفتم

Pnu-Soal.ir

✓ کویدگی^۱

در سیستم صفحه بندی یا قطعه بندی حافظه مجازی هر زمان که صفحه یا قطعه ای از برنامه نیاز باشد با مراجعه به حافظه ثانویه آن را به حافظه اصلی می آوریم که به این عمل در اصطلاح swap in می گویند. به همین شکل زمانی که صفحه یا قطعه ای از حافظه اصلی خارج می شود به آن swap out می گویند. البته در سیستم صفحه بندی یا قطعه بندی حافظه مجازی، مادامی که حافظه خالی کافی برای صفحات یا قطعات وجود دارد صفحات یا قطعات موجود در حافظه از آن خارج نمی شوند.

اگر سیستم به حالتی برسد که مدام مشغول خارج ساختن یک صفحه یا قطعه و جایگزین کردن آن باشد یا به عبارتی نرخ swap in و swap out بسیار زیاد باشد اصطلاحاً کویدگی رخ می دهد که در این حالت کارایی سیستم به شدت کاهش می یابد.

✓ نقص صفحه (فقدان صفحه)^۲

چنانچه صفحه ای از یک فرآیند، مورد آدرس دهی (رجوع) قرار گیرد اما در حافظه اصلی نباشد اصطلاحاً نقص صفحه اتفاق می افتد. در هنگام وقوع نقص صفحه، لازم است صفحه مورد نظر از حافظه ثانویه به حافظه اصلی کپی شود. در این حالت چنانچه فریم یا قاب خالی در حافظه داشته باشیم صفحه مورد نظر در آن محل قرار می گیرد. اما چنانچه تمامی قاب های حافظه اصلی پر باشند لازم است یکی از قاب ها خالی و صفحه مورد نظر جایگزین آن شود.

برای انتخاب یک صفحه از میان صفحات موجود در حافظه جهت جایگزینی آن با یک صفحه جدید از الگوریتم های جایگزینی صفحه استفاده می شود.

✓ الگوریتم های جایگزینی صفحه

این الگوریتم ها بر اساس تعداد قاب های حافظه و دنباله رجوع به صفحات یک فرآیند عمل می کنند. دنباله مراجعات یا دنباله رجوع به صفحات، دنباله ای است که در آن ترتیب مراجعه به صفحات مختلف یک فرآیند مشخص می شود. الگوریتم های جایگزینی صفحه عبارتند از :

روش فروج به ترتیب ورود^۳

در این روش هر گاه حافظه پر شد برای جایگزین کردن یک صفحه از میان صفحات درون حافظه صفحه ای را که زودتر در حافظه قرار گرفته از حافظه خارج می کنیم. در این روش معمولاً تعداد نقص صفحه زیاد است هرچند پیاده سازی این روش بسیار آسان است.

^۱ Trashing

^۲ Page Fault

^۳ FIFO

روش بهینه^۱

در این روش هرگاه حافظه پر باشد برای جایگزینی یک صفحه به دنباله مراجعات (از آن لحظه به بعد. توجه می-شود. صفحه ای را که در آینده دیرتر استفاده خواهد شد انتخاب و از حافظه اصلی خارج می کنیم.

این روش دارای حداقل تعداد نقص صفحه است اما به دلیل نیاز به دانستن اطلاعاتی در مورد آینده مراجعه به صفحات پیاده سازی آن تقریباً غیرممکن خواهد بود.

روش LRU^۲ (کمترین استفاده در گذشته)

در این روش هنگامی که حافظه اصلی پر باشد و بخواهیم صفحه جدیدی را جایگزین کنیم با توجه به گذشته دنباله رجوع، (توجه از نقطه حال) صفحه ای را که با توجه به حال دیرتر استفاده شده از حافظه خارج می کنیم. مزیت این روش نسبت به روش دوم آنست که با توجه به گذشته سیستم تصمیم می گیرد.

مثال ۷ - ۱ با توجه به دنباله مراجعات زیر به صفحات یک فرآیند و نیز با در نظر گرفتن سه قاب برای حافظه اصلی تعداد نقص صفحه در هریک از سه الگوریتم فوق را به دست آورید.

