امتحان میانترم

Started on	Wednesday, 26 Aban 1400, 4:31 PM
State	Finished
Completed on	Wednesday, 26 Aban 1400, 6:21 PM
Time taken	1 hour 49 mins
Marks	4.45/5.00
Grade	**8.90** out of 10.00 (**89**%)

Question 1

Correct

Mark 0.25 out of 0.25

Flag question

تعریف مدهای متنوع در سیستم‌های کامپیوتری یک مکانیزم صرفا نرم افزاری برای ایجاد protection است

Select one:

- True
- False ✔

The correct answer is 'False'.

Question 2

Incorrect

Mark 0.00 out of 0.25

Flag question

در پیاده سازی Infrastructure as a Service در کلادها، مدیریت منابع از طریق ارایه سرویس به صورت VM مناسبتر از container است.

Select one:

- True
- False ✖

The correct answer is 'True'.

Question 3

Correct

Mark 0.25 out of 0.25

Flag question

در پیاده‌سازی مفسر خط فرمان (Command Line Interpreter) بهتر است دستورات خط فرمان، built-in باشد، یعنی جزئی از پیاده‌سازی خط فرمان باشد.

Select one:

- True
- False ✔

The correct answer is 'False'.

Question 4

Correct

Mark 0.25 out of 0.25

Flag question

اگر یک برنامه را از خط فرمان shell را ببنیدم برنامه مورد نظر یتیم می شود shell، اجرا کنیم و قبل از اتمام آن

Select one:

- True ✔
- False

The correct answer is 'True'.

Question 5

Complete

Mark 1.00 out of 1.00

Flag question

الف) توضیح دهید PCBدر کجا ذخیره می‌شود و چه نقشی در تعویض متن پروسسها دارد؟

ب) آیا جهت تعویض اجرای سرویس روتین اینتراپت به جای پروسسی که قبل از وقوع اینتراپت در CPU در حال اجرا بوده است، از PCB استفاده میشود؟ اگر بله به چه صورت و اگر خیر پ تعویض متن در این مواقع چگونه انجام می‌شود.

الف)

از ساختار های مربوط به کرنل است و در حافظه کرنل ذخیره میشود، اطلاعات مختلف مربوط به پروسه ها از جمله PC، حافظه ها، اطلاعات رجیستر ها و ... در PCB ذخیره میشود که هنگام تعویض متن، امکان اجرای ادامه پروسه ای که از پردازنده خارج شده را میدهد، اطلاعات پروسه در حال اجرا در PCB آن ذخیره شده و اطلاعات پروسه آماده اجرا از PCB آن لود میشود

ب)

خیر استفاده نمیشود، هنگام اینتراپت اطلاعات رجیستر ها، PC و ... داخل interrupt stack ذخیره و از آن استفاده میشود

الف) ۰.۵ نمره

ب) (۰.۵ نمره) وقتی حین اجرای پروسسی، اینتراپت رخ میدهد در حالت کلی پس از اجرای سرویس روتین اینتراپت قصد داریم دوباره به اجرای پروسس قبلی برگردیم بنابراین نیاز نیست تمام اطلاعات pcb آپدیت شوند کافیست اطلاعات مورد نیاز حین اجرای پروسس در استکی push شوند و پس از اجرای سرویس روتین، دوباره این موارد از استک popشود و اجرای پروسس از سر گرفته شود لذا در این حالت تعویض متن بدان معنی که در تعویض متن پروسسها وجود دارد نداریم و کار تعویض متن از پروسس به اینتراپت و برعکس هم سریعتر و با سربار کمتری انجام میگیرد

Comment:

Question 6

Complete

Mark 1.70 out of 2.00

الف) یک سیستم تعاملی که زمانبندی نوبت گردشی (Round Robin) را به کار می برد بدین صورت عمل می‌کند: پس از کامل شدن یک دور اجرای همه پروسس‌های آماده (ready)، time slice هر پروسس آماده موجود را برای دور بعدی اجرا محاسبه می‌کند. این time slice برای هر پروسس از تقسیم ماکزیمم زمان پاسخ دور قبلی بر تعداد پروسسهای آماده فعلی بدست می‌آید. این سیاست را چگونه می‌توان توجیه کرد؟ یا این سیاست منطقی است؟ بحث کنید (از تحلیل تغییرات یارامترهای زمان بندی مثل متوسط زمان انتظار، درصد بهره وری سیستم،

Flag question

متوسط زمان پاسخ و ... استفاده کنید)

ب) به بهترین نحوی که میتوانید توضیح دهید یا نشان دهید الگوریتم **CFS** که برای زمان بندی در لینوکس استفاده می‌شود عادلانه است. سپس توضیح دهید که آیا ممکن است شرایطی در سیستم ایجاد شود که گرسنگی برای برخی پروسس‌ها با اجرای این الگوریتم به وجود آید؟ (اگر بله، توضیح دهید چه شرایطی و اگر خیر، بنویسید چرا)

ج) زمان ورود و اندازه burst تعدادی پروسس در زیر مشخص شده است. **متوسط زمان انتظار و متوسط زمان پاسخ** را برای هر کدام از الگوریتمهای **Shortest Remaining Time First** و **Round Robin با کوانتوم ۴** با رسم جزییات گانت چارت و انجام محاسبات لازم بدست آورید.

