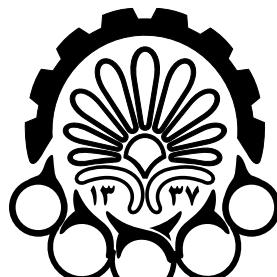


به نام خدا



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)

## سیستم‌های عامل (پاییز ۱۴۰۰)

# تمرین اول

استاد درس:

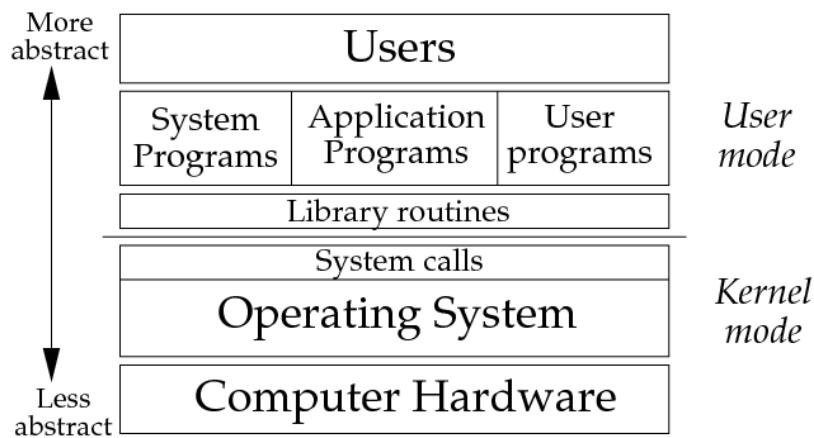
آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

۳۰ مهر ۱۴۰۰ ساعت ۵:۲۳

نکته مهم؛ دقت کنید که تمدید نخواهیم داشت و صرفاً می‌توانید ۱ تا ۵ روز از ۱۵ روز مجاز برای تاخیر ارسال تمامی تمرین‌های تئوری در این ترم را استفاده کنید. اگر بودجه ۱۵ روز شما تمام شود، به ازای هر روز تاخیر ۱۰ درصد از نمره تمرین را از دست خواهید داد.

۱) سه مورد از مزایا و معایب جداسازی سطح **kernel** و **user** را نام ببرید.



۲) درباره multiprogramming و multitasking به صورت جداگانه توضیح دهید و تفاوت‌های آن‌ها را بیان کنید.

۳) چگونگی بالا آمدن و بارگذاری سیستم عامل در لحظه شروع به کار سیستم کامپیوتری را با جزئیات توضیح دهید.

۴) دلیل اینکه برنامه‌ها مختص به هر سیستم عامل هستند و براحتی روی سیستم‌های عامل دیگر اجرا نمی‌شوند، چیست؟

۵) دربارهٔ نحوه عملکرد DMA توضیح دهید. نحوه همکاری DMA و پردازنده به چه صورت است؟ پردازنده به چه صورت از به پایان رسیدن کار DMA مطلع می‌شود؟

موفق باشید

تیم درس سیستم‌های عامل

## به نام خدا

### سوال ۱) سه مورد از مزایا و معایب جداسازی سطح user و kernel را نام ببرید.

از مزایای جداسازی سطح user و kernel این است که برای هر برنامه در سطح user یک محدوده حافظه در نظر گرفته می‌شود و برنامه‌های مختلف امکان دسترسی به حافظه یکدیگر را ندارند. همچنین در صورت ایجاد مشکل در یک روند در سطح user، این مشکل باعث از کار افتادن کل سیستم نمی‌شود و تاثیری بر روی سایر پردازه‌ها نمی‌گذارد. علاوه بر این برنامه‌ای که در سطح user نوشته شده به وسیله system call‌ها با سطح kernel ارتباط برقرار می‌کند و در بسیاری از موارد به طور مستقیم با قطعات سخت افزاری درگیر نمی‌شود.

از معایب این جداسازی این است که برنامه‌های که در سطح user اجرا می‌شود به دلیل این که پردازنده به طور مداوم باید بین سطوح user و kernel تغییر وضعیت دهد باعث کند شدن آن می‌شود. همچنین برنامه در سطح user دسترسی مستقیم به  $i/o$  ندارد و برای هر دسترسی باید یک فرآخوانی سیستمی انجام دهد که باعث کند شدنش می‌شود. علاوه بر این چون در سطح user معمولاً تعداد زیادی پردازه در حال اجرا هستند و پردازنده دائمًا بین آنها تعویض می‌شود و این تعویض بسیار زمان‌گیر است، اگر برنامه‌ی خاصی را در سطح kernel برد و به دور از بقیه پردازه‌ها اجراش کنیم بسیار سریعتر خواهد بود.

### سوال ۲) درباره multitasking و multiprogramming به صورت جدأگانه توضیح دهید و تفاوت‌های آن‌ها را بیان کنید.

در multiprogramming ما یکسری از کارها(jobs) در حافظه خود داریم و به این صورت مطمئن می‌شویم که دائمًا در حال اجرای برنامه‌ای است.

در multitasking هم به مانند multiprogramming هست که علاوه بر کار بالا به سرعت(با یک اندازه‌ای مشخص زمان به برنامه اختصاص می‌دهد) بین برنامه‌ها جا به جا می‌شود به طوری که کاربر متوجه جابه‌جایی برنامه‌ها نشود.

در هر دو حالت زمان اگر برنامه‌ای به دستورات غیر از استفاده از cpu برسرد (برای مثال  $I/O$ ) برنامه خارج می‌شود از cpu.(البته در multitasking بجز این حالت اگر زمانی که اختصاص دادیم به برنامه تمام شود برنامه از cpu خارج شده و به صف انتظار می‌رود).

از دیگر تفاوت‌ها می‌شود که این اشاره کرد که multiprogramming استفاده از حافظه کمتری نسبت به دیگری دارد چرا که فقط یک برنامه می‌امد تا تمام شود ولی در حالت دیگه کلی برنامه در حافظه است دیگر تفاوت این است که multiprogramming بیشتر استفاده از cpu را ماقسیم می‌کند ولی دیگری دنبال افزایش reaction cpu هست.

سوال ۳) چگونگی بالا آمدن و بارگذاری سیستم عامل در لحظه شروع به کار سیستم کامپیوتري را با جزئیات توضیح دهید.

از آنجایی که حافظه ROM به صورت Read Only میباشد و همچنین مثل RAM فرار نمیباشد برنامه bootstrap در آن ذخیره میشود سپس Bootloader سیستم عامل های موجود را که ممکن است چندتا باشند پیدا میکند و سپس کاربر تعیین میکند تا کدام یک اجرا شود. بعد از انتخاب کاربر، سیستم عامل موردنظر به RAM منتقل شده تا از آنجا خوانده شود و شروع به اجرا کند. بعد از این مرحله کنترل کامل سیستم کامپیوتري ما در اختیار سیستم عامل قرار میگيرد.

سوال ۴) دلیل اینکه برنامه ها مختص به هر سیستم عامل هستند و بر احتیاج روی سیستم های عامل دیگر اجرا نمیشوند چیست.

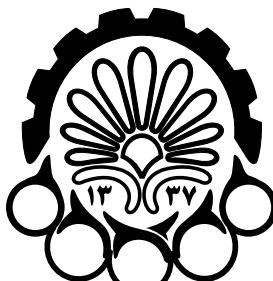
چون برنامه ها در هنگام اجرا از فرآخوانی های سیستمی مختلف استفاده میکنند و از آنجایی که در سیستم های عامل مختلف فرآخوانی های سیستمی متفاوتی وجود دارد، برنامه های (کامپایل شده در) یک سیستم عامل ممکن است روی سیستم عامل دیگری اجرا نشود.

سوال ۵) نحوی عملکرد DMA را توضیح دهید. نحوی همکاری DMA و پردازنده به چه صورت است؟ پردازنده به چه صورت از پایان رسیدن کار DMA مطلع میشود؟

دسترسی مستقیم به حافظه از طریق DMA برای دستگاه های ورودی-خروجی با سرعت بالا مورد استفاده قرار میگیرد تا مانع افزایش زمان اجرای عملیات بارگیری در پردازنده ها شود.

پردازنده میتواند با نوشتن مقادیر در ثبات های ویژه ای که بطور مستقل توسط دستگاه های ورودی-خروجی قابل دسترس میباشند، عملیات DMA را شروع کند. دستگاه با فرمان پردازنده، کار خود را شروع میکند و زمانی که کار خود را به اتمام میرساند یک سیگنال وقفه برای پردازنده ارسال میکند تا آن را از پایان کار خود مطلع و متوقف کند.

به نام خدا



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)

سیستم‌های عامل (پاییز ۱۴۰۰)

## تمرین دوم

استاد درس:

آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

۱۷ آبان ۱۴۰۰ ساعت ۲۳:۵۹

نکته مهم: دقت کنید که تمدید نخواهیم داشت و صرفا می‌توانید ۱ تا ۵ روز از ۱۵ روز مجاز برای تاخیر ارسال تمامی تمرین‌های تئوری در این ترم را استفاده کنید. اگر بودجه ۱۵ روز شما تمام شود، به ازای هر روز تاخیر ۱۰ درصد از نمره تمرین را از دست خواهید داد.

لطفا به سوالات زیر با دقت پاسخ دهید و پاسخ نهایی و کامل خود را در سایت درس بارگذاری کنید.

۱) استفاده از فراخوانی سیستمی (`fork()`) روشی پرکاربرد برای ایجاد پردازه فرزند در سیستم عامل‌های بر پایه POSIX است. به طور خلاصه پس از اجرای این دستور، پردازه فرزند که در اغاز کارش یک کپی از پردازه پدر است، به شکل همرونده (یا همزمان) با پردازه پدر اجرا می‌شود. مهم‌ترین تفاوت در نحوه رفتار `fork()` است که در پردازه پدر مقدار شناسه یکتاًی پردازه فرزند (عددی بزرگ‌تر از صفر) را بر می‌گرداند و در پردازه فرزند عدد صفر را بر می‌گرداند. همچنین اگر مقدار برگشت داده شده، عددی کمتر از صفر باشد به این معنی است که بر اثر رخداد خطایی، پردازه فرزند ایجاد نشده است. با این توصیفات اگر قطعه کد زیر اجرا شود، به ترتیب، خطوطی که در پردازه پدر و فرزند چاپ می‌شوند، چه خواهد بود؟ دلیل پاسخ خود را به اختصار توضیح دهید.

```
int main() {
    pid_t = pid
    a = 30; /* address of a in memory is 2000 */
    pid = fork();
    if (pid<0) {
        fprintf(stderr, "fork failed");
        return 1;
    }else if (pid == 0) {
        a = a + 10;
        printf("%d, %dn", a, &a);
    }else {
        a = a - 10;
        printf("%d, %dn", a, &a);
    }
    return 0;
}
```

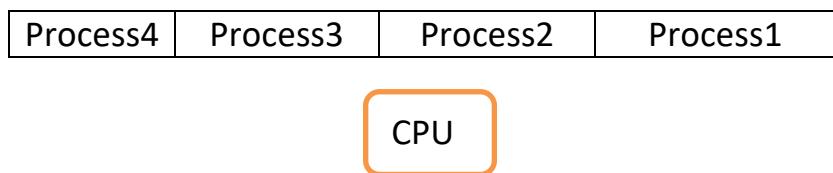
۲) خروجی کد زیر چیست؟ نمودار (درخت) پردازه‌های آن را رسم کرده و درمورد آن به اختصار توضیح دهید.

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    if ((fork() && !fork()) || (fork() && fork()))
        printf("3 ");
    return 0;
}
```

۳) در مورد نحوه بوجود آمدن پردازه‌های زامبی (Zombie) و یتیم (Orphan) توضیح دهید. نحوه رسیدگی سیستم عامل به آن‌ها به چه شکل است؟

۴) زمانبند پردازه به این نحو عمل می‌کند که هر پردازه ۱۵۰ کلاک برای اجرای روی CPU زمان دارد و بعد از پایان زمانش روی صفحه انتظار می‌رود که در لحظه ابتدایی (صفر) ۴ پردازه در صفحه موجود است و CPU خالی است. پردازه اول بعد از ۲۰۰ کلاک اجرای خود به فراخوانی سیستمی fork() می‌رسد. پردازه دوم بعد از هر ۱۰۰ کلاک به یک دستور  $i/o$  می‌رسد که ۴۰۰ کلاک زمان خواهد برد، پردازه سوم تنها به ۴۰۰ کلاک نیاز دارد تا به پایان برسد، پردازه چهارم بعد از ۵۰ کلاک نیاز به خواندن از حافظه اصلی دارد که ۵۰ کلاک زمان می‌برد. صفحه پردازه‌ها پس از گذشت ۱۰۰۰ کلاک به چه نحوی خواهد بود؟



۵) تکه کد زیر چند بار عبارت "hello" را چاپ می کند؟ (با این فرض که تمامی فراخوانی های سیستمی با موفقیت اجرا شوند)

```
...
{
    if(fork() && fork())
    {
        fork();
    }

    if(fork() || fork())
    {
        fork();
    }

    printf("hello");
    return 0;
}
```

موفق باشد

تیم درس سیستم های عامل

## سوال ۱)

هنگامی که فرایند فرزند ایجاد می شود ، از بخش های stack و heap فرایند والد یک کپی گرفته و اجرا می شود . با این حساب آدرس فیزیکی متغیر a در فرایند فرزند با پدر متفاوت خواهد بود.

اما در اینجا دو نکته مهم وجود دارد که باید به آن توجه کنیم:

نکته ۱ : اینکه اول قسمت کد والد اجرا خواهد شد یا فرزند را نمی توان به صورت قطعی از قبل تعیین کرد. ولذا ممکن است ترتیب چاپ شدن خطوط فرق کند .

نکته ۲ : علی رقم آنکه آدرس فیزیکی متغیر a در دو فرایند فرق می کند ، اما اگر این قطعه کد را در زبان C پیاده کنیم خواهیم دید که آدرس متغیر های a در هر دو فرایند یکسان خواهد بود. تصویر زیر نشان دهنده این مورد است:

The screenshot shows a terminal window titled "main.c" with the following C code:

```
1 // ****
2
3 // Write C code here
4 // Online C Compiler.
5 // Code, Compile, Run and Debug C program online.
6 // Write your code in this editor and press "Run" button to compile and execute it.
7 ****
8
9 // Online C compiler to run C program online
10 #include <stdio.h>
11
12 int main() {
13     // Write C code here
14     int a = 30;
15     if (fork() == 0)
16     {
17         a = a + 10;
18         printf("child : %d, %p \n", a, &a);
19     }
20     else
21     {
22         a = a - 10;
23         printf ("parent : %d, %p \n", a,&a);
24     }
25     return 0;
26 }
```

The output window shows the results of the program execution:

```
main.c:15:9: warning: implicit declaration of function 'fork' [-Wimplicit-function-declaration]
parent : 20, 0x7ffecd209dc
child : 40, 0x7ffecd209dc

...Program finished with exit code 0
Press ENTER to exit console.
```

علت این اتفاق در این است که سیستم های امروزی بجای آنکه با آدرس های physical متغیر ها کار کند ، با آدرس های logical آنها کار می کند. این موضوع باعث می شود که هنگام باز سازی آدرس متغیر ها مقادیر یکسانی ایجاد شود. لذا خروجی به صورت زیر است:

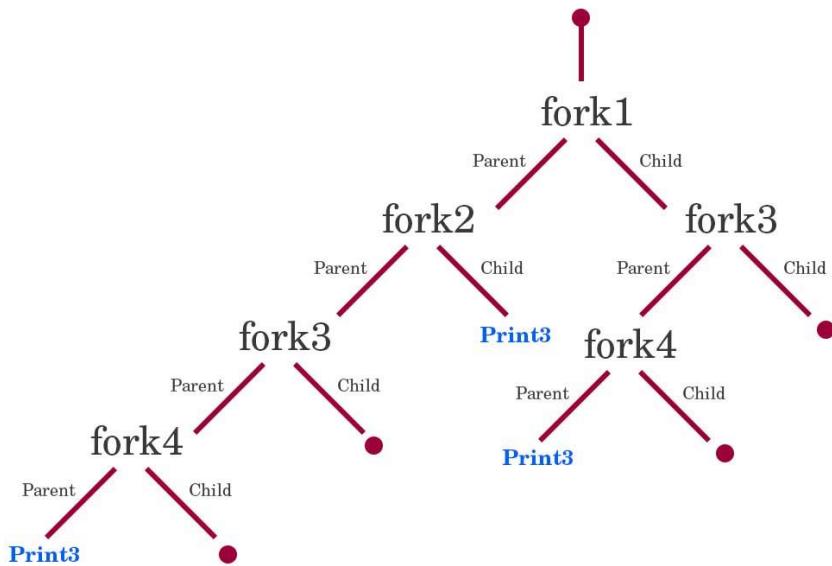
parent : "20, 2000"

child : "40, 2000"

## سوال (۲)

اگر fork ها را به ترتیب از چپ به راست شماره گذاری کنیم روند اجرای کد به صورت زیر است:

نکته مهمی که در اجرای روند کد وجود دارد مبحث Lazy Evaluation در زبان C است. به این صورت که اگر در سمت چپ یک عبارت And مقدار False ظاهر شود، سمت راست آن هر مقداری باشد حاصل And مقدار False خواهد بود. پس با ظاهر شدن مقدار False یا همان صفر در سمت چپ این عملگر، دیگر نیازی به اجرا و بررسی عبارت سمت راست نداریم. همین مبحث در مورد عملگر Or در صورتی که سمت چپ عملگر مقدار True ظاهر شود نیز وجود دارد.



## سوال (۳)

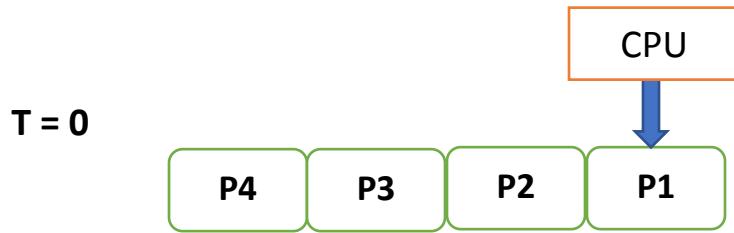
پردازه zombie : پردازه ایی است که وظایف خود را به طور کامل انجام داده و زمان kill شدن آن فرا رسیده اما به دلایل مختلف عدم ارسال سیگنال SIGCHLD توسط این پردازه (child) یا عدم اجرای() توسط پردازه پدر (parent) هنوز در process table باقی مانده است.

روش handle کردن پردازه zombie : با فرستادن سیگنال kill نمیتوان این نوع پردازه ها را از بین برد چون به هیچ سیگنالی واکنش نشان نمیدهد ( فقط توصیف گر آن در حافظه حضور دارد بنابراین اگر آن ها را هندل نکنیم با انباشته شدن سیستم را مختل میکنند) اما میتوان به پردازه پدر سیگنال SIGCHLD را ارسال کرد تا () wait را اجرا کند. اگر در پردازه پدر به درستی این امر پیاده سازی نشده باشد و قادر به اجرای () wait نباشد خود پردازه پدر را kill میکنیم تا پردازه zombie به پردازه orphan تبدیل شود.

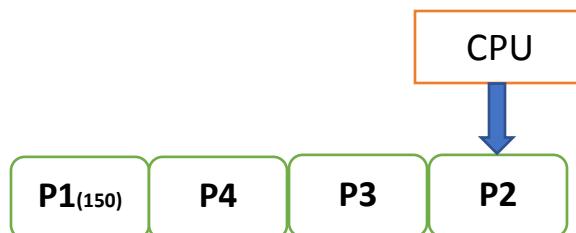
پردازه orphan : پردازه ایی است که قبل از اتمام وظایف خود به هر دلیلی پردازه پدر خود را از دست میدهد و پردازه پدرش بدون اجرای wait() برای این پردازه به پایان میرسد.

روش handle کردن پردازه orphan : پس از مدت کوتاهی که این پردازه orphan میشود پردازه init (اولین پردازه ایی که از زمان شروع به کار سیستم فعال میشود و تا زمان خاموش شدن آن از بین نمیرود با pid برابر با ۱) را به عنوان پدر این نوع پردازه ها اختصاص میدهیم. این پردازه در دوره های زمانی مشخص wait() را برای این نوع پردازه ها صدا میزند تا terminate شوند.

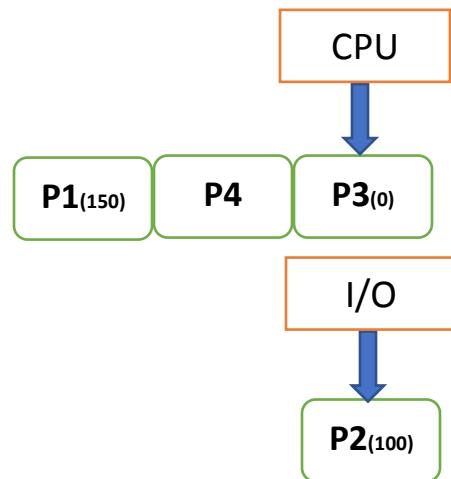
سوال (٤)



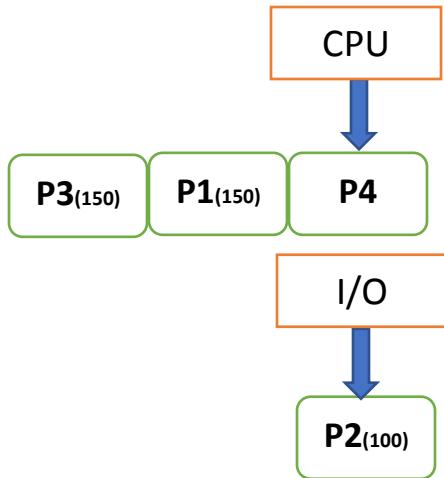
**T = 150**



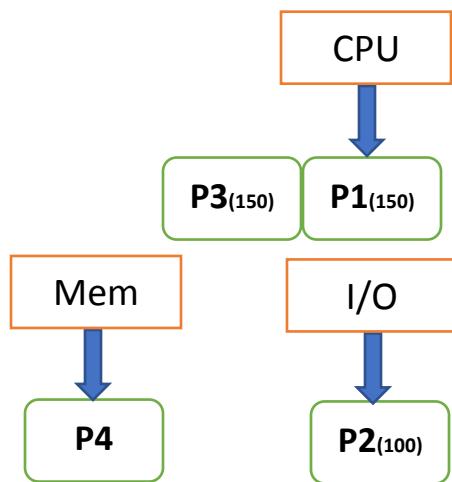
**T = 250**



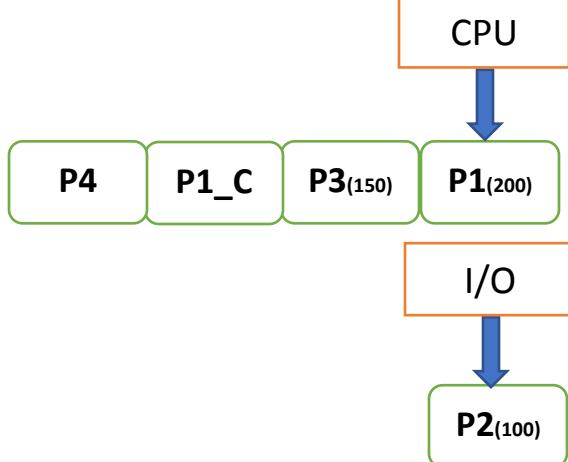
**T = 400**



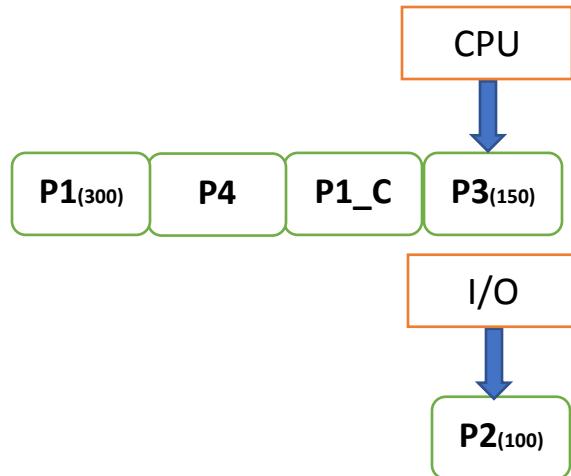
**T = 450**



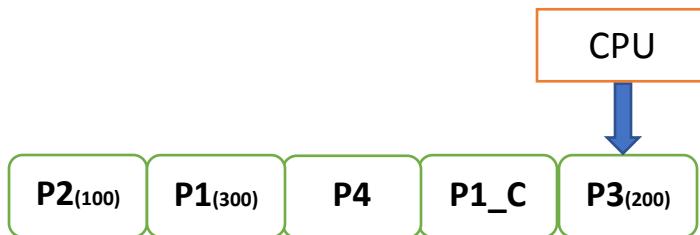
**T = 500**



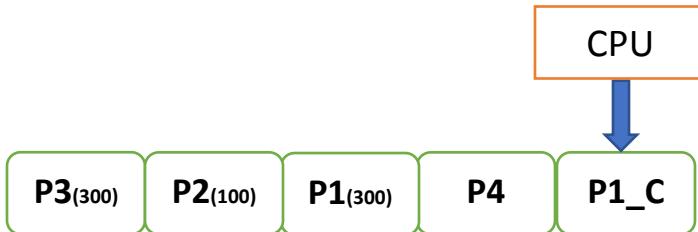
**T = 600**



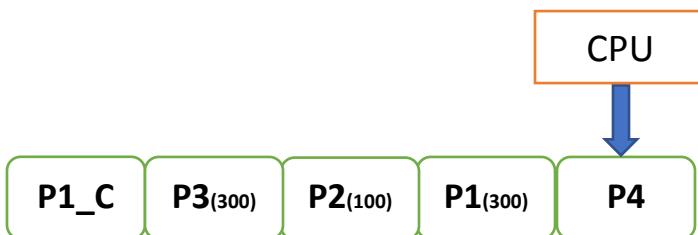
**T = 650**



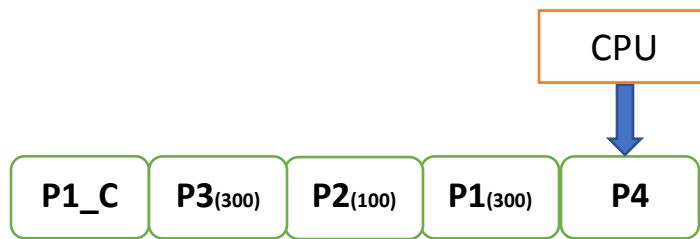
**T = 750**



**T = 900**



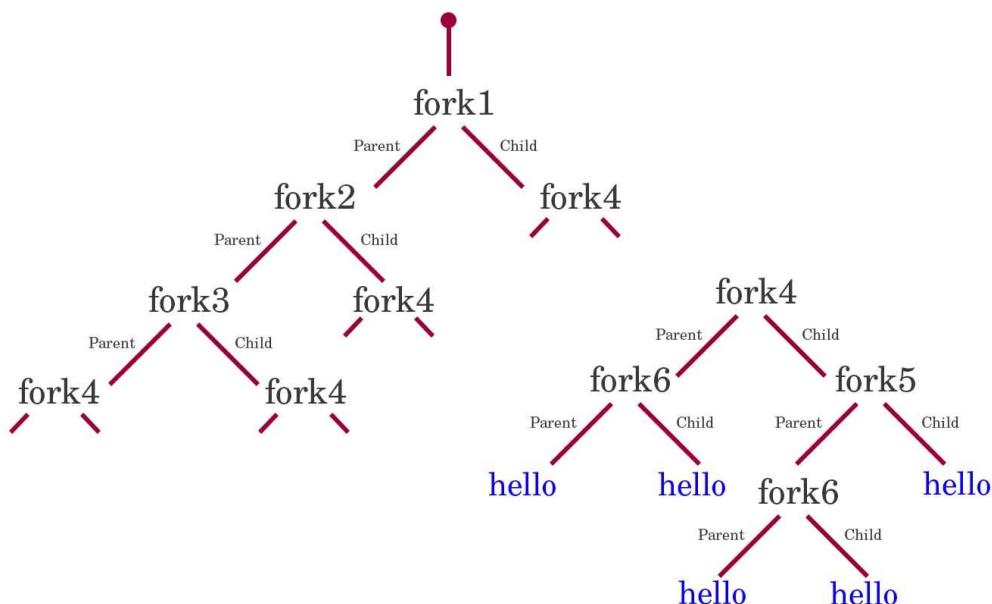
**T = 1000**



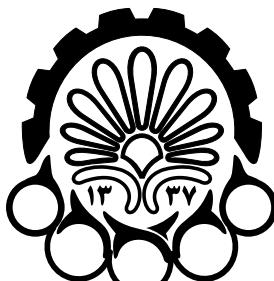
## سؤال ۵)

اگر به ترتیب از بالا به پایین و از سمت چپ به راست fork ها را شماره گذاری کنیم روند اجرای برنامه به صورت زیر خواهد بود:

با توجه به این شکل، عبارت شرطی اول که شامل سه fork است در نهایت منجر به ۴ بار اجرای fork4 می‌شود. یک بار اجرای fork4 باعث ۵ بار چاپ شدن عبارت hello می‌شود. پس در نهایت به اندازه ۴ ضربدر ۵ پردازه (۲۰ عدد) داریم که هر کدام در انتهای اجراشان عبارت hello را چاپ می‌کنند.



به نام خدا



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)

سیستم‌های عامل (پاییز ۱۴۰۰)

## تمرین سوم

استاد درس:

آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

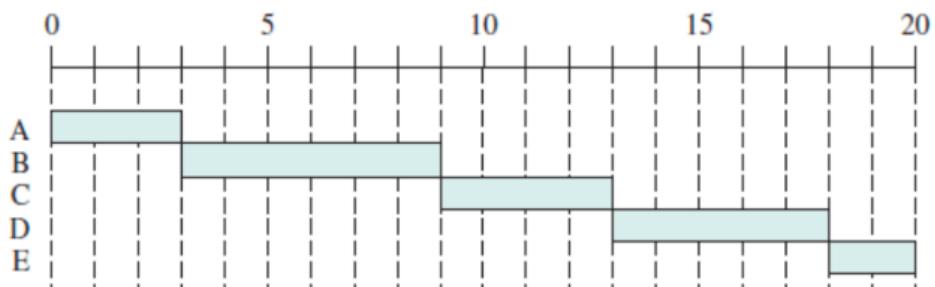
۹ آذر ۱۴۰۰ ساعت ۲۳:۵۹

نکته مهم: دقت کنید که تمدید نخواهیم داشت و صرفا می‌توانید ۱ تا ۵ روز از ۱۵ روز مجاز برای تاخیر ارسال تمامی تمرین‌های تئوری در این ترم را استفاده کنید. اگر بودجه ۱۵ روز شما تمام شود، به ازای هر روز تاخیر ۱۰ درصد از نمره تمرین را از دست خواهید داد.

۱) در سیستمی با شرایط زیر، زمان‌بندی پردازه‌ها را با استفاده از الگوریتم‌های SJF و RR و priority (به صورت non-pre-emptive) را مانند مثال زیر رسم کنید و زمان رفت و برگشت (turnaround time) و زمان انتظار (waiting time) را بدست آورید.

	زمان ورود	زمان سرویس	اولویت
A	0	8	2
B	1	1	1
C	3	4	3
D	3	6	4
E	3	2	2

زمان‌بندی را مانند مثال زیر رسم نمایید.



۲) پردازه‌های p1، p2 و p3 به ترتیب با زمان‌های (CPU burst) ۲ و ۴ و ۵ میلی ثانیه در زمان صفر وارد می‌شوند. چنانچه در آغاز اولویت آن‌ها عکس زمان اجراشان باشد و با گذر زمان به ازای هر ۰.۱ میلی ثانیه به اولویت پردازه‌های منتظر اضافه شود هر پردازه در چه زمانی به اتمام می‌رسد؟

۳) دو پردازه در لحظه صفر وارد یک سیستم تک پردازنده‌ای می‌شوند. هر پردازه در آغاز اجرای خود از یک نخ برخوردار است. با توجه به مفروضات زیر، زمان کامل اجرای نخ‌های پردازه‌ها را محاسبه کنید :

۱ - نخ اول پردازه اول که از ۳ میلی ثانیه زمان اجرا برخوردار است در انتهای هر ۱ میلی ثانیه از اجرای خود نخ جدیدی را با زمان اجرای ۲ میلی ثانیه تولید می‌کند. (multithreading)

۲—نخ اول پردازه دوم که از ۵ میلی ثانیه زمان اجرا برخوردار است، در انتهای هر ۲ میلی ثانیه از (multithreading) اجرای خود نخ جدیدی را با زمان اجرای ۳ میلی ثانیه تولید می‌کند.

۳\_الگوریتم زمانبندی استفاده شده RR با کوانتوم زمانی ۳ میلی ثانیه است.

۴\_سیاست اولویتدهی ما (Last Come First Served) است LCFS ۵\_زمان context switch برای پردازه‌ها ۱ میلی ثانیه و برای نخ‌ها ۰.۵ میلی ثانیه است.

۴) در قطعه کد زیر چه تعداد پردازه و ریسمان متمایز ساخته می‌شود؟

