به نام خدا

سیستم‌های عامل (پاییز ۱۴۰۱)

پاسخ‌نامه تمرین سوم

استاد درس:

دکتر جوادی

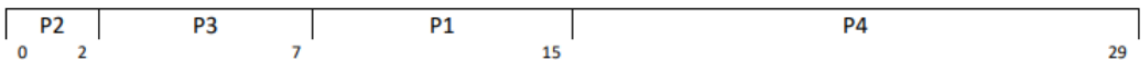
طراحی وجمع‌آوری سوالات:

خانم ها سروقد و میرزازاده و آقای پولاد

۱ - به سوالات پاسخ دهید.

الف) با در نظر گرفتن جدول زیر و با استفاده از روش اول کوتاه ترین کار نمودار گانت زمان بندی پردازنده ها را رسم کرده و میانگین زمان تاخیر را بدست آورید.

فرایند	Burst time
p1	4
p2	8
p3	14
p4	7



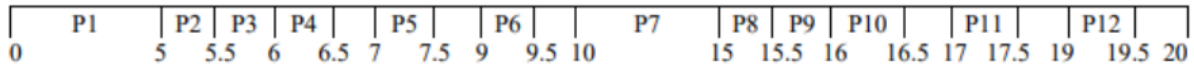
میانگین زمان تاخیر:

$(2 + 7 + 15 + 0)/4 \ = \ 6ms$

ب) فرض کنید در هر 10ثانیه یک فرآیند با مدت اجرای 5 ثانیه به صف اجرا اضافه میشود. همچنین فرآیندهایی با مدت اجرای 0.5 ثانیه نیز در هر 2 ثانیه ساخته میشوند. بخشی از الگوی ترتیب اضافه شدن این فرآیندها را در جدول زیر مشاهده میکنید.

Process	Arrival time	Burst time
p1	0	5
p2	1	0.5
p3	3	0.5
p4	5	0.5
p5	7	0.5
p6	9	0.5
p7	10	5
p8	11	0.5
p9	13	0.5
...	...	...

نمودار گانت اجرای فرآیندها با زمانبندی FCFS را برای مدت 20 ثانیه رسم کرده و میانگین زمان انتظار فرآیندها را به دست آورید.



Average waiting time = (0 + 4 + 2.5 + 1 + 0 + 0 + 0 + 4 + 2.5 + 1 + 0 + 0) / 12 = 1.25

۲ - می‌خواهیم با استفاده از دستور compare_and_swap، یک تابع برای جمع کردن دو عدد به صورت اتمیک طراحی کنیم. فرم تابع به صورت زیر است و خروجی تابع باید مقدار *+ p1 p2 باشد. توجه کنید که مقدار حافظه‌ای که *p1 به آن اشاره می‌کند ممکن است هر لحظه توسط یک ترد دیگر تغییر کند و منظور از اتمیک این است که عملکرد تابع شما نباید در این صورت دچار اختلال گردد.

int atomic_add(int *p1, int p2);

برای استفاده از دستور compare_and_swap می‌توانید از تابع زیر استفاده کنید:

int compare_and_swap(int *value, int expected, int new_value);

ایده‌ی کلی این است که انقدر عملیات CAS را تکرار کنیم تا با موفقیت انجام شود. پاسخ هایی که در آنها از فرمی از CAS استفاده شده که مقدار boolean موفقیت یا عدم موفقیت عملیات را بر می‌گردانند نیز صحیح هستند.

```
int atomic_add(int *p1, int p2)
{
    if (p2 == 0) return *p1;
    int done = 0;
    while (!done)
    {
        int value = *p1;
        int new = compare_and_swap(p1, value, value + p2)
        if (new == value + p2) done = 1;
    }
    return *p1;
}
```

۳ - دو پردازه برای حل مسئله‌ی ناحیه‌ی بحرانی از روش زیر استفاده کرده‌اند. متغیر های S1 و S2 بین دو پردازه مشترک هستند و یک مقدار Boolean دارند که در ابتدای اجرای برنامه به صورت تصادفی مقدار دهی شده‌اند.

P2	P1

while (S1 != S2);
 	
//Critical Section
 	
S2 = !S1
	while (S_1 == S_2);
	
	//Critical Section
	
	S_1 = S_2

بررسی کنید و توضیح دهید که هر کدام از ۳ شرط Mutual Exclusion, Progress و Bounded Waiting برآورده می‌شوند یا خیر.

Progress: نداریم. فرض کنیم S1 = S2 است و پردازه‌ی ۱ قصد ورود به ناحیه بحرانی را ندارد. در این صورت پردازه‌ی ۲ که قصد ورود به ناحیه بحرانی را دارد باید منتظر پردازه‌ی اول بماند و نمی‌تواند وارد ناحیه‌ی بحرانی شود.

Mutual Exclusion: داریم. در هر لحظه یا S1==S2 است یا نیست و در هر کدام از این حالت فقط یکی از پردازه ها می‌توانند در ناحیه‌ی بحرانی باشند.

Bounded Waiting: داریم. بعد از حداکثر یک بار ورود یک پردازه به ناحیه‌ی بحرانی، نوبت ورود پردازه‌ی دیگر می‌شود. به عبارتی پردازه‌ی اول حداکثر باید به اندازه‌ی یک پردازه برای ورود به ناحیه‌ی بحرانی صبر کند.

۴ - تعداد n پردازه داریم که هر کدام از دو بخش پردازشی A و B تشکیل شده اند. می‌خواهیم با استفاده از Semaphore ها به گونه‌ای این پردازه ها را هماهنگ کنیم که اول قسمت A همه‌ی پردازه ها به طور کامل اجرا شود و بعد از آن قسمت B پردازه ها اجرا شود و فرض کنید تعداد پردازه ها را در متغیر n داریم. شبه کد زیر را به گونه‌ای تکمیل کنید که این هماهنگی ایجاد شود. سمافور هایی که استفاده کرده‌اید و مقدار اولیه آن ها را بالای کدتان بنویسید.

```
A()

//add your code here

B()
```

راهنمایی: می‌توانید از سمافور ها و متغیر های زیر استفاده کنید:

- **Count: متغیری که تعداد پردازه هایی که بخش A را اجرا کرده‌اند را نشان می‌دهد.**
- **Mutex: سمافور با مقدار اولیه‌ی ۱ برای تغییر دادن Count.**
- **Barrier: سمافور با مقدار اولیه‌ی ۰ که تا زمانی که همه‌ی پردازه ها قسمت A را تمام نکرده‌اند مقدار آن بیشتر از صفر نمی‌شود.**

از همان متغیر ها و سمافور هایی که در راهنمایی سوال آمده استفاده می‌کنیم.

```
A()

mutex.wait()
count = count + 1
mutex.signal()
if count == n : barrier.signal()
barrier.wait()
barrier.signal()

B()
```