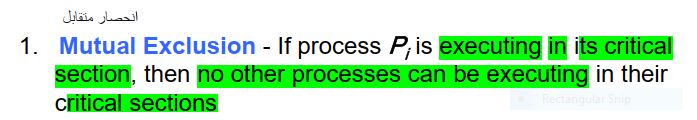
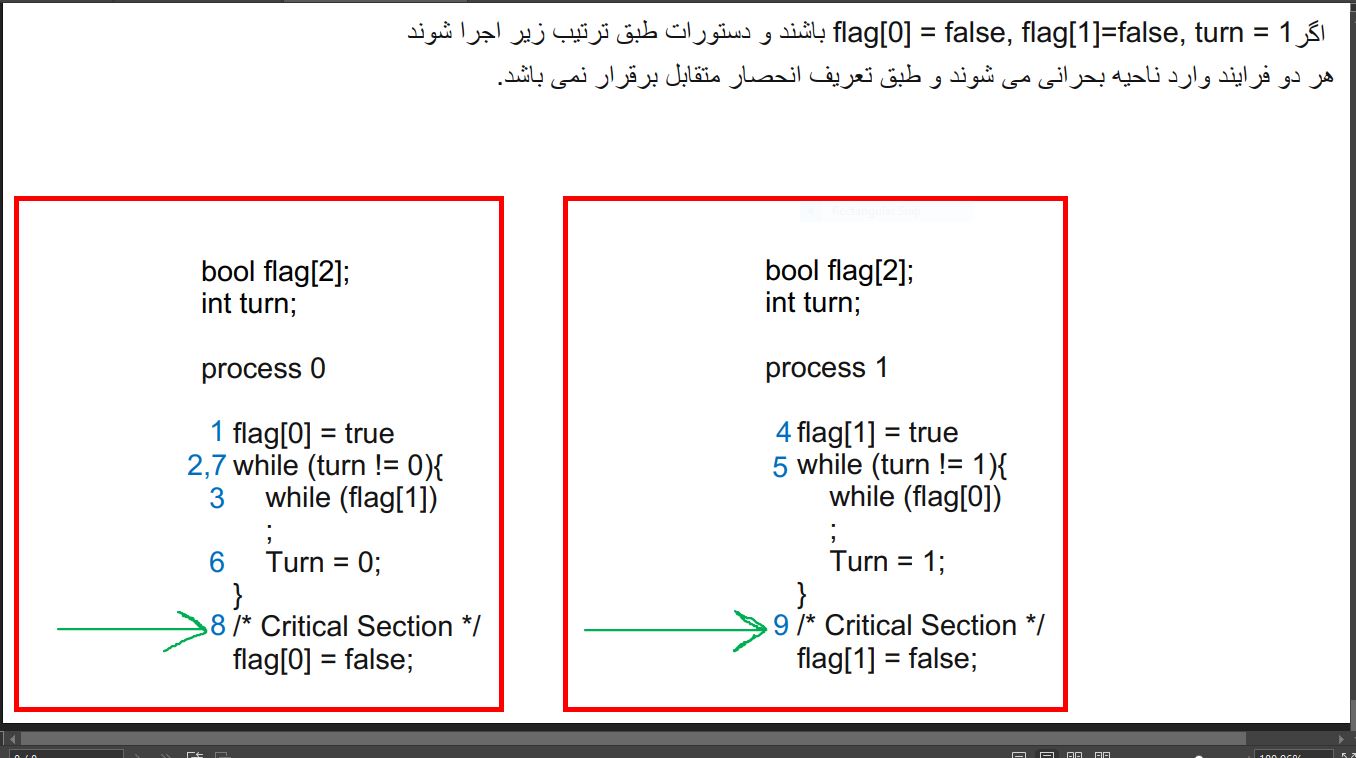
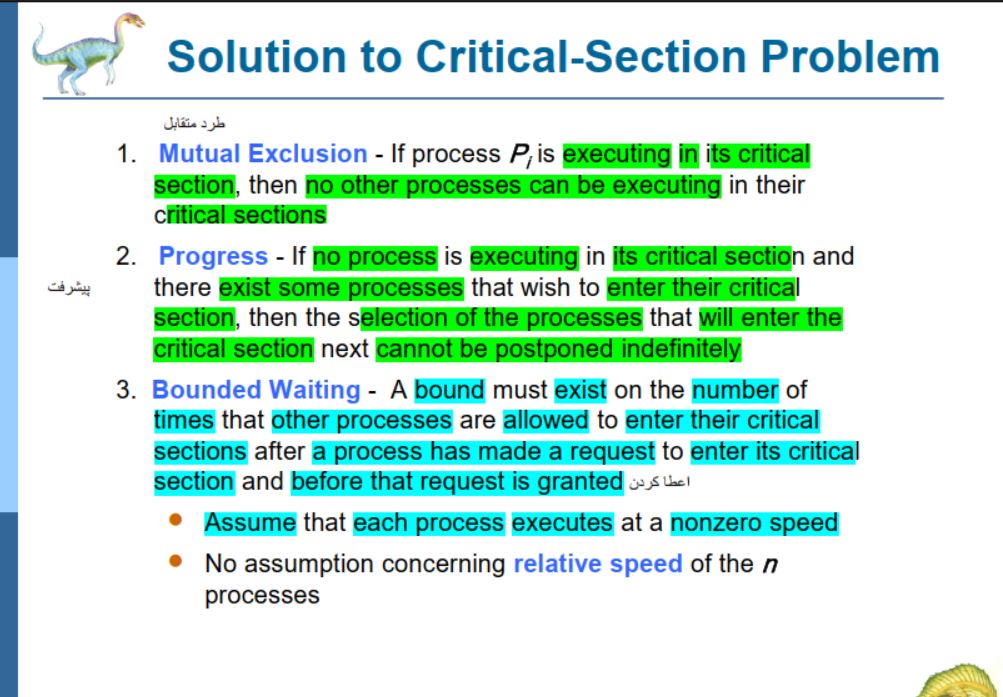