```
interleave () {  
    pthread_t th0, th1, th2;  
    int count=0;  
    pthread_create(&th0 , 0 , test , 0);  
    pthread_create(&th1 , 0 , test , 0);  
    pthread_create(&th2 , 0 , test , 0);  
    pthread_join(th0 , 0);  
    pthread_join(th1 , 0);  
    pthread_join(th2 , 0);  
    printf(count);  
}  
  
test () {  
    for (int j=0; j < 1000 ; j++)  
        count=count+1;  
}
```

۵) متوسط زمان برگشت (Waiting Time) و متوسط زمان انتظار (Turnaround Time) برای پردازهای جدول زیر را با هر یک از الگوریتم‌های زیر بدست آورید (شامل Gantt Chart). اندازه کوانتوم ۳ واحد زمان است و فرآیند وارد شده به سیستم می‌تواند فرآیند موجود را قبضه کند.(Preemptive Scheduling)

الف) Shortest Remaining Time First

ب) Shortest Job First

پ) Round-Robin

Process	Burst	Arrival
A	5	2
B	1	3
C	8	3
D	5	1
E	6	4

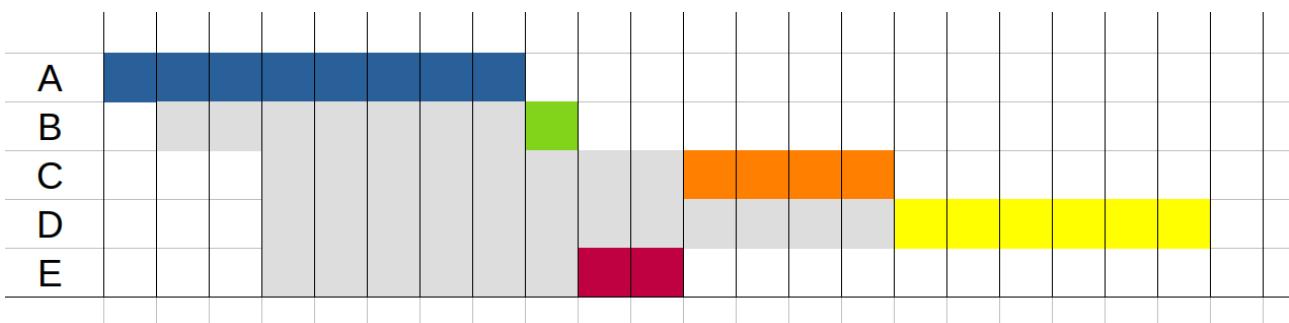
ت) کدام الگوریتم کمترین میزان متوسط زمان انتظار را تولید می‌کند؟ دلیل این امر چیست؟

موفق باشید

تیم درس سیستم‌های عامل

سؤال اول)

برای دو روش SJF و Priority نمودار به صورت زیر می‌شود:

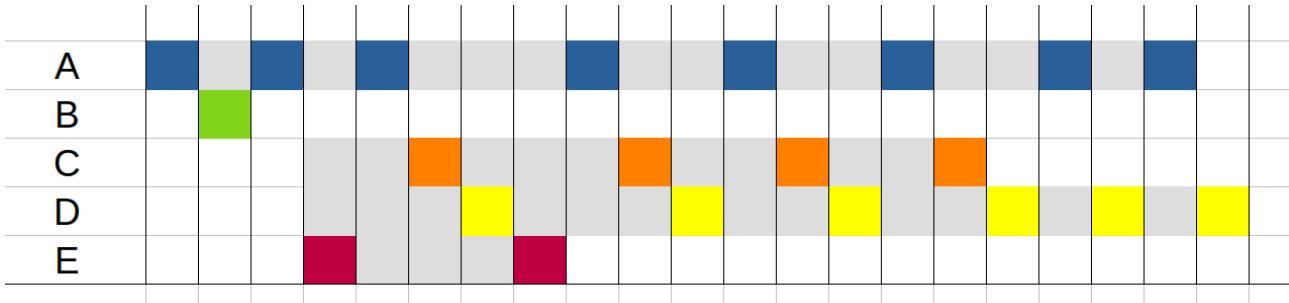


	Waiting Time	Turnaround Time
A	0	8
B	7	8
C	8	12
D	12	18
E	6	8

$$\text{Average Waiting Time} = (0 + 7 + 8 + 12 + 6) / 5 = 6.6$$

$$\text{Average Turnaround Time} = (8 + 8 + 12 + 18 + 8) / 5 = 10.8$$

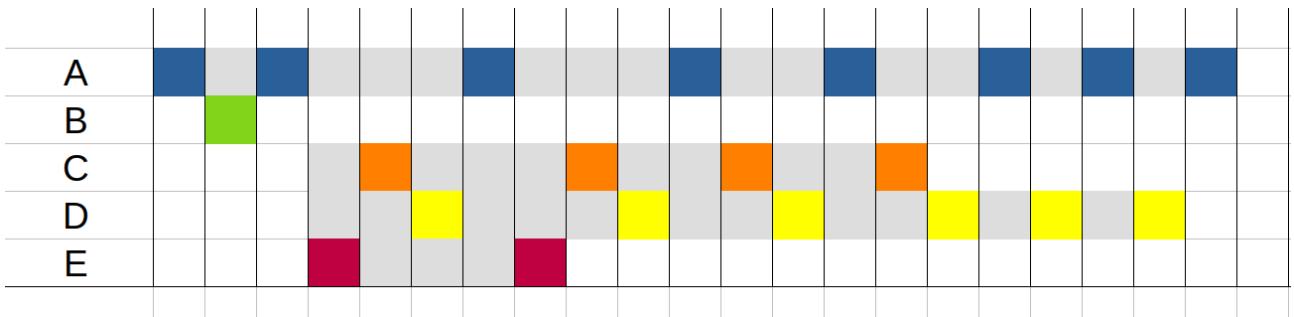
برای روش RR هر یک از نمودارهای زیر قابل قبول است. علاوه بر این‌ها اگر زمان ورود همه پردازه‌هارا صفر گرفته باشید نیز قابل قبول است.



	Waiting Time	Turnaround Time
A	12	20
B	0	1
C	9	13
D	12	18
E	3	5

$$\text{Average Waiting Time} = (12 + 0 + 9 + 12 + 3) / 5 = 7.2$$

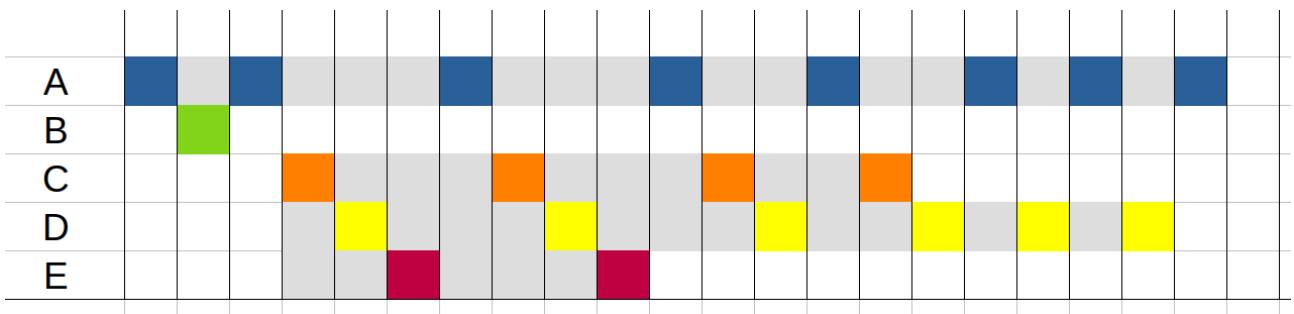
$$\text{Average Turnaround Time} = (20 + 1 + 13 + 18 + 5) / 5 = 11.4$$



	Waiting Time	Turnaround Time
A	13	21
B	0	1
C	8	12
D	11	17
E	3	5

$$\text{Average Waiting Time} = (13 + 0 + 8 + 11 + 3) / 5 = 7$$

$$\text{Average Turnaround Time} = (21 + 1 + 12 + 17 + 5) / 5 = 11.2$$



	Waiting Time	Turnaround Time
A	13	21
B	0	1
C	8	12
D	11	17
E	5	7

$$\text{Average Waiting Time} = (13 + 0 + 8 + 11 + 5) / 5 = 7.4$$

$$\text{Average Turnaround Time} = (21 + 1 + 12 + 17 + 7) = 11.6$$

سوال (۲)

	0		5		10	
P1	0.5	0.5				
P2	0.25	0.35	0.35	0.45	0.45	0.55
P3	0.2	0.3	0.4	0.4	0.5	0.5

Processes	Turn-around time	Waiting time
P1	2	0
P2	9	5
P3	11	6

با تشکر از آقای امیرحسین فضل المئی

سوال (۳)

با فرض محاسبه شدن context switch نخ های پردازه ها در quantum time زمان بندی زیر به دست می آید :

0 - 1	1 - 1.5	1.5 - 3	3 - 4	4 - 6	6 - 6.5	6.5 - 7	7 - 8	8 - 8.5	8.5 - 9
T11	CST	T12	CSP	T21	CST	T22	CSP	T12(F)	CST
9 - 10	10 - 10.5	10.5 - 11	11 - 12	12 - 14.5	14.5 - 15	15 - 16	16 - 17.5	17.5 - 18	18 - 19
T11	CST	T13	CSP	T22(F)	CST	CSP	T13(F)	CST	T11(F)
19 - 20	20 - 22	22 - 22.5	22.5 - 23	23 - 24	24 - 26	26 - 27	27 - 29.5	29.5 - 30	30 - 31
CSP	T21	CST	T23	CSP	T14(F)	CSP	T23(F)	CST	T21(F)

به معنای context switch thread است CST

به معنای context switch process است CSP

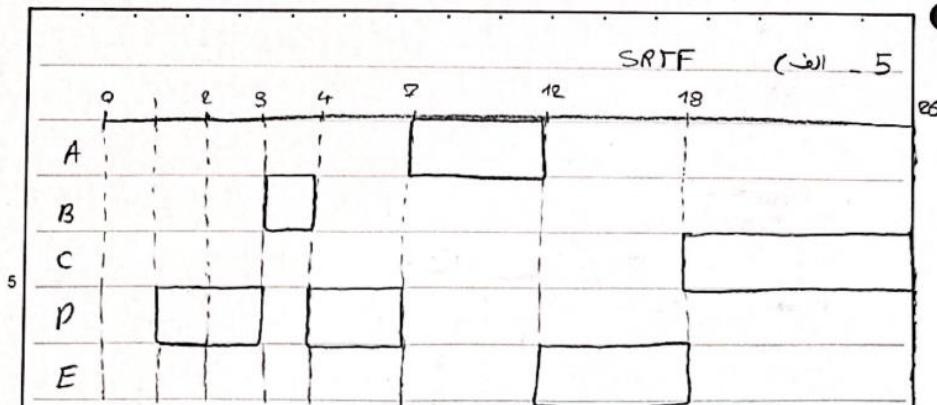
T11 به معنای thread اصلی آن پردازه است

(F) به معنای پایان یافتن آن thread است

سوال (۴)

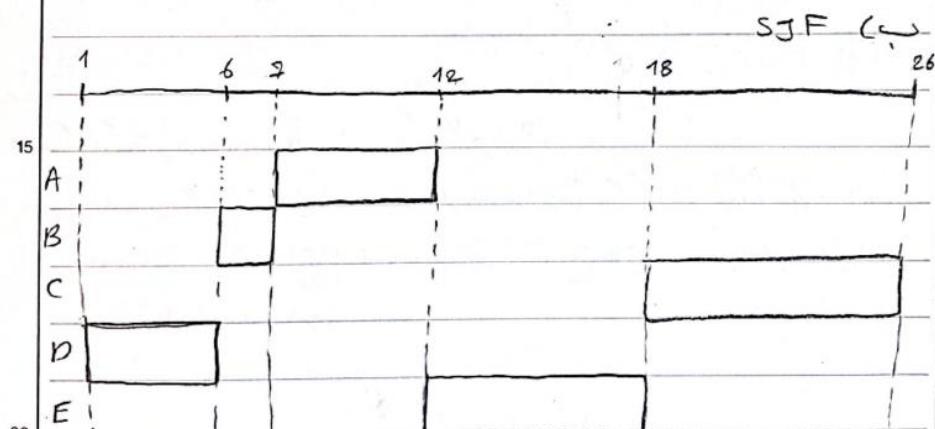
در ابتدا یک پردازه اصلی داریم. چون تابع فورک صدا زده نشده پردازه جدیدی ایجاد نمیشود. هر پردازه حداقل یک ترد اصلی دارد و بنابراین پردازه اصلی یک ترد اصلی دارد. در این کد ۳ ترد جدید با توجه به pthread\_create ها ایجاد میشود. در مجموع نیز ۴ ترد و ۱ پردازه داریم.

سؤال (٥)



$$\text{average waiting time} = \frac{A + B + C + D + E}{5} = 5.8$$

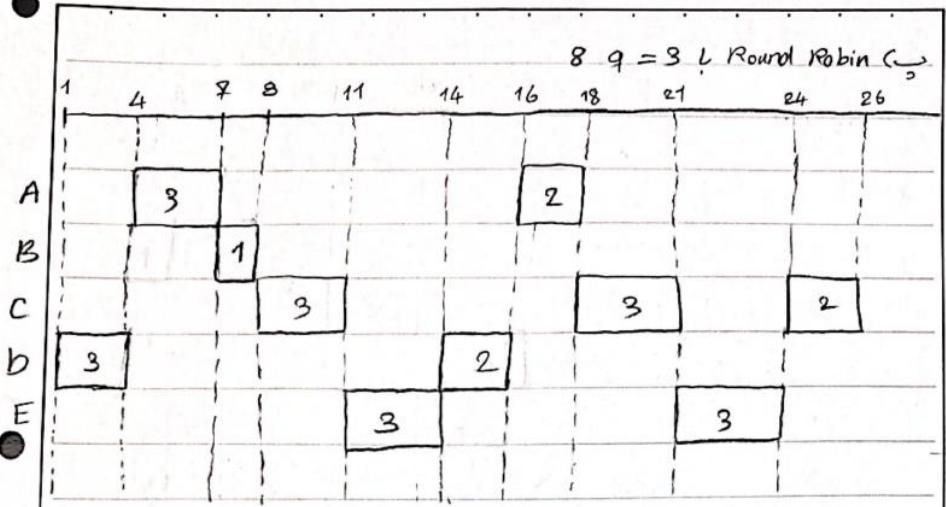
$$\text{average turnaround time} = \frac{10 + 1 + 23 + 6 + 14}{5} = 10.2$$



$$\text{average waiting time} = \frac{A + B + C + D + E}{5} = 6.2$$

$$\text{average turnaround time} = \frac{10 + 4 + 23 + 5 + 14}{5} = 11.2$$





$$\text{average waiting time} = \frac{A + B + C + D + E}{5} = 10.8$$

$$\text{average turnaround time} = \frac{A + B + C + D + E}{5} = 15.8$$

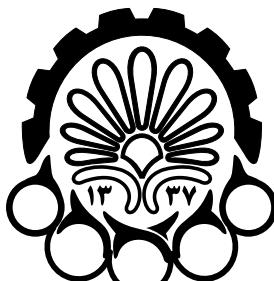
کدام کمترین زمان انتظار را تولید می‌کند؟

SRTF - زیرا هر بار آن بردازه‌ها را با حفظ کمترین <sup>remaining</sup> CPU burst دارد.

نه تن باید صورت ممکن سود و الورخ  
optional SRTF Waiting time  
(قابلیت پاغن)



به نام خدا



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)

سیستم‌های عامل (پاییز ۱۴۰۰)

## تمرین چهارم

استاد درس:

آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

۳۰ آذر ۱۴۰۰ ساعت ۵۹:۲۳

نکته مهم: دقت کنید که تمدید نخواهیم داشت و صرفا می‌توانید ۱ تا ۵ روز از ۱۵ روز مجاز برای تاخیر ارسال تمامی تمرین‌های تئوری در این ترم را استفاده کنید. اگر بودجه ۱۵ روز شما تمام شود، به ازای هر روز تاخیر ۱۰ درصد از نمره تمرین را از دست خواهید داد.

۱) سیستمی با ۵ پردازه و ۴ منبع در حالت زیر قرار دارد:

Allocation					Max				
	A	B	C	D		A	B	C	D
P <sub>0</sub>	0	0	1	2	P <sub>0</sub>	0	0	1	2
P <sub>1</sub>	1	0	0	0	P <sub>1</sub>	1	7	5	0
P <sub>2</sub>	1	3	5	4	P <sub>2</sub>	2	3	5	6
P <sub>3</sub>	0	6	3	2	P <sub>3</sub>	0	6	5	2
P <sub>4</sub>	0	0	1	4	P <sub>4</sub>	0	6	5	6

Available				
A	B	C	D	
1	5	2	0	

الف ) محاسبه کنید که آیا سیستم در حالت امن است؟

ب ) اگر درخواستی از پردازه p<sub>1</sub> به صورت (0,4,2,0) به شکل تعداد منابع درخواستی برای (A, B, C, D) وارد شود آیا می توان فورا به آن تخصیص داد؟ بعد از تخصیص، سیستم در وضعیت امن قرار میگیرد؟

۲) می دانیم که الگوریتم پترسون تنها برای 2 پردازه کار می کند. این الگوریتم را برای N پردازه بازنویسی کنید و شروط 3 گانه (انحصار متقابل، پیشرفت و انتظار محدود) را در الگوریتم جدید بررسی کنید.

۳) در ادامه یک قطعه کد برای تعمیم الگوریتم Peterson برای سه فرایند ارائه شده است. این راه حل را با توجه به سه شرط انحصار متقابل، پیشرفت و انتظار محدود بررسی کنید. هر کدام از پردازه‌ها برای ورود به ناحیه بحرانی بایستی `enter_region` را صدای زده و بلافصله بعد از خروج از ناحیه بحرانی `leave_region` را صدای بزنند.

