موسسه بابان

انتشارات بابان و انتشارات راهیان ارشد

درس و کنکور ارشد

سیستم عامل

(حل تشریحی سوالات دولتی ۱۳۹۶)

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی کامپیوتر و IT

براساس کتب مرجع

آبراهام سیلبرشاتز، ویلیام استالینگز و اندرو اس تنن‌بام

ارسطو خلیلی‌فر

کلیه‌ی حقوق مادی و معنوی این اثر در سازمان اسناد و کتابخانه‌ی ملی ایران به ثبت رسیده است.

تست‌های سیستم عامل مهندسی کامپیوتر دولتی ۱۳۹۶

۱- درباره ویژگی محلیت (Locality) برنامه زیر، کدام مورد درست است؟

(مهندسی کامپیوتر- دولتی ۹۶)

```
int sum (int v[n]){
      int i, sum=0
      for(i=0;i<n;i++)
            sum +=v[i]
}
```

۱) متغیر sum دارای ویژگی محلیت زمانی (Temporal Locality) خوب و متغیر v دارای ویژگی محلیت زمانی بد و محلیت مکانی (Spatial Locality) خوب می‌باشد.

۲) متغیر sum دارای ویژگی محلیت زمانی (Temporal Locality) خوب و متغیر v دارای ویژگی محلیت زمانی بد و محلیت مکانی (Spatial Locality) بد می‌باشد.

۳) متغیر sum دارای ویژگی محلیت مکانی (Spatial Locality) خوب و متغیر v دارای ویژگی محلیت زمانی (Temporal Locality) بد و محلیت مکانی خوب می‌باشد.

۴) متغیر sum دارای ویژگی محلیت مکانی (Spatial Locality) خوب و متغیر v دارای ویژگی محلیت زمانی (Temporal Locality) خوب و محلیت مکانی بد می‌باشد.

۲- یک کامپیوتری با 8 گیگابایت حافظه را در نظر بگیرید که اندازه هر صفحه 8 کیلوبایت و هر خانه از جدول 4 بایت باشد. در صورتیکه این کامپیوتر از جدول چند سطحی استفاده نماید که هر جدول صفحه در یک صفحه ذخیره شود و بخواهیم آدرس مجازی 46 بیتی را به آدرس فیزیکی تبدیل نماییم. برای خواندن یک کلمه 32 بیتی به چند دسترسی به حافظه نیاز است؟

(مهندسی کامپیوتر – دولتی ۹۶)

۱) 4 ۲) 3 ۳) 2 ۴) 1

۳- با توجه به جدول، متوسط زمان برگشت (Turnaround Time) و زمان انتظار (Waiting Time) پردازه‌های زیر را به ازای الگوریتم Preemptive Shortest Remaining Job First چه عددی است؟

(مهندسی کامپیوتر – دولتی ۹۶)

پردازه	زمان ورود به سیستم	زمان موردنیاز برای اجرا
P_1	1	10
P_2	3	8
P_3	7	6
P_4	11	3
P_5	15	7

۱) 14 و 6.2 ۲) 16 و 6.2 ۳) 14 و 7.2 ۴) 16 و 7.2

۴- فرض کنید دیسکی دارای 100 سیلندر است (سیلندر 0 تا 99) و رفتن هِد خواندن و نوشتن از یک سیلندر به سیلندر مجاور، یک واحد زمان طول می‌کشد. در زمان صَفر، هِد در سیلندر صفر قرار دارد و هیچ درخواستی موجود نیست. شش درخواست در زمان‌هایی کَه در جدول نشان داده است مطرح می‌شوند. اگر از الگوریتم آسانسور استفاده شود، زمان لازم برای پاسخ دادن به همه درخواست‌ها چقدر است؟ (در زمان‌هایی که هد در حال حرکت است، رسیدن یک درخواست باعث تغییر مقصد هد در آن حرکت نمی‌شود. از زمان گردش دیسک هم صرف‌نظر کنید.)

(مهندسی کامپیوتر – دولتی ۹۶)

زمان	شماره سیلندر
0	21
10	75
20	16
70	68
80	2
90	17

۱) 163 ۲) 172 ۳) 199 ۴) 200

پاسخ تست‌های سیستم عامل مهندسی کامپیوتر دولتی ۱۳۹۶

۱- گزینه (۱) صحیح است.

بـه طـور کلـی محلیـت در دو نـوع، (۱) محلیـت مکـانی (Spatial Locality) و (۲) محلیـت زمانی(Temporal Locality) وجود دارد.

محلی بودن مکانی را مانند شعاع **متغیر** یک دایره در نظر بگیرید، هر چه قدر این شـعاع کـوچکتر شود محلی بودن مکانی در **کناره‌ها و اطراف** بیشتر و بیشتر می‌شـود و هـر چـه قـدر ایـن شـعاع بزرگتر شود محلی بودن مکانی در **کناره‌ها و اطراف** کمتر و کمتر می‌شود. شما بـا هـم محلـه‌های خود محلی‌تر مکانی هستید، اما هر چه قدر از محله و مکان خود دور می‌شـوید، اهـالی محلـه‌های دیگر با شما کمتر محلی‌تر مکانی هستند. به عبـارت دیگـر محلـی بـودن مکـانی خـوب می‌گویـد مراجعه بعدی، در **نزدیکی** همین مراجعه فعلی است. آرایه مطرح شده در صورت سوال یعنی v[i] دارای خاصیت محلی بودن مکانی **خوب** است. عناصر آرایه v[i] در داخل یک حلقه for از اندیس صفر تا n به ترتیب و پشت سرهم خوانده می‌شوند، یعنی مراجعه بعدی، مـدام در نزدیکـی همـین مراجعه فعلی است.

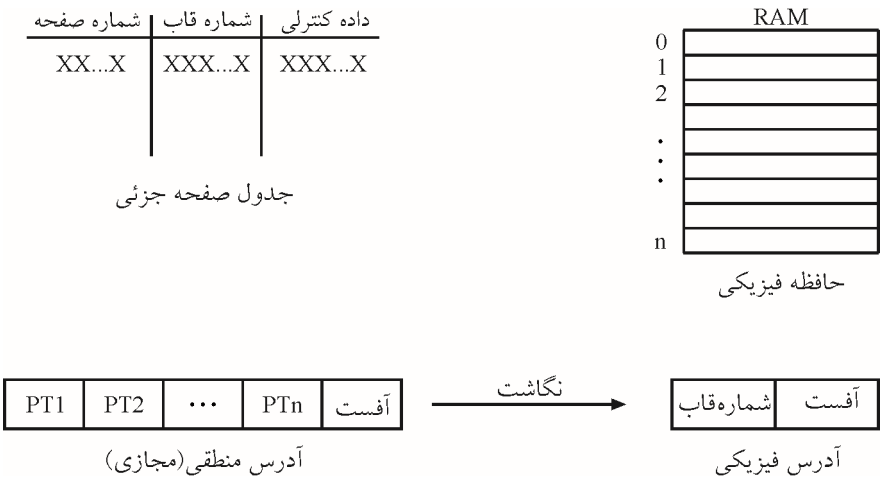
محلی بودن زمانی را مانند شعاع **ثابت** یک دایره در نظر بگیرید، که محدوده ایـن شـعاع هیچگـاه تغییر نکند. شما با خانواده خود محلی زمانی هستید. به عبارت دیگر محلـی بـودن زمـانی **خـوب** می‌گوید مراجعه بعدی، در **محل** همین مراجعه فعلی است. متغیر مطـرح شـده در صـورت سـوال یعنی sum دارای خاصیت محلی بودن زمانی **خوب** است. متغیر sum در داخـل یـک حلقـه for از اندیس صفر تا n خوانده می‌شود، یعنی مراجعه بعدی، مدام در محل همین مراجعه فعلی است.

همانطور که گفتیم آرایه v[i] دارای خاصیت محلی بودن مکانی **خوب** اسـت، امـا از آنجـا کـه بـه هریک از عناصر آرایه v[i] در حرکت حلقه فقط و فقط یکبار مراجعه می‌شود، بنـابراین آرایـه v[i] داری خاصیت محلی بودن زمانی **بد** است. همچنین همانطور که گفتیم متغیر sum دارای خاصـیت محلی بودن زمانی **خوب** است، اما از آنجا که به کناره‌هـا و اطـراف متغیـر sum در حرکـت حلقـه مراجعه نمی‌شود، بنابراین متغیر sum داری خاصیت محلی بودن مکانی **بد** است.

۲- گزینه (۱) صحیح است.

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

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



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

بنابراین داریم:

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

$4B$ = عرض جدول صفحه جزئی

توجه: مطابق فرض سؤال، هر مدخل جدول صفحه (عرض جدول صفحه جزئی) 4 بایت در نظر گرفته شده است.

$$\text{تعداد سطرهای جدول صفحه جزئی} = \frac{\text{اندازه قاب}}{\text{عرض جدول صفحه}} = \frac{2^3 \times 2^{10} B}{2^2 B} = 2^{11} = 2048$$

$2^{46} B$ =اندازه فرآیند (فضای آدرس مجازی)

$8KB = 8192B = 2^3 \times 2^{10} B = 2^{13} B$ =اندازه صفحه = اندازه قاب

$$f : \text{تعداد صفحات فرآیند} = \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه}} = \frac{2^{46}}{2^{13}} = 2^{33}$$

$r : \text{تعداد سطرهای جدول صفحه جزئی} = 2048 = 2^{11}$

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

روش تجزیه

تعداد صفحات فرآیند باید در اندازه r(2^{11}) تجزیه گردد:

$$\text{تعداد صفحات فرآیند} = 2^{33} = 2^{\overset{\uparrow}{(11)}} \times 2^{\overset{\uparrow}{(11)}} \times 2^{\overset{\uparrow}{(11)}}$$

PT1 PT2 PT3

توجه: تعداد عملوندها برابر 3 است، بنابراین تعداد سطوح جدول چند سطحی نیز برابر 3 است.

روش لگاریتم

$$\text{تعداد سطوح جدول چند سطحی} = d = \left\lceil \log_r^f \right\rceil = \left\lceil \log_{2^{11}}^{2^{33}} \right\rceil = 3$$

روش تقسیم متوالی

$$\text{تعداد جداول صفحه جزئی در سطح سوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{2^{33}}{2^{11}} = 2^{22}$$

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد جداول صفحه جزئی در سطح سوم}}{r} = \frac{2^{22}}{2^{11}} = 2^{11}$$

$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{2^{11}}{2^{11}} = 1$$

توجه: سطح اول، یک جدول به حساب می‌آید، که 2^{11} سطر دارد.

