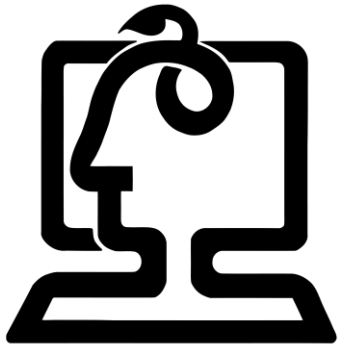
«به نام ایزد یکتا»



دانشکده مهندسی کامپیوتر

تمرین چهارم درس سیستم عامل

استاد: دکتر جوادی

تهیه کننده: بردیا اردکانیان

۹۸۳۱۰۷۲

1)

الف)

R = 9

process	used	Max
P_1	2	7
P_2	1	6
P_3	2	5
P_4	1	4

Contex of the need matrix:

```
Need [i] = Max [i] - Allocation [i]
```

```
Available = 9 – (2+1+2+1) = 3
```

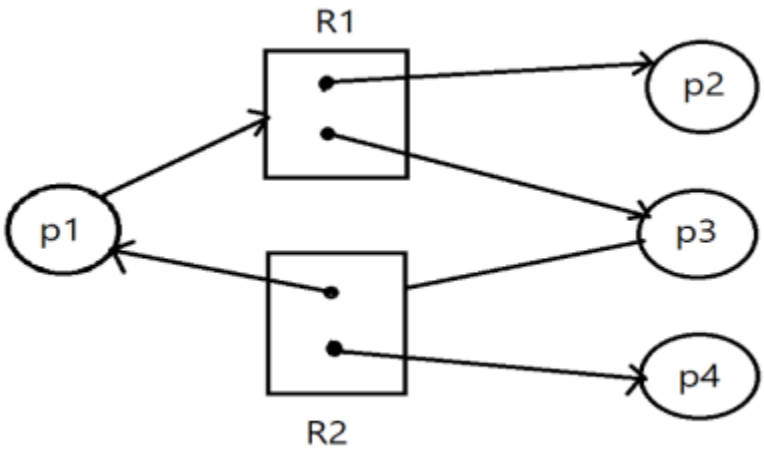
NEED
5
5
3
3

```
Apply the bankers algorithm:
```

1. P1: Need <= Available?
   is 5 <= 3? -> false
2. P2: Need <= Available?
   is 5 <= 3? -> false
3. P3: Need <= Available?
   is 3 <= 3 -> true

   New_Available = Available + Allocation
   New_Available = 3 + 2 = 5
4. P4: Need <= Available?
   is 3 <= 5 -> true
   New_Available = Available + Allocation
   New_Available = 5 + 1 = 6
5. P1 ((5 % 4)+1): Need <= Available?
   is 5 <= 6 -> true
   New_Available = Available + Allocation
   New_Available = 6 + 2 = 8
6. P2: Need <= Available?
   is 5 <= 8 -> true
   New_Available = Available + Allocation
   New_Available = 8 + 1 = 9
7. Banker’s Algorithm is finished
8. A suggested sequence will be <P3, P4, P1, P2>
9. We will not have deadlock in above sequence

ب)



1) آیا در گراف حلقه وجود دارد؟

```
P1 -> R1 -> P3 -> R2 -> P1
```

بله بین P1 و P3 حلقه وجود دارد.

2) آیا بن‌بست وجود دارد؟ خیر

در صورتی که P2 و P4 منابع R1 و R2 را آزاد کند، بن‌بست رخ نمی‌دهد و P1 و P2 وابسته به منابع هم نیستند.

2)

process	MAX A B C D	Allocation A B C D	Available A B C D
p_0	2 1 0 6	1 0 0 4	1 1 2 3
P_1	0 5 7 1	0 0 1 1	
P_2	6 5 2 3	4 5 2 1	
P_3	3 5 6 1	3 3 6 0	
P_4	6 5 6 1	2 1 2 0	

الگوریتم بانکداران را برای این پردازه‌ها اجرا می‌کنیم تا ببینیم راهی وجود دارد تا بن‌بست رخ ندهد. اگر وجود نداشته باشد پس بن‌بست می‌توانیم داشته باشیم و در حالت امن نیست.

```
<Resources> = <Allocation> + <Available>
<A, B, C, D> = <10, 9, 11, 6> + <1, 1, 2, 3>
<A, B, C, D> = <11, 10, 13, 9>
```

Contex of the need matrix:

```
Need [i] = Max [i] - Allocation [i]
```

NEED			
1	1	0	2
0	5	6	0
2	0	0	2
0	2	0	1
4	4	4	1

```
Apply the bankers algorithm: Available<1, 1, 2, 3>
```

برای اینکه جواب کوتاه‌تر باشد شرط (Need <= Available) برای تمامی پردازه‌ها نمی‌نویسم و فقط برای کاندیداهایی که این شرط true هست اعمال می‌کنم.