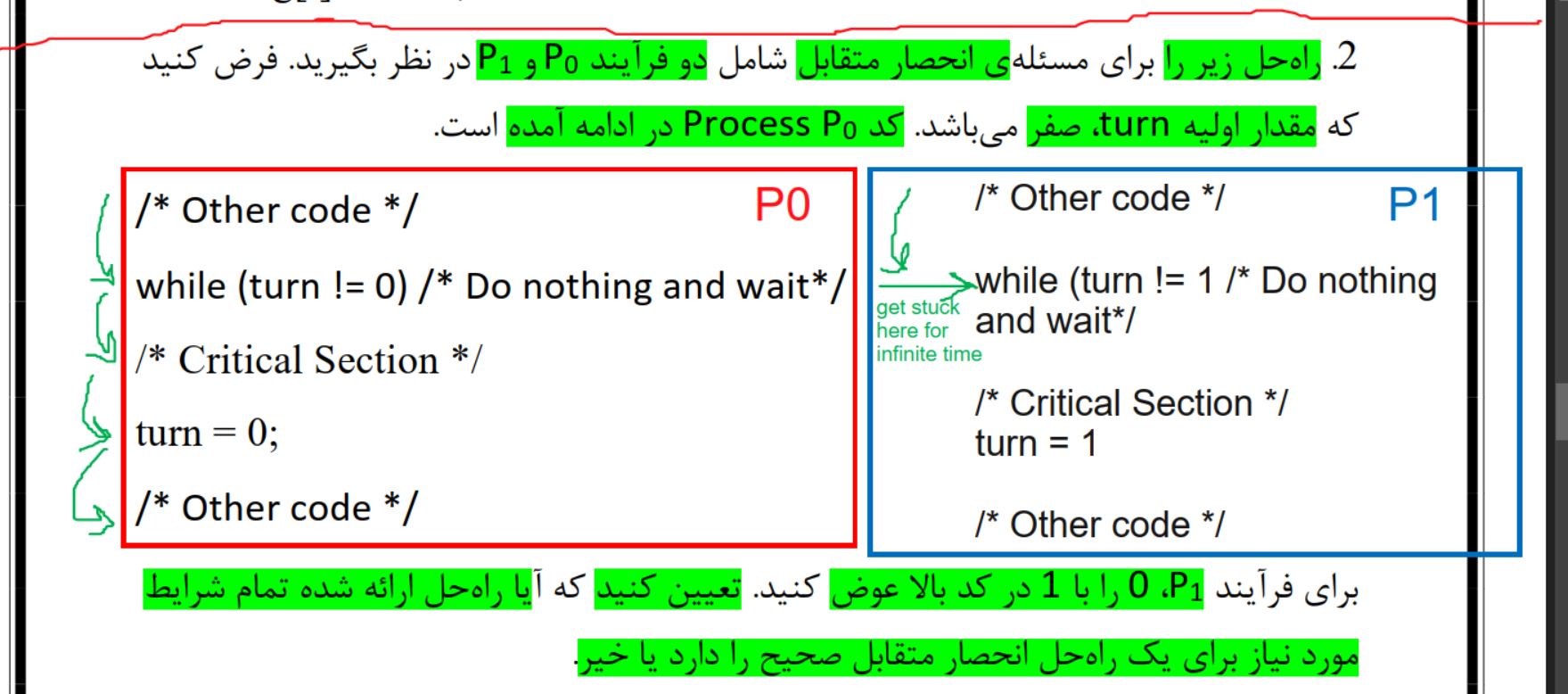
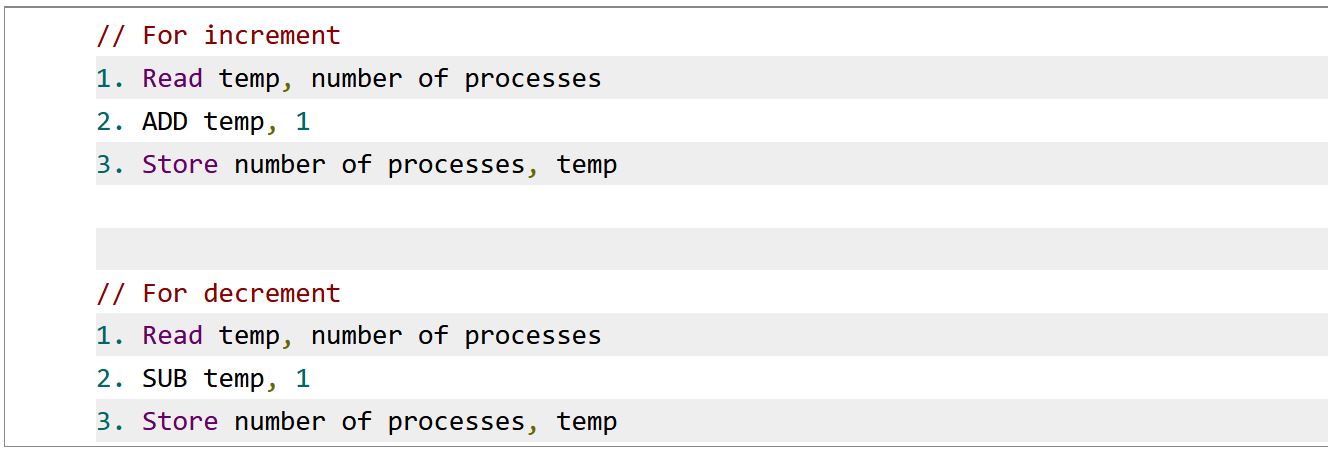
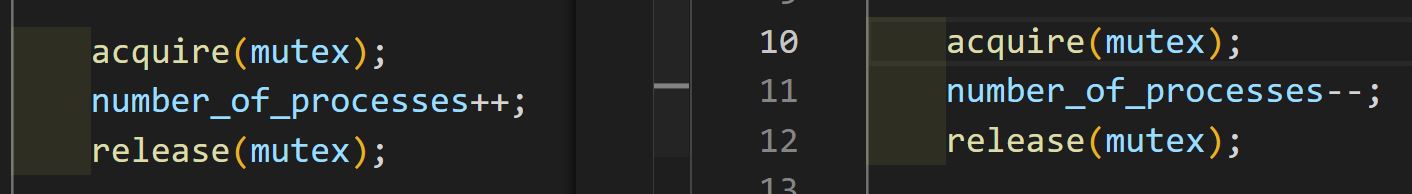
به نام خدا