```
void enter_region(int process) {
    int other1 = (process + 1) % 3;
    /* other1 and other 2 are the ids */
    int other2 = (process + 2) % 3;
    /* of the other two processes */
    interested[process] = TRUE;
    turn = other1;
    while (turn != process && (interested[other1]
    || interested[other2]));
    /* do nothing */
}

void leave_region(int process) {
    interested[process] = false;
}
```

۴) سیستم ما داری 4 منبع A, B, C, D و به ترتیب 6, 4, 4, 2 تا از هر کدام از این منابع را دارا است. با استفاده از الگوریتم بانکدار به سوالات زیر پاسخ دهید؟

		اختصاص داده شده				حداکثر نیاز				مورد نیاز				موجود			
		A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	1	1	3	2	1	1					6	4	4	2	
P1	1	1	0	0	1	2	0	2									
P2	1	0	1	0	3	2	1	0									
P3	0	1	0	1	2	1	0	1									

الف) در ابتدا حساب کنید کدام یک از پردازه‌ها می‌تواند درخواست منابع کند (الگوریتم بانکدار را انجام دهید)؟

ب) ایا سیستم ما به بن بست می‌خورد؟ چرا؟

ج) ایا سیستم ما امن هست؟ چرا؟

د) اگر  $p_3$  در این لحظه (2,1,0,0) منبع درخواست کند، آیا می‌توانیم درخواستش را اجابت کنیم؟

ه) یک مثال دیگر مانند قسمت د بزنید و الگوریتم بانکدار را جلو ببرید.

۵) شروط انحصار متقابل، پیشرفت و انتظار محدود را برای الگوریتم های 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);
```

(ب)

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

موفق باشید

تیم درس سیستم‌های عامل

(سؤال اول)

۱) سیستمی با ۵ پردازه و ۴ منبع در حالت زیر قرار دارد:

Allocation				
	A	B	C	D
P <sub>0</sub>	0	0	1	2
P <sub>1</sub>	1	0	0	0
P <sub>2</sub>	1	3	5	4
P <sub>3</sub>	0	6	3	2
P <sub>4</sub>	0	0	1	4

Max				
	A	B	C	D
P <sub>0</sub>	0	0	1	2
P <sub>1</sub>	1	7	5	0
P <sub>2</sub>	2	3	5	6
P <sub>3</sub>	0	6	5	2
P <sub>4</sub>	0	6	5	6

Available				
	A	B	C	D
	1	5	2	0

الف) محاسبه کنید که آیا سیستم در حالت امن است؟ بله، زیرا:

	allocation	Max	available	Safety check
P <sub>0</sub>	<0,0,1,2>	<0,0,1,2>	<1,5,2,0>	<0,0,0,6> ✓
P <sub>1</sub>	<1,0,0,0>	<1,7,5,0>	<1,5,3,2>	<0,7,5,0> ✓
P <sub>2</sub>	<1,3,5,4>	<2,3,5,6>	<1,1,6,4>	<1,0,0,2> ✓
P <sub>3</sub>	<0,6,3,2>	<0,6,5,2>	<2,14,11,8>	<0,0,2,0> ✓
P <sub>4</sub>	<0,0,1,4>	<0,6,5,6>	<3,14,11,18>	<0,6,4,2> ✓
			<3,14,12,22>	

Need[i,j] = Max[i,j] - Allocation[i,j] ← / راهساب مارکیم ابتدا

حال برسی مارکیم با استفاده از متتابع  $\Rightarrow$  درخواست های را مارکیم انجام دهیم

$$\begin{array}{r}
 + 1 \ 5 \ 2 \ 0 \\
 0 \ 0 \ 1 \ 2 \\
 \hline
 1 \ 5 \ 3 \ 2
 \end{array}
 \quad
 \leftarrow \text{available} = \text{available} + \begin{array}{l} P_0 \\ \text{انتخاب} \end{array}
 \quad
 \begin{array}{r}
 + 1 \ 1 \ 6 \ 4 \\
 1 \ 3 \ 5 \ 4 \\
 \hline
 2 \ 14 \ 11 \ 8
 \end{array}
 \quad
 \leftarrow \text{available} = \text{available} + \begin{array}{l} P_1 \\ \text{انتخاب} \end{array}
 \quad
 \begin{array}{r}
 + 1 \ 5 \ 3 \ 2 \\
 0 \ 6 \ 3 \ 2 \\
 \hline
 1 \ 11 \ 6 \ 4
 \end{array}
 \quad
 \leftarrow \text{available} = \text{available} + \begin{array}{l} P_2 \\ \text{انتخاب} \end{array}$$

$$\begin{array}{r}
 + 2 \ 14 \ 11 \ 18 \\
 1 \ 0 \ 0 \ 0 \\
 \hline
 3 \ 14 \ 11 \ 18
 \end{array}
 \quad
 \leftarrow \text{available} = \text{available} + \begin{array}{l} P_3 \\ \text{انتخاب} \end{array}
 \quad
 \begin{array}{r}
 + 3 \ 14 \ 11 \ 18 \\
 0 \ 0 \ 1 \ 4 \\
 \hline
 3 \ 14 \ 12 \ 22
 \end{array}
 \quad
 \leftarrow \text{available} = \text{available} + \begin{array}{l} P_4 \\ \text{انتخاب} \end{array}$$

safety در این سیستم  $\langle P_0, P_1, P_2, P_3, P_4 \rangle$  را باعثیت سرت

اجر ایتم  $\Rightarrow$  True safe  $\Leftarrow$  اجر ایتم

ب) اگر درخواستی از پردازه  $p_1$  به صورت  $(0,4,2,0)$  به شکل تعداد منابع درخواستی برای  $(A, B, C, D)$  وارد شود آیا می‌توان فوراً به آن تخصیص داد؟ بعد از تخصیص، سیستم در وضعیت امن قرار می‌گیرد؟

	allocation	Max	Need	Available
$P_0$	$\langle 0, 0, 1, 2 \rangle$	$\langle 0, 0, 1, 2 \rangle$	$\langle 0, 0, 0, 0 \rangle$	$\langle 1, 5, 1, 0 \rangle$
$P_1$	$\cancel{\langle 1, 4, 2, 0 \rangle}$	$\langle 1, 7, 5, 0 \rangle$	$\cancel{\langle 0, 7, 5, 0 \rangle}$	$\cancel{\langle 1, 4, 0, 0 \rangle}$
$P_2$	$\langle 1, 3, 5, 4 \rangle$	$\langle 2, 3, 5, 6 \rangle$	$\langle 1, 0, 0, 2 \rangle$	$\cancel{\langle 1, 1, 0, 2 \rangle}$
$P_3$	$\langle 0, 6, 3, 2 \rangle$	$\langle 0, 6, 5, 2 \rangle$	$\langle 0, 0, 2, 0 \rangle$	$\cancel{\langle 2, 4, 0, 6 \rangle}$
$P_4$	$\langle 0, 0, 1, 4 \rangle$	$\langle 0, 6, 5, 6 \rangle$	$\langle 0, 6, 4, 2 \rangle$	$\langle 2, 10, 9, 8 \rangle$