1. P0: Need <= Available?
   is <1, 1, 0, 2> <= <1, 1, 2, 3> -> true
   New_Available = Available + Allocation
   New_Available = <2, 1, 2, 7>
2. P2: Need <= Available?
   is <2, 0, 0, 2> <= <2, 2, 2, 4> -> true
   New_Available = <6, 6, 4, 8>
3. P3: Need <= Available?
   is <0, 2, 0, 1> <= <6, 6, 4, 8> -> true
   New_Available = <9, 9, 10, 8>
4. P4: Need <= Available?
   is <4, 4, 4, 1> <= <9, 9, 10, 8> -> true
   New_Available = <11, 10, 12, 8>
5. P1: Need <= Available?
   is <0, 5, 6, 0> <= <11, 10, 12, 8> -> true
   New_Available = <11, 10, 13, 9>
6. Bankers algorithm is finished. Available = Resources
7. A suggested sequence will be <P0, P2, P3, P4, P1>
8. We will not have deadlock in above sequence

P0 -> P2 -> P3 -> P4 -> P1

3)

الف)

Method used by P_1

```
while(true){
      while(turn!=1);
            //critical section
      turn = 2;
            //outside of critical section
}
```

Method used by P2

```
while(true){
      while(turn!=2);
            //critical section
      turn = 1;
            //outside of critical section
}
```

Mutual Exclusion:

وجود دارد.

با توجه به وجود متغیر turn هیچ دو پردازه‌ای به طور همزمان در CS نیستند.

Progress:

وجود ندارد.

زمانی که پردازه اول از CS بیرون بیاید نوبت را به دومی می‌دهد و اجرای قسمت Non-CS را شروع می‌کند. دومی نیز CS را اجرا می‌کند و بعد از تمام شدن CS پردازه دوم، نوبت را به پردازه اول می‌دهد و Non-CS را شروع می‌کند. ممکن است قسمت Non-CS پردازه دوم بسیار کوتاه باشد و زود تمام بشود. بعد از اتمام درخواست ورود به CS را دارد ولی می‌بایست منتظر بماند تا پردازه اول Non-CS خود را به اتمام برسد و در مدت زمانی هیچ پردازه‌ای در عین وجود درخواست نمی‌تواند وارد CS شود.

Bounded Waiting:

وجود دارد.

چون فقط دو پردازه داریم و با تمام شدن قسمت CS یک پردازه، مقدار turn عوض می‌شود؛ پس بعد از لحظه‌ای که درخواست ورود به CS را بدهد نهایتا یک پردازه می‌تواند CS خود را اجرا کند تا نوبت پردازه درخواست دهنده بشود.
Bound = 1

ب)

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

این الگوریتم بسیار شبیه الگوریتم Peterson است. فقط شرط while و مشخص کردن turn قبل while متفاوت است. این تفاوت تاثیر زیادی در اجرای الگوریتم ندارد چرا که دقیقا بیانگر جمله «اگر پردازه دوم آماده نبود و نوبت پردازه من بود وارد CS شو»

به عنوان مثال

Mutual Exclusion:

وجود دارد.

شرایطی وجود ندارد که شرط while هردو پردازه غلط و هردو وارد CS بشوند. پس Mutual Exclusion داریم.

Progress:

وجود دارد.

فرض کنیم پردازه اول اجرا شود، وارد CS شود و از CS خارج شود. به مخض خارج شدن flag خود را غلط می‌کند تا پردازه دیگر بتواند وارد CS شود. دیگر نیازی نیست تا پردازه دوم منتظر بماند تا پردازه اول Non-CS را اجرا کند تا وارد CS شود.

Bounded Waiting:

وجود دارد.

پس از لحظه‌ای که پردازه درخواست ورود به CS را می‌دهید (flag = true) امکان ندارد بیش از یک پردازه بتواند وارد CS شود تا نوبت او برسد.

4)

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

int multiplication(int op1, int *P_op2){
     // TODO
     return *P_op2, *op1;
}
```

TODO:

```
int val;
flag = true;
while(flag){
     val = *P_op2;
     flag = compare_and_swap(P_op2, val, val*op1);
}
```

Mutual Exclusion:

وجود دارد.

Progress:

وجود دارد.

Bounded Waiting:

وجود ندارد.

مثلا اگر همه فرایندها به دنبال تغییر متغیر P_op2 باشد ممکن است نوبتی رعایت نشود و bound وجود ندارد.

در این صورت به صورت اتمی نیز نمی‌توان ضرب را انجام داد.

5)

Busy waiting چیست؟ انتظار مشغول، همچنین به عنوان Spinning یا Busy looping شناخته می‌شود، یک تکنیک همگام‌سازی فرآیند است که در آن یک فرآیند منتظر می‌ماند و قبل از ادامه اجرای آن شرط را بررسی می‌کند تا برآورده شود و هیچ کار مفیدی انجام نمی‌دهد. به عبارت دیگر وقت CPU را هدر می‌دهد.

سایر انتظارهای موجود در سیستم‌عامل کدامند؟

دو رویکرد کلی برای انتظار در سیستم عامل ها وجود دارد: اول، Busy waiting. ثانیا، یک فرآیند می تواند بدون مصرف پردازنده منتظر بماند. در چنین حالتی، هنگامی که شرایط برآورده شد، فرآیند هشدار داده می شود یا بیدار می شود. مورد دوم به عنوان Sleeping، Blocked waiting یا Sleep waiting شناخته می شود. همچنین انتظارهای دیگری وجود دارند مانند I/O Waiting که در آن فرایند منتظر براورده شدن درخواستی از دستگاه‌های ورودی/خروجی می‌باشد.

آیا به صورت کلی می‌توان از Busy waiting جلوگیری کرد؟ بله. با استفاده از Sleep waiting که بالاتر توضیح داده شد می‌توان از انتظار مشغول جلوگیری کرد.