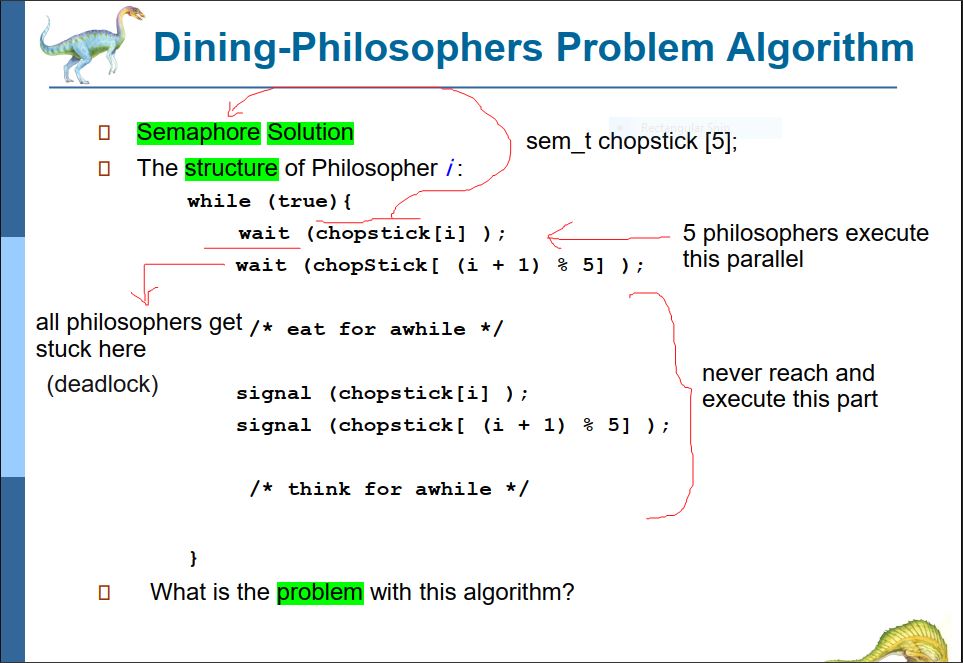
# فرزان رحمانی – 99521271 تمرین سوم تئوری سیستم عامل