request  $\langle 0, 4, 2, 0 \rangle$  from  $P_0$

ابتدا بیشترین راهکار کنیم:

$$\text{Request}_i \leq \text{Need}_i \rightarrow \langle 0, 4, 2, 0 \rangle \leq \langle 0, 7, 5, 0 \rangle \checkmark$$

$$\text{Request}_i \leq \text{Available} \rightarrow \langle 0, 4, 2, 0 \rangle \leq \langle 1, 5, 2, 0 \rangle \checkmark$$

$$\begin{array}{r} 1 \ 5 \ 2 \ 0 \\ - 0 \ 4 \ 2 \ 0 \\ \hline 1 \ 1 \ 0 \ 0 \end{array} \quad \begin{array}{r} 1 \ 0 \ 0 \ 0 \\ + 0 \ 4 \ 2 \ 0 \\ \hline 1 \ 4 \ 2 \ 0 \end{array} \quad \begin{array}{r} 0 \ 7 \ 5 \ 0 \\ - 0 \ 4 \ 2 \ 0 \\ \hline 0 \ 3 \ 3 \ 0 \end{array}$$

$$\begin{array}{r} 2 \ 4 \ 6 \ 6 \\ + 0 \ 6 \ 3 \ 2 \\ \hline 2 \ 1 \ 0 \ 9 \ 8 \end{array} \xleftarrow{\text{انتخاب } P_3} \begin{array}{r} 1 \ 1 \ 1 \ 2 \\ + 1 \ 3 \ 5 \ 4 \\ \hline 2 \ 4 \ 6 \ 6 \end{array} \xleftarrow{\text{انتخاب } P_2} \begin{array}{r} 0 \ 0 \ 1 \ 2 \\ + 1 \ 1 \ 0 \ 0 \\ \hline 1 \ 1 \ 1 \ 2 \end{array} \xleftarrow{\text{انتخاب } P_0} \text{حال را اجرا مکنیم: safety check}$$

$$\begin{array}{r} 2 \ 1 \ 0 \ 9 \ 8 \\ + 0 \ 0 \ 1 \ 4 \\ \hline 2 \ 1 \ 0 \ 1 \ 2 \end{array} \xleftarrow{\text{انتخاب } P_4} \begin{array}{r} 3 \ 1 \ 4 \ 1 \ 1 \ 8 \\ + 1 \ 4 \ 2 \ 0 \\ \hline \end{array} \xleftarrow{\text{انتخاب } P_1}$$

در حال حاضر safe نیست.

با توجه به این نتیجه نتوان تخصیص داد.

با تشکر از خانم مینا بیکی

## سوال ۲

یک صف (queue) مجازی به اندازه تعداد process ها می‌سازیم در اینجا چون N عدد داریم به اندازه N. هر پردازه ایی که بخواهد وارد ناحیه بحرانی (critical section) شود به این صف وارد می‌شود سپس با توجه به مکانیزم صف موقعی که زمان خارج شدنش از صف فرا رسید اجازه ورود به ناحیه بحرانی را پیدا می‌کند.

بدین منظور دو تابع به نام های lock, unlock تعریف می‌کنیم و با اجرای تابع lock برای هر process باعث می‌شویم تا زمانی که تا اخر صف پیش نرقته است وارد ناحیه بحرانی نشود و بعد از آن که همه process های جلوتر کارشان تمام شد و به اخر صف رسیدیم وارد ناحیه بحرانی می‌شود و در نهایت unlock را اجرا می‌کند.

```
LOCK( Process  
      PID) {
```

```
for( int i = 0; i < N; ++i ) {  
    Turn[i] = PID;  
    Flag[PID] = i;  
    while( (for all k != PID, Flag[k] < i) or (Turn[i]  
!= PID))  
        ;  
    }  
/* Critical Section */  
UNLOCK(Process PID){  
    Flag[PID] = -1  
    else  
    Flag[PID] = 0  
}
```

اکنون شروط ۳ گانه را بررسی می‌کنیم :

شرط انحصار متقابل (mutual exclusion) : با توجه به اینکه در هر لحظه فقط یک process میتواند در انتهای صف باشد بنابراین پس فقط یک process میتواند در ناحیه بحرانی باشد پس این شرط برقرار است.

شرط پیشرفت (progress) : با توجه به اینکه هر process پس از خروجش از ناحیه بحرانی با اجرای تابع unlock باید flag خود را ۱- کند پس این اجازه را به بقیه process ها میدهد که وارد ناحیه بحرانی بشوند بنابراین این شرط نیز برقرار است.

شرط انتظار محدود (bounded waiting) : با توجه به اینکه سایز صف را میدانیم امکان ندارد بیش از ۱ - N پردازه طول بکشد که نوبت process جدید شود چون هر process ایی که خارج می‌شود با false کردن نوبت خود باعث می‌شود بقیه process ها بتوانند وارد شوند و اگر بخواهد دوباره وارد شود دوباره

باید به اول صفت بروود پس نمیتواند بی نهایت بار وارد و خارج شود و موجب انتظار باقی process ها شود بنابراین زمان محدودی طول میکشد تا هر process وارد ناحیه بحرانی بشود.

### سوال (۳)

الگوریتم ارائه شده انحصار متقابل ندارد ، پیش روی ندارد ، انتظار محدود دارد.

انحصار متقابل نداریم ، برای مثال:

فرایند ۰ وارد enter\_region می شود ، به دلیل برقرار نبود شرط:

Interested[other1] || interested[other2]

وارد بخش بحرانی می شود. سپس فرایند ۲ وارد این تابع شده و تا خط قبل از while اجرا می شود. سپس

فرایند ۱ وارد شده و تا خط قبل از while اجرا می شود. حال مقدار turn برابر id فرایند ۲ است و به

همین خاطر شرط  $turn \neq process$  برای فرایند ۲ برقرار نیست و وارد ناحیه بحرانی می شود (که فرایند

۰ هم هنوز ممکن است داخل آن باشد)

پیش روی نداریم ، برای مثال:

فرض کنید دو فرایند ۰ و ۱ درخواست ورود به بخش بحرانی را داشته باشند اما فرایند ۲ نه. درین صورت

هرگز فرایند های ۰ و ۱ از خط while عبور نخواهند کرد. چرا که turn برابر فرایند ۲ است و بخش

[interested] || other2[interested]

انتظار محدود داریم چرا که تعداد دفعاتی که یک فرایند می تواند پشت سر هم وارد ناحیه بحرانی شود

محدود است و اگر فرایند بعد درخواست ورود داشته باشد ، همان فرایند وارد خواهد شد (پس دچار قحطی

نمی شویم) چرا که با هر دفعه ورود فرایند قبلی turn به فرایند بعدی داده می شود ، هرچند برای ورود

فرایند بعدی نیاز به ورود فرایند قبلی هم داریم که مشکل عدم پیش روی را ایجاد می کرد.

#### سوال (۴)

(۴) سیستم ما داری ۴ منبع A, B, C, D و به ترتیب ۶, ۴, ۴, ۲ تا از هر کدام از این منابع را دارا است. با استفاده از الگوریتم بانکدار به سوالات زیر پاسخ دهید؟

$$\text{Available} = \langle 6, 4, 4, 2 \rangle$$

$\max$

$\text{Need}$

	احتصار داده شده				حداکثر نیاز				مورد نیاز				موجود			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	1	1	3	2	1	1	1	2	0	0	6	4	4	2
P1	1	1	0	0	1	2	0	2	0	1	0	2				
P2	1	0	1	0	3	2	1	0	2	2	0	0				
P3	0	1	0	1	2	1	0	1	2	0	0	0				

الف) در ابتدا حساب کنید کدام یک از پردازه‌ها می‌تواند درخواست منابع کند (الگوریتم بانکدار را انجام دهید)؟

ابتدا  $\text{Need} = \max - \text{alloc}$  را حساب می‌کنیم که از طبق نیاز دسترسی آن.

حال بازجنبه به حوصله  $R$  ها بررسی کلیم کلام  $P_i$  را توانیم اطمینان دهیم که ریاضی بسطه  $\text{Need} \leq \text{Available}$  است (نهایی متراند درخواست مبالغه کنند). برای اینجا از توانیم Safety راهنم حساب کلیم کدرست محاسبه کرد.

ب) ایا سیستم ما به بن بست می‌خورد؟ چرا؟

ضریزه زیرا اگر از تک پردازنده سرچرخ لیم دباشیم  $\text{need}$ ، منابع را احتفاظ نمی‌کنیم بردازه‌های بعدی نیز مبالغه با ارزش‌های طاف داریم  $\rightarrow$  سیستم در حالات Safety check نیز توانیم به این نتیجه برسیم.

ج) ایا سیستم ما امن هست؟ چرا؟

بله، همانطور که در صفت ب استاد کردم با اجرای الگوریتم safety check مطمن شویم:

$$\begin{array}{r}
 \text{انتخاب } P_1 \leftarrow \\
 + \begin{array}{rrrr}
 1 & 2 & 0 & 3 \\
 1 & 1 & 0 & 0 \\
 \hline
 9 & 0 & 0 & 3
 \end{array} \\
 \text{انتخاب } P_2 \leftarrow \\
 + \begin{array}{rrrr}
 6 & 4 & 2 & 0 \\
 2 & 0 & 1 & 1 \\
 \hline
 8 & 4 & 0 & 3
 \end{array} \\
 \text{انتخاب } P_3 \leftarrow \\
 + \begin{array}{rrrr}
 10 & 0 & 6 & 3 \\
 0 & 1 & 0 & 1 \\
 \hline
 10 & 0 & 4 & 2
 \end{array} \\
 \text{انتخاب } P_0 \leftarrow \\
 + \begin{array}{rrrr}
 9 & 0 & 0 & 3 \\
 1 & 0 & 1 & 0 \\
 \hline
 10 & 0 & 4 & 3
 \end{array}
 \end{array}$$

د) اگر  $P_3$  در این لحظه (2, 1, 0, 0) منبع درخواست کند، آیا می‌توانیم درخواستش را اجابت کنیم؟ حیره زیرا:

ابتدا پس سیرایه را حکم می‌کنیم:

$$\text{Request}_i \leq \text{Need}_i \rightarrow \langle 2, 1, 0, 0 \rangle \leq \langle 2, 0, 0, 0 \rangle \quad X$$

$$\text{Request}_i \leq \text{Available} \rightarrow \langle 2, 1, 0, 0 \rangle \leq \langle 6, 4, 2, 2 \rangle \quad \checkmark$$

سرطانی برقرار نیست، یعنی مقادیر مناسب که در خواسته از مقدار کافی بیانز دارد و بیتل محدود نمایند بود در  $\max$  بیشتر است  $\rightarrow$  پس نمی‌توانیم درخواستش را اجابت کنیم.

۵) یک مثال دیگر مانند قسمت د بزنید و الگوریتم بانکدار را جلو ببرید.

آخر پ= دخواست <۱,۱,۰,۰> نهد:

$$\text{Request}_i \leq \text{Need}_i \rightarrow <۱,۱,۰,۰> \leq <۱,۲,۰,۰> \checkmark$$

$$\text{Request}_i \leq \text{Available} \rightarrow <۱,۱,۰,۰> \leq <۶,۴,۴,۲>$$

حال چه میکنیم آیا سیستم درحال این است؟

$$\begin{array}{r} 1 \ 2 \ 0 \ 0 \\ - 1 \ 1 \ 0 \ 0 \\ \hline 0 \ 1 \ 0 \ 0 \end{array} \quad + \quad \begin{array}{r} 2 \ 0 \ 1 \ 1 \\ 1 \ 1 \ 0 \ 0 \\ \hline 3 \ 1 \ 1 \ 1 \end{array} \quad - \quad \begin{array}{r} 1 \ 2 \ 0 \ 0 \\ - 1 \ 1 \ 0 \ 0 \\ \hline 0 \ 1 \ 0 \ 0 \end{array}$$

available ↗

alloc ↗

need ↗

P. <2, 0, 1, 1>

<3, 2, 1, 1>

<1, 2, 0, 0>

<0, 1, 0, 0>

P<sub>1</sub> <1, 1, 0, 0>

<1, 2, 0, 2>

<0, 1, 0, 2>

P<sub>2</sub> <1, 0, 1, 0>

<3, 2, 1, 0>

<2, 2, 0, 0>

P<sub>3</sub> <0, 1, 0, 1>

<2, 1, 0, 1>

<2, 0, 0, 0>

سیستم درحال این ترتیب نیست.

نیاز دخواست را میگیرد.

با تشکر از خاتم مینا بیکی

سؤال (۵)

(الف)

انحصار متقابل:

وجود دارد، فرآیندها برای ورود به ناحیه‌ی بحرانی نیاز دارند که نوبت به آن‌ها تعارف شود.  
فرض می‌کنیم فرآیند  $z$  زودتر به حلقه while رسیده است. پس از آنکه فرآیند  $z$  نوبت را به فرآیند  $j$  تعارف کرد، فرآیند  $j$  وارد ناحیه‌ی بحرانی می‌شود. اما فرآیند  $j$  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] != 0، 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 ناحیه‌ی 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)

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.

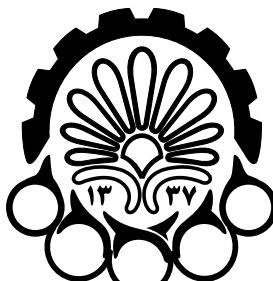
انتظار محدود:

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

راه حل ساده‌تر:

این الگوریتم همان الگوریتم پترسون است زیرا تنها اندیس‌های  $i$  و  $j$  در همه کاربردهای متغیر flag باهم جا به جا شده‌اند. پس همه شروط برقرار هستند.

به نام خدا



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)

سیستم‌های عامل (پاییز ۱۴۰۰)

## تمرین پنجم

استاد درس:

آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

۱۷ دی ۱۴۰۰ ساعت ۲۳:۵۹

نکته مهم: دقت کنید که تمدید نخواهیم داشت و صرفا می‌توانید ۱ تا ۵ روز از ۱۵ روز مجاز برای تاخیر ارسال تمامی تمرین‌های تئوری در این ترم را استفاده کنید. اگر بودجه ۱۵ روز شما تمام شود، به ازای هر روز تاخیر ۱۰ درصد از نمره تمرین را از دست خواهید داد.

