حل تمرین هفتم درس سیستم‌های عامل

دکتر زرندی

پاییز ۹۹

۱- شروط انحصار متقابل، پیشرفت و انتظار محدود را برای الگوریتم‌های زیر بررسی کرده و دلیل خود را بنویسید.

الف)

```
do {
        flag[i] = true;
        turn = j;
        while (!flag[j] || turn = = j);
                critical section
        flag[i] = false;
                remainder section
} while (true);
```

انحصار متقابل:
وجود دارد، فرآیندها برای ورود به بخش بحرانی نیاز دارند که نوبت به آن‌ها تعارف شود.
فرض می‌کنیم فرآیند **i** زودتر به حلقه **while** رسیده است. پس از آنکه فرآیند **j** نوبت را به فرآیند **i** تعارف کرد، فرآیند **i** وارد بخش بحرانی می‌شود. اما فرآیند **i** دیگر نمی‌تواند نوبت را به فرآیند **j** تعارف کند. پس فرآیند **j** نمی‌تواند وارد شود و انحصار متقابل حفظ می‌شود.

پیشرفت:
به دلیل شرط **!flag[j]**، اگر فقط یکی از فرآیندها اجرا شود، پیشرفت نخواهیم داشت اما اگر هر دو اجرا شوند و در قسمت **remainder** دچار قفل نشوند، پیشرفت وجود خواهد داشت.

انتظار محدود:
وجود دارد، زیرا فرآیندی که از بخش بحرانی خارج شود، ناچار است دو مرتبه نوبت را به فرآیند دیگر تعارف کند. البته این مورد نیز به شرط آن است که قسمت **remainder** در زمان متناهی به اتمام برسد.

ب)

```
do {
        flag[j] = true;
        turn = j;
        while (flag[i] && turn = = j);
                critical section
        flag[j] = false;
                remainder section
} while (true);
```

انحصار متقابل:
وجود دارد، فرآیندی که نوبت را به فرآیند دیگر بدهد خود نخواهد توانست وارد بخش بحرانی شود تا هنگامی که فرآیند دیگر از بخش بحرانی خارج شود.

پیشرفت:
وجود دارد، زیرا حالتی وجود ندارد که هر دو فرآیند قادر به عبور از حلقه while نباشند.

انتظار محدود:
وجود دارد، چون به دلیل تعارف نوبت امکان ندارد که یک فرآیند برای همیشه پشت حلقه while باقی بماند.

راه حل ساده‌تر:
این الگوریتم همان الگوریتم پترسون است زیرا تنها اندیس‌های i و j در همه کاربردهای متغیر flag با هم جا به جا شده‌اند. پس همه شروط برقرار هستند.

۲- بدون استفاده از قفل و تنها با استفاده از دستور *compare-and-swap* تابع زیر را به گونه‌ای کامل کنید که به صورت اتمی عملیات جمع را انجام دهد. منظور از عملیات جمع اضافه شدن مقدار *v* به حافظه‌ای است که *p* به آن اشاره دارد. سپس توضیح دهید که تضمینی برای انجام شدن این عملیات وجود دارد یا خیر.

```
bool compare_and_swap(int *p, int old, int new)
{
        if(*p != old)
                return false;
        *p = new;
        return true;
}
```

تضمینی برای انجام این عملیات وجود ندارد. به عبارت دیگر شرط انتظار محدود در اینجا برقرار نمی‌باشد. زیرا ممکن است همیشه فرآیندهای دیگر به طور همزمان در حال تغییر مقدار خانه p باشند و در اینصورت هیچگاه فرصت انجام عملیات جمع به صورت اتمی فراهم نخواهد شد.

```
int add(int *p, int v)
{
        bool done = false;
        int value;
        while(!done) {
                value = *p;
                done = compare_and_swap(p, value, value + v);
        }
        return *p + v;
}
```

۳- می‌دانید که برای پیاده‌سازی توابع *acquire()* و *release()* در قفل *mutex* باید از دستورات سخت‌افزاری اتمی استفاده کرد. این کار را با استفاده از دستور *test-and-set* انجام دهید.

به این روش توابع acquire() و release() به صورت اتمی اجرا خواهند شد.

از آنجایی که این دو تابع تنها توابعی هستند که به متغیر available دسترسی دارند، بنابراین عملیات خواندن و نوشتن هیچگاه به طور همزمان در این متغیر انجام نخواهد شد.

برای سادگی می‌توان دستور acquire() را به صورت زیر نیز پیاده‌سازی کرد:

```
acquire() {
        while ( ! test_and_set(&available));
}
```

```
bool lock = false;
bool available = false;

acquire() {
        while (test_and_set(&lock));
        while (!available);
        available = false;
        lock = false;
}

release() {
        while (test_and_set(&lock));
        available = true;
        lock = false;
}
```