1. خیر mutual exclusion برقرار نیست. در زیر مثالی آمده است که نقض انحصار متقابل را نشان می دهد.  
   تعریف انحصار متقابل:  
     
     
     
   ترتیب اجرای فرایند ها در مثال نقض انحصار متقابل:  
     
     
   در شکل بالا هر دو فرایند وارد وارد ناحیه بحرانی شده اند و با توجه به تعریف این شرط رعایت نمی شود و mutual exclusion برقرار نیست.
2. خیر شرط دوم و سوم را ندارد.  
     
   تعاریف شرط ها :  
     
     
   فرایند P0 اجرا می شود و داخل ناحیه بحرانی می رود و دوباره نوبت را به خودش می دهد. به همین دلیل برای همیشه فرایند P1 در قسمت  
    while(turn != 1); /\* Do nothing and wait \*/ می ماند و هیچ گاه از این قسمت عبور نمی کند و پیشرفتی ندارد.  
   بنابراین:  
   این راه حل شرط انحصار متقابل (Mutual Exclusion) را ارضا می کند چرا که هیچ وقت هر دو فرایند وارد ناحیه بحرانی نمی شوند چرا که while(turn != 1); اجازه ورد به ناحیه بحرانی را به فرایند P1 نمی دهد و هر دو فرایند با هم وارد ناحیه بحرانی نمی شوند.  
   شرط دوم یعنی پیشرفت (Progress) را رعایت نمی کند. چرا که فرایند P1 هیچ وقت از قسمت  
    while(turn != 1); /\* Do nothing and wait \*/ عبور نمی کند و پیشرفت نمیکند و حتی اگر بینهایت منتظر بمانیم و P0 از ناحیه بحرانی خارج شود باز هم پیشرفت نخواهد کرد.  
   شرط سوم (Bounded Waiting) هم برآورده نمی شود. چرا که فرایند P1 درخواست می دهد که وارد شود ولی به طور نامحدود منتظر می ماند. سرعت فرایند P1 صفر می شود و هیچ گاه وارد ناحیه بحرانی نمی شود. بذا طبق تعریف این شرط هم ارضا نمی شود.  
   