توجه: تعداد تقسیم متوالی برابر 3 است، بنابراین تعداد سطوح جدول صفحه چند سطحی برابر 3 است.

$$\text{تعداد بیت آفست} = \log_2^{\text{اندازه صفحه}} = \log_2^{2^{13}} = 13 \text{ بیت}$$

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

PT1	PT2	PT3	آفست
11 بیت	11 بیت	11 بیت	13 بیت
33 بیت			

مطابق آنچه گفتیم سه دسترسی به جداول صفحه جزئی سطح اول، دوم و سوم برای ترجمه آدرس و یک دسترسی به داده اصلی (مقصد مورد نظر) لازم است، که مجموع آن شامل چهار دسترسی به حافظه می‌گردد. بنابراین برای خواندن یک کلمه 32 بیتی نیاز به چهار دسترسی به حافظه است.

۳- گزینه (۳) صحیح است.

الگوریتم SJF (Shortest Job First)

در این روش ابتدا کاری برای اجرا انتخاب می‌شود که از همه کوتاهتر باشد (زمان اجرای کمتری داشته باشد).

توجه: این الگوریتم، **SPN** (Shortest Process Next) و

SPT (Shortest Process Time) نیز نامیده می‌شود.

توجه: **SJF** یک الگوریتم **انحصاری** (Non Preemptive) است.

توجه: یک نقص عمده الگوریتمSJF این است که ممکن است باعث قحطی‌زدگی فرآیندهای طولانی شود. به این ترتیب که اگر همواره تعدادی فرآیند کوچک وارد سیستم شوند، اجرای فرآیندهای بزرگ به طور متناوب به تعویق می‌افتد. این روال حتی می‌تواند تا بینهایت ادامه یابد و هیچگاه نوبت به فرآیندهای بزرگ نرسد!!!!!

توجه: در این روش اگر دو فرآیند مدت زمان اجرای برابر داشته باشند، بر اساس FCFS زمان‌بندی می‌شوند.

توجه: هدف الگوریتم SJF به حداقل رساندن میانگین زمان انتظار، میانگین زمان پاسخ و میانگین زمان گردش کار (بازگشت) فرآیندهاست.

توجه: در عمل نمی‌توان الگوریتم SJF را پیاده‌سازی کرد، زیرا سیستم عامل زمان اجرای فرآیندها را از قبل نمی‌داند و تنها کاری که می‌تواند انجام دهد این است که زمان اجرای فرآیندها را فقط حدس زده و به طور تقریبی بدست آورد.

الگوریتم SRT (Shortest Remaining Time)

این الگوریتم نسخه **غیرانحصاری** (Preemptive) الگوریتم SJF است. در این الگوریتم اگر حین اجرای یک فرآیند، فرآیندی وارد شود که زمان اجرای کوتاه‌تری داشته باشد، پردازنده را در اختیار می‌گیرد.

توجه: این الگوریتم، **SRPT** (Shortest Remaining Processing Time)،
SRTF (Shortest Remaining Time First) و
SRTN (Shortest Remaining Time Next) نیز نامیده می‌شود.

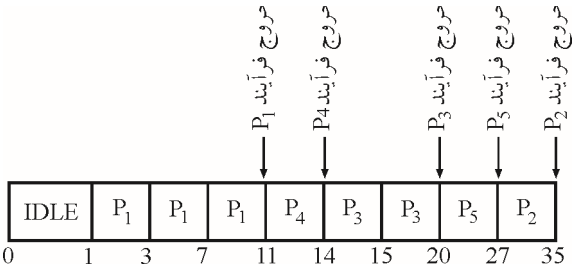
توجه: اگر لحظه ورود همه فرآیندها یکی باشد، الگوریتمSRT مشابه SJF عمل می‌کند.

توجه: در الگوریتم **SRT** نیز همانند الگوریتم SJF، احتمال وقوع **قحطی‌زدگی برای کارهای بزرگ** وجود دارد.

با توجه به مفروضات مطرح شده در صورت سؤال داریم:

فرآیند	زمان ورود	زمان اجرا	زمان انتظار+	زمان بازگشت =
P_1	1	10		
P_2	3	8		
P_3	7	6		
P_4	11	3		
P_5	15	7		

با توجه به مفروضات مساله، نمودار گانت زیر را داریم:



زمان ورود فرآیند – زمان خروج فرآیند = زمان بازگشت فرآیند

P_1 زمان بازگشت $= 11-1=10$

P_2 زمان بازگشت $= 35-3=32$

P_3 زمان بازگشت $= 20-7=13$

P_4 زمان بازگشت $= 14-11=3$

P_5 زمان بازگشت $= 27-15=12$

میانگین زمان بازگشت $= ATT = \frac{10+32+13+3+12}{5} = \frac{70}{5} = 14$

زمان اجرای فرآیند – زمان بازگشت فرآیند = زمان انتظار فرآیند

P_1 زمان انتظار $= 10-10=0$

P_2 زمان انتظار $= 32-8=24$

P_3 زمان انتظار $= 13-6=7$

P_4 زمان انتظار $= 3-3=0$

P_5 زمان انتظار $= 12-7=5$

میانگین زمان انتظار $= AWT = \frac{0+24+7+0+5}{5} = \frac{36}{5} = 7.2$

میانگین زمان اجرا $= AST = \frac{10+8+6+3+7}{5} = \frac{34}{5} = 6.8$

AVG Turnaround Time=AVG Service Time + AVG Waiting Time

میانگین زمان انتظار + میانگین زمان اجرا = میانگین زمان بازگشت

14= 6.8 + 7.2

توجه: مطابق رابطه فوق، تفاضل میانگین زمان بازگشت و میانگین زمان انتظار باید برابر میانگین زمان اجرا باشد، که فقط در گزینه سوم این مورد رعایت شده است. بنابراین گزینه‌های اول، دوم و چهارم پاسخ سؤال نیستند.

توجه: همچنین مطابق رابطه فوق، میانگین زمان بازگشت همواره از میانگین زمان انتظار بیشتر است، بنابراین از این نگاه همه گزینه‌ها درست هستند، که در اینجا این نگاه کارآمد نیست.

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

فرآیند	زمان ورود	زمان اجرا	زمان انتظار+	زمان بازگشت=
P_1	1	10	0	10
P_2	3	8	24	32
P_3	7	6	7	13
P_4	11	3	0	3
P_5	15	7	5	12

میانگین زمان بازگشت = میانگین زمان انتظار + میانگین زمان اجرا
14 7.2 6.8

۴- گزینه ( ) صحیح است.

توجه: مطابق فرض سؤال، در زمان صفر هد بر روی سیلندر صفر است و درخواستی از گذشته وجود ندارد.

شش درخواست در زمان‌های مختلف مطابق جدول زیر وارد می شوند:

زمان ورود درخواست: 0 10 20 70 80 90

سیلندر درخواست شده: 21 75 16 68 2 17

فرض سوال: در زمان حرکت هد به سمت یک سیلندر، ورود درخواست جدید تأثیری بر حرکت ندارد.

توجه: هد دیسک تحت کنترل نرم‌افزار زمان‌بندی دیسک و کنترل‌کننده هد دیسک قرار دارد، و به خودی خود هد دیسک فهم و شعور ندارد، یعنی بدون نرم‌افزار زمان‌بندی دیسک و کنترل‌کننده هد دیسک، هد دیسک نمی‌داند کجا برود و کجا نرود، چه کند و چه نکند. بلکه نرم‌افزار زمان‌بندی دیسک به واسطه استفاده از یک الگوریتم خاص، توسط کنترل‌کننده هد دیسک، هد دیسک را کنترل و جابه‌جا می‌کند، در واقع این نرم‌افزار زمان‌بندی دیسک و کنترل‌کننده هد دیسک است که به هد دیسک فهم و شعور می‌هد و به هد می‌گوید که کجا برود و کجا نرود، چه کند و چه نکند. بنابراین در زمان حرکت مکانیکی هد به سمت یک سیلندر، هد شخصا تصمیم جدیدی نمی‌گیرد، بلکه تابع تصمیمات نرم‌افزار زمان‌بندی و کنترل‌کننده خود است، که فرض سوال این است که ورود درخواست جدید تأثیری بر حرکت هد ندارد که این خواست نرم‌افزار زمان‌بندی دیسک برای هد دیسک است.

توجه: در الگوریتم SCAN، اگر درخواستی به صف برسد که جلوی هد باشد، این درخواست سریعاً سرویس داده می‌شود ولی اگر درخواست پشت سر هد باشد، بایستی صبر شود، تا هد به

انتهای دیسک رفته، تغییر جهت داده و برگردد. لذا در این مسأله در ابتدا هد به سیلندر 21 رفته و سپس با توجه به درخواست‌های وارد شده تا زمان 21 (یعنی درخواست‌های 75 و 16) به سیلندر 75 می‌رود. (جهت حرکت فعلاً صعودی است).

توجه: درصورت سوال مطرح شده است که در زمان حرکت هد به سمت یک سیلندر، ورود درخواست جدید تأثیری بر حرکت ندارد، یعنی ابتدا شخص بدون توجه به حواشی، وارد یک جزیره می‌شود، به جزیره که رسید، دعوت‌های خود را بررسی می‌کند، سپس دعوتی را قبول می کند که در جهت حرکتش باشد، سپس مجدداً بدون توجه به حواشی، وارد جزیره بعدی می‌شود و این روال تا انتها به همین منوال ادامه می‌یابد.

توجه: در صورت سوال بیان شده است که زمان لازم برای ورود هد از یک سیلندر به سیلندر مجاور یک واحد زمانی است. بنابراین هنگامی که در سیلندر 21 قرار داریم، به معنی حضور در زمان 21 است. بنابراین واضح است که تا قبل از زمان 21 درخواست سیلندر 75 در زمان 10 و درخواست سیلندر 16 در زمان 20 صادر شده است. بنابراین ابتدا وارد جزیره (سیلندر) 21 شدیم، سپس وارد جزیره (سیلندر) 75 شدیم، حال که در سیلندر 75 قرار داریم، به معنی حضور در زمان 75 است.