(۱) حملات buffer overflow را میتوان به روش :  
سخت افزاری (special hardware support)  
یا نرم افزاری (better programming methodology)  
رفع کرد. هر دوی این راه حل ها را توضیح دهید و با یکدیگر از جنبه های مختلف مقایسه کنید.

(۲) اگر نرخ برخورد (hit) برای TLB در سیستمی برابر ۹۵ درصد باشد و این سیستم به ۱۰ نانوثانیه زمان برای دسترسی به TLB و ۱۰۰ نانوثانیه زمان برای دسترسی به حافظه نیاز داشته باشد، زمان موثر دسترسی به حافظه (EAT) در این سیستم را حساب کنید.

(۳) یکی از قابلیت‌های سیستم عامل UNIX این است که به کاربر اجازه می‌دهد یک watchdog program برای هر فایل دلخواه اختصاص دهد که وظیفه آن این است هر زمان برنامه‌ای درخواست دسترسی به آن فایل را داشته باشد، فعال شود و درخواست برنامه را رد یا تایید کند. دو مورد از مزایا و دو مورد از معایب watchdog program را در بحث امنیت را توضیح دهید.

(۴) اگر لیست حفره‌های درون حافظه (تکه‌های خالی) از چپ به راست برابر ۲KB، ۷KB، ۵KB، ۸KB، ۱0KB، ۱0KB، ۳0KB باشد، اگر فرآیندهایی با اندازه‌های ۲5KB، ۸KB و ۹KB (از راست به چپ) را در حافظه بارگذاری گردد، نتیجه هر یک از الگوریتم‌های زیر را مشخص کنید (برای روش تشخیص حافظه پیوسته). منظور از نتیجه، حفره انتخاب شده (در صورت امکان) و حفره‌های باقیمانده در اثر تشخیص حافظه است.

الف) First Fit

ب) Best Fit

ج) Worst Fit

۵) یک سیستم صفحه بندی بر اساس تقاضا (demand-paging) را با داده‌های محاسبه شده زیر برای بهره‌وری براساس زمان درنظر بگیرید.

CPU utilization: 20%

Paging disk: 97.7%

Other I/O devices: 5%

برای هر یک از گزاره‌های زیر توضیح دهید که آیا بهره‌وری CPU افزایش می‌یابد یا خیر:

الف) نصب یک CPU سریع‌تر

ب) نصب یک paging disk بزرگ‌تر

ج) افزایش اندازه multiprogramming سیستم

د) کاهش اندازه multiprogramming سیستم

ه) افزایش حافظه main memory

و) نصب یک دیسک سریع‌تر

ز) افزایش page size

موفق باشد

تیم درس سیستم‌های عامل

## سوال ۱

یک مدل از hardware support که اطمینان میدهد که حملات buffer overflow اتفاق نیوفتد این است که از اجرای کدی که در بخش stack از فضای آدرس پردازه قرار دارد، جلوگیری کند. میدانیم که حملات buffer overflow با سرریز کردن buffer در داخل stack frame به وقوع میپیوندد. در واقع با Overwrite کردن آدرس برگشتی یک تابع و سپس در نتیجه‌ی آن پریدن به قسمت دیگری از stack frame که برنامه بدخواه و خرابکار در آن قرار دارد که در نتیجه‌ی buffer overflow در آنجا قرار گرفته، به وقوع میپیوندد. با جلوگیری از اجرای کد در stack segment این مشکل حل میشود.

رویکرد دیگری که از یک better programming methodology استفاده میکند به صورت معمول با توجه به استفاده bounds-checking ساخته میشوند تا در برابر buffer overflow محافظت کنند. Buffer overflow معمولاً در زبان‌های مثل جاوا که هر دسترسی آرایه‌ای تضمین میشود در محدوده‌ای که نرم افزار میتواند چک کند است، رخ نمیدهد. این ویکردها به پشتیبانی سخت افزاری نیاز ندارند ولی خب هزینه‌ی اجرایی دارد که مرتبط با اجرای bounds-checking میباشد.

## سوال ۲

در ابتدا برای دسترسی به TLB به ۱۰ نانوثانیه زمان نیاز داریم. اگر TLB Miss رخ دهد در این صورت باید ابتدا از حافظه و از Page Table آدرس فیزیکی را لود کنیم که خود ۱۰۰ نانوثانیه زمان نیاز دارد. سپس دسترسی به این آدرس به ۱۰۰ نانوثانیه دیگر زمان نیاز دارد. اگر hit رخ دهد در این صورت فقط به یک ۱۰ نانوثانیه دیگر زمان نیاز داریم.

$$EAT = 0.95*(10 + 100) + 0.05*(100 + 100 + 10) = 115 \text{ ns}$$

## سوال ۳

اولین مزیت watchdog این است که یک سیستم مرکزی (centralized) است و زمانی که برای تمامی فایل‌ها اعمال شود اگر بخواهیم تغییری در آن ایجاد کنیم به راحتی قابل انجام است و نیازی به فعال سازی دوباره برای تک به تک فایل‌ها نیست. مزیت دوم آن این است که چون اولین برنامه ایی است که برای دسترسی به فایل با آن مواجه میشویم، اگر بتوان اطمینان حاصل کرد که تکنیک‌های به کار رفته در آن قابل اعتماد هستند این برنامه میتواند گارانتی بگذارد که تمامی فایل‌های ما در وضعیت امن قرار خواهند داشت.

اولین عیب watchdog دقیقاً به مزیت دوم بازمیگردد. روی طرف دیگر اولین مکانیزم امنیتی بودن این است که این مکانیزم تبدیل به bottleneck میشود و هر دسترسی باید از این مکانیزم عبور کند که میتواند سربار اضافی داشته باشد و برای هر بار دسترسی به فایل باید از این گلوبگاه عبور کرد. دومین و اصلی ترین عیب آن این است که اگر این برنامه و تکنیک هایش به درستی پیاده سازی نشده باشند و دارای حفره امنیتی باشند هیچ سیستم پشتیبانی برای file protection وجود ندارد و این مشکل امنیتی به دلیل مرکزی بودن watchdog برای تمامی فایل‌هایی که با این سیستم امن شده اند وجود خواهد داشت.

#### سوال ۴

در ابتدا حافظه مانند شکل زیر است.

2KB
7KB
5KB
8KB
30KB
10KB
10KB

اگر از first fit برای تخصیص حافظه به برنامه‌ها استفاده کنیم، داریم:

2KB
7KB
5KB
Program B
Program A 5KB
Program C 1KB
10KB

اگر از best fit برای تخصیص حافظه به برنامه‌ها استفاده کنیم، داریم:

2KB
7KB
5KB
Program B
Program A
5KB
Program C
1KB
10KB

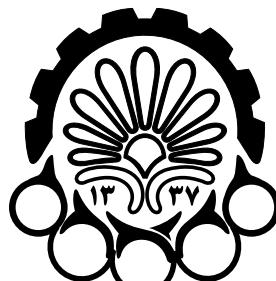
اگر از worst fit برای تخصیص حافظه به برنامه‌ها استفاده کنیم، داریم:

2KB
7KB
5KB
8KB
Program A
5KB
Program B
2KB
Program C
1KB

## سوال ۵

- الف) نصب یک پردازنده سریعتر باعث می‌شود بهره‌وری سیستم کاهش یابد، زیرا کارها در زمان کمتری روی پردازنده قرار می‌گیرند.
- ب) نصب یک دیسک بزرگتر برای paging فضای ذخیره سازی page‌ها در حافظه را افزایش می‌دهد و از بهره‌وری disk می‌کاهد.
- ج) افزایش اندازه‌ی page fault سیستم تعداد page‌ها را افزایش می‌دهد، بنابراین از بهره‌وری CPU می‌کاهد.
- د) کاهش اندازه‌ی page fault سیستم تعداد page‌ها را کاهش می‌دهد، بنابراین به بهره‌وری CPU می‌افزاید.
- ه) افزایش حافظه‌ی main memory از تعداد page fault‌ها کاسته و بهره‌وری CPU را افزایش می‌دهد.
- و) نصب یک دیسک سریعتر زمان swap page‌ها را کاهش داده و به بهره‌وری پردازنده می‌افزاید.
- ز) افزایش page size از تعداد page fault‌ها کاسته و به زمان swap page‌ها می‌افزاید، بنابراین با توجه به شرایط می‌تواند باعث افزایش یا کاهش بهره‌وری پردازنده شود.

به نام خدا



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)

## سیستم‌های عامل

# فاز اول پروژه نهائی

استاد درس:

آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

۳۰ مهر ۱۴۰۰ ساعت ۵:۵۹

نکته مهم: دقت کنید که تمدید نخواهیم داشت و هفته اول ابان تحويل اسکایپی خواهید داشت و تنها دانشجویانی که فاز اول را به موقع انجام داده‌اند، خواهند توانست وارد فاز دوم شوند.

## مقدمه

همانطور که در کلاس درس بیان شد، پروژه درس سیستم‌های عامل در مورد شناخت کامل سیستم عامل آموزشی xv6 و اضافه کردن قابلیت‌های جدید به آن است. با انجام دقیق این پروژه و نگاشت مفاهیم بیان شده در کلاس درس به معادل عملیاتی آنها، به یک یادگیری عمیق و ماندگار دست خواهد یافت. مهم‌تر اینکه برنامه‌نویسی در سطح سیستم عامل به شما کمک می‌کند تا یک تجربه بی‌نظیر از برنامه‌نویسی سیستمی داشته باشید.

هدف از دو فاز ابتدایی پروژه آشنایی شما دانشجویان عزیز با این سیستم عامل آزمایشی است و روند کار به این صورت است که به صورت انفرادی باید تغییراتی در این سیستم عامل ایجاد کنید تا بتوانید درک خوبی از این سیستم عامل بدست آورید. در فاز سوم از شما درخواست خواهیم کرد که به گروه‌های دونفره تقسیم شوید و تغییراتی که در این فاز انجام خواهد داد بسیار مهم‌تر و جدی‌تر خواهد بود که موضوع این قسمت در زمان مناسب به شما اعلام خواهد شد.

در فاز اول پروژه شما بایستی که فراخوانی‌های سیستمی جدیدی به xv6 اضافه کنید. تیم تدریسیاری ویدئوهایی بسیار خوب برای آشنایی شما با این سیستم عامل در سایت درس قرار داده‌اند.

برای آشنایی بیشتر با این سیستم عامل می‌توانید به لینک زیر مراجعه کنید :

<https://pdos.csail.mit.edu/6.828/2012/xv6.html>

برای اجرای سیستم عامل xv6 باید از یک نرم افزار مجازی‌سازی به اسم qemu استفاده کنید. پیشنهاد می‌شود با استفاده از ماشین مجازی VirtualBox یا VMware یک سیستم عامل ubuntu نصب کنید و برروی سیستم عامل ubuntu خود نرم افزار مجازی‌ساز qemu را راه اندازی کرده و سیستم عامل xv6 را اجرا کنید. تمامی مراحل نصب و راه اندازی xv6 در قالب ویدئوهای آموزشی در سایت درس قرار گرفته است. ضمن اینکه حجم زیادی از منابع مفید را می‌توانید در وب پیدا کنید.

## فاز اول

این فاز را با نصب و اجرای xv6 اغاز کنید و سپس با استفاده از راهنمایی‌هایی که برای شما فراهم شده است، فراخوانی‌های سیستمی زیر را پیاده سازی کنید.

### • **int getProcCount(void)**

در این فراخوانی سیستمی از شما می‌خواهیم که تعداد پردازه‌ایی (processes) را که در سیستم در لحظه فراخوانی این فراخوانی سیستمی وجود دارند را برگردانید.

همچنین پس از ساخت این فراخوانی سیستمی از شما می‌خواهیم که یک فایل به نام `getProcCountTest.c` ایجاد کنید که بتوانید صحت کارکرد این فراخوانی سیستمی را چک کنید. وقت کنید که اگر قبل از اجرای برنامه `getProcCountTest` در سیستم xv6 شما مثلاً ۲۰ پردازه وجود داشته باشد، `getProcCount` عدد ۲۱ را برمی‌گرداند چرا که خود برنامه `getProcCountTest` یک پردازه جدید است.

## • **int getCount(void)**

در این فراخوانی سیستمی از شما می‌خواهیم که تعداد دفعاتی که فراخوانی سیستمی **Read** توسط هر پردازه‌ی دیگر کاربر فراخوانی شده است (از زمانی که کرنل بوت شده) را برگردانید.

همچنین پس از ساخت این فراخوانی سیستمی از شما می‌خواهیم که یک فایل به نام **getCountTest.c** ایجاد کنید که بتوانید صحت کارکرد این فراخوانی سیستمی را چک کنید.

از شما در خواست داریم که یک **private repository** در گیت هاب درست کنید و تغییرات کد خود را مرحله به مرحله **commit** کنید و در صورت تمایل می‌توانید هریک از تدریسیاران را به پروژه‌ی خود اضافه کنید. دقت کنید که شما نبایستی برنامه‌های خود را با دیگر دانشجویان به اشتراک بگذارید.

### نکات مهم در ارتباط با این فاز

- این فاز پیش‌نیاز قطعی فازهای بعدی است و انجام ندادن ان باعث می‌شود که نتوانید فاز دوم را شروع کنید و همچنین نمی‌تواند برای انجام پروژه نهائی گروهی را تشکیل دهید.
- پروژه شما تحويل اسکایپی خواهد داشت بنابراین از استفاده از کدهای یکدیگر یا کدهای موجود در وب که قادر به توضیح دادن عملکرد انها نیستید، بپرهیزید.
- ابهامات خود را در سایت درس مطرح کنید و ما در سریع‌ترین زمان ممکن به انها پاسخ خواهیم داد.

### انچه که باید ارسال کنید

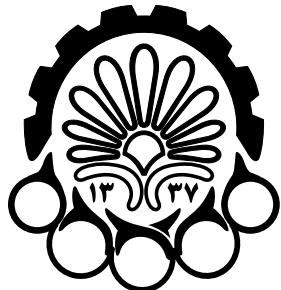
یک فایل زیپ با نام **sid\_hw1.zip** را با شماره دانشجویی خود جایگزین کنید) که شامل دو مورد زیر است:

- گزارش خیلی مختصر از انچه که انجام داده‌اید تا دو فراخوانی سیستمی خواسته شده را به **xv6** اضافه کنید.
- پوشاهای که در ان کدهای شما وجود دارد. دقت کنید که تنها و تنها فایل‌هایی را که تغییر داده‌اید یا اضافه کرده‌اید را برای ما بفرستید.