3. الف ) زمانی که هر فرآیندی سعی می‌کند دستورات زیر را اجرا کند، شرط مسابقه  
   (race condition) ایجاد می‌شود.   
   ++number\_of\_processes;  
   --number\_of\_processes;  
   در زبان اسمبلی این دستورات در 3 مرحله مختلف پیاده سازی می شود:  
     
     
   بیش از یک فرآیند می‌تواند کد بالا را به صورت درهم‌پیچیده(interleaved) اجرا کند و در نتیجه شرایط مسابقه ایجاد شود.  
     
   ب) برای جلوگیری از شرایط مسابقه از کار های زیر را انجام می دهیم:  
   قفل acquire() برای mutex زمانی استفاده خواهد شد که فرآیند سعی در انجام عملیاتهای ++number\_of\_processes; و --number\_of\_processes; داشته باشد و بعد از انجام آن از release() برای آزاد کردن قفل استفاده میکنیم.  
     
     
   ج) بله، atomic integer variable می تواند برای جلوگیری از شرایط مسابقه استفاده شود که در آن زمان فقط یک فرآیند می تواند number\_of\_processes را افزایش یا کاهش دهد. چرا که یکی از ابزار های synchronization است.

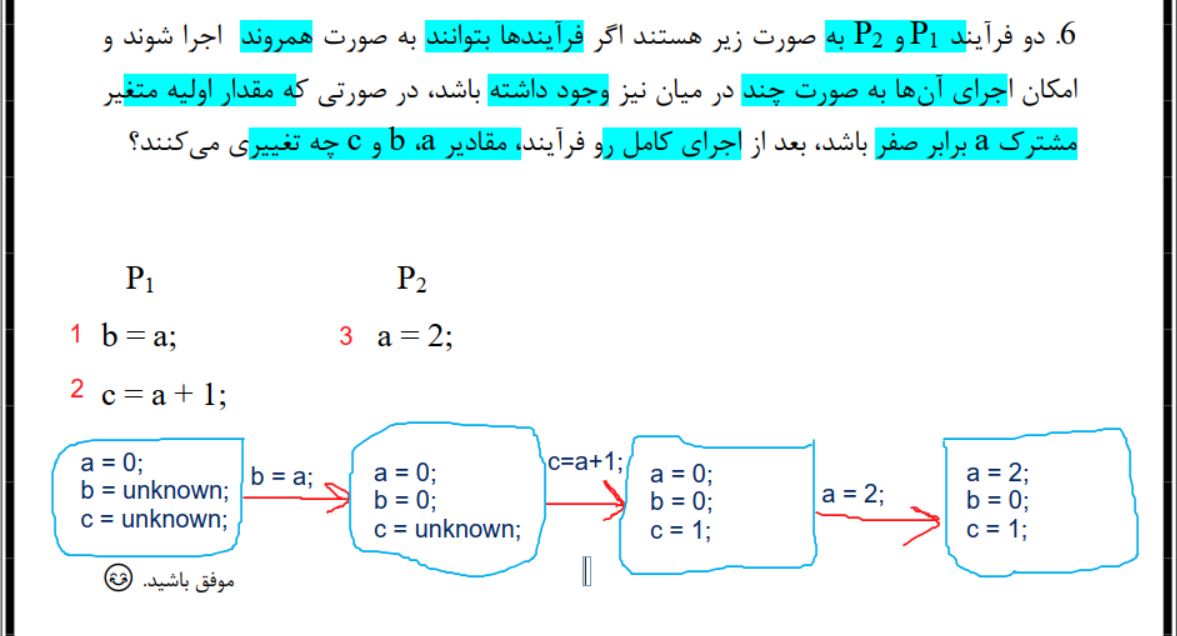
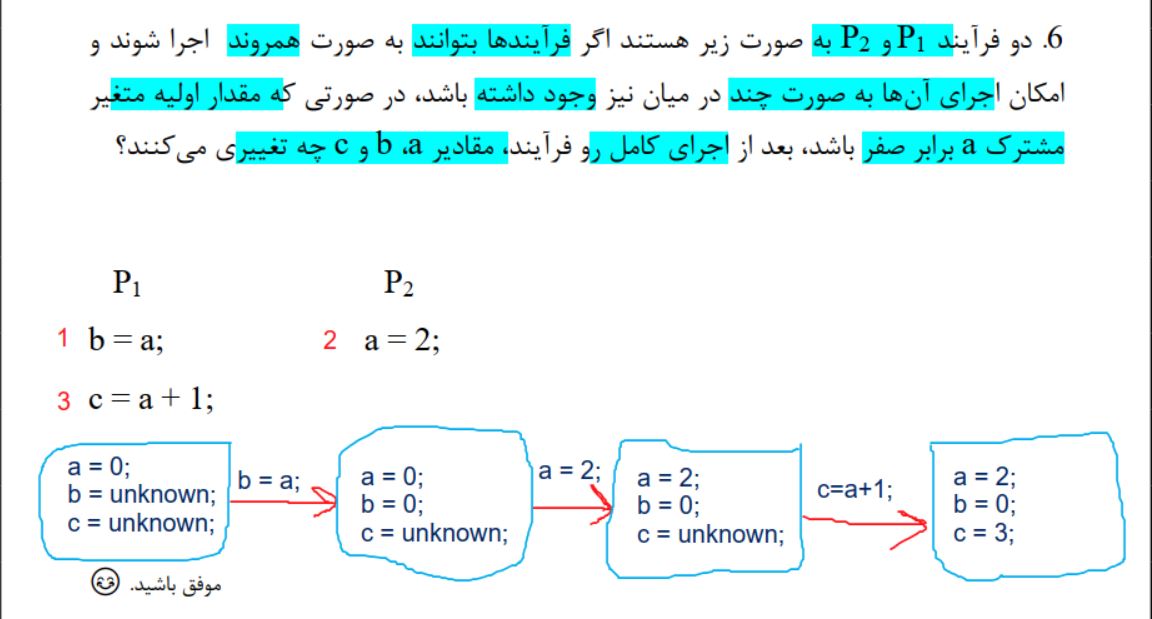
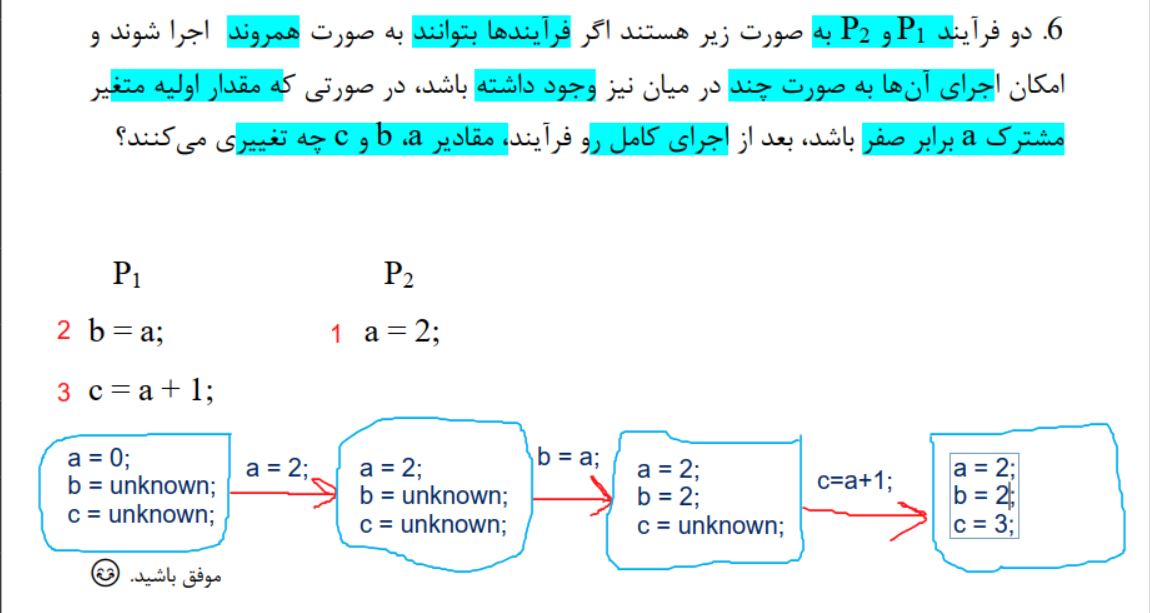
One tool is an **atomic variable** that provides atomic (uninterruptible) updates on basic data types such as integers and booleans.  
For example, the **increment()** operation on the atomic variable **sequence** ensures **sequence** is incremented without interruption:  
**increment(&sequence);**