بنابراین واضح است که تا قبل از زمان 75، درخواست سیلندر 68 در زمان 70 صادر شده است. بنابراین در لحظه 75، درخواست‌های 16 و 68 در سیستم وجود دارند، ولی با توجه به اینکه درخواست‌ها در پشت سر هد قرار دارند، در این لحظه به هیچ یک سرویس داده نمی شوند و هد با توجه به الگوریتم SCAN تا انتهای دیسک (سیلندر 99) حرکت کرده و جهت آن برعکس می شود. حال که در سیلندر 99 قرار داریم، به معنی حضور در زمان 99 است. بنابراین واضح است که تا قبل از زمان 99، درخواست سیلندر 2 در زمان 80 و درخواست سیلندر 17 نیز در زمان 90 صادر شده است. بنابراین در لحظه 99، درخواست‌های 16، 68، 2 و 17 در سیستم وجود دارند، حال جهت حرکت هد به شکل نزولی از سیلندر 99 به سیلندر صفر تغییر می‌کند و در این پیمایش و حرکت به ترتیب به درخواست‌های 68، 17، 16 و 2 سرویس داده می‌شود، یعنی داریم:

$$0 \rightarrow 21 \rightarrow 75 \rightarrow 68 \rightarrow 17 \rightarrow 16 \rightarrow 2$$

در نتیجه، تعداد کل حرکات هد به صورت زیر خواهد بود:

$$21+54+7+51+1+14=148$$

مطابق فرض سوال، اگر حرکت از هر سیلندر به سیلندر دیگر (سیلندرهای متوالی) یعنی مجاور یک واحد زمانی طول بکشد، مجموع زمان جستجو در این درخواست‌ها، به صورت زیر خواهد بود:

$$T_{Total}=148\times 1=148$$

که در هیچ کدام از گزینه‌ها این ترتیب و به تبع زمان پاسخ به همه درخواست‌ها وجود نـدارد. امـا اگر فرض کنیم که در الگوریتم SCAN هد تا جایی حرکت می‌کنـد کـه درخواسـتی وجـود داشـته باشد و سپس جهت آن بالعکس شود (یعنی الگوریتم Look) داریم:

$0 \rightarrow 21 \rightarrow 75 \rightarrow 68 \rightarrow 16 \rightarrow 2 \rightarrow 17$

توجه کنید هنگامی که هد به سیلندر 75 می‌رسد جهت آن پس از آن بالعکس و در جهـت نزولـی می‌شود چون درخواست‌های موجـود 16 و 68 اسـت. در واقـع حضـور در سـیلندر 75 بـه معنـی حضور در زمان 75، درخواست سیلندر با شماره بیشتر وجود ندارد، بلکه سـیلندرهای 16 و 68 بـا شماره کمتر وجود دارد. بنابراین در حرکت نزولی از سیلندر 75 ازبین درخواست‌هـای موجـود 16 و 68، ابتدا درخواست 68 پاسخ داده می‌شود. هنگامی که در سیلندر 68 قرار مـی‌گیـریم، در زمـان $(75+(\overset{7}{\cancel{75}}-\cancel{68}))=82$ نیز حضور داریم.

بنابراین هنگامی که در سیلندر 68 قرار داریم، به معنی حضور در زمان 82 اسـت. بنـابراین واضـح است که تا قبل از زمان 82، درخواست سیلندر 2 در زمان 80 صادر شده است. بنـابراین در لحظـه 82، درخواست‌های 16 و 2 در سیستم وجود دارند. بنابراین در ادامه حرکت نزولـی از سـیلندر 68 به سیلندر 16 می‌رویم. هنگامی که در سیلندر 16 قرار می‌گیریم، در زمان $(82+(\overset{52}{\cancel{68}}-\cancel{16}))=134$ نیز حضور داریم. بنابراین هنگامی که در سیلندر 16 قرار داریـم، بـه معنـی حضـور در زمـان 134 است. بنابراین واضح است که تا قبل از زمان 134، درخواست سیلندر 17 نیـز در زمـان 90 صـادر شده است. بنابراین در لحظه 134، درخواست‌های 2 و 17 در سیستم وجود دارند. بنابراین در ادامه حرکت نزولی از سیلندر 16 به سیلندر 2 می‌رویم. هنگامی که در سیلندر 2 قرار می‌گیریم، در زمان $(134+(\overset{14}{\cancel{16}}-\cancel{2}))=148$ نیز حضور داریم. بنابراین هنگامی که در سیلندر 2 قرار داریم، بـه معنـی حضور در زمان 148 است. در این زمان همه درخواست‌هـا صـادر شـده‌انـد و درخواسـت جدیـد دیگری وجود ندارد و تنها درخواست باقی مانده از قبل، درخواست سـیلندر 17 مـی‌باشـد. توجـه کنید هنگامی که هد به سیلندر 2 می‌رسد جهت آن پس از آن بالعکس و در جهت صعودی می‌شود چون تنها درخواست موجود سیلندر 17 است. در واقع هنگام حضور در سیلندر 2 به معنی حضور در زمان 148، درخواست سیلندر با شماره کمتر وجود نـدارد، بلکـه سـیلندر 17 بـا شـماره بیشـتر وجود دارد. بنابراین در حرکـت صـعودی از سـیلندر 2 بـه سـیلندر 17 حرکـت مـی‌کنـیم. تـا بـه درخواست سـیلندر 17 پاسـخ داده شـود. هنگـامی کـه در سـیلندر 17 قـرار مـی‌گیـریم، در زمـان $(148+(\overset{15}{\cancel{17}}-\cancel{2}))=163$ نیز حضور داریم. بنابراین هنگامی که در سیلندر 17 قرار داریم، به معنی حضور در زمان 163 است و کار تمام می‌شود، بنابر روال مطرح شده داریم:

$0 \rightarrow 21 \rightarrow 75 \rightarrow 68 \rightarrow 16 \rightarrow 2 \rightarrow 17$

در نتیجه، تعداد کل حرکات هد به صورت زیر خواهد بود:

$$21+54+7+52+14+15=163$$

مطابق فرض سوال، اگر حرکت از هر سیلندر به سیلندر دیگر(سیلندرهای متوالی) یعنی مجاور یک واحد زمانی طول بکشد، مجموع زمان جستجو در این درخواست‌ها، به صورت زیر خواهد بود:

$$T_{Total}=163\times1=163$$

روال فوق با گزینه اول تطابق دارد، بنابراین گزینه اول می‌تواند درست باشد.

توجه: البته بهتر بود طراح محترم الگوریتم Look را مورد پرسش قرار می‌داد و نه SCAN که بعد الگوریتم Look در گزینه‌ها باشد.

توجه: سازمان سنجش آموزش کشور، در کلید اولیه خود، گزینه اول را به عنوان پاسخ اعلام کرده بود. اما در کلید نهایی این سؤال حذف گردید، که کار درستی بوده است.

تست‌های سیستم عامل مهندسی فناوری اطلاعات دولتی ۱۳۹۶

۱- در یک کامپیوتر، روتین سرویس‌دهی به وقفه ساعت (Interrupt Service Routine) به 2 میلی‌ثانیه زمان برای اجرا نیاز دارد. این زمان شامل زمان اجرا و زمان تعویض زمینه می‌باشد. این روتین سرویس‌دهی به وقفه ساعت در هر پالس ساعت اجرا شده و پالس ساعت دارای فرکانس 75 هرتز می‌باشد. چه درصدی از زمان پردازنده برای این روتین سرویس‌دهی به وقفه اختصاص می‌یابد؟ (مهندسی IT- دولتی ۹۶)

۱) 5 ۲) 7.5 ۳) 10 ۴) 15

۲- در یک سیستم صفحه‌بندی که دارای 34 بیت آدرس است 23 بیت اول برای شماره صفحه و 11 بیت بعدی برای آدرس‌دهی درون صفحه است. در یک سیستم با صفحه‌بندی وارونه (Inverted Page Table) با 128 مگابایت حافظه، جدول صفحه دارای چند خانه است؟

(مهندسی IT – دولتی ۹۶)

۱) 2^{16} ۲) 2^{17} ۳) 2^{23} ۴) 2^{34}

۳- کدام مورد، درباره ایمن بودن یک تابع در سطح نخ (Thread Safe) درست است؟

(مهندسی IT- دولتی ۹۶)

۱) یک تابع در سطح نخ ایمن است و اگر و فقط اگر از نخ‌های مختلف فراخوانی شود همیشه نتیجه درست را برگرداند.

۲) یک تابع در سطح نخ ایمن است اگر و فقط اگر از نخ‌های همروند فراخوانی شود همیشه نتیجه درست را برگرداند.

۳) یک تابع در سطح نخ ایمن است اگر از نخ‌های همروند فراخوانی شود همیشه نتیجه درست را برگرداند.

۴) یک تابع در سطح نخ ایمن است اگر از نخ‌های مختلف فراخوانی شود همیشه نتیجه درست را برگرداند.

۴- برای یک نخ (Thread) کدام‌یک از تغییر وضعیت‌ها، امکان‌پذیر است؟ (مهندسی IT – دولتی ۹۶)

۱) رفتن از حالت آماده به خاتمه ۲) رفتن از حالت آماده به انتظار

۳) رفتن از حالت آماده اجرا به انتظار ۴) رفتن از حالت آماده انتظار به اجرا

۵- سه سیستم مدیریت حافظه S_1، S_2 و S_3 را در نظر بگیرید. S_1 دارای یک TLB با زمان پاسخ 120ns است که نرخ برخورد آن 60% می‌باشد. S_2 دارای یک TLB بزرگ‌تر ولی کندتر می‌باشد که زمان پاسخ آن 150ns و نرخ برخورد آن 80% می‌باشد. S_3 دارای TLB با زمان پاسخ 120ns

و نرخ برخورد 70% است. دو سیستم S_1 و S_2 همزمان با ارسال درخواست بـه TLB، آن را بـه حافظهٔ اصلی نیز ارسال می‌کنند تا در صورت عدم یافتن آدرس در TLB، زمـان پاسـخ کـاهش یابد. سیستم S_3 پس از دریافت پاسخ از TLB و عدم یـافتن آدرس، درخواسـت را بـه حافظـهٔ اصلی ارسال می‌کند. کمترین و بیشترین میانگین زمان پاسخ این سه سیستم به ترتیب از راسـت به چپ چند نانوثانیه است؟ **(مهندسی IT – دولتی ۹۶)**

۱) 312-300 ۲) 312-264 ۳) 300-312 ۴) 300-264

۶- در سیستمی با پنج پردازه و سه نوع منبع، وضعیت تخصیص منابع به صورت زیر است. اگـر در این وضعیت، درخواستی برای یک واحد دیگر از منبع A توسط پردازه P_3 صادر شود، کـدام مورد درست است؟ **(مهندسی IT – دولتی ۹۶)**

(تعداد منابع اولیه)

A	B	C
8	6	10

(حداکثر نیاز)

	A	B	C
P_0	3	6	8
P_1	7	3	6
P_2	5	3	3
P_3	4	5	9
P_4	2	3	3

(تخصیص‌یافته)

	A	B	C
P_0	0	1	2
P_1	2	0	3
P_2	3	2	0
P_3	1	0	2
P_4	1	1	0

۱) بعد از انجام درخواست فوق بن‌بست قطعی است.
۲) قبل از انجام درخواست فوق وقوع بن‌بست قطعی است.
۳) قبل از درخواست فوق احتمال وقوع بن‌بست وجود دارد.
۴) بعد از انجام درخواست فوق احتمال وجود بن‌بست وجود دارد.

حل تشریحی تست‌های سیستم عامل مهندسی فناوری اطلاعات دولتی ۱۳۹۶

۱- گزینه (۴) صحیح است.

