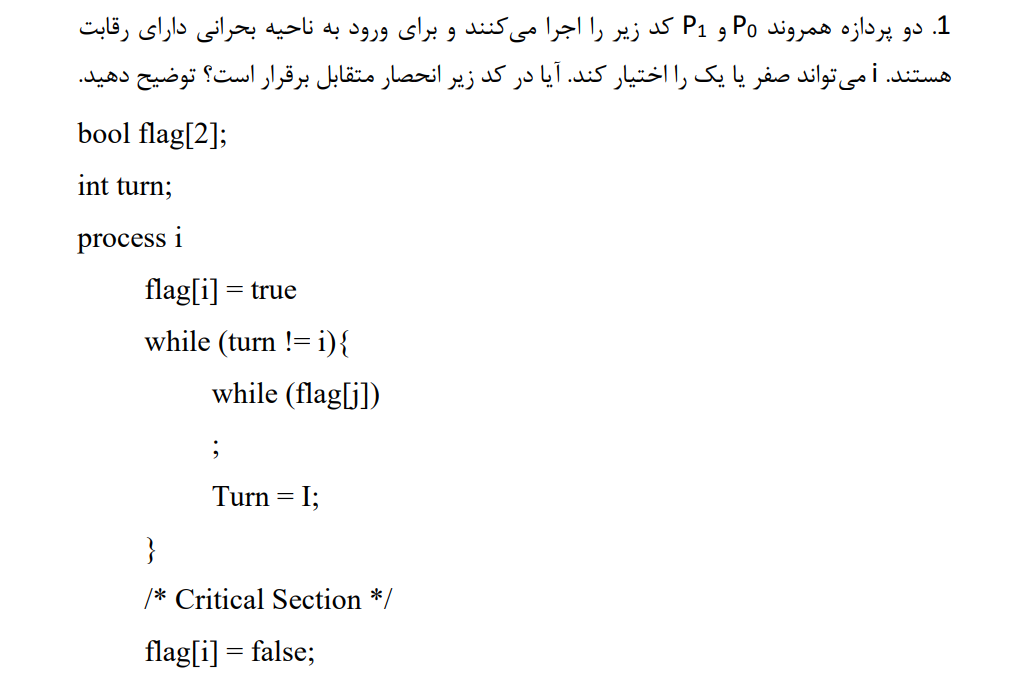
امیرعلی فرازمند

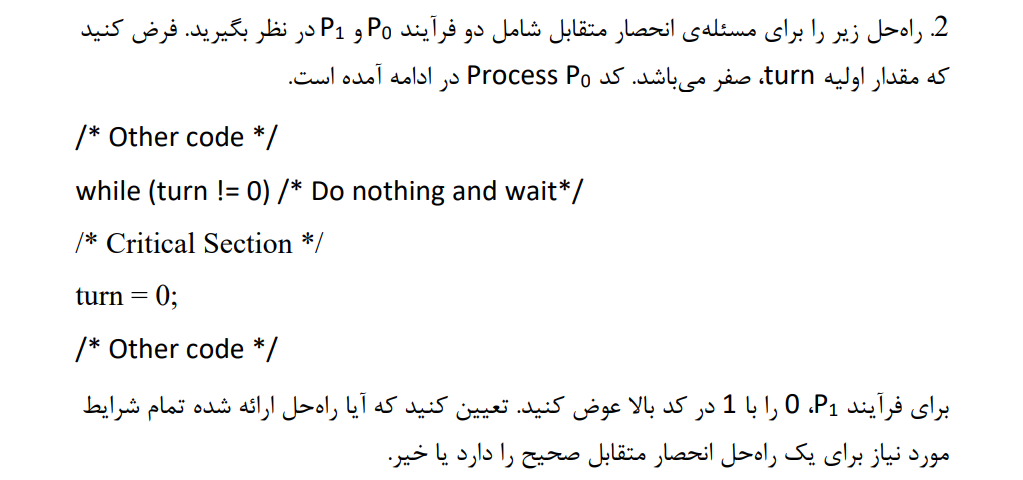
99522329

تمارین تئوری فصل 6 O.S.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| turn | Flag [1] | Flag [0] | P1 | po |
| 0 | 0 | 0 | فرض مقادیر اولیه |  |
| 0 | 1 | 1 | هر دو به لوپ اول میرسند |  |
| 0 | 1 | 1 | داخل لوپ اول میرود، وارد لوپ دوم هم میشود و انجا باید wait کند | وارد لوپ نمیشود و به critical-section میرود |
| 0 | 1 | 0 |  | از critical-section خارج میشود |
| 0 | 1 | 0 | از لوپ دوم خارج میشه(هنوز turnرو تغیر نداده) |  |
| 0 | 1 | 1 |  | پراسس 0 همین لحظه دوباره اجرا شده و flagاش را 1 میکند،به داخل لوپ اول نمیرود و دوباره وارد ناحیه بحرانی میشود. |
| 1 | 1 | 1 | خط بعد لوپ دوم turn را تغیر میدهد |  |
| 1 | 1 | 1 | از لوپ اول می اید بیرون و به ناحیه بحرانی میرود | همچنان در ناحیه بحرانی هست |