1. بستگی به نحوه پیاده سازی و اجرای وقفه ها دارد، اما صرف نظر از نحوه پیاده سازی آن، انتخاب ضعیفی از تکنیک های موجود برای اجرای اصول synchronization در سیستم های چند پردازنده ای است.  
   مورد 1 - وقفه ها(interrupts) فقط برای یک پردازنده(processor) غیرفعال هستند - نتیجه این است که رشته‌ (thread) هایی که روی پردازنده ‌های دیگر اجرا می‌شوند می‌توانند اصول همگام سازی (synchronization) را نادیده بگیرند و به داده‌های مشترک دسترسی پیدا کنند.(access the shared data)  
   مورد 2 - وقفه‌ها(interrupts) برای همه پردازنده‌ها(processors) غیرفعال هستند - این به این معنی است که ارسال کار (task dispatching) ، مدیریت تکمیل I/O   
   (handling I/O completion) و غیره نیز در همه پردازنده‌ها غیرفعال است، بنابراین رشته‌ (thread)های در حال اجرا بر روی همه پردازنده‌ها هستند می‌توانند تا متوقف شوند و عملا دیگر سیستم چند پردازنده ای نیست.  
   و همچنین می توان گفت:

Interrupts are not sufficient in multiprocessor systems since disabling interrupts only prevents other processes from executing on the processor in which interrupts were disabled; there are no limitations on what processes could be executing on other processors and therefore the process disabling interrupts cannot guarantee mutually exclusive access to program state.

1. همان طور که در شکل زیر آمده است اگر بخواهیم این مسئله را با سمافور ها حل کنیم به deadlock ممکن است برسیم. چرا که اگر 5 فیلسوف داشته باشیم و هر 5 فیلسوف خط  
    wait (chopstick[i] ) را اجرا کنند هر 5 chopstick در گرو یکی از آنها هست و کس دیگری نمی تواند آن را بگیرد. و هر 5 فیلسوف به طور چرخشی به منابع همدیگر نیاز دارند و این دور باعث deadlock می شود همگی در خط wait (chopstick[ ( i + 5) % 5 ] ) گیر می کنند.  
     
     
   همچنین توضیحات بیشتر در زیر آمده است:

The philosophers never speak to each other, which creates a dangerous possibility of deadlock. Deadlock could occur if every philosopher holds a left chopstick and waits perpetually for a right chopstick (or vice versa). Originally used as a means of illustrating the problem of deadlock, this system reaches deadlock when there is a ’cycle of unwarranted requests’. In this case philosopher P1 waits for the chopstick grabbed by philosopher P2 who is waiting for the chopstick of philosopher P3 and so forth, making a circular chain.  
  
The lack of available chopsticks is an analogy to the locking of shared resources in real computer programming. Locking a resource is a common technique to ensure the resource is accessed by only one program or chunk of code at a time. The challenge occurs when there are multiple resources which must be acquired individually. When several programs are involved in locking multiple resources, deadlock can occur. For example, one program needs two files to process. When two such programs lock one file each, both programs wait for the other one to unlock the other file, which will never happen.

1. چون فرایند ها می توانند همروند اجرا شوند 3 حالت مختلف ممکن است رخ بدهد و ترتیب اجرای دستور ها در مقدار متغیر ها تاثیر گذار است. لذا مقادیر مختلفی داریم.  
     
   در حالت اول همان طور که در شکل زیر آمده است a = 2 , b=0 , c=1 می شوند.  
     
     
     
     
     
     
     
     
     
     
     
     
   در حالت دوم همان طور که در شکل زیر آمده است a = 2 , b=0 , c=3 می شوند.  
     
     
   در حالت سوم همان طور که در شکل زیر آمده است a = 2 , b=2 , c=3 می شوند.  
   

پایان