حل تمرین هفتم درس سیستم‌های عامل

دکتر زرندی

پاییز ۰۰

۱. الف) یک سیستم مدیریت حساب بانکی را تصور کنید. این سیستم با فراهم کردن دو تابع *add()* و *withdraw()*، به مشتریان خود امکان افزایش موجودی و برداشت از حساب را می‌دهد. فرض کنید دو شخص A و B یک حساب مشترک دارند و می‌توانند در هر لحظه مبلغی را به حساب خود اضافه یا از آن برداشت کنند. در چنین سیستمی چگونه ممکن است شرایط رقابت *(race condition)* ایجاد شود؟ مثال بزنید.
ب) راه حل‌های پیشنهادی برای حل مشکل بخش (الف) باید دارای چه ویژگی‌هایی باشند؟ هر کدام را مختصرا شرح دهید.

الف) بله. فرض کنید موجودی حساب ۵ هزار تومان باشد. شخص A می‌خواهد ۲ هزار تومان به حساب اضافه کند. ۲ با ۵ هزار تومان جمع شده و حساب نهایی ۷ هزار تومان می‌شود. پیش از آنکه نتیجه نهایی ثبت شود، B هزار تومان از حساب برداشت می‌کند و نتیجه نهایی آن می‌شود ۴ هزار تومان. پس از آن موجودی به ۷ هزار تومان و سپس به ۴ هزار تومان تغییر می‌کند. در این صورت مبلغ نهایی موجود در حساب بر اثر race condition دچار مشکل می‌شود.

ب)

۱. انحصار متقابل: درصورتی که فرایند A در منطقه بحرانی خود است هیچ فرایند دیگر حق ورود به منطقه بحرانی خود را ندارد.
۲. پیشرفت: درصورتی که هیچ فرایند در منطقه بحرانی خود نباشد، تنها آن دسته از فرایندهایی که هنوز به منطقه بحرانی خود نرسیده‌اند می‌توانند تصمیم بگیرند کدام فرایند وارد ناحیه بحرانی خود شود.
۳. انتظار محدود: هیچ فرایندی نباید به‌طور نامحدود منتظر ورود به ناحیه بحرانی خود بماند.

۲. برای هریک از دو الگوریتم زیر شروط انحصار متقابل، پیشرفت و انتظار محدود را بررسی کنید.

انحصار متقابل: فرض می‌کنیم ابتدا به فرایند j نوبت می‌رسد. شرط while در آن چک شده و نمی‌تواند وارد منطقه بحرانی خود می شود. حالا در سمت فرایند دیگر نوبت به i می‌رسد. شرط while چک شده و اینبار فرایند j نیز نمی‌تواند وارد منطقه بحرانی خود شود. در سمت فرایند i اینبار نوبت i است و بنابراین شرط برقرار نشده و وارد منطقه بحرانی خود می‌شود. تا زمانی که از منطقه بحرانی خود خارج شود فرایند دیگر نمی‌تواند وارد شود و بنابراین شرط انحصار متقابل برقرار است.

پیشرفت: پیشرفت ندارد چون درصورت عدم وجود یک فرایند، فرایند دیگر نمی‌تواند وارد شود.

انتظار محدود: وجود ندارد. چرا که فرایندی که خارج می‌شود تنها نوبت خود را برمی‌دارد و دیگر نوبت را به فرایند دیگر تعارف نمی‌کند.

```
do{
    flag[i] = true;
    turn = j;
    while(flag[i] && turn==j);
    // CRITICAL SECTION //
    flag[i] = false;
    // REMAINDER SECTION //
} While(true)
```

انحصار متقابل: فرض می‌کنیم ابتدا به فرایند j نوبت می‌رسد. شرط while در آن چک شده و وارد منطقه بحرانی می‌شود. در فرایند دیگر از خط اول شروع به اجرا شده و turn را به i تغییر میدهد. شرط برقرار نیست و وارد منطقه بحرانی می‌شود. بنابراین انحصار متقابل نقض می‌شود.

پیشرفت: پیشرفت دارد چون بهرحال یک فرایند وارد می‌شود و پشت while نمی‌ماند.