هر وقفه یک روتین یا روال پاسخ به وقفه دارد که به آن ISR که سرواژه عبارت Interrupt Service Routine و به معنی روتین سرویس‌دهی به وقفه است، گفته می‌شود. ISR ها توسط سازنده سیستم عامل نوشته می‌شود و بخشی از هسته سیستم عامل محسوب می‌شود. در واقع با وقوع یک وقفه، برنامه کاربر قطع و یک تکه برنامه از هسته سیستم عامل به نام ISR در پاسخ به وقفه اجرا می‌شود. به عبارت بهتر با توجه به نوع Interrupt ای که آمده، ISR مربوط به آن اجرا می‌شود. هر Interrupt برای خود یک ISR مختص به خود را دارد، یعنی به تعداد وقفه‌ها، ISR داریم. برای مثال وقتی دوستتان یا پست‌چی زنگ می‌زند، روال برخورد با آن‌ها متفاوت است، کسی که وقفه را می‌دهد باید مشخصات خود را نیز بدهد، وقتی که وقفه‌دهنده شناخته شد، روال نوع پاسخ به آن وقفه پیگیری می‌شود. برای اینکار جدول وقفه بررسی می‌شود تا مشخص شود کدام ISR باید فراخوانی شود. برای مثال در قدیم کوبه درب مخصوص زنان و مردان متفاوت بود و صدای هر کوبه عکس‌العمل مختص به خود را توسط اهالی خانه در پی داشت، سیستم عامل جهت تسریع در پاسخ دادن به وقفه‌ها، آدرس روال‌های سرویس‌دهنده به آن‌ها را در جدولی به نام جدول توصیف وقفه (Interrupt Descriptor Table) نگهداری می‌کند که به ازای هر وقفه یک درایه در این جدول وجود دارد که به آن بردار وقفه (Interrupt Vector) نیز گفته می‌شود.

شروع و کشف نوع وقفه سخت‌افزاری است اما پاسخ به وقفه نرم‌افزاری و توسط سیستم عامل و اجرای ISR مرتبط با نوع وقفه است.

مطابق فرض سؤال روتین سرویس‌دهی به وقفه ساعت در هر پالس ساعت اجرا شده و پالس ساعت دارای فرکانس 75 هرتز می‌باشد. در واقع ISR با فرکانس 75Hz اجرا می‌شود، یعنی 75 بار در ثانیه، همچنین مطابق فرض سؤال روتین سرویس‌دهی به وقفه ساعت (Interrupt Service Routine) به 2 میلی ثانیه زمان برای اجرا نیاز دارد که این زمان شامل زمان اجرا و زمان تعویض زمینه (متن) می‌باشد.

پس در مجموع زمان پردازش ISR در هر ثانیه به صورت زیر محاسبه می‌گردد:

$$\text{زمان پردازش ISR در هر ثانیه} = 75 \times 2\,\text{ms} = 150\,\text{ms}$$

یعنی در هر ثانیه 150ms برای ISR مصرف می‌شود.

خواسته سؤال این است که چه درصدی از زمان پردازنده برای این روتین سرویس‌دهی به وقفه اختصاص می‌یابد، که در تناسب زیر داریم:

زمان	درصد

$$\begin{matrix} 150\,\text{ms} & x \\ 1\text{sec} = 1000\,\text{ms} & 100 \end{matrix} \quad \rightarrow \quad x = \frac{150 \times 100}{1000} = 15\%$$

۲- گزینه (۱) صحیح است.

در تکنیک صفحه‌بندی، برای هر فرآیند یک جدول صفحه تشکیل می‌شود که در آن به ازای همه صفحه‌های یک فرآیند، درایه وجود دارد و هر درایه مشخص می‌کند که کدام قاب فیزیکی به این صفحه اختصاص یافته است. در این روش وقتی فرآیندها بزرگ باشند، هزینه نگهداری جداول صفحه بسیار زیاد می‌شود، در ضمن به ازای هر فرآیند نیز باید یک جدول صفحه داشته باشیم. به عبارتی وقتی تعداد و اندازه فرآیندها بزرگ شود، این روش مقرون به صرفه نیست.

برای حل این مشکل از جداول صفحه معکوس استفاده می‌کنیم. در این حالت به جای اینکه در جداول صفحه مربوط به هر فرآیند، به ازای هر صفحه مجازی یک درایه داشته باشیم، به ازای هر قاب در حافظه فیزیکی یک درایه در جدول صفحه معکوس نگهداری می‌کنیم. در واقع به ازای هر قاب حافظه اصلی اینکه در حال حاضر کدام صفحه مربوط به کدام فرآیند در این قاب ذخیره شده است، نگهداری می‌شود. بنابراین تعداد درایه‌های جدول صفحه معکوس برابر تعداد قاب‌های حافظه فیزیکی(اصلی) است.

با استفاده از تکنیک جدول صفحه معکوس، مقدار زیادی در حافظه صرفه‌جویی می‌شود اما تبدیل آدرس مجازی به فیزیکی سخت‌تر و زمان‌گیرتر می‌شود.

در واقع وقتی یک فرآیند با PID مختص به خود به صفحه مجازی P# مراجعه می‌کند، سخت‌افزار دیگر نمی‌تواند از P# به عنوان اندیس جدول صفحه استفاده کند و صفحه فیزیکی را بیابد و از این جهت باید سرتاسر جدول صفحه وارونه را برای یافتن درایه (PID, P#) جستجو کند.

تعداد درایه‌های جدول صفحه معکوس، مطابق رابطه زیر محاسبه می‌گردد:

تعداد قاب‌های حافظه فیزیکی = تعداد درایه‌های جدول صفحه معکوس

$$\text{تعداد درایه‌های جدول صفحه معکوس} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{2^7 \times 2^{20}}{2^{11}} = 2^{16}$$

همانطور که گفتیم تعداد درایه‌های جدول صفحه معکوس برابر تعداد قاب‌های حافظه فیزیکی است، بنابراین تعداد درایه‌های جدول صفحه معکوس به صورت زیر خواهد بود:

$$\text{تعداد درایه‌های جدول صفحه معکوس} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{2^7 \times 2^{20}}{2^{11}} = 2^{16}$$

بنابراین گزینه اول پاسخ سوال است.

همچنین اندازه حافظه منطقی (فرآیند) مطابق رابطه زیر قابل محاسبه است:

$$\text{اندازه حافظه منطقی (فرآیند)} = 2^{\overset{\text{تعداد بیت آدرس منطقی}}{\text{تعداد بیت شماره صفحه}} \times 2 \overset{\text{تعداد بیت آفست}}{=}}$$

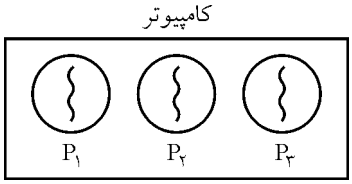
$$\text{اندازه حافظه منطقی (فرآیند)} = 2^{34} = 2^{23} \times 2^{11} = 2^{34}\,\text{Byte} = 2^4 \times 2^{30} = 16\,\text{GB}$$

۳- گزینه (۲) صحیح است.

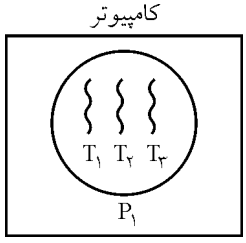
نخ یا ریسمان (Thread)

در سیستم‌های قدیمی‌تر، به ازای هر فرآیند یک رشته نخ یا رشته اجرایی و به تبع یک شمارنده برنامه (PC) وجود داشت اما در سیستم عامل‌های امروزی به ازای هر فرآیند می‌توان چند نخ یا رشته اجرایی داشت.

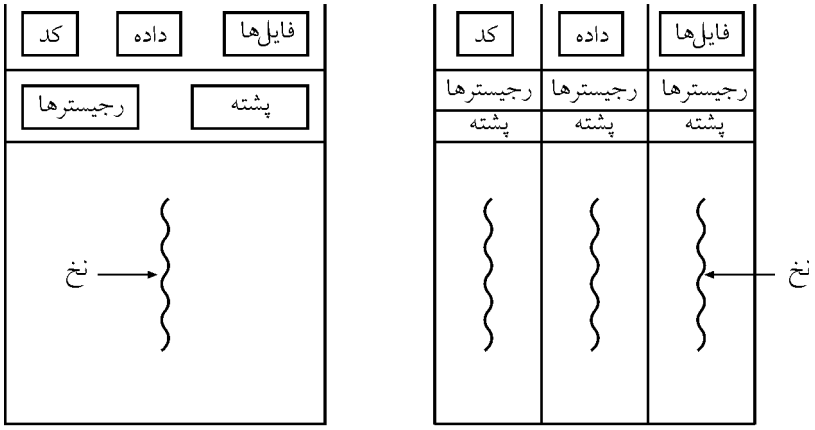
شکل زیر سه فرآیند معمولی را نشان می‌دهد که هریک برای خودشان یک رشته اجرایی و یک حافظه مختص به خود را دارند.



ولی در شکل زیر یک فرآیند، سه رشته اجرایی دارد که هر یک رجیستر، پشته و شمارنده برنامه (PC) مجزای خود را دارند و مانند فرآیندها می‌توانند همروند (در سیستم‌های تک‌پردازنده‌ای) و موازی (در سیستم‌های چندپردازنده‌ای یا چندهسته‌ای) اجرا شوند.



توجه: نخ‌های همتا که در یک فرآیند قرار دارند، از کد، داده و منابع مشترک استفاده می‌کنند اما هر نخ، شمارنده برنامه، مجموعه رجیستر و فضای پشته جداگانه‌ای در اختیار دارد. در واقع هر نخ، TCB مجزایی دارد.



فرآیند تک نخی　　فرآیند چند نخی

توجه: از آن‌جا که نخ‌های همتا در یک فرآیند قرار داشته و اشتراکات زیادی با هم دارند، عمل تعویض متن بین آنها به راحتی و با هزینه کمتری صورت می‌گیرد، در واقع TCB مربوط به نخ‌ها، محتوی کمتری نسبت به PCB فرآیندها دارد، مثلاً لیست فایل‌های باز مربوط به فرآیندها است، بنابراین این لیست به هنگام تعویض متن فرآیندها باید داخل PCB مربوط به فرآیند ذخیره گردد، در حالی که به هنگام تعویض متن بین نخ‌ها نیازی به ذخیره‌سازی لیست فایل‌های باز مربوط به یک فرآیند در TCB یک نخ نیست. بنابراین تعویض متن بین نخ‌ها نسبت به فرآیندها ارزان‌تر است.

توجه: گاهی از نخ به عنوان Light Weight Process (فرآیند سبک وزن) نیز یاد می‌کنند و به کل یک فرآیند، Heavy Weight Process (فرآیند سنگین وزن) نیز می‌گویند.

توجه: نخ‌ها هم مانند فرآیندها می‌توانند حالت‌های مختلفی را تجربه کنند مانند آماده، در حال اجرا یا منتظر. در واقع پردازنده می‌تواند بین نخ‌ها به اشتراک گذاشته شود.

چند نخی در زبان #C

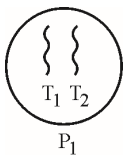
#C پیاده‌سازی چند نخی را پشتیبانی می‌کند. در زبان #C، هر برنامه به طور پیش فرض از یک نخ تشکیل شده است و درصورت ایجاد نخ‌های دیگر، مفهوم چندنخی پیاده‌سازی می‌گردد.

توجه: نخ اول به صورت پیش‌فرض وجود دارد و برنامه با نخ اول شروع به اجرا می‌کند.