در ردیف آخر همانطور که دیدید 2 پراسس همزمان داخل critical-section هستند و این به این معنی هست که mutual-exclusion در این الگوریتم رعایت نمیشود.



|  |  |  |
| --- | --- | --- |
| Turn | P1 | Po |
| 0 | وارد critical section میشود. | وارد لوپ میشود و wait میخورد |
| 1 | از ناحیه بحرانی خارج میشود و turn را 1 میکند |  |
| 1 | وارد لوپ میشود و wait میخورد | وارد critical section میشود. |
| 0 |  | از ناحیه بحرانی خارج میشود و turn را 0 میکند |

وارد حالت اولیه میشویم...

بررسی 3 شرط:

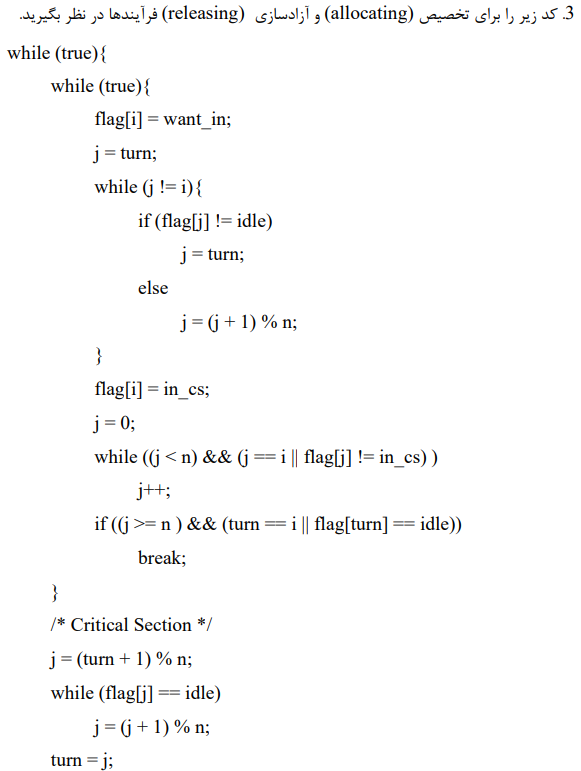
Mutual exclusion:

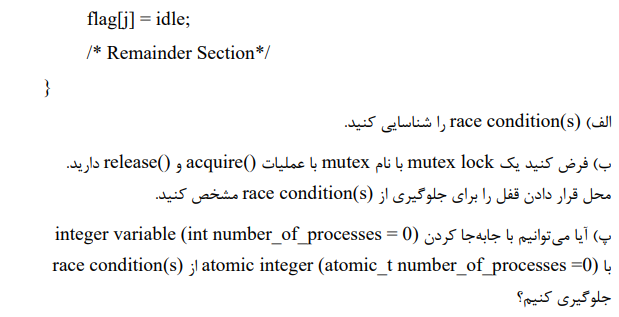
وقتی یکی از پراسس ها داخل ناحیه بحرانی باشه turn نمیگذاره آن یکی داخل ناحیه بحرانی اش بشود . متغیر turn این را تعیین میکنه که کدام داخل لوپ wait بخوره.