انتظار محدود: وجود ندارد چون فرایند که خارج می‌شود ممکن است دوباره خود نوبت را دست بگیرد.

```
do{
    flag[i] = true;
    turn = j;
    while(flag[j]&& !turn==j);
    // CRITICAL SECTION //
    flag[i] = false;
    // REMAINDER SECTION //
} While(true)
```

۳. قطعه کد زیر را در نظر بگیرید، در این تابع قصد داریم عملیات ضرب بین مقدار *op1* و مقداری که *P_op2* به آن اشاره دارد را انجام دهیم. حاصل ضرب باید در حافظه‌ای که *P_op2* به آن اشاره می‌کند ذخیره شود. این تابع را طوری با دستور *compare-and-swap* کامل کنید که عملیات ضرب به صورت اتمی انجام شود. همچنین برقرار بودن یا نبودن هر کدام از شروط سه‌گانه را با دلیل شرح دهید.

```
int Multiplication(int op1, int *P_op2)
    bool flag = false;
    int value;
    while(!flag) {
        value = *P_op2;
        flag = compare_and_swap (P_op2, value, value * op1);
     }
      return  *P_op2   *  op1;
```

شروط انحصار متقابل و پیشرفت برقرار هستند ولی شرط انتظار محدود در اینجا برقرار نمی باشد. زیرا ممکن است همیشه فرآیندهای دیگر به طور همزمان در حال تغییر مقدار خانه P_op2 باشند و در اینصورت هیچگاه فرصت انجام عملیات ضرب به صورت اتمی فراهم نخواهد شد.

۳.الف) پیاده‌سازی‌های سخت‌افزاری زیادی برای دستور test-and-set وجود دارد. دو مورد از آن‌ها را شرح داده و به امکانات سخت‌افزاری مورد استفاده در هر کدام از آن‌ها اشاره کنید.
ب) پس از ارائه ویندوز 8.1، برخی از پردازنده‌ها به دلیل عدم پشتیبانی از یک دستور مشابه با دستور compare-and-swap، در اجرای این نسخه از ویندوز دچار مشکل شدند. نام این دستور چیست و چه کاری انجام می‌دهد؟

پیاده‌سازی نوع ۱:
فرض کنید که پردازنده ۱ درخواست استفاده از test-and-set دارد. در این هنگام DPRAM آدرس حافظه مورد نظر را جایی ذخیره مي‌کند(اصطلاحا یک نوت برای خود تهیه می‌کند). سپس اگر پردازنده ۲ نیز درخواست test-and-set برای همان مکان از حافظه را داشته باشد، DPRAM با بررسی آدرسی که پیشتر ذخیره کرده است، وضعیت را شناسایی کرده و به پردازنده ۲ یک وقفه مشغول می‌دهد که یعنی پردازنده ۲ باید منتظر بماند و منجر به انتظار مشغول پردازنده ۲ می‌شود.سپس DPRAM تست را انجام می‌دهد و درصورت قبولی مقدار را به‌روز رسانی می‌کند

پیاده‌سازی نوع ۲:

در این حالت اگر پردازنده ۱ خواستار test-and-set برای نوشتن در خانه‌ای از حافظه باشد، DPRAM ابتدا مقدار فعلی را در ثبات مخصوصی ذخیره می‌کند و محتوای آن را به پرچم مخصوصی تنظیم می‌کند. سپس در صورتی که پردازنده ۲ درخواست test-and-set به همان خانه داشته باشد، DPRAM وضعیت پرچم حافظه را دیده و وقفه مشغول می‌دهد. سپس DPRAM تست را انجام می‌دهد و درصورت قبولی مقدار را به‌روز رسانی می‌کند، و در غیر اینصورت مقدار قبلی را از ثبات گرفته و دوباره در خانه حافظه کپی می‌کند.

Double-wide compare-and-swap

روی دو خانه از حافظه همسایه عمل می‌کند. در پردازنده‌های x86 به بعد دستورات CMPXCHG8B و CMPXCHG16B این نقش را دارند و پردازنده های ۶۴ بیتی AMD دیگر از آن پشتیبانی نمی‌کنند. و این مشکل در ویندوز ۸.۱ به بعد خود را نشان داد چرا که نیازمند پشتیبانی سخت افزاری CMPXCHG8B بود.