موفق باشید

تیم درس سیستم‌های عامل

به نام خدا



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)

سیستم‌های عامل (پاییز ۱۴۰۰)

## فاز دوم

استاد درس:

آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

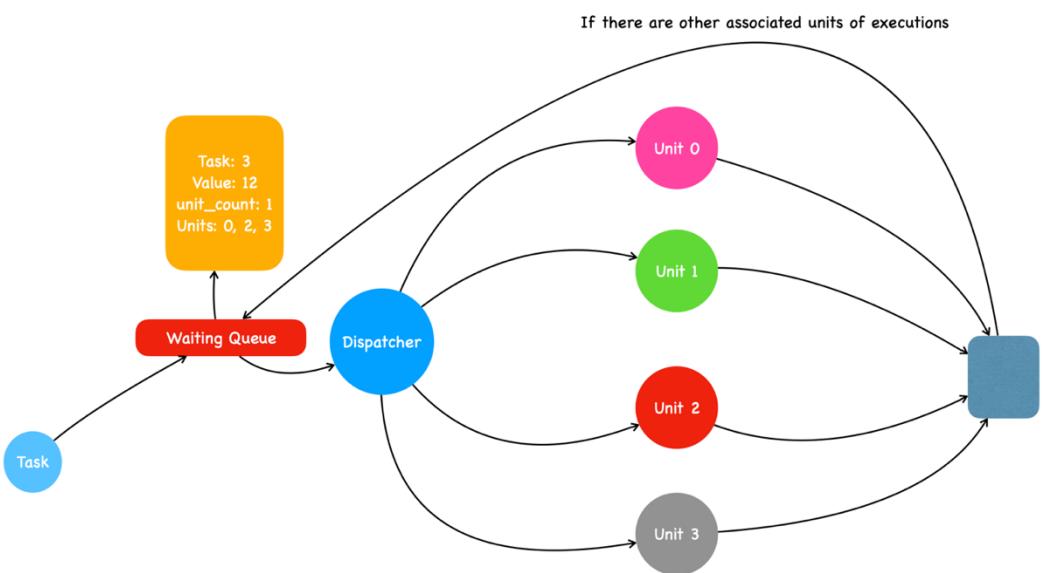
۱۴۰۰ آذر ماه ۲۶

دقیقت کنید که امکان تمدید وجود ندارد.

## توصیف پروژه

یک سیستم پیچیده مانند یک سازمان بزرگ را درنظر بگیرید که واحدهای پردازشی جدایی برای هر کار دارد. در این سازمان برای آنکه هر کدام از کارها بطور کامل انجام شوند نیازمند مجموعه‌ای از پردازش‌ها در هر کدام از واحدهای پردازشی هستند. تمامی کارها در صف انتظار منتظر می‌مانند تا توزیع کننده، کار مورد نظر را انتخاب کند و به واحد پردازشی مربوطه تحويل دهد.

شکل زیر ساختار سیستم موردنظر را نشان می‌دهد :



شما باید بتوانید چنین مکانیزمی را در پروژه خود طراحی کنید.

## جزئیات پیاده‌سازی

به منظور پیاده‌سازی مکانیزم خواسته شده باید فراخوانی‌های سیستمی زیر را پیاده‌سازی کنید :

```
int thread_create(void *stack, int status)  
  
int thread_wait()  
  
int unit0_operation()  
  
int unit1_operation()  
  
int unit2_operation()  
  
int unit3_operation()
```

همچنین لازم است که یک تابع سطح کاربر با امضای زیر پیاده‌سازی کنید که به عنوان تابع پوششی برای `thread_create` ایفای نقش می‌کند و هدف آن استفاده آسان‌تر از این فراخوانی سیستمی است.

```
int thread_creator(void (*fn)(void *), void *arg, int status)
```

فراخوانی سیستمی `thread_create` بسیار شبیه به `fork` عمل می‌کند با این تفاوت که در این تابع به جای کپی کردن فضای آدرس به یک `page directory` جدید، این فراخوانی سیستمی وضعیت پردازه جدید را طوری مقداردهی می‌کند که پردازه فرزند و پردازه پدر فضای آدرس را به اشتراک بگذارند (به عبارت دیگر از یک `page directory` مشابه استفاده کنند). اگر این فراخوانی سیستمی به این شکل نوشته شود دو پردازه فضای حافظه را به اشتراک خواهند گذاشت و این دو عمل ریسمان هستند.

اگرچه دو ریسمان پدر و فرزند فضای آدرس را به اشتراک می‌گذارند، هر کدام به یک پشته جداگانه نیاز دارند. فضای پشته جدید احتمالاً با استفاده از `malloc` تخصیص داده می‌شود. به عنوان مثال فرض کنید که ریسمان `T1` یک کپی از خود به عنوان `T2` را ایجاد می‌کند. `T1` ابتدا حافظه‌ای به اندازه یک صفحه که `page-aligned` است را به پشته `T2` اختصاص داده و اشاره‌گر به آن را به فراخوانی سیستمی `thread_create` ارسال می‌کند. از طرفی دیگر، فراخوانی سیستمی پشته ریسمان `T1` را به این نقطه از حافظه کپی می‌کند و ثبات‌های پشته و پایه `T2` را تغییر می‌دهد تا از پشته جدید استفاده کند. توجه شود که `T2` باید از ساقه اجرای `T1` مطلع باشد و این یعنی باید بتواند به آیتم‌های موجود در پشته `T1` به عنوان مقدار اولیه پشته جدید دسترسی داشته باشد.

تابع `thread_creator`، اشاره‌گر به یک تابع و آرگومان‌های ریسمان و یک `status` را به عنوان ورودی می‌گیرد و کارهای زیر را انجام می‌دهد:

- یک فضای `page-aligned` به پشته اختصاص دهد.
- فراخوانی سیستمی `thread_create` را صدا بزند که ID را به پدر بر می‌گرداند.
- در ریسمان فرزند، `thread_creator` باید اشاره‌گر به تابع ارسالی را صدا بزند و آرگومان‌های ارسالی به عنوان پارامترهای ورودی در اختیار این تابع قرار دهد، همچنین باید مقدار `status` نیز پاس داده شود.
- هنگامی که اجرای تابع ارسالی به اتمام رسید، `thread_creator` باید پشته را خالی کرده و `exit` را صدا بزند.
- فراخوانی سیستمی `thread_wait` بسیار شبیه فراخوانی سیستمی `wait` می‌باشد و در واقع منتظر می‌ماند تا اجرای ریسمان فرزند تمام شود و `wait` منتظر می‌ماند تا پردازه فرزند تمام شود.

متغیر `status` در پیاده‌سازی شما سه حالت دارد:

۱. اگر صفر باشد یعنی `thread` ای که باید ساخته شود باید از نوع `Unit` باشد.
۲. اگر یک باشد یعنی ریسمانی که باید ساخته شود باید از نوع `Task` باشد.

۳. اگر دو یا بیشتر باشد باشد یعنی ریسمانی که باید ساخته شود باید از نوع معمولی می‌باشد (هیچ یک از Task یا Unit نیست).

در struct proc باشد یک متغیر به اسم status تعریف شود که مشخص کند که ترد ساخته شده از نوع Unit می‌باشد یا Task.

### مشخصات ریسمان از نوع Unit:

تردهایی که از نوع Unit می‌سازید ویژگی‌های زیر را دارا هستند:

مقدار متغیر status برای آنها در struct proc برابر با صفر می‌باشد.

یک متغیر به عنوان unit\_num که مشخص کننده شماره واحد پردازشی موردنظر می‌باشد.

### مشخصات ریسمان از نوع Task:

ریسمان‌هایی که از نوع Task می‌سازید ویژگی‌های زیر را دارا هستند:

- مقدار متغیر status برای آنها در struct proc برابر با یک می‌باشد.
- یک متغیر struct proc unit\_count در مربوط به این ریسمان‌ها تعریف می‌شود که از صفر شروع می‌شود و با هر بار ورود به یک واحد پردازشی و پردازش شدن، مقدار آن یک واحد زیاد می‌شود.
- یک لیست از واحدهای پردازشی که Task مورد نظر نیاز دارد تا پردازش‌های لازم بر روی آن صورت گیرد.
- یک متغیر Value که مقدار ریسمان Task مورد نظر را در خود نگهداری می‌کند.

### فراخوانی‌های سیستمی مربوط به هر واحد پردازشی:

- `int unit0_operation()`

این فراخوانی سیستمی پردازش مربوط به واحد پردازشی اول را انجام می‌دهد که در واقع مقدار Value یک Task را می‌گیرد و عملیات زیر را بر روی آن انجام می‌دهد:

Adds 7 to the value then modulate by M

- `int unit1_operation()`

این فراخوانی سیستمی پردازش مربوط به واحد پردازشی دوم را انجام می‌دهد که در واقع مقدار Value یک Task را می‌گیرد و عملیات زیر را بر روی آن انجام می‌دهد:

Multiplies by 2 then modulate by M

- **int unit2\_operation()**

این فراخوانی سیستمی پردازش مربوط به واحد پردازشی سوم را انجام می‌دهد که در واقع مقدار Value یک Task را می‌گیرد و عملیات زیر را بروی آن انجام می‌دهد:

Multiplies by 3 then modulate by M

- **int unit3\_operation()**

این فراخوانی سیستمی پردازش مربوط به واحد پردازشی چهارم را انجام می‌دهد که در واقع مقدار Value یک Task را می‌گیرد و عملیات زیر را بروی آن انجام می‌دهد:

Prints out the value

متغیر M یک مقدار **const** است، مقدار آن را ۲ قرار دهید.

نکته: در واقع برای پیاده‌سازی خود باید هر یک از فراخوانی‌های سیستمی مربوط به هر واحد پردازشی را در یک تابع جدا فراخوانی کرده و اشاره‌گر به این تابع را به تابع پوششی **thread\_creator** متناظر با آن واحد پردازشی پاس دهید.

دقت کنید که هر واحد پردازشی باید بین تمامی ریسمان‌های Task جستجو کند و ریسمانی را پردازش می‌کند که list(unit\_count) برای آن Task برابر با واحد پردازشی فعلی باشد.

توجه شود که در حالت عادی ریسمان‌های متعلق به یک پردازه باید process id یکسان داشته باشند. با این وجود در این پروژه اشکالی ندارد که getpid برای ریسمان‌های متعلق به یک پردازه مقادیر متفاوتی را برگرداند. در واقع شما می‌توانید از این کار می‌توانید از مکانیزم شمارش ارجاعات (reference counting) استفاده کنید تا بتوانید با کمینه تغییرات، پیاده‌سازی خود را انجام دهید.

توجه شود که باید اطمینان حاصل کنید که در هنگام اجرای exit توسط یک ریسمان، پیاده‌سازی شما جدول صفحه (page table) این ریسمان را در صورت وجود دیگر ریسمان‌هایی که به این جدول صفحه اشاره می‌کنند آزاد (free) نمی‌کند، برای این کار می‌توانید از مکانیزم شمارش ارجاعات (reference counting) استفاده کنید یا می‌توانید با اسکن کردن جدول پردازه (process table) چک کنید که اگر ارجاع دیگری به جدول صفحه پیدا نشد، جدول صفحه در نتیجه‌ی آزاد شود.

همچنین برای پیاده‌سازی فراخوانی سیستمی **thread\_create** باید به نکات زیر توجه داشته باشید:

۱) جلوگیری از شرایط مسابقه یا race condition در هنگام گسترش فضای آدرس

۲) آزاد (free) نکردن جدول صفحه (page table) تا وقتی که تمامی ریسمان‌های متعلق به یک پردازه به اتمام برسند.

راهنمایی: می‌توانید یک تابع dispatcher درست کنید و در این تابع چهار ریسمان جدا برای هر واحد پردازشی بسازید و برای هر واحد پردازشی تابعی تعریف کنید که سیستم کال مربوط به واحد پردازشی موردنظر را فراخوانی کند و این تابع را به

مربوط به واحد پردازشی موردنظر پاس دهید و همچنین چند ریسمان از نوع Task ایجاد کنید و آنها را به حالت Busy waiting یا sleep ببرید تا پردازش‌های مورد نیاز آنها در هر کدام از واحدهای پردازشی انجام شود (دققت کنید که هر واحد پردازشی خود جدول پردازه‌ها اسکن می‌کند و تردهایی که نیاز به پردازش دارند را پردازش می‌کند (در واقع این پیاده‌سازی راحت‌ترین نوع پیاده‌سازی است که می‌توانید انجام دهید<sup>(۱)</sup>، در واقع ریسمان‌های Task کاری انجام نمی‌دهند و فقط منتظر آن هستند که پردازش‌های آنها توسط واحدهای پردازشی موردنظر انجام شده تا تمام شوند).

همچنین می‌توانید از فراخوانی سیستمی getprocs() که در فاز اول پروژه پیاده‌سازی کردید، که تعداد پردازه‌های فعال در سیستم را به شما بر می‌گرداند، کنترل کنید که تعداد پردازه‌های سیستم در هر لحظه چطور تغییر می‌کند و با استفاده از این سیستم کال می‌توانید متوجه شوید که چه تعداد از Task شما تمام شده‌اند و چه تعداد همچنان نیاز به پردازش دارند.

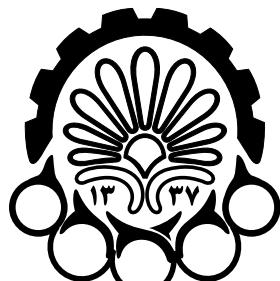
توجه شود که تستی که می‌نویسید باید بتواند عملکرد سیستم را به خوبی نشان دهد.

توجه شود که تصویری که در ابتدای پروژه قرار داده شده است صرفا برای دید بهتر شما نسبت به عملکرد سیستم قرار داده شده است و پیاده‌سازی شما باید مطابق توضیحات گفته شده باشد.

موفق باشید

تیم درس سیستم‌های عامل

به نام خدا



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)

سیستم‌های عامل (پاییز ۱۴۰۰)

## فاز سوم

پیاده سازی الگوریتم‌های زمانبندی

استاد درس:

آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

۴ بهمن ۱۴۰۰ (۵۹:۲۳)

تمدیدی نخواهیم داشت.