دنبال مراجعات : 2 3 2 1 5 2 4 5 3 2 5 2

با استفاده از روش FIFO:

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

PF=9

با استفاده از روش Optimal:

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

PF=6

با استفاده از روش LRU:

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

PF=7

^۱ Optimal

^۲ Last Recently Used

✓ الگوریتم ساعت^۱

در این الگوریتم برای هر صفحه از یک بیت استفاده شده بنام u استفاده می شود. هر صفحه به محض اینکه در حافظه قرار می گیرد دارای بیت $u=0$ خواهد بود. اگر چنانچه صفحه ای در حافظه باشد و دوباره در دنباله رجوع به آن مراجعه شود (دوباره مورد استفاده قرار گیرد. بیت u برای آن یک می شود. بهترین حالت برای پیاده سازی این الگوریتم استفاده از یک لیست حلقوی خواهد بود. این لیست دارای اشاره گری است که در جهت حرکت عقربه های ساعت حرکت می کند. هرگاه صفحه جدیدی بخواهد به حافظه وارد شود اگر اشاره گر به یک قاب خالی اشاره کند صفحه در آن قرار گرفته و اشاره گر یک واحد جلو می رود اما اگر تمامی قاب ها پر باشند با توجه به بیت u در مورد قابی که اشاره گر به آن اشاره می کند عمل می کنیم. اگر بیت u برابر صفر باشد صفحه درون این قاب از حافظه خارج شده و صفحه جدید جایگزین می شود، اشاره گر نیز یک واحد جلو می رود. اما اگر بیت u ، یک باشد به این صفحه اجازه ماندن در حافظه داده می شود هر چند بیت u ی آن را از یک به صفر تغییر می دهیم. اشاره گر حرکت می کند تا صفحه ای با بیت $u = 0$ پیدا کند که در صورت وجود مثل مورد قبل با آن رفتار می شود.

اگر همه بیت های u ، یک باشند پس از آنکه اشاره گر، لیست را یک دور کامل پیمایش کند همه بیت ها صفر می شوند.

چنانچه صفحه ای در حافظه باشد و دوباره به آن رجوع شود اشاره گر لیست هیچ حرکتی نخواهد داشت. در مورد بیت u ی آن صفحه نیز اگر بیت u صفر باشد به یک تغییر می یابد اما اگر بیت u یک باشد هیچ تغییری نخواهد داشت.

مثال ۷ - ۲ باتوجه به تعداد قاب های حافظه و دنباله مراجعه به صفحات مثال قبل و با استفاده از الگوریتم کلاک تعداد نقص صفحه را مشخص کنید.

2 5 2 3 4 2 5 1 2 3 2

تعداد قاب ها : ۳

\square	$\square \xrightarrow{2}$	$\square \xleftarrow{2}$	$\square \xleftarrow{2^1}$	$\square \xleftarrow{2^1}$	$\square \xleftarrow{2}$	$\square \xleftarrow{2^1}$	$\square \xleftarrow{2^1}$	$\square \xleftarrow{2^1}$	$\square \xleftarrow{2}$	$\square \xleftarrow{2^1}$	$\square \xleftarrow{2^1}$
$\square \uparrow$	$\square \rightarrow$	$\square \leftarrow 3$	$\square \leftarrow 3$	$\square \uparrow 3$	$\square \leftarrow 5$	$\square \leftarrow 5$	$\square \uparrow 5$	$\square \uparrow 5$	$\square \uparrow 5$	$\square \uparrow 5$	$\square \uparrow 5$
	*	*	*	*	*	*	*	*	*	*	*

