باسمه تعالی

تکلیف ۳ درس سیستم عامل – دکتر جوادی

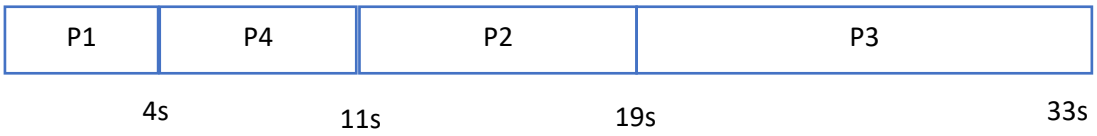
سید امیرمهدی میرشریفی -۹۸۳۱۱۰۵

۱ - به سوالات پاسخ دهید.

الف) با در نظر گرفتن جدول زیر و با استفاده از روش اول کوتاترین کار[1] نمودار گانت زمان‌بندی پردازنده‌ها را رسم کرده و میانگین زمان تاخیر را بدست آورید.

process	burst time
p1	4
p2	8
p3	14
p4	7

پاسخ:



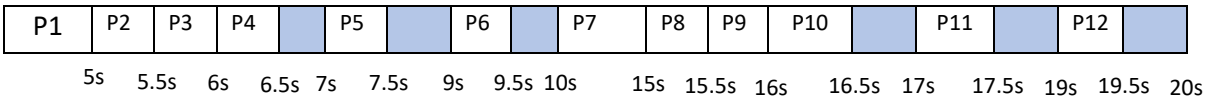
waitTime(p1)=0s waitTiem(p2)=11s waitTime(p3)=19s waitTime(p4)=4s

⇨ Average waitTime = (0+4+11+19)/4 s = 8.5s

ب) فرض کنید در هر ۱۰ ثانیه یک پردازه با مدت اجرای ۵ ثانیه به صف اجرا اضافه می‌شود. همچنین پردازه هایی با مدت اجرای ۰.۵ ثانیه نیز در هر ۲ 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 ۲۰ ثانیه رسم کرده و میانگین زمان انتظار فرآیندها را به دست آورید.



waitTimes:

P1=0s	P2=4s	P3=2.5s	P4=1s
P5=0s	P6=0s	P7=0s	P8=4s
P9=2.5s	P10=1s	P11=0s	P12=0s

⇨ Avrage waitTime = (0+4+2.5+1+0+0+0+4+2.5+1+0+0)/12=1.25s

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);
```

```
Int atomic_add(int *p1, int p2){
    Int value;
    While (True) {
        value=*p;
        int old_value = compare_and_swap(*p,value,value+p2);
        if(value==old_value) {break;}}
    return value + p2;
```

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.

پاسخ:

شرط اول که هر دو نتوانند در یک زمان وارد ناحیه بحرانی شوند برقرار است زیرا دو متغیر در یک زمان یا مساوی هستند یا نیستند.

حال اگر هر کدام از پردازه ها پس از پایان کار خود مقدار متغیر ها را تغییر دهند باید حتما منتظر بمانند تا پردازه دیگر ، عملیات خود را انجام دهد تا پس از پایان آن شرایط را برای ورود پردازه دیگر محیا کند.

بنابراین شرط دوم که پیشرفت است در این کد نقض شده است.

و اما شرط سوم برقرار است زیر هر پردازه می بایست یکبار منتظر بماند تا پردازه دیگر شرایط را محیا کند برایش و سپس حتما نوبت پردازه ای است که منتظر ایستاده است . بنابراین هر پردازه به اندازه یک عملیات بحرانی پردازه دیگر منتظر می ماند

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

```
A()
//add your code here
B()
```

راهنمایی: می‌توانید از سمافورها و متغیرهای زیر استفاده کنید:

- count: متغیری که تعداد پردازه‌هایی که بخش A را اجرا کرده‌اند را نشان می‌دهد.
- mutex: سمافور با مقدار اولیه‌ی 1 برای تغییر دادن count.
- barrier: سمافور با مقدار اولیه‌ی 0 که تا زمانی که همه‌ی پردازه‌ها قسمت A را تمام نکرده‌اند مقدار آن بیشتر از صفر نمی‌شود.

پاسخ: با فرض این که متغیر counter متغیر داخلی سمافور mutex است که دو تابع signal&wait تغییرات را روی آن اعمال می کنند:

```
Semaphore mutex=1;
Semaphore barrier=0;

Wait(mutex);
A ();
If (mutex.count==n) {
        B ();
        signal(barrier);
        exit ();}
Signal(mutex);

Wait(barrier);
B ();
Signal(barrier);
```