با فرض اینکه برای 2 پراسس عمل میکنیم این شرط برآورده میشود و نه برای n پراسس که معلومه که نمیتونه. 😐

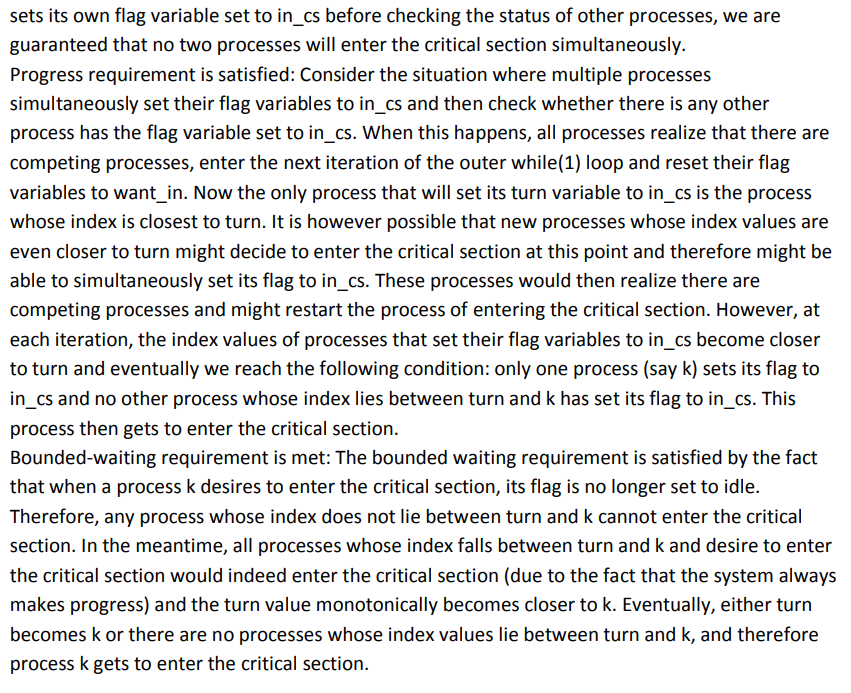
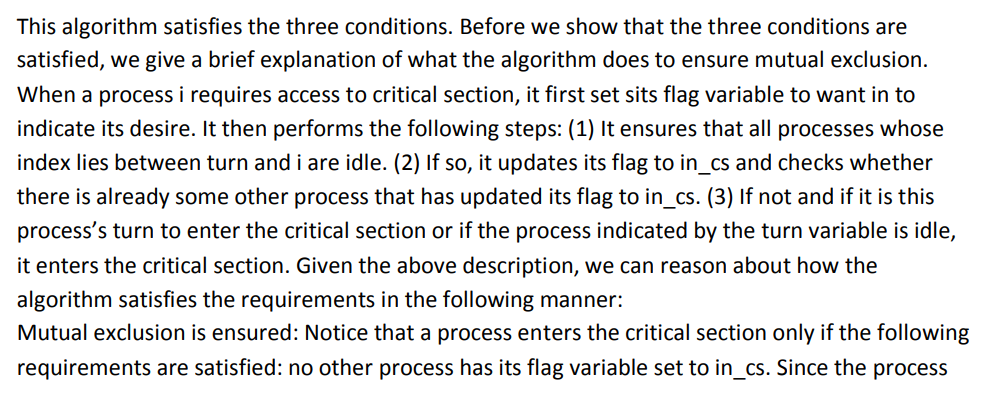
Progress & Bounded-waiting:

وقتی p0 داخل ناحیه بحرانی هست وturn همپنان 0 هست اگر قبل اینکه مقدار turn را 1 کنیم ،p0 فراخوانی شود بخاطر اینکه turn همچنان مقدارش عوض نشده است ،دوباره وارد ناحیه بحرانی میشود و این موضوع هم میتواند تا ابد ادامه پیدا کند ، پس این 2 شرط را برآورده نمیشوند.





الف)race condition رخ نمیدهد و 3 شرط را هم برآورده میکند:



[منبع جواب](https://personal.cis.strath.ac.uk/sotirios.terzis/classes/CS.304/Remaining%20Contemplation%20Questions.pdf)(سوال 2ش میشود)

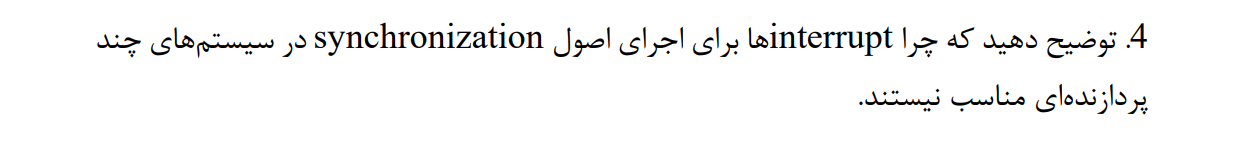
ب) دو خط

turn =j;

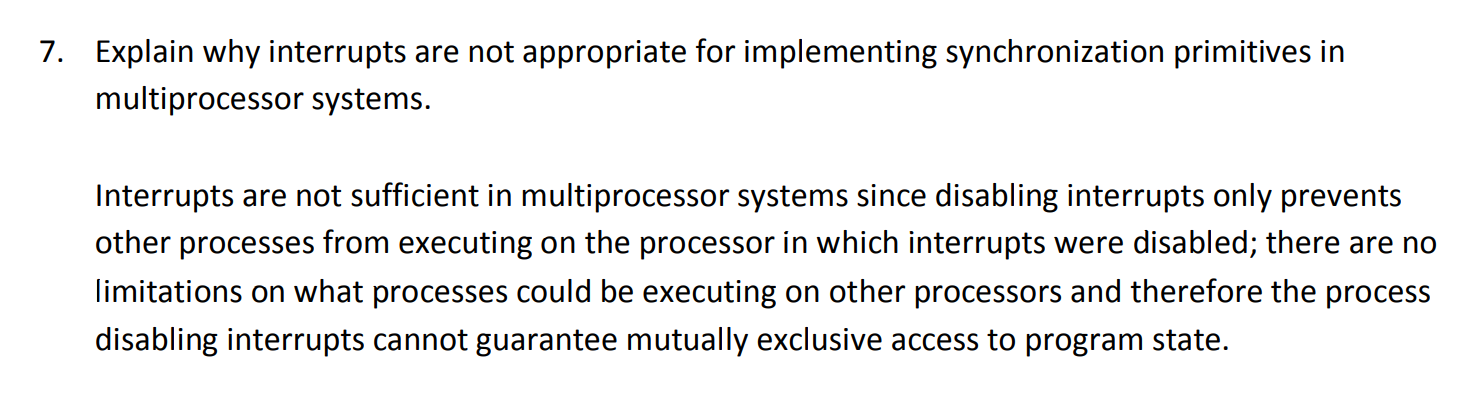
flag[j] = idle;

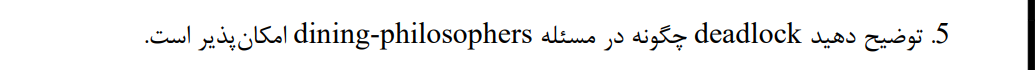
در خط های آخر برنامه را با mutex lock کنترل میکنیم که اگر 2 پراسس به آنجا رسیدند به race-condition برنخورند.