مثال: در قطعه کد زیر نخ T1 به طور پیش‌فرض وجود دارد و نخ T2 ایجاد می‌گردد.

```
static void main ()
    {
Thread    T2=new    Thread (Go);
                          ↓
                  در این بخش نخ T2 ایجاد
                  می‌گردد و فعالیت Go
                  داخل این نخ قرار می‌گیرد.
→   T2.Start( )
                  نخ T2 اجرا می‌گردد.
→   Go ( );
                  فعالیت Go داخل نخ T1 قرار داده
                  شده است. می‌شود فعالیت دیگری
                  تعریف شود و در این بخش فراخوانی
                  شود.
```

```
        }
static void Go( )
{
for (int i = 0 ; i<5 ; i++)
        console.write("*");
}
```



توجه: فعالیت Go داخل نخ T1 قرار دارد، اما فعالیت Go در نخ T2 هم قرار داده شده است. در دو نخ T1 و T2 که فعالیت Go داخل آن قرار دارد، متغیر محلی i در داخل پشته مربوط به هر نخ ایجاد می‌گردد.

بنابراین خروجی این برنامه به صورت زیر خواهد بود:

****

چاپ 10 عدد ستاره به دلیل اجرای همروند (سیستم تک‌پردازنده‌ای) یا موازی (سیستم چندپردازنده‌ای یا چند هسته‌ای) دو نخ T1 و T2 است!

مثال: در قطعه کد زیر نخ T1 به طور پیش‌فرض وجود دارد و نخ T2 ایجاد می‌گردد.

```
static void main ()
    {
      Thread    T2=new    Thread (func);
T2.Start( ) ;          → نخ T2 اجرا می‌گردد.
console.write("X");    → فعالیت چاپ "X" داخل نخ T1 قرار داده شده است.
      }
static void func( )
{
console.write("Y");
}
```

در این بخش نخ T2 ایجاد می‌گردد و فعالیت func داخل این نخ قرار می‌گیرد. ↓

بنابراین خروجی این برنامه به صورت زیر خواهد بود:

XY

مزایای فرآیندهای چندنخی

۱- ساختار بسیاری از برنامه‌های کاربری ذاتاً از بخش‌های کاملاً مستقل تشکیل می‌شوند که جدا نکردن آن‌ها باعث پیچیدگی بالا و کاهش خوانایی در برنامه می‌گردد. مهندسی نرم‌افزار نیز بر ساخت برنامه‌های کاربردی توسط پیمانه‌های مستقل نیز تأکید دارد. برای مثال، خطاست اگر بیندیشید که برنامه شبیه‌سازی 11 بازیکن یک تیم فوتبال در یک نخ قرار گیرد. به عنوان مثالی دیگر یک برنامه واژه‌پرداز می‌تواند از نخ‌های مستقلی مانند کنترل املا و گرامر، صفحه‌آرایی، مدیریت ورودی‌های کاربر و غیره تشکیل شده باشد.

۲- در فرآیند تک‌نخی، هرگاه فراخوان سیستمی مسدود کننده‌ای اجرا شود، کل فرآیند مسدود می گردد. در حالی‌که در فرآیندهای چندنخی در صورتی که سیستم عامل زمان‌بندی چند نخی را پشتیبانی کند، فقط نخی که فراخوان سیستمی مسدودکننده دارد، مسدود می‌گردد و مابقی نخ‌های یک فرآیند می‌توانند پس از در اختیار گرفتن پردازنده، اجرا گردند. فرآیند تک‌نخی مانند قانونی می‌باشد که اگر یک نفر در خانواده خطا کند، همه خانواده محکوم می‌گردند و فرآیند چندنخی مانند قانونی می‌باشد که اگر یک نفر در خانواده خطا کند، فقط همان یک نفر محکوم می‌گردد و بقیه خانواده می‌توانند به زندگی طبیعی خود ادامه دهند.

۳- ایجاد هم‌روندی (در سیستم تک‌پردازنده‌ای) و توازی (در سیستم چندپردازنده‌ای یا چند هسته‌ای) در نخ‌های یک فرآیند و فرآیندهای دیگر.

مثال: کاربرد چندنخی در فرآیند سمت سرویس‌دهنده.

در این مدل، فرآیند سمت سرویس‌دهنده از چندین نخ جهت پاسخ به درخواست‌های کاربر یعنی سرویس‌گیرنده تشکیل شده است. پاسخ هر کاربر می‌تواند توسط یک نخ از سمت سرویس‌دهنده داده شود. چنانچه نخی در فرآیند سرویس‌دهنده جهت تبادل داده از روی دیسک به سمت سرویس‌گیرنده مسدود گردد، نخ‌های دیگر فرآیند سرویس‌دهنده می‌توانند به درخواست‌های دیگر، پاسخ دهند. زیرا کارکرد آن‌ها وابسته به نخ مسدود شده نیست.

توجه: شاید بگویید به جای قرار دادن کارهای مختلف یک سرویس‌دهنده در داخل نخ‌های یک فرآیند، می‌شد هریک از کارها را در داخل یک فرآیند قرار داد و چند فرآیندی را در مقابل چندنخی ابداع کرد. اما به دلایل زیر استفاده از چندنخی معقولانه‌تر به‌نظر می‌رسد:

- هزینه زمانی ایجاد (بارگذاری TCB) و پایان دادن (ذخیره‌سازی TCB) یک نخ در یک فرآیند به مراتب کمتر از ایجاد (بارگذاری PCB) و پایان دادن (ذخیره‌سازی PCB) یک فرآیند است. نخ‌های داخل یک فرآیند، از برخی منابع به صورت مشترک استفاده می کنند، در حالی که فرآیندها، منابع مختص به خود را در اختیار می‌گیرند.
- نخ‌های همتا در یک فرآیند، اشتراکات زیادی باهم دارند، بنابراین عمل تعویض متن بین آنها با هزینه کمتری انجام می‌گردد. در حالی که تعویض متن بین فرآیندها به دلیل عدم اشتراکات با هزینه بیشتری انجام می‌گردد.

توجه: وجه اشتراک نخ‌های داخل یک فرآیند شامل سگمنت داده (داده سراسری)، فضای آدرس، فایل‌های باز و وجه اختلاف نخ‌های داخل یک فرآیند شامل شمارنده برنامه (PC)، رجیسترها و پشته می‌باشد.

مدیریت نخ‌های هم‌روند(Thread Safety)

در اغلب سیستم‌های امروزی، تعدادی از فرآیندها یا نخ‌ها به صورت همروند بر روی یک پردازنده و یا به صورت موازی بر روی چندین پردازنده یا چندین هسته اجرا می‌شوند. در سیستم‌های چندبرنامگی، چندپردازنده‌ای و چندهسته‌ای، هم‌روندی و توازی فرآیندها و نخ‌ها، یک پدیده‌ی عادی به شمار می‌آید.

فرآیندها و نخ‌های هم‌روند و همکار، به ارتباط با یکدیگر نیاز دارند. آن‌ها برای دستیابی به یک هدف مشترک، نیازمند همکاری، هماهنگی، تبادل داده و استفاده از داده‌ها و سایر منابع مشترک هستند. بنابراین مدیریت اجرای همروند چند فرآیند یا چند نخ بر روی یک پردازنده و اجرای موازی چند فرآیند یا چند نخ بر روی چندین پردازنده یا چندین هسته، حائز اهمیت فراوان است. این مدیریت باید به گونه‌ای باشد که اجرای یک فرآیند یا نخ آسیبی به اجرای فرآیندها یا نخ‌های همکار و هم‌روند دیگر نرساند.

در هنگام طراحی سیستم عامل، در زمینه‌ی ارتباط بین فرآیندها یا نخ‌ها با سه مسأله‌ی اساسی زیر مواجه هستیم:

الف) تبادل داده

گاهی یک فرآیند یا یک نخ، به نتیجه‌ی محاسبات یک فرآیند یا نخ دیگر نیاز دارد، بنابراین به یک مکانیسم برای ارتباط بین فرآیندها یا نخ‌ها نیاز است. انواع مکانیزم‌های تبادل داده بین فرآیندها یا نخ‌ها به روش‌های زیر است:

- **حافظه مشترک**
- **فایل مشترک**

توجه: تبادل داده برای نخ‌های همتا و هم‌خانواده درون یک فرآیند ساده می‌باشد، زیرا نخ‌ها یک فضای آدرس مشترک دارند. اما نخ‌های متعلق به فرآیندهای جداگانه یعنی غیرهمتا که در فضای آدرس متفاوت قرار دارند، در صورت نیاز به ارتباط باید از مکانیسم‌های ارتباط فرآیندها، استفاده کنند.

ناحیه بحرانی

اگر چند فرآیند یا چند نخ قصد دسترسی به یک منبع مشترک را داشته باشند، قطعه کدی از هر فرآیند یا نخ را که در آن به دستکاری این منبع مشترک می‌پردازد، ناحیه‌ی بحرانی می‌گویند.

توجه: در همه‌ی نواحی بحرانی، دسترسی به منبع مشترک، وجود دارد، اما عکس آن همیشه صادق نیست و هرگونه دسترسی به منبع مشترک باعث رقابت و ایجاد ناحیه‌ی بحرانی نمی‌شود.

منبع بحرانی

منبعی که توسط ناحیه‌ی بحرانی مورد دستیابی قرار می‌گیرد، منبع بحرانی نام دارد، مانند متغیرهای مشترک رقابت‌زا.

ب) شرایط رقابتی (مسابقه)

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

مثال: شرایط رقابتی

دو نخ هم‌روند Thread1 و Thread2 در یک سیستم اشتراک زمانی که از متغیر مشترک سراسری S، در بخشی از کد خود استفاده می‌کنند در نظر بگیرید، بعد از اجرای کامل دو نخ، مقدار نهایی S چه خواهد شد؟ (مقدار اولیه متغیر سراسری S برابر صفر است).

Thread1 :

S=S+1

Thread2 :

S=S-1

از آنجا که این نخ‌ها به زبان اسمبلی یک ماشین فرضی در نظر گرفته می‌شوند، لذا در ادامه، دستورات فوق را به صورت سطح غیر انتزاعی‌تر (نمایش جزئیات) و در سطح اسمبلی بازنویسی می‌کنیم:

Thread1:

```
MOVE REGISTER, S
INC REGISTER
MOVE S, REGISTER
```

Thread2:

```
MOVE REGISTER, S
DEC REGISTER
MOVE S, REGISTER
```

حالت اول:

فرض کنید Thread1 کامل اجرا شود و سپس Thread2 کامل اجرا شود (یا برعکس).

Thread1:

① REGISTER ← 0
② REGISTER ← 1
③ S ← 1

Thread2:

④ REGISTER ← 1
⑤ REGISTER ← 0
⑥ S ← 0

بنابراین مقدار نهایی متغیر S برابر صفر خواهد بود.(S=0)

حالت دوم:

فرض کنید ابتدا ترتیب زیر اجرا شود.