2^1	2^1
$3 \uparrow 5$	$3 \uparrow 5$

¹ Clock

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

[illegible]

$$\begin{array}{cccccccccccccccc}
1^1 & 1 & 1 & 6 & 6 & 6 & 6 & 6 & 6^1 & 6^1 \\
6 \uparrow 2^1 & 6 \leftarrow 2 & 7 \uparrow 2 & 7 \rightarrow 2 & 7 \rightarrow 2 & 7 \rightarrow 2^1 & 1 \uparrow 2 & 1 \uparrow 2^1 & 1 \uparrow 2^1 & 1 \uparrow 2^1 & 1^1 \uparrow 2^1 \\
5 & 3 & 3 & 3 & 3^1 & 3^1 & 3 & 3 & 3^1 & 3^1 & 3^1
\end{array}$$

$$\begin{array}{cc|c} & 4 & \\ 1 & & 2 \\ & 3 & \end{array} \quad \text{PF} = 10$$

سوالات تستی

✓ نیم سال اول ۸۹-۹۰

۱- با افزایش سطح چند برنامه‌گی، کدامیک از حالات زیر اتفاق می افتد؟

الف) درصد استفاده از پردازنده ابتدا افزایش میابد ولی سپس شروع به کاهش میکند.

ب) درصد استفاده از پردازنده ابتدا کاهش میابد ولی سپس شروع به افزایش میکند.

ج) درصد استفاده از پردازنده، رو به افزایش خواهد بود.

د) درصد استفاده از پردازنده، رو به کاهش خواهد بود.

۲- فرض کنید که به یک برنامه ۳ قالب از حافظه اصلی اختصاص داده شده است و هر سه قالب در ابتدا خالی هستند. اگر برنامه به ترتیب از چپ به راست به صفحات زیر با رویکرد FIFO رجوع کند چند نقص صفحه رخ خواهد داد؟

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

ب) ۴ نقص صفحه

الف) ۳ نقص صفحه

د) ۵ نقص صفحه

ج) ۶ نقص صفحه

✓ نیم سال دوم ۸۹-۹۰

۱- فرآیندی به ترتیب زیر از چپ به راست به صفحات حافظه مجازی اش مراجعه میکند:

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

اگر این فرآیند سه قالب صفحه در اختیار داشته باشد و هیچ یک از صفحات آن در شروع کار در حافظه اصلی موجود نباشد و برای جایگزینی از سیاست بهینه (optimal) استفاده شود، تعداد خطای صفحات برابر است با:

د) ۵

ج) ۶

ب) ۷

الف) ۸

۲- اگر حافظه اصلی یک کامپیوتر که تحت مدیریت حافظه مجازی کار میکند، دارای سه صفحه باشد و به صفحات زیر از چپ به راست مراجعه شود. چند خطای صفحه در روش جایگزینی LRU خواهیم داشت؟

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

د) ۹

ج) ۵

ب) ۶

الف) ۷

سوالات تشریحی**✓ نیم سال اول ۸۹-۹۰**

- ۱- میانگیر دم دستی ترجمه چیست؟ (۵/۰ نمره)
- ۲- ترجمه آدرس را در یک سیستم صفحه بندی _قطعه بندی با رسم شکل نشان دهید (۱ نمره)

✓ نیم سال دوم ۸۹-۹۰

- ۱- مکانیزم ترجمه آدرس در یک سیستم صفحه بندی دو سطحی را با رسم شکل شرح دهید (۲۵/۱ نمره)

پاسخنامه سوالات تستی

[illegible]

فصل هشتم^۱

زمانبندی دیسک

Pnu-Soal.ir

^۱ این فصل، فصل یازدهم کتاب می باشد.

برای خواندن داده‌ها از روی دیسک سه زمان برای دستیابی به آن داده طول خواهد کشید: زمان پیگرد، تاخیر چرخشی و زمان انتقال این سه زمان هستند که از میان اینها زمان پیگرد به نسبت بقیه طولانی تر خواهد بود. اگر برای یک دیسک مغناطیسی درخواست‌های متعددی برای خواندن یا نوشتن در استوانه‌های مختلف مطرح شود ترتیب پاسخ دهی به این درخواست‌ها می‌تواند در متوسط زمان پاسخ دهی تاثیرگذار باشد. اگر فرض کنیم درخواست‌های مطرح شده در یک صف نگه داری می‌شوند می‌توانیم برای پاسخ دهی به این درخواست‌ها از الگوریتم‌های مختلف زمانبندی دیسک استفاده کنیم. معروف ترین این الگوریتم‌ها عبارتند از:

✓ الگوریتم خروج به ترتیب ورود

در این روش شیارهای درخواست شده به همان ترتیب درخواستشان پاسخ دهی می‌شوند.

