Department of
Computer Engineering

Homework 3-Solution

Operating Systems

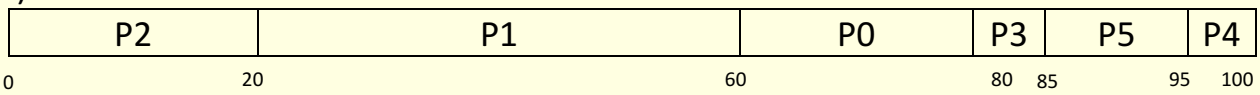
Fall 2023

Dr. Javadi

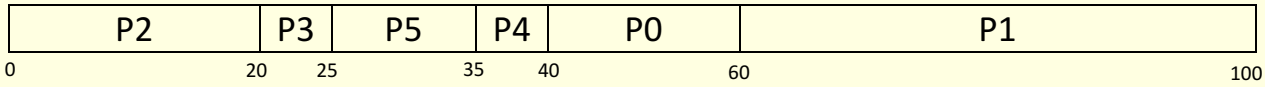
پاسخ سوال ۱:

الف) با نمودار Gantt نشان می دهیم:

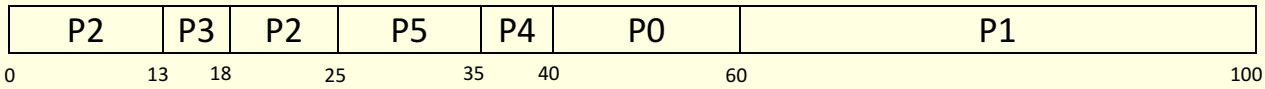
i)FCFS:



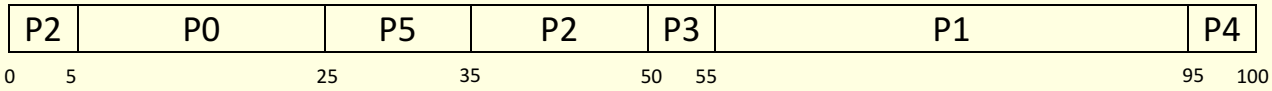
ii)SJF:



iii)SRT:



iv)preemptive priority:



ب)با توجه به نمودار های فوق:

FCFS:

Turnaround time : $\frac{20+60+75+72+78+70}{6} = 62.5$

Waiting time : $\frac{0+20+55+67+68+65}{6} = 45.8$

Response time : $\frac{0+20+55+67+68+65}{6} = 45.8$

CPU utilization : 1

SJF:

Turnaround time : $\frac{20+12+18+10+55+100}{6} = 35.8$

Waiting time : $\frac{0+7+8+5+35+60}{6} = 19.1$

Response time : $\frac{0+7+8+5+35+60}{6} = 19.1$

CPU utilization : 1

SRT:

Turnaround time : $\frac{25+5+18+10+55+100}{6} = 35.5$

Waiting time : $\frac{5+0+8+5+35+60}{6} = 18.8$

Response time : $\frac{0+0+8+5+35+60}{6} = 18$

CPU utilization : 1

Preemptive priority:

Turnaround time : $\frac{50+20+18+42+95+70}{6} = 49.2$

Waiting time : $\frac{30+0+8+37+55+65}{6} = 32.5$

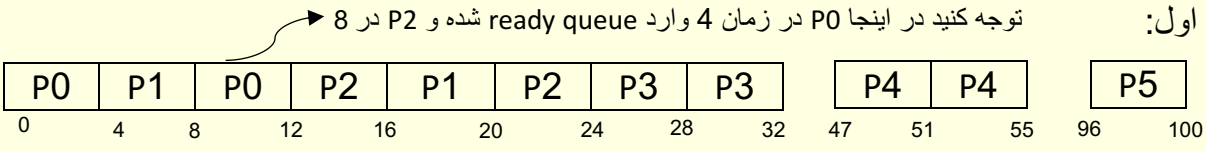
Response time : $\frac{0+0+8+37+55+65}{6} = 27.5$

CPU utilization : 1

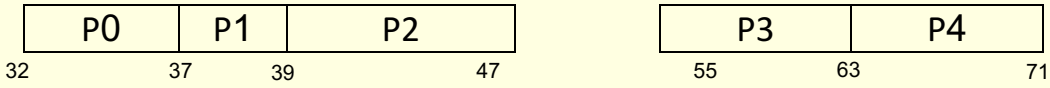
پاسخ سوال۲:

الف)با دنبال کردن روند ذکر شده در سوال،ابتدا برای هر لایه نمودارهای جداگانه می کشیم و سپس نمودار ها را به هم می چسبانیم:

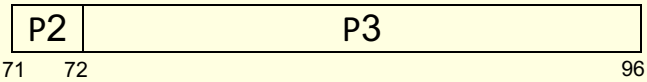
لایه اول:



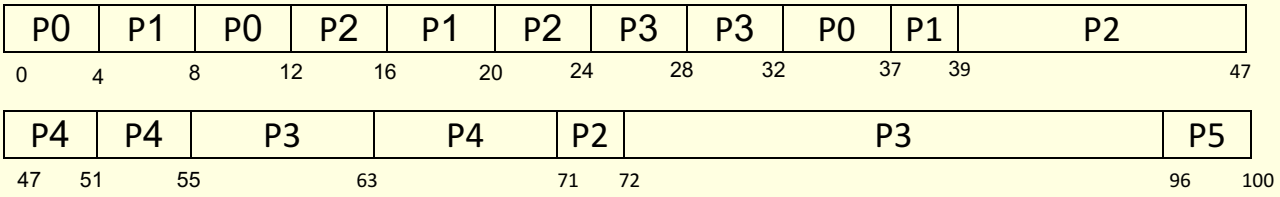
لایه دوم:



لایه سوم:

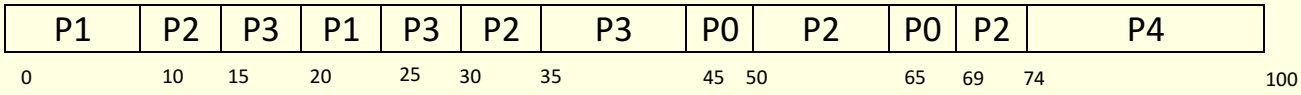


و در نهایت نمایش آن با یک نمودار Gantt:



...

ب) نمودار Gantt زمانبند مذکور(توجه کنید context switch ها هم به علت تغییر اولویت ها بر اساس مکانیزم aging مطرح شده و هم به علت CPU burst کوتاه تر در شرایط اولویت یکسان اتفاق می افتد.):



پاسخ سوال ۳:

الف)

Process i

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

Process j

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

ابتدا شرط mutual exclusion رو بررسی میکنیم. فرض کنیم ابتدا دو خط اول i اجرا شود سپس به حلقه میرسیم حال شرط برقرار است چون فلگ j فالس است. سپس دو خط اول j اجرا مشود حال برای اجرای حلقه ی j شرط turn==i برقرار است. پس حلقه برقرار است. حال چون فلگ j فالس نیست و turn=i است پس شرط حلقه برقرار نیست و وارد critical section میشود. و تا وقتی که این بخش تمام نشود پردازه ی j نمیتواند وارد critical section پس mutual exclusion داریم.
حال progress را بررسی میکنیم. در ادامه ی فرایند بالا اگر خط flag[i]=flase اجرا شود بازهم شرط حلقه برقرار است و j نمیتواند وارد critical section شود پس progress نداریم.
و Bounded Waiting داریم.

ب)

Process j

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

Process i

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

ابتدا شرط mutual exclusion را بررسی میکنیم که اگر خط های اول پردازه ها ابتدا اجرا شوند دیگر هیچکدام از دو شرط حلقه ها برقرار نمیشوند و هردو میتوانند وارد critical section شوند پس mutual exclusion نداریم.
حال progress را بررسی میکنیم. فرض کنیم پردازه j کامل اجرا شود سپس پردازه i ،حال شرط حلقه برقرار است و مانع ادامه ی اجرای i میشود پس progress نداریم. Bounded Waiting داریم.

پاسخ سوال ۴:

الف)

progress:

نداریم، فرض کنیم s1 = s2 است و پردازه ی ۱ قصد ورود به ناحیه بحرانی را ندارد. در این صورت پردازه ی۲ که قصد ورود به ناحیه بحرانی را دارد باید منتظر پردازه ی اول بماند و نمی تواند وارد ناحیه ی بحرانی شود.

Mutual Exclusion:

داریم، در هر لحظه یا s1 == s2 است یا نیست و در هر کدام از این حالت ها فقط یکی از پردازه ها می توانند در ناحیه بحرانی باشند.

Bounded Waiting:

داریم، بعد از حداکثر یکبار ورود یک پردازه با ناحیه ی بحرانی، نوبت ورود پردازه ی دیگر میشود. به عبارتی پردازه ی اول حداکثر باید به اندازه ی یک پردازه برای ورود به ناحیه بحرانی صبر کند.

ب)

برای این بخش یک عکس از اسلاید ها قرار میدهیم، در این عکس همانطور که مطالعه کرده اید، تمام شرط ها برقرار است. flag همان نقش s1 و s2 را دارد و یک متغیر دیگر اضافه کرده ایم. هر پاسخی که همین کار را انجام دهد مورد قبول است.

```
while (true){
    flag[i] = true;
    turn = j;
    while (flag[j] && turn = = j);

        /* critical section */

    flag[i] = false;

    /* remainder section */

}
```

پاسخ سوال ۵:

الف)

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

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

ب) از آنجا که اجرای دستورات تابع compare and swap اتمیک هستند یعنی میتوانیم اطمینان داشته باشیم که وسط اجرا آن context switch رخ نمیدهد و از کاربردهای آن این است که میتواند برای critical section بکار برود.