حل تمرین هشتم درس سیستم‌های عامل

دکتر زرندی

پاییز ۰۰

2- سه فرآیند همروند زیر را در نظر بگیرید که دارای سه سمافور باینری هستند. سمافورها به صورت S2=0، S0=0، S1=1 مقداردهی اولیه شده‌اند.

فرایند P1 چند بار عبارت "Hello world" را چاپ خواهد کرد؟

Process P2	Process P1	Process P0
wait(S2); release (S1);	while(True) { wait (S1); print ' Hello world '; release (S0); release (S2); }	wait(S0); release (S1);

در ابتدا P1 اجرا می شود زیرا فقط S1 برابر 1 است. پس یکبار **"Hello world"** چاپ خواهد شد.

سپس وقتی S0 و S2 توسط P1 releases , می شوند و هر یک از P0 و P2 می توانند اجرا شوند.

فرض کنیم P0 اجرا شده و S1 را release می کند (اکنون مقدار S1 ، 1 است).

اکنون دو احتمال وجود دارد که P1 یا P2 می توانند اجرا شوند.

فرض کنیم P2 اجرا و release ، S1 شود. بنابراین در پایان P1 اجرا شده و چاپ **"Hello world"** انجام خواهد شد (به معنی دو **"Hello world"**).
اما اگر P1 قبل از P2 اجرا شود، در مجموع 3 چاپ **"Hello world"** انجام خواهد شد (یکی در زمان اجرای P1 و سپس اجرای P2 که S1 را آزاد می کند و سپس P1 دوباره اجرا می شود).

بنابراین پاسخ کامل حداقل دو **"Hello world"**است.

۱. الف) یک فراخوانی دستورات acquire() و release() در فقل mutex باید به صورت اتمی انجام شود و به همین دلیل معمولا با استفاده از روش‌های اتمی سخت‌افزاری پیاده‌سازی می‌شوند. شما برای این کار از دستور compare_and_swap() استفاده کنید. دقت کنید که پیاده‌سازی شما نباید در هیچ حالتی دچار بن‌بست (deadlock) شود.

```
int compare_and_swap(int *value, int expected, int new_value) {
    int temp = *value;
    if (*value == expected)
    *value = new_value;
    return temp;     /* old value */
}
```

پاسخ:

```
int lock = 0;
int available = 1;
int acquire() {
        while (compare_and_swap( &available, 1, 0) != 1);
}
int release() {
        while (compare_and_swap(&lock, 0, 1) != 0);
        available = 1;
        lock = 0;
}
```

3- در مورد دستورات سخت‌افزاری حل ناحیه بحرانی Test_and_Set و Compare_and_Swap به سوالات زیر پاسخ دهید:

الف) چطور دستور Test_and_Set را می‌توان با استفاده از Compare_and_Swap پیاده‌سازی کرد؟

ب) هنگام استفاده از هر یک در زبان‌های سطح بالا (مثل C) داخل حلقه while قبل از ناحیه بحرانی، آیا اتمیک بودن دستورات while و دیگر دستورات سطح بالا مهم نیست؟ چرا؟

ج) چطور در سیستم‌های تک هسته‌ای (تک program counter)، اتمیک بودن دستورات فوق، مشکل ناحیه بحرانی را حل می‌کند (به عبارتی توضیح دهید چرا حتی با تعویض متن، شرط مسابقه و ناسازگاری پیش نخواهد آمد و چطور انحصار متقابل تامین می‌شود)؟

د) در سیستم‌های چند هسته‌ای یا چند پردازنده‌ای که دارای چندین سخت‌افزار موازی من جمله program counter هستند، آیا دستورات فوق مشکل‌ساز نیستند؟ آیا شرایطی پیش نمی‌آید که دو فرآیند در دو هسته بصورت موازی (مثالی از همروندی) اجرا شوند طوریکه هر دو در حین استفاده از دستور Test_and_Set (یا Compare_and_Swap) روی یک متغیر مشترک از حافظه باشند و در این حالت، شرط مسابقه و ناسازگاری پیش آید یا انحصار متقابل تامین نشود؟ بحث کنید.

الف)

```
test_&_set(int * lock){
    Return compare_&_swap(lock,0,1);
}
```

ب) در واقع دستورات و متغیرهای تاثیر گذار روی فرایند اجرای دستوراتی که قصد اجرای اتمیک آنها را داریم، برایمان مهم هستند اما بقیه خیر. زیرا با استفاده از روش های TAS و CAS که از اتمیک بودن آنها اطمینان داریم میتوانیم از اجرای اتمیک ناحیه ی بحرانی مورد نظر اطمینان حاصل کنیم و بقیه ی دستورات while اهمیتی ندارند.

ج) در سیستم های تک هسته ای از آنجایی که دستورات اتمیک هستند، وقتی زمان context switching یا وقفه برسد، اگر فرایندی در ناحیه ی بحرانی باشد تا پایان اجرای آن ناحیه ی بحرانی آن فریاند به اجرا ادامه میدهد پس شرایط مسابقه ای رخ نخواهد داد چون همچنان اتمیک بودن و فعالیت بک برنامه در ناحیه ی بحرانی حفظ خواهد شد. همچنین چون یک program counter در این شرایط وجود دارد عملا فقط یک فرایند وجود دارد در هر لحظه که میتواند ناحیه ی بحرانی را اجرا کند که آن هم کاملا اتمیک اجرا خواهد شد و انحصار متقابل وجود دارد.

د) اگر دو دستور TAS یا CAS به طور همزمان روی هسته های پردازشی متفاوت اجرا شوند، سخت افزار سیستم مدیریت این که این دو فرایند به طور همزمان وارد ناحیه ی بحرانی نشوند را میتواند انجام دهد. در واقع متغیری مشترک، مثلا به نام lock را برای این موضوع در نظر گرفته و به کمک مقادیر آن، این مدیریت را انجام میدهد در نتیجه انحصار متقابل تضمین شده و شرایط مسابقه رخ نخواهد داد. این کنترل سخت افزاری توسط ماژولی به نام DPROM انجام میشود و از این موضوع جلوگیری میکند که چند دستور TAS به داده ها دسترسی پیدا کنند.