Thread2:

② REGISTER ← 0
③ REGISTER ← -1
④ S ← -1

Thread1:

① REGISTER ← 0
INC REGISTER
MOVE S, REGISTER

سپس پردازنده در اثر تعویض متن به نخ Thread1 باز گردد.

توجه: هر نخ محتویات رجیسترهای خودش را قبل از تعویض متن در TCB ذخیره می‌کند.

بنابراین در این لحظه مقدار رجیستر در TCB فرآیند Thread1، برابر صفر است.

حال در ادامه دستورات باقی مانده نخ Thread1 اجرا می‌شوند.

Thread1:

.
.
.

⑤ REGISTER ← 1
⑥ S ← 1

بنابراین مقدار نهایی متغیر S برابر مثبت یک خواهد بود.(S= +1)

حالت سوم:

فرض کنید ابتدا ترتیب زیر اجرا شود.

Thread2:

① REGISTER ← 0
DEC REGISTER
MOVE S, REGISTER

Thread1:

② REGISTER ← 0
③ REGISTER ← 1
④ S ← 1

سپس پردازنده در اثر تعویض متن به نخ Thread2 باز گردد.

توجه: هر نخ محتویات رجیسترهای خودش را قبل از تعویض متن در TCB ذخیره می‌کند.

بنابراین در این لحظه مقدار رجیستر در TCB نخ Thread2، برابر صفر است.

حال در ادامه دستورات باقی مانده نخ Thread2 اجرا می‌شوند.

Thread2:

.
.
.

⑤ REGISTER ← -1
⑥ S ← -1

بنابراین مقدار نهایی متغیر S برابر منفی یک خواهد بود. (S= -1)

توجه: اما مشکل نهایی اینجاست که مقدار نهایی متغیر S، به نحوه‌ی **تعویض متن پردازنده** یا به عبارتی، به ترتیب اجرای دستورالعمل‌ها، بستگی دارد و می‌تواند مقادیر 0، 1، 1- را داشته باشد. این پدیده، حاصل رقابت بر سر تصاحب یک عامل مشترک (متغیر مشترک S) است.

توجه: وجود پدیده‌ی رقابت در مثال قبل در سیستم‌های تک‌پردازنده‌ای ناشی از وقفه‌ای است که می‌تواند اجرای دستورالعمل‌ها را در هر کجای نخ متوقف نماید. (پدیده‌ی تعویض متن). این وضعیت در سیستم‌های چندپردازنده‌ای و چندهسته‌ای نیز ممکن است پیش بیاید، به علاوه اینکه دو یا چند نخ می‌توانند به موازات هم اجرا شده و برای دسترسی به یک عامل مشترک در رقابت باشند.

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

۱- شرط انحصار متقابل

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

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

```
Thread () {
while (TRUE) {
entry_section () ; // تلاش برای کسب اجازه‌ی ورود به ناحیه‌ی بحرانی
critical_ selection (); // ناحیه‌ی بحرانی
exit_ selection () ; // اعلام خروج از ناحیه‌ی بحرانی
remainder_ section () ; // ناحیه‌ی باقی مانده
  }
}
```

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

۲- شرط پیشرفت

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

۳- شرط انتظار محدود

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

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

قحطی (گرسنگی)

در عالم زندگی قحطی زمانی رخ می‌دهد که عده‌ای مدام از منابع مشترک استفاده کنند و عده‌ای دیگر قادر به استفاده از منابع مشترک نباشند. زیرا دسته‌ی اول از اختصاص منابع به دسته‌ی دوم به طور مداوم و بدون رعایت یک حد بالای مشخص جلوگیری می‌کنند. در عالم نخ‌ها نیز هرگاه نخی به مدت نامعلوم و بدون رعایت یک حد بالای مشخص در انتظار گرفتن یک منبع بحرانی یا دسترسی به یک عامل مشترک بماند و نخی دیگر مدام در حال استفاده از منبع بحرانی باشد، در این حالت نخ اول دچار قحطی شده است. بنابراین در صورت اقدام یک نخ برای ورود به ناحیه‌ی بحرانی، باید محدودیتی برای تعداد دفعاتی که سایر نخ‌ها می‌توانند وارد ناحیه‌ی بحرانی شوند، وجود داشته باشد تا قحطی رخ ندهد.

بن‌بست

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

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

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

ج) همگام سازی

گاهی اوقات خواسته‌ی ما این است که نخ‌ها به یک ترتیب مشخص و از قبل تعیین شده بر روی یک عامل مشترک (داده مشترک) عملیاتی را انجام دهند. واضح است که در این حالت تبادل داده به روش استفاده از حافظه مشترک است. همچنین از آنجا که پای یک عامل مشترک در میان است، پس رقابت بر سر تصاحب این عامل مشترک هم در میان است. بنابراین راه‌حل همگام‌سازی باید به گونه‌ای باشد که نخ‌های همکار دچار شرایط رقابتی نشوند. به بیان دیگر باید مانع اثر مخرب **تعویض متن پردازنده** بین نخ‌های همکار که عامل ایجاد شرایط رقابتی است، شد.

برای مثال اگر نخ Thread1 داده‌ای را باید تولید کند و سپس نخ Thread2 آن را مصرف کند، نخ Thread2 باید منتظر باشد تا نخ Thread1، داده‌ی مورد نیازش را آماده کند و بعد شروع به مصرف نماید.

مثال: همگام سازی دو نخ

دو نخ Thread1 و Thread2 را در نظر بگیرید که در یک سیستم تک‌پردازنده‌ای اشتراک زمانی به صورت هم‌روند اجرا می‌شوند. فرض کنید نخ Thread1 در بخشی از کد خود مقدار متغیر x را می‌خواند و نخ Thread2 نیز در بخشی از برنامه‌اش باید مقدار متغیر x خوانده شده توسط Thread1 را چاپ کند. هدف مسأله، نوشتن این دو نخ به صورتی است که Thread2 در چاپ متغیر x بر Thread1 پیشی نگیرد و اگر زودتر به بخش چاپ متغیر x رسید و هنوز مقدار متغیر x خوانده نشده بود، صبر کند تا Thread1 متغیر x را مقداردهی کند. بنابراین باید یک مسأله‌ی همگام‌سازی حل شود. واضح است که در این حالت تبادل داده به روش استفاده از حافظه مشترک است. یکی از راه‌حل‌های مناسب برای همگام‌سازی نخ‌های همکار، استفاده از کلاس سمافور و یا کلاس مانیتور است.

راه‌حل‌های کنترل شرایط رقابتی نخ‌های همکار

۱- راه‌حل‌های نرم‌افزاری (برعهده‌ی برنامه‌نویس)

۲- راه‌حل‌های سخت‌افزاری

۳- راه‌حل کلاس سمافور

۴- راه‌حل کلاس مانیتور

راه‌حل‌های همگام‌سازی نخ‌های همکار

۱- راه‌حل کلاس سمافور

۲- راه‌حل کلاس مانیتور

توجه: راه‌حل‌های فوق، در سیستم‌های تک‌پردازنده‌ای، چندپردازنده‌ای و چندهسته‌ای با حافظـه‌ی اشتراکی قابل استفاده‌اند.

خواسته سوال این است که چه وقت یک تابع در سطح نخ **ایمن** است؟

جهت پاسخ به خواسته سوال شبه کد زیر را در نظر بگیرید:

```
#include <pthread.h>
  static void main ( )
     {
      Thread T2=new Thread (increment_counter);
         T2.Start ( )
         increment_counter ( );
     }
int increment_counter ( )
{
static int counter = 0;
static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
//only allow one thread to increment at a time
pthread_mutex_lock (&mutex);
counter= counter+1;
//store value before any other threads increment it further
int result = counter;
pthread_mutex_unlock (&mutex);
return result;
}
```

توجـــه: فعالیـــت increment_counter داخـــل نـــخ Thread1 قـــرار دارد، همچنـــین فعالیـــت increment_counter در نخ Thread2 هم قرار داده شده است. هر نخ پشته و رجیستر مخـتص بـه خود را دارد و متغیرهای محلی یک نخ در داخل پشته مربوط به هر نخ ایجاد می‌گـردد. امـا در دو نخ Thread1 و Thread2 که فعالیت increment_counter داخـل آن قـرار دارد، متغیـر counter از نوع static است که داخل سگمنت داده نگهداری می‌شود که به تبع در دو نخ به اشـتراک گذاشـته می‌شود، بنابراین متغیر counter منبع بحرانی و دستور counter= counter+1 ناحیه بحرانـی مـابین دو نخ محسوب می‌گردد که باید محافظت گردد. **بنابراین یک تابع همچون increment_counter در سطح نخ ایمن است اگر و فقط اگر از نخ‌های همروند فراخوانی شود، همیشه نتیجه درست را برگرداند.** دقت کنید که این رابطه دو طرفه صـادق اسـت، بنـابراین شـرط لازم و کـافی در هـر طرف برقرار است و طرف دیگر را نتیجه می دهد. بنابراین گزینه دوم پاسخ سوال است.

۴- گزینه ( ) صحیح است.

نخ‌ها هم مانند فرآیندها می‌توانند حالت‌های مختلفی را تجربه کنند مانند آماده، در حال اجرا یا منتظر (مسدود). در واقع پردازنده می‌تواند بین نخ‌ها به اشتراک گذاشته شود. حالت آماده اجرا همان معنی حالت آماده را می‌دهد بنابراین گزینه دوم و سوم یکسان هستند، سازمان سنجش آموزش کشور در کلید اولیه خود ابتدا گزینه سوم را به عنوان پاسخ اعلام کرده بود، سپس در کلید نهایی نظر خود را عوض کرد و گزینه سوم را با تأثیر مثبت اعلام کرد.

در روش سطح کاربر یا مدل چند به یک (Many to One) فقط زمان‌بند پردازنده و زمان‌بند چندنخی در سطح کاربر وجود دارد و زمان‌بند چند نخی در سطح هسته در این روش مورد استفاده قرار نمی‌گیرد. در واقع هسته سیستم عامل فقط فرآیندها را می‌شناسد و هیچ اطلاعاتی از نخ‌ها ندارد. در واقع اولویت‌بندی نخ‌ها، مدیریت نخ‌ها و زمان‌بند چندنخی در سطح کاربر و توسط یک بسته نرم‌افزاری انجام می‌گردد. بدین معنی که نخ‌ها را برنامه‌نویس مشخص می‌کند و مدیریت آن‌ها را نیز بر عهده می‌گیرد. بنابراین زمان‌بند پردازنده، براساس الگوریتم مشخصی مثلاً نوبت چرخشی پردازنده را در اختیار یکی از فرآیندهای آماده قرار می‌دهد. سپس زمان‌بند چند نخی در سطح کاربر، متناسب با کاربردی که در آن فرآیند به کار گرفته می‌شود، الگوریتم زمان‌بند را انتخاب کرده و تصمیم می‌گیرد که پردازنده در اختیار کدام‌یک از نخ‌های آماده در فرآیند موردنظر قرار گیرد و تا زمانی که پردازنده در تملک فرآیند باشد و یا تا قبل از پایان برش زمانی مربوط به فرآیند، نخ‌های یک فرآیند از پردازنده بهره‌مند می‌شوند و به محض مسدود شدن یک نخ، و یا پایان برش زمانی یک فرآیند، یا اتمام فرآیند، پردازنده به فرآیند بعدی تعلق می‌گیرد.

