تمرین سری پنجم اصول سیستم‌های عامل **یاسمن امی (۹۶۱۳۰۰۵)**

۷.۵) حالات رقابتی در میان بسیاری از سیستم‌های کامپیوتری امکان‌پذیر است. یک سیستم بانکی که موجودی حساب را با دو تابع نگه میدارد در نظر بگیرید: deposit(amount) و withdraw(amount) .برای این توابع مقداری ارسال می‌شود که به حساب گذاشته یا برداشته شود. فرض کنید یک زن و شوهر حساب بانکی مشترکی دارند. به طور همروند، شوهر تابع withdraw() را فرا می‌خواند و همسر deposit() را فراخوانی می‌کند. توضیح دهید که چگونه یک حالت رقابت ممکن است رخ دهد و برای جلوگیری از رخ دادن رقابت چه باید کرد؟
فرض کنید مقدار ۲۰۰ در حساب بانکی موجود است و همسر تابع deposit(100) و شوهر تابع withdraw(50) را فراخوانی کنند. قاعدتا مقدار نهایی حساب باید ۲۵۰ شود. چون این دو فرایند به ترتیب انجام می‌شوند قبل از آن‌که شوهر بتواند عملیاتش را انجام دهد، فرایند واریز صورت می‌گیرد و مقدار موجودی محلی حساب را برای او به مقدار ۳۵۰ به‌روز می‌کند که غلط است.

۸.۵) اولین راه حل نرم افزاری صحیح شناخته شده در مسئله منطقه بحرانی برای دو فرایند به وسیله ذکر ارائه شد. دو فرایند P0 و P1 متغیرهای زیر را به اشتراک دارند:

```
boolean flag[2];  /*initially false */
int turn;
```

ساختار فرایند Pi (i=0 یا 1)در شکل ۵.۲۱ نشان داده شده است. فرایند دیگر Pj(i=0 یا 1) می‌باشد. ثابت کنید که الگوریتم همه نیازهای سه‌گانه را برای مسئله منطقه بحرانی برآورده میکند.
به ترتیب نیازهای مسئله‌ی منطقه بحرانی را بررسی می‌کنیم.
(۱) محرومیت متقابل از طریق استفاده از متغیرهای flag و turn تضمین می شود. اگر هر دو فرآیند flag خود را true کنند، تنها یک مورد موفق خواهد بود. یعنی فرآیندی که نوبت آن است. فرآیند منتظر فقط می‌تواند وارد منطقه بحرانی شود تا وقتی که فرایند دیگر مقدار turn را به‌روز کند.
(۲) پیشرفت دوباره از طریق متغیرهای flag و turn ارائه می‌شود. این الگوریتم جایگزین دقیق را ارائه نمی‌دهد. بلکه، اگر این فرآیند می‌خواهد به منطقه بحرانی خود دسترسی پیدا کند ، می تواند flag خود را true کند و وارد منطقه بحرانی خود شود. این فرایند تنها پس از خروج از منطقه بحرانی خود، می‌تواند مقدار turn را فرایند دیگر قرار دهد. اگر این فرآیند بخواهد دوباره -قبل از فرآیند دیگر- وارد منطقه بحرانی خود شود روند ورود به منطقه بحرانی خود را تکرار می‌کند و قبل از خروج مقدار turn را به فرایند دیگری می‌دهد.
(۳) انتظار محدود شده از طریق استفاده از متغیر TTturn حفظ می‌شود. فرض کنید دو فرآیند مایل به ورود به منطقه بحرانی مربوط به خود هستند. هر دو مقدار flag خود را به true تنظیم می‌کنند، با این وجود فقط نخی که نوبت آن است می‌تواند ادامه یابد و نخ دیگر منتظر است. اگر انتظار محدود نشده باشد، ممکن است که فرآیند منتظر به‌طور نامحدود منتظر بماند در حالی که اولین فرآیند بارها و بارها وارد بخش بحرانی آن می شود. با این حال، الگوریتم Dekker یک فرآیند را دارد که مقدار turn را با فرآیند دیگر تعیین می‌کند و اطمینان می‌دهد که فرایند دیگر نیز وارد منطقه بحرانی خود خواهد شد.

۵.۱۶) پیاده‌سازی قفل‌های انحصاری که در بخش ۵.۵ ارائه شده از انتظار مشغولی رنج می‌برد. چه تغییراتی لازم است تا یک فرایند منتظر در بدست آوردن یک قفل انحصاری مسدود شده و در صف انتظار بماند تا قفل آزاد شود.
همراه با هر قفل انحصاری یک صف شامل فرآیندهای منتظر خواهد بود. هنگامی که یک فرآیند تعیین می‌کند که قفل در دسترس نیست ، آن‌ها در صف قرار می‌گیرند. هنگامی که یک فرآیند قفل را آزاد می‌کند ، اولین فرآیند را از لیست فرآیندهای منتظر حذف و بیدار می کند تا اجرا شود.

۵.۲۰) نمونه کد برای تخصیص و رهاسازی فرایندها در شکل ۵.۲۳ را در نظر بگیرید.
الف. حالات رقابت را شناسایی کنید.
روی متغیرnumber_of_processess حالت رقابت وجود دارد.

ب. فرض کنید یک قفل انحصاری به نام mutex با عملیات ()acquire و ()release دارید. مشخص کنید که قفل‌گذاری کجا قرار گیرد تا مانع حالات رقابتی شود.
قبل از ورود به هر تابع باید ()acquire و درست قبل از خروج از تابع باید ()release صدا زده شود.

پ. آیا میتوانیم متغیر صحیح زیر را

```
int number_of_processess = 0
```

با متغیر اتمیک

```
atomic_t number_of_processess = 0
```

برای جلوگیری از رقابت جایگزین نماییم؟
خیر. دلیل آن این است که رقابت در تابع ()allocate_process رخ می‌دهد که در شرط if اول مقدار number_of_processes چک می‌شود و بعد با توجه به مقدار نتیجه به‌روز می‌شود. ممکن است در یک مرحله مقدار number_of_processes در هنگام چک‌کردن برابر ۲۵۴ باشد، ولی به‌خاطر رقابت موجود توسط نخ دیگری قبل از زیاد شدن دوباره به ۲۵۵ تغییر داده شده باشد.