## مقدمه

در این قسمت می خواهیم الگوریتم های زمانبندی خودمان را جایگزین الگوریتم پیش فرض در xv6 کنیم. توصیه می کنیم که با جستجو و مطالعه فایل های سیستم عامل به دنبال چگونگی و کار کرد زمانبندی و تخصیص CPU به پردازه ها بگردید. زمانبندی یکی از مهم ترین و پایه ای ترین مفاهیم موجود در هر سیستم عاملی است جایی که زمانبند باید یک سری اهداف که در بعض اضاد هم هستند را برآورده کند، مانند:

- زمان پاسخ سریع<sup>۱</sup>
- بازده خوب برای پردازه های پس زمینه<sup>۲</sup>
- جلوگیری از قحطی<sup>۳</sup>
- برآورده کار کردن توامان نیازهای پردازه های با الویت پایین و الویت بالا<sup>۴</sup>
- و ...

به مجموعه ای از قوانین که با استفاده از آنها یک پردازه برای اجرا انتخاب می شود را خطمنشی زمانبندی scheduling policy) می گوییم. ابتدا باید با مطالعه کد xv6 و منابع موجود درباره ان یاد بگیریم که xv6 policy موجود در xv6 چیست و چه پردازه ای وظیفه انتخاب پردازه ها برای اجرا را بر عهده دارد. بخش زمانبندی در کد xv6 را پیدا کنید و سعی کنید به سوالات زیر پاسخ دهید (صرفا برای شروع کار):

- خطمنشی پیش فرض چه پردازه ای برای اجرای انتخاب می کند؟
- وقتی که یک پردازه از رخداد IO برمی گردد، چه اتفاقی می افتد؟
- وقتی که یک پردازه ایجاد می شود، چه اتفاقی می افتد و زمانبندی در چه زمان هایی و با چه فاصله زمانی انجام می شود؟

## بخش اول : پیاده سازی الگوریتم های زمانبندی

### ۱) الگوریتم در Round-Robin

در کد پیش فرض، زمانبندی یا round-robin از سیاست scheduler با clock tick کار تخصیص CPU را انجام می دهد. در این پروژه می خواهیم با تعیین زمان بین دو تخصیص CPU، بینیم روند پردازه ها چگونه بهبود می یابد. ابتدا باید در فایل param.h یک پارامتر جدید مانند QUANTUM را با مقدار اولیه ای مانند ۱۰ تعریف کنیم.

```
#define QUANTUM <Number>
```

---

fast process response time<sup>۱</sup>  
good throughput for background jobs<sup>۲</sup>  
avoidance of process starvation<sup>۳</sup>  
reconciliation of the needs of low-priority and high priority processes<sup>۴</sup>

سپس در مکان مربوط به الگوریتم‌های زمان‌بندی، تغییرات لازم برای اینکه زمان‌بند به جای اینکه هر یک clock tick کار زمان‌بندی را انجام دهد، هر QUANTUM clock tick کار زمان‌بندی را انجام دهد.

لازم به ذکر است که در فایل گزارش خود طبق آزمون‌های تعریف شده در جلوتر، باید با تغییر مقدار QUANTUM ببینید که عملکرد پردازه‌ها بهبود می‌یابد یا خیر.

## (۲) الگوریتم زمان‌بندی طبق اولویت‌بندی (non-preemptive priority scheduling)

در این قسمت می‌خواهیم الگوریتم زمان‌بندی طبق اولویت را پیاده‌سازی کنیم که cpu طبق اولویت پردازه‌ها تخصیص داده می‌شود. این الگوریتم را از نوع غیر-قبضه‌ای پیاده کنید، به این معنی که اگر پردازه‌ای جدیدی ایجاد شود یا اینکه پردازه‌ای از رخداد IO برگردد و اولویت ان از پردازه در حال اجرای پردازه باشد، پردازه در حال اجرا، کار خود را تا زمانی که کوانتم زمانی اجازه دهد، برای یک IO بلوک شود یا terminate شود، ادامه می‌دهد. در این الگوریتم به هر پردازه باید یک عدد بین ۱ تا ۶ به عنوان اولویت تخصیص داده شود. اولویت پیش فرض را ۳ در نظر بگیرید و همچنین اگر اولویتی خارج از این بازه وارد شد به عنوان ۵ آن را بشناسد. در این روش، پردازه با عدد اولویت کمتر (به سمت ۱) شанс بیشتری برای اجرا شدن نسبت به دیگر پردازه‌ها دارد. از طرف دیگر، پردازه‌ها با اولویت ۶ کمترین شанс برای اجرا شدن را دارند. برای پیاده‌سازی، به ساختمان داده proc در فایل proc.h باید متغیرهای لازم اضافه شوند.

این خطمنشی زمان‌بندی باید اینگونه کار کند که تخصیص CPU خود را به طور چرخشی (round-robin)، اسلاید ۱۳ از lecture 13) به پردازه‌های دارای اولویت بالاتر بدهد و پس از نبود برنامه در اولویت‌های بالاتر، به سراغ اولویت‌های کمتر برود.

در ادامه برای تعیین و تغییر اولویت یک رویه نیاز به یک system call setPriority به نام داریم. در این سیستم کال یک عدد به عنوان ورودی گرفته می‌شود و طبق توضیحات گفته شده باید به عنوان اولویت پردازه‌ای که سیستم کال را صدای زده ثبت شود.

## (۳) الگوریتم زمان‌بندی طبق صفت‌های (به شکل preemptive)

در این قسمت می‌خواهیم زمان‌بندی چند صفحه‌ای و یا صفحه‌ای چند لایه پیاده‌سازی ابتدا باید ۶ صفحه‌سازیم (صفه‌ای شماره ۱ تا ۶) و هر صفحه با الگوریتم زمان‌بندی round-robin اجرا می‌شود (با این تفاوت که صفحه‌ای با شماره کمتر دارای QUANTUM بیشتر نسبت به صفحه‌ای با شماره بیشتر هستند). اینکه QUANTUM زمانی صفحه‌ای با اولویت بیشتر (شماره کمتر) چقدر بیشتر باشد در اختیار شما است. دقت کنید که پردازه‌های موجود در صفحه شماره ۲ در صورتی اجرا می‌شوند، که صفحه شماره ۱ خالی باشد. به همین ترتیب پردازه‌های موجود در صفحه شماره ۳، تنها در صورتی اجرا می‌شوند که صفحه شماره ۱ و ۲ خالی باشند. به شکل کلی پردازه‌های موجود در صفحه ۱ تنها در صورتی اجرای می‌شوند که صفحه‌ای شماره ۱ تا ۱-۱ خالی باشند. دقت کنید که صفحه مورد نظر یک پردازه با استفاده از سیستم کال setPriority و با توجه به توضیحات بخش ۲ تعیین می‌شوند. به تفاوت کلیدی دیگر این بخش با بخش ۲ دقت کنید. در این بخش اگر پردازه‌ای با اولویت بالاتر به سیستم وارد شود در حالی که پردازه با اولویت پایین‌تر در حال اجرا است، پردازه با اولویت بالا بایستی که جایگزین پردازه با اولویت پایین‌تر شود. این تفاوت را جدی بگیرید و برای پیاده‌سازی ان زمان بگذارید و به نظر ما خیلی جذاب است.

#### ۴) الگوریتم زمانبندی طبق صفت چند لایه‌ی پویا (اختیاری-نمره اضافی)

این قسمت مانند قسمت قبل است با این تفاوت که یک پردازه نمی‌تواند به صورت دستی اولویت خود را تغییر دهد. در حقیقت یک پردازه کار خود را با اولویت پیش‌فرض یعنی ۳ شروع می‌کند و سپس بر اساس مجموعه‌ای از قوانین که در زیر آورده شده است، اولویت آن در طول اجرا تغییر می‌کند. این قوانین سعی می‌کنند که اولویت پردازه‌های که بیشتر کار IO انجام می‌دهند (احتمال زیاد کاربردهای تعاملی یا interactive) را افزایش دهند و اولویت پردازه‌های cpu-intensive را افزایش دهند.

رعایت قوانین زیر الزامی است:

۱) فرآخوانی سیستم کال exec اولویت پردازه را به حالت پیش‌فرض ریست می‌کند (default priority).

۲) برگشت از حالت sleep mode یا همان IO باعث افزایش اولویت پردازه به بیشترین اولویت می‌شود یا همان عدد ۱ می‌شود (highest priority).

۳) واگذاری (yielding) cpu به صورت دستی اولویت پردازه را تغییر نمی‌دهد.

۴) اجرا شدن به اندازه‌ی یک full quanta باعث کاهش اولویت پردازه به اندازه ۱ واحد می‌شود.

توجه شود که برای این قسمت ممکن است نیاز به تعریف سیستم کال‌های جدید یا متغیرهای جدیدی برای پیاده‌سازی خود شوید.

بخش دوم: کدهای کمکی یا ارزیابی پیاده‌سازی‌های انجام شده

#### ۱) changePolicy

جدای موارد خواسته شده همانطور که مشخص است نیاز داریم تا روند تخصیص CPU و زمان‌بندی را تغییر دهیم. برای این کار نیاز به پیاده‌سازی system call به نام changePolicy داریم. برای این کار ما به هر الگوریتم و قاعده زمانبندی‌مان یک عدد مانند ۰ و ۱ و ۲ اختصاص می‌دهیم (سعی کنید به الگوریتم پیش‌فرض عدد صفر و به باقی الگوریتم‌ها اعداد متناظر آنها در متن پروژه را تخصیص دهید). در این system call باید یک عدد به عنوان ورودی بگیریم و سپس روند زمان‌بندی برنامه را تغییر دهیم. با استفاده از این system call باید بتوانیم بین الگوریتم‌های گفته شده بالا و همچنین الگوریتم اصلی و پیش‌فرض خود ۶XV تغییر وضعیت دهیم (مجموعاً ۴ الگوریتم و با الگوریتم امتیازی مجموعاً ۵ الگوریتم).

#### ۲) قابلیت اندازه‌گیری زمان

در این قسمت می‌خواهیم به ساختمان داده هر پردازش در proc.h متغیرهایی اضافه کنیم تا با این‌ها بتوانیم بینیم CBT waitingTime و turnAroundTime (Cpu Burst Time) هر برنامه چقدر است.

برای این کار نیاز است تا متغیر هایی مانند ready time ,running time ,termination time ,creation time و sleeping time را نگه داریم و با هر کلاک CPU این مقادیر را برای هر process با توجه به موقعیت آن به روزرسانی کنیم.

لازم به ذکر است که برای باز پس‌گیری این مقادیر هنگام پایان کار نیز نیازمند به متدها و یا فراخوانی‌های سیستمی دارد که طبق سلیقه خودتان انها را پیاده سازی کنید.

### ۳) تست نویسی

برای هر کدام از الگوریتم‌های پیاده‌سازی شده، نیاز است فایل تستی نوشته شود تا صحت عملکرد آن چک شود. همچنین نیاز داریم در انتهای آنها با قابلیت‌های اضافه شده برای اندازه‌گیری زمان، بتوانیم به بررسی این الگوریتم‌ها بپردازیم.

#### roundRobinTest (۱-۳)

ابتدا برای الگوریتم Round-Robin نیاز به یک تست داریم. در این تست برنامه اصلی ما بوسیله fork ۱۰ فرزند می‌سازد و سپس هر کدام از فرزندان در یک حلقه ۱۰۰۰ بار خط زیر را چاپ می‌کنند که در این خط یک شمارنده از ۱ تا ۱۰۰۰ است.

```
/PID/ : /i/
```

در انتهای آنها نیز Waiting Time ,Turn Around Time و CBT هر کدام از فرزندان باید نمایش داده شود و همچنین میانگین این‌ها نیز گفته شود.

سپس در ادامه چک شود با افزایش و کاهش مقدار Quantum خروجی ما چه تغییری می‌کند؟

#### prioritySchedTest (۲-۳)

این بار برای الگوریتم Priority Scheduling که در قبل پیاده سازی کردیم یک تست باید بنویسیم. در این تست ابتدا process ما باید ۳۰ فرزند تولید کند که ۵ فرزند اول دارای اولویت ۶، ۵ فرزند بعدی اولویت ۵ و ... و در انتهای به ۵ فرزند آخر اولویت ۱ داده شود. سپس هر کدام از فرزندان در یک حلقه ۲۵۰ بار خط زیر را چاپ می‌کنند که در این خط یک شمارنده از ۱ تا ۲۵۰ است.

```
/ChildNumber/ : /i/
```

برای این نیز Waiting Time ,Turn Around Time و CBT هر کدام از فرزندان باید نمایش داده شود و همینطور میانگین این پارامترها برای کل فرزندان و هر کلاس اولویت گفته شود.

#### multiLayeredQueuedTest (۳-۳)

این بار برای الگوریتم Multi layered queue که در قبل پیاده سازی کردہ‌اید یک تست بنویسید. برای این کار باید ۶۰ فرزند تولید کنید و ۱۰ فرزند اول را به صف اول، ۱۰ فرزند دوم به صف دوم و ... اختصاص دهید. در نتیجه هر فرزند در یک حلقه ۲۰۰ بار خط زیر را چاپ کند که در این خط یک شمارنده از ۱ تا ۲۰۰ است.

/ChildNumber/ : /i/

برای این نیز CBT و Waiting Time .Turn Around Time میانگین این پارامتر ها برای کل فرزندان و هر لایه صف گفته شود.

#### DynamicMultiLayeredQueuedTest (۴-۳ اختیاری-نمره اضافی)

این بار برای الگوریتم Dynamic Multi Layered Queue که در قبل پیاده سازی کردہ اید یک تست بنویسید.

#### نحوه تحويل پروژه

- گزارش پروژه (شامل خروجی تمامی تست‌ها) و فایل‌های اضافه شده یا تغییر داده شده در XV6 را در قالب یک فایل zip با نام project\_report\_group\_id\_sid1\_sid2.zip بارگذاری کنید.
- پروژه دارای تحويل به صورت آنلاین است و توضیح کد و توضیح عملکرد سیستم عامل پرسیده می‌شود و بخش مهمی از نمره را در بر می‌گیرد. پس ضروری است که همه اعضای گروه به XV6 و پیاده‌سازی انجام شده تسلط داشته باشند.
- با توجه به انجام پروژه به شکل گروهی، تعداد commit‌های هر فرد باید متناسب باشد و خود گیت در حین تحويل چک می‌شود.
- لازم به ذکر است که تمامی پروژه‌ها پس از تحويل بررسی می‌شوند و هرگونه شباهت، به منزله نمره ۰ برای پروژه خواهد بود.

موفق باشید

تیم درس سیستم‌های عامل