توجه: در این روش نخ ماهیت منطقی دارد و از دید کاربر فقط وجود دارد، در واقع از نظر سیستم عامل ماهیت فیزیکی ندارد، بنابراین نخ کاربر در این روش همانند یک تابع در فرآیند می‌باشد که از رجیستر و پشته مختص به خود نیز بهره‌مند نمی‌باشد.

توجه: مدل غیرکامپیوتری این روش نیز وجود دارد، مانند حالتی که درآمدهای دولت حاصل از منابع کشور، بین پدران خانواده‌ها تقسیم گردد و این پدران خانواده‌ها باشند که تصمیم بگیرند به هر عضو خانواده چه مقدار نقدینگی تعلق بگیرد. در این روش فقط پدران شماره حساب مختص به خود را دارند. اما اگر یکی از اعضای خانواده خطایی انجام دهد و محکوم گردد، آنگاه تمام اعضای خانواده برای مدتی از خدمات دولت محکوم می‌گردند، زیرا در این مدل، دولت فقط پدران خانواده را می‌شناسد و از اعضای خانواده اطلاعی ندارد. بنابراین حساب پدر خانواده برای مدتی مسدود می‌گردد.

توجه: عمل تعویض متن مابین نخ‌های یک فرآیند کاربر، کاملاً در سطح کاربر و با سربار بسیار ناچیز (در حد فراخوانی نخ بعدی) و بدون تغییر حالت پردازنده به مد هسته پردازنده، توسط زمان‌بند چندنخی در سطح کاربر (برنامه‌های کاربر) انجام می‌گردد. اما عمل تعویض متن مابین فرآیندها (فرآیندهای کاربر یا فرآیندهای سیستم عامل) در سطح هسته سیستم عامل و در مد هسته

پردازنده، توسط زمان‌بند پردازنده برای انتخاب یک فرآیند جدید بر اساس یک الگوریتم خاص انجام می‌گردد.

توجه: به دلیل آنکه نخ‌های یک فرآیند کاربر، تماماً در فضای کاربر مدیریت می‌شوند، اگر نخ موجود در یک فرآیند کاربر، یک فراخوان سیستمی مسدودکننده را اجرا نماید، هسته سیستم عامل نه تنها آن نخ، بلکه کل فرآیند کاربر را که شامل تمام نخ‌های دیگر می‌باشد، مسدود می‌کند، زیرا هسته سیستم عامل خبری از نخ‌های داخل فرآیند کاربر ندارد. در واقع در این روش هسته سیستم عامل نخ‌ها را همانند توابع داخل یک فرآیند می‌بیند، یعنی هسته سیستم عامل، یک فرآیند چند نخی سطح کاربر را، مانند یک فرآیند تک نخی اما دارای چند تابع مختلف می‌بیند! بنابراین در اینجا انتقال از حالت آماده نخ‌های همتای دیگر به حالت مسدود (منتظر) به دلیل اجرای فراخوان سیستمی مسدود کننده توسط یک نخ همتای دیگر را داریم، بنابراین گزینه دوم و سوم هم می‌تواند در این حالت درست باشد ولی نه در حالت کلی.

توجه: این راه‌کار، منجر به عدم امکان هم‌روندی (در سیستم‌های تک‌پردازنده‌ای) نخ‌های داخل یک فرآیند کاربر در حالت انسداد یک نخ داخل یک فرآیند کاربر می‌گردد.

البته اگر نخی داخل یک فرآیند کاربر مسدود نگردد، امکان هم‌روندی میان نخ‌های داخل فرآیند کاربر برقرار است. مانند یک تیم فوتبال ۱۱ نفره که اگر بازیکنی مرتکب خطا گردد، از آن‌جا که داور فقط نام تیم را می‌شناسد و نه تک‌تک بازیکنان تیم را، آن‌گاه کل تیم را جریمه، اخراج و مسدود می‌کند. اما اگر هیچ‌یک از بازیکنان تیم مرتکب خطایی نگردد، واضح است که هم‌روندی برقرار است.

توجه: فرض کنید یک کیک داریم که آن را به چهار قسمت مساوی تقسیم کرده‌ایم، همچنین فرض کنید خوردن هر بخش کیک یک ساعت زمان بخواهد، اگر یک نفر بخواهد تمام این کیک را بخورد، پس ۴ ساعت طول می‌کشد اگر ۴ نفر بخواهند تمام این کیک را بخورند و به هر نفر یک بخش کیک داده شود، آنگاه خوردن تمام کیک به‌طور موازی ۱ ساعت طول خواهد کشید. در روش سطح کاربر، یک فرآیند چند نخی کاربر نمی‌تواند از امتیازات چند پردازنده‌ای بهره ببرد، زیرا در روش سطح کاربر، هسته سیستم عامل در هر لحظه فقط یک پردازنده را در اختیار نخ‌های یک فرآیند کاربر قرار می‌دهد. حتی اگر چند پردازنده موجود باشد. یعنی در این روش خوردن کیک بخش‌بندی شده به صورت چند نفری امکان‌پذیر نیست. بنابراین در این روش امکان پردازش موازی نخ‌های داخل یک فرآیند کاربر وجود ندارد.

توجه: نرم‌افزارهای POSIX P-threads و Mach C-thread به عنوان یک بسته نرم‌افزاری، می‌توانند جهت مدیریت نخ‌ها و زمان‌بند چند نخی در سطح کاربر مورد استفاده قرار گیرند.

توجه: در این روش تخصیص منابع و زمان‌بندی پردازنده، بر روی فرآیندها انجام می‌شود. همچنین زمان‌بندی نخ‌های سطح کاربر، بر عهده زمان‌بند چندنخی سطح کاربر خواهد بود.

توجه: سیستم عامل سولاریس، مدل چند به یک را پیاده سازی می‌کند.

حذف نخی (Thread Cancellation)

حذف نخی به معنی **خاتمه‌دادن** یک نخ قبل از تکمیل‌شدن آن است. برای‌مثال اگر چند نخ بطور همروند درحال جستجو در یک پایگاه داده باشند، در صورتی که یکی از نخ‌ها نتیجه را برگرداند، **احتمالا** ادامه کار مابقی نخ‌ها کنسل می‌شود. حالت دیگر وقتی است که یک کاربر، دکمه‌ای را در یک مرورگر وب می‌فشارد که صفحه وب را از ادامه بارگذاری باز می‌دارد، اغلب یک صفحه وب با استفاده از چندین نخ بارگذاری می‌شود (هر تصویر توسط یک نخ مجزا بارگذاری می‌شود). وقتی که یک کاربر، دکمه stop را در مرورگر می‌فشارد، تمامی نخ‌های بارگذاری صفحه کنسل می‌شوند. نخی که باید کنسل شود اغلب **نخ هدف** (Target Thread) معرفی می‌شود. حذف (کنسل‌کردن) نخ هدف ممکن است در دو سناریوی مختلف رخ دهد:

۱- حذف ناهمگام (Asynchronous Cancellation)

یک نخ بلافاصله **نخ هدف** را خاتمه دهد، حذف نخ در این حالت به شکل نا ایمن انجام می‌گردد، و بدون بررسی اشتراکات نخ حذف شده با نخ‌های دیگر.

بنابراین در این حالت، انتقال از حالت آماده نخ هدف به حالت خاتمه وجود دارد، پس گزینه اول هم می‌تواند درست باشد ولی نه در حالت کلی.

۲- حذف همراه تعویق (Asynchronous Cancellation)

نخ هدف به طور متناوب چک می‌شود که آیا می‌تواند خاتمه پیدا کند یا نه، و به آن اجازه داده می‌شود که بنا به درخواستش در صورت امکان به شکل امن، خاتمه یابد. حذف نخ در این حالت به شکل ایمن انجام می‌گردد، عموم سیستم‌عامل‌ها حذف همراه تعویق را پشتیبانی می‌کنند. به طور مثال سیستم عامل لینوکس حذف همراه تعویق را پشتیبانی می‌کند.

بنابراین در این حالت، انتقال از حالت اجرای نخ هدف به خاتمه وجود دارد، ولی نه در حالت کلی.

۵- گزینه ( ) صحیح است.

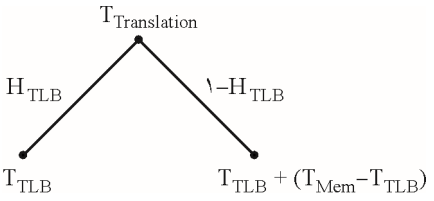
مطابق فرض سوال دو سیستم S_1 و S_2 همزمان با ارسال درخواست به TLB، آنرا به حافظه اصلی نیز ارسال می‌کنند تا در صورت عدم یافتن آدرس در TLB، زمان پاسخ کاهش یابد. به عبارت دیگر نحوه عملکرد دو سیستم S_1 و S_2 **موازی** در نظرگرفته شده‌است، به صورت زیر:

$\mathbf{T_{Translation}}$: میانگین زمان ترجمه در حالت **موازی**

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

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

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



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

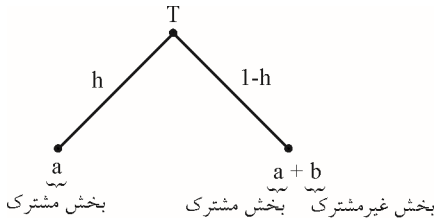
$$T_{Translation} = T_1 = T_{TLB} + (1\text{-}H_{TLB}) \times (T_{Mem} - T_{TLB})$$

$$T_{Translation} = T_1 = 120 + (1\text{-}0.6) \times (T_{Mem} - 120) = 120 + 0.4 \times T_{Mem} - 48 = 72 + 0.4 \times T_{Mem}$$

$$T_{Translation} = T_2 = T_{TLB} + (1\text{-}H_{TLB}) \times (T_{Mem} - T_{TLB})$$

$$T_{Translation} = T_2 = 150 + (1\text{-}0.8) \times (T_{Mem} - 150) = 150 + 0.2 \times T_{Mem} - 30 = 120 + 0.2 \times T_{Mem}$$

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



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

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

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

$$T_{Translation} = T_1 = T_{TLB} + (1\text{-}H_{TLB}) \times (T_{Mem} - T_{TLB})$$

$$T_{Translation} = T_1 = 120 + (1\text{-}0.6) \times (T_{Mem} - 120) = 120 + 0.4 \times T_{Mem} - 48 = 72 + 0.4 \times T_{Mem}$$

$$T_{Translation} = T_2 = T_{TLB} + (1\text{-}H_{TLB}) \times (T_{Mem} - T_{TLB})$$

$$T_{Translation} = T_2 = 150 + (1\text{-}0.8) \times (T_{Mem} - 150) = 150 + 0.2 \times T_{Mem} - 30 = 120 + 0.2 \times T_{Mem}$$