✓ الگوریتم خروج به ترتیب عکس ورود^۱

در این روش درخواست‌ها به ترتیب عکس پاسخ دهی می‌شوند.

✓ الگوریتم کوتاهترین زمان خدمت اول^۲

در این روش ابتدا شیارهای که کوتاهترین فاصله تا موقعیت فعلی هد را دارد پاسخ دهی می‌شود. این روش دارای حداقل متوسط زمان پیگرد خواهد بود. در این روش اگر دو درخواست دارای کمترین فاصله یکسان داشته باشیم به دلخواه یکی را انتخاب می‌کنیم.

✓ روش مرور^۳

در این روش، هد از موقعیت فعلی و در یک جهت حرکت می‌کند و درخواست‌های مطرح شده در مسیرش را جواب می‌دهد. با رسیدن به آخرین استوانه در این جهت به سمت مخالف حرکت کرده و مجدداً درخواست‌های در مسیرش را پاسخ می‌دهد.

در حالت اصلاح شده این روش، هد تا آخرین استوانه حرکت نمی‌کند بلکه زمانی که به بالاترین درخواست مطرح شده رسید جهت حرکت را معکوس می‌کند. در جهت معکوس نیز تا پایین ترین درخواست مطرح شده حرکت و مجدداً جهت را عوض می‌کند.

✓ روش مرور مدور^۴

در این روش مشابه روش SCAN عمل می‌شود با این تفاوت که درخواست‌ها فقط در یک جهت پاسخ داده می‌شوند، یعنی زمانی که هد در حرکت به سمت بالا به آخرین (بالاترین) درخواست مطرح شده رسید از آنجا

^۱ LIFO

^۲ SSTF

^۳ SCAN

^۴ CSCAN

به پایین ترین درخواست مطرح شده پرش می کند و مجددا در حرکت به سمت بالا درخواست ها را پاسخ دهی می کند.

معمولا در ارائه یک مثال، موقعیت اولیه هد مشخص می شود. برای به دست آوردن متوسط طول پیگرد در هر روش پس از مشخص کردن ترتیب پاسخ دهی به درخواست ها و تعیین حاصل جمع تعداد پیگردها این مقدار را بر تعداد درخواست های مطرح شده تقسیم می کنیم تا متوسط طول پیگرد به دست آید.

مثال ۸ - ۱ موقعیت هد در یک دیسک بر روی شیار صد و جهت حرکت آن به سمت بالا خواهد بود اگر درخواست های مطرح شده به ترتیب از چپ به راست به صورت زیر باشند متوسط طول پیگرد در روش های FIFO، SSTF، SCAN و CSCAN را به دست آورید.

درخواست ها: 55 58 39 18 90 160 150 38 184

FIFO : 100 45 55 3 58 16 39 21 18 72 90 70 160 10 150 112 38 146 184

$$\text{متوسط طول پیگرد} = \frac{45 + 3 + 19 + 21 + 72 + 70 + 10 + 112 + 146}{9}$$

SSTF : 100 10 90 32 58 3 55 16 39 1 38 20 18 132 150 10 160 24 184

$$\text{متوسط طول پیگرد} = \frac{10 + 32 + 3 + 16 + 1 + 20 + 132 + 10 + 24}{9}$$

SCAN: 100 50 150 10 160 24 184 94 90 32 58 3 55 16 39 1 38 20 18

$$\text{متوسط طول پیگرد} = \frac{50 + 10 + 24 + 94 + 32 + 3 + 16 + 1 + 20}{9}$$

C-SCAN: 100 50 150 10 160 24 184 166 18 20 38 1 39 16 55 3 58 32 90

$$\text{متوسط طول پیگرد} = \frac{50 + 10 + 24 + 166 + 20 + 1 + 16 + 3 + 32}{9}$$

سوالات تستی

☑ نیم سال اول ۸۹-۹۰

۱- کدام گزینه مربوط به سطوح RAID است که از دسترسی موازی شود می برد؟

الف) سطوح ۱ و ۲ ب) سطوح ۳ و ۴ ج) سطوح ۲ و ۳ د) سطوح ۴ و ۵ و ۶

۲- در یک دستگاه دیسک خوان و نوشتن روی سیلندر ۱۰۰ قرار دارد و تقاضاهایی برای دستیابی به سیلندرها دیگرم به ترتیب زیر از چپ به راست واصل شده است:

55, 58, 39, 18, 90, 160, 150, 38, 184

اگر از الگوریتم SSTF برای دستیابی به سیلندرها استفاده شود، میانگین طول پیگرد چقدر خواهد بود؟

الف) ۲۷/۵ ب) ۲۷/۸ ج) ۲۷ د) ۵۳/۵

۳- کدام یک از سیاست های زمانبندی دیسک، عادلانه ترین روش محسوب میشود؟

الف) SSTF ب) C-SCAN ج) FIFO د) SCAN

☑ نیم سال دوم ۸۹-۹۰

۱- اگر شماره شیارهای درخواستی به صورت:

183, 37, 122, 14, 124, 65, 67, 98

باشد و هد در ابتدا در شیار ۵۳ باشد، مجموع تعداد شیارهای طی شده توسط هد در الگوریتم SSTF چه اندازه خواهد بود؟

الف) ۲۳۳ ب) ۲۷۶ ج) ۲۳۰ د) ۲۳۹

۲- کدامیک از الگوریتم های زمانبندی دیسک مشکل گرسنگی دارد؟

الف) SSTF ب) SCAN ج) C.SCAN د) FIFO

از آنجایی که فصل ۴ کتاب، در جزوه وجود ندارد این قسمت مختص تستهای امتحانی این فصل می باشد.

☒ نیم سال اول ۸۹-۹۰

۱- کدام گزینه صحیح نیست؟

الف) نخهای داخل فرآیند با اینکه در حافظه و پرونده ها مشترک هستند، میتوانند بدون دخالت هسته با یکدیگر ارتباط برقرار کنند.

(ب) مسدود شدن یک نخ، از اجرای دیگر نخ های آماده آن فرآیند جلوگیری نمی کند.

ج) نخب‌های یک فرآیند در یک فضای آدرس هستند و بنابراین به یک فضای آدرس مشترک دسترسی دارند.

(د). اگر فرآیندی به بیرون مبادله گردد، الزاما تمام نخبایش به بیرون مبادله نخواهد شد.

۲- کدام گزینه جزء امتیازات نخهای سطح کاربر (ULTs) نسبت به نخهای سطح هسته (KLTs) نمی باشد؟

(الف) نخهای سطح کاربر میتوانند روی هر سیستم عاملی اجرا شوند.

(ب) در راهبرد نخ سطح کاربر محض، کاربرد چند نخ می‌تواند از امتیازات چندپردازی استفاده کند.

(ج) هر کاربرد از نخ میتواند زمانبندی خاص خود را داشته باشد.

(د) تعویض نخ، به حالت امتیاز هسته نیاز ندارد.

۳- معماری چند پردازشی (SMP) جزء کدام مجموعه معماری های زیر است؟

الف) SIMD ب) MIMD یا حافظه توزیعی

(ج) MIMD با حافظه مشترک

☒ نیم سال دوم ۸۹-۹۰

۱- کدام موارد جزء موارد بکارگیری نخواهند بود؟

۱. کار ییش زمینہ و یس زمینہ

۲. پردازش ناهمگام

۳. سرعت اجرا

۴. افزایش اطمینان در بکارگیری منابع اشتراکی

(الف) ۲ او (ب) ۳ او ۲ او (ج) ۳ او ۲ او (د) ۳ او ۳ او

(ج) ۱ و ۲ و ۳

(ب) ۱ و ۲ و ۳

(د) ۲ و ۳ و ۳

پاسخنامه سوالات تستی

نیم سال اول ۸۹-۹۰	
د	۱
ب	۲
ج	۳
نیم سال دوم ۸۹-۹۰	
د	۴