پ) چون ما 2 متغیر را در 2 خط مختلف باعث race-condition میشوند ، استفاده از متغیر های اتمی فایده ای ندارد و بهتر است از یکی از انواع لاک استفاده شود.



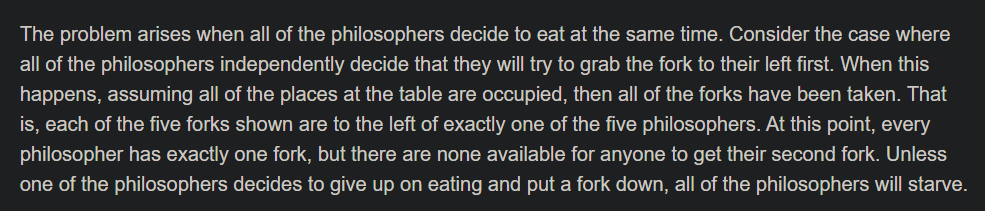
مشکل ما در synchronization این بود که پراسس ها به resource های یکسانی دسترسی دارند و تغیر دادن در آنها وقتی که همزمان اجرا میشوند میتواند باعث race -condition شود واسه همین پراسس ها باید بعضی وقتا صبر کنند تا کار پراسس دیگه ای روی دیتای مشترک تمام بشه. در سیستم های مالتی پروسسور وقتی روی پراسس interrupt بزنیم روی پروسسور دیگه ای میتونه اجرا کنه و وقفه ای به اون شکل ایجاد نمیشه که بتونه شرط mutual-exclusion بین 2 پراسس برآورده کنه.

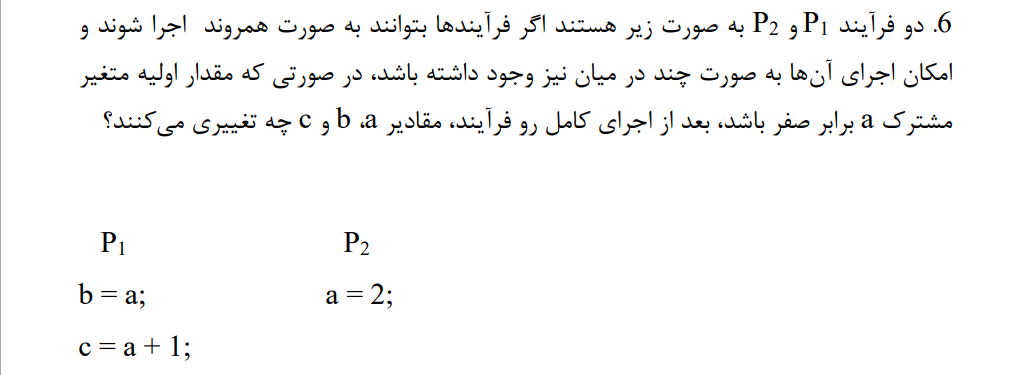




هنگامی که همه ی فیلسوف ها باهم همزمان تصمیم به غذا خوردن بگیرند، هرکدامشان چاپ استیک یکطرفش را بردارد(مثلا همه سمت چپیشان را برمیدارند). اینطور هر چاپ استیک دست یک نفر هست و همه منتظر میمانند تا آن یکی آزاد شود و آنرا بردارند که همچین چیزی رخ نمیدهد ، پس ددلاک رخ میدهد.

[جواب کامل تر](https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/DiningPhil.html):





a was initialized to 0;

All 3 possible cases:

1)

P2

P1(line1)

P1(line2)

* a = 2, b = 2, c = 2

2)

P1(line1)

P2

P1(line2)

* a = 2, b = 0, c = 3

3)

P1(line1)

P1(line2)

P2

* a = 2, b = 0, c = 1