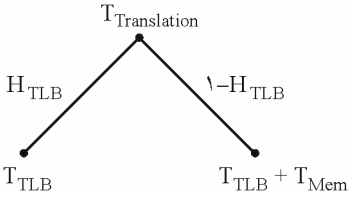
همچنین مطابق فرض سوال سیستم S_3 پس از دریافت پاسخ از TLB و عدم یافتن آدرس، درخواست را به حافظه اصلی ارسال می‌کند. به عبارت دیگر نحوه عملکرد سیستم S_3 **ترتیبی** (سری) در نظرگرفته شده‌است، به صورت زیر:

$\mathbf{T_{Translation}}$: میانگین زمان ترجمه در حالت **ترتیبی**

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

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

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



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

$T_{Translation} = T_3 = T_{TLB} + (1\text{-}H_{TLB}) \times T_{Mem}$

$T_{Translation} = T_3 = 120 + (1\text{-}0.7) \times T_{Mem} = 120 + 0.3 \times T_{Mem}$

خواسته سوال این است که **کمترین** و **بیشترین** میانگین زمان پاسخ این سه سیستم به ترتیب از راست به چپ چند نانوثانیه است؟

که مطابق آنچه پیشتر گفتیم، داریم:

$T_{Translation} = T_1 = 120 + (1\text{-}0.6) \times (T_{Mem} - 120) = 120 + 0.4 \times T_{Mem} - 48 = 72 + 0.4 \times T_{Mem}$

$T_{Translation} = T_2 = 150 + (1\text{-}0.8) \times (T_{Mem} - 150) = 150 + 0.2 \times T_{Mem} - 30 = 120 + 0.2 \times T_{Mem}$

$T_{Translation} = T_3 = 120 + (1\text{-}0.7) \times T_{Mem} = 120 + 0.3 \times T_{Mem}$

متاسفانه مقدار T_{Mem} در صورت سوال توسط طراح محترم بیان نشده است، که باعث شده ادامه حل مساله امکان‌پذیر نباشد، اما اگر مقدار T_{Mem} را برابر 600 ns در نظر بگیریم، آنگاه نتایج زیر را خواهیم داشت:

$T_{Translation} = T_1 = 72 + 0.4 \times T_{Mem} = 72 + 240 = 312$ ns

$T_{Translation} = T_2 = 120 + 0.2 \times T_{Mem} = 120 + 120 = 240$ ns

$T_{Translation} = T_3 = 120 + 0.3 \times T_{Mem} = 120 + 180 = 300$ ns

اگر مقدار T_{Mem} از 600 ns بیشتر شود، آنگاه **بیشترین** میانگین زمان پاسخ که مربوط بـه T_1 و برابـر 312 است، از 312 بیشتر می‌شود که در گزینه‌ها بیشتر از مقدار 312 وجود ندارد. بنـابراین در ایـن حالت **کمترین** و **بیشترین** میانگین زمان پاسخ این سه سیستم به ترتیب برابـر 312 و 240 نانوثانیـه است که در هیچیک از گزینه‌ها وجود ندارد.

توجه: منظور طرح از زمان پاسخ، محاسبه زمان ترجمه یعنـی $T_{Translation}$ در شـرایط مطـرح شـده است، و نه محاسبه زمان دسترسی به یک مقصد مورد نظـر، بنـابراین در محاسـبات فـوق از زمـان مقصد یعنی $T_{Destination}$ استفاده نکردیم.

توجه: سازمان سنجش آموزش کشور در کلید اولیه خود ابتدا گزینه دوم را به عنـوان پاسـخ اعـلام نمود، سپس در کلید نهایی نظر خود را عوض کرد و کلاً تست را حذف نمود، که عمل درسـتی را انجام داده است که البته عمل درست‌تر آن است که سؤال از ابتدا، درست طرح شود.

۶- گزینه (۴) صحیح است.

ابتدا وضعیت سیستم را قبل از اجابت درخواست فرآیند P_3 بررسی می‌کنیم:

با توجه بـه مـاتریس تخصـیص منـابع (Allocation) و منـابع اولیـه، بـردار Available را بدسـت می‌آوریم، برای یافتن بردار Available ابتدا برای هر منبع، منابع تخصیص یافته به هر فرآینـد را بـا هم جمع کرده و سپس از منابع اولیه کسر می‌نماییم.

A منبع موجود $= 8-(2+3+1+1) = 1$

B منبع موجود $= 6-(1+2+1) = 2$

C منبع موجود $= 10-(2+3+2) = 3$

بنابراین بردار Available به صورت زیر است:

Available	1	2	3

براساس رابطه زیر داریم:

$$Need = MAX - Allocation$$

فرآیند	Need		
	A	B	C
P_0	3	5	6
P_1	5	3	3
P_2	2	1	3
P_3	3	5	7
P_4	1	2	3

=

فرآیند	MAX		
	A	B	C
P_0	3	6	8
P_1	7	3	6
P_2	5	3	3
P_3	4	5	9
P_4	2	3	3

−

فرآیند	Allocation		
	A	B	C
P_0	0	1	2
P_1	2	0	3
P_2	3	2	0
P_3	1	0	2
P_4	1	1	0

Available	1	2	3

$$(1,2,3) \xrightarrow[+(1,1,0)]{P_4} (2,3,3) \xrightarrow[+(3,2,0)]{P_2} (5,5,3) \xrightarrow[+(2,0,3)]{P_1} (7,5,6)$$

$$\xrightarrow[+(0,1,2)]{P_0} (7,6,8) \xrightarrow[+(1,0,2)]{P_3} (8,6,10)$$

به این ترتیب و با توجه بـه ماتریس‌هـای فـوق، دنبالـه‌ی امـن $< P_4, P_2, P_1, P_0, P_3 >$ (از چـپ بـه راست) برای این سیستم موجود است. بنابراین قبـل از اجابـت درخواسـت فرآینـد P_3، سیسـتم در حالت امن قرار دارد و گزینه‌های دوم و سوم نادرست هستند.

الگوریتم درخواست منبع بانکداران

هنگامی که فرآیند P_i، بردار Request[i] را به عنوان اعلام نیاز به منابع مطـرح می‌کنـد، بایـد ایـن درخواست مطابق مراحل زیر بررسی گردد:

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

۲) اگر $Request[i] \leq Availabe[i]$ است به مرحله 3 بـرو، در غیـر اینصـورت منبـع کـافی جهـت تخصیص به این فرآیند وجود نداشته و این فرآیند باید منتظر بماند.

۳) با فرض این که منابع مورد نیاز به این فرآیند اختصـاص می‌یابـد، در ایـن صـورت بـا اصـلاح ماتریس‌ها و بردارها یک الگوریتم تشخیص وضعیت امن اجرا می‌شود. با اجرای این الگوریتم اگر سیستم در وضعیت امن باقی بماند، این منابع به فرآیند اختصاص می‌یابند، در غیراینصورت اگر بـا این تخصیص سیستم به وضعیت ناامن وارد شود، ماتریس‌ها و بردارها به حالـت قبـل بازگردانـده شده و فرآیند تا آزاد شدن منابع بیشتر منتظر می‌ماند.

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

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

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

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

مطابق فرض صورت سؤال، اگر در این وضعیت، درخواستی بـرای یـک واحـد دیگـر از منبـع A، توسط فرآیند P_3 صادر شود، مطابق الگوریتم درخواست منبع بانکداران، ابتدا بررسـی می‌شـود کـه آیا درخواست فرآیند، کم‌تر از نیاز اعلام شده آن است یا خیر.

$Request[P_3] \leq Need[P_3]$

$[1,0,0] \leq [3,5,7]$

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

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

$Request[P_3] \leq Available$

$[1,0,0] \leq [1,2,3]$

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

فرآیند	Need (new)		
	A	B	C
P_0	3	5	6
P_1	5	3	3
P_2	2	1	3
P_3	②	5	7
P_4	1	2	3

=

فرآیند	MAX		
	A	B	C
P_0	3	6	8
P_1	7	3	6
P_2	5	3	3
P_3	4	5	9
P_4	2	3	3

−

فرآیند	Allocation (new)		
	A	B	C
P_0	0	1	2
P_1	2	0	3
P_2	3	2	0
P_3	②	0	2
P_4	1	1	0

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

منبع موجود A $= 8-(2+3+2+1) = 0$

منبع موجود B $= 6-(1+2+1) = 2$

منبع موجود C $= 10-(2+3+2) = 3$

بنابراین بردار Available(new) به صورت زیر است:

Available(new)	⓪	2	3

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

Available (new) = Available (old) – Request [P_3]
Available (new) = [1,2,3] – [1,0,0] = [0,2,3]

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

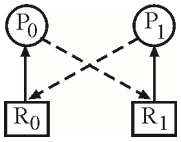
توجه: ناامنی نه به معنی وقوع بن‌بست در گذشته و نه به معنی وقوع حتمی بن‌بست در آینده است، بلکه به معنی احتمال وقوع بن‌بست در آینده است.

توجه: در حالت ناامن نیز ممکن است، فرآیندها در موقعیت رهاسازی منابع در اختیار خود قرار بگیرند و سیستم به بن‌بست نرسد. به بیان دیگر در حالت ناامن وقوع بن‌بست قطعی نیست. اما اگر در حالت ناامن فرآیندها علاوه بر نگهداری منابع تحت تملک خود، منابع دیگری را مورد درخواست قرار دهند، در این حالت فرآیندهایی که نیاز آن‌ها برطرف نمی‌شود یک به یک در

صف انتظار قرار می‌گیرند و همچون سرنوشت‌های به هم گره خورده، تا ابد منتظر یکدیگر باقی می‌مانند، یعنی بن‌بست رخ داده است.

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

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



فرآیند	MAX	
	R_0	R_1
P_0	1	1
P_1	1	1

=

فرآیند	Allocation	
	R_0	R_1
P_0	1	0
P_1	0	1

+

فرآیند	Need	
	R_0	R_1
P_0	0	1
P_1	1	0

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

R_0 منبع موجود $= 1-(1) = 0$

R_1 منبع موجود $= 1-(1) = 0$

بنابراین بردار Available به صورت زیر است:

Available	0	0

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

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

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

P_1 نیز در صف انتظار، می‌خوابد (نگهداری و انتظار). این سرنوشت‌های بـه هـم گـره خـورده، در ادامه بن‌بست را به ارمغان می‌آورد.

توجه: از دو توجه فوق این نتیجه استنباط می‌گردد که در شرایط ناامن، **احتمـال وقـوع بن‌بسـت** وجود دارد. در واقع در شرایط ناامن وقوع بن‌بست و عدم بن‌بسـت بـه رفتـار فرآینـدها در آینـده، بستگی دارد. یعنی فرآیندها رفتار نگهداری و درخواست را در پیش گیرند و یا رفتـار رهاسـازی و درخواست را.