پروسس	زمان ورود	اندازه burst
p1	2	4
p2	0	10
p3	3	2
p4	4	4

الف) این سیاست منطقی نیست

زیرا اگر نحوه وارد شدن پروسه ها به صف به نحوی باشد که زمان پاسخ ماکزیموم پایین بماند، و همینطور تعداد پروسه های در صف نسبت به آن زیاد باشد، تایم اسلایس به شدت کم شده و overhead سیستم بالا میرود و cpu utilization کاهش میابد

به طور مثال

process	arrive time	burst time
p0	0	1000
p1	4	1000
p2	7	1000

حال اگر فرض کنیم در ابتدای اجرا کوانتوم 4 باشد، پس از اولین دور ماکزیموم زمان پاسخ 1 میشود که به نسبت 3 کوچک است

حال اگر این سناریو را در مقیاس بزرگ تر در نظر بگیریم، تایم اسلایس به شدت کم میشود

ب)

با استفاده از تعاریف niceness و vruntime، اولویت بندی و انتخاب از صف به صورت عادلانه صورت میگیرد

مقیاس niceness معکوس اولویت یک پروسه است، هر چه یک پروسه nice تر باشد، منابع را بیشتر در اختیار دیگر پروسه ها قرار میدهد

مقیاس vruntime، زمان ران تایم یک پروسه است که نسبت به niceness آن نرمالایز شده است

زمانبند برای انتخاب، پروسه ای با کمترین vruntime را انتخاب میکند، و زمان اجرای آن را به نسبت niceness آن تنظیم میکند، هر چه یک کمتر nice باشد، زمان اجرای بیشتری میگیرد

مقیاس sched_latency این اطمینان خاطر را میدهد که از گرسنگی جلوگیری میشود و در یک بازه زمانی، همه پروسه ها حداقل یک بار اجرا شده و برای جلوگیری از overhead تعویض متن، زمان اجرای آن ها یک حداقل مقداری دارد (مثلا 6ms)

ج)

پیوست شده است

photo_2021-11-17_18-17-22.jpg

الف))(۰.۵ نمره) فقط تا وقتی که تعداد پروسسها نسبتا کم باشد این سیاست منطقی است. وقتی کوانتوم زمانی کوچکتر میشود تا تعداد پروسسهای بیشتری در مدت مشخصی پاسخ داده شوند، دو اتفاق می افتد: ۱) میزان بهره وری CPU کاهش میابد. در یک لحظه ای کوانتوم

اونقدر کوچک میشود که اکثر درخواستها در این زمان تکمیل نمیشود و پروسسها باید دفعات زیادی در صف نوبت - چرخشی قرار گیرند پس اگر پروسسها IO bound باشند و تعدادشان هم کم باشد این الگوریتم خوب است از این جهت که تلاش میکند متوسط زمان پاسخ و انتظار را کاهش دهد اما اگر تعداد پروسسها خیلی زیاد شود و پروسسها CPU bound باشند بهره وری کم میشود و متوسط زمان انتظار هم میتواند بالا رود چون دفعات زیادی هر پروسس باید در صف بماند تا دوباره نوبتش شود و burstش پایان یابد البته سیستم مورد سوال یک سیستم تعاملی است که پروسسها در آن IO bound هستند پس تا وقتی که تعداد پروسسها خیلی زیاد نشود این راهکار نسبتا منطقی است

ب) (۰.۵ نمره) از آنجایی که این الگوریتم سعی میکند یک بازه زمانی مشخص را بین پروسسهای موجود سیستم با توجه به اولویتشان تقسیم کند، عادلانه است. میزان سهم هر پروسس در این بازه زمانی بر اساس اولویت و میزان CPU که تا به حال استفاده کردند میباشد. لذا از بین پروسسهای با الویت یکسان، انهایی که تاکنون cpuکمتری گرفته اند سهم بیشتری میگیرند تا میزان cpu ی که گرفته اند کم کم در اندازه همدیگر شود. از طرفی چنین پروسسهایی زودتر از بقیه هم cpuمیگیرند که این هم با توجه به اینکه تا به حال cpu کمتری دریافت کرده اند عادلانه است
گرسنگی ممکن است به وجود اید اگر مرتبا پروسسهای جدیدی به سیستم اضافه شوند که چون runtime آنها صفر است اگر اولویت بالایی هم داشته باشند بللافاصله بعد ورود cpu میگیرند و این کار میتواند با ورود چنین پروسسهایی تکرار شود و پروسسهای قبلی سیستم گرسنه بمانند

ج) ۱ نمره

Comment:

الف) ۰.۴

ب) ۰.۳

ج) ۱

Question 7

Complete

Mark 1.00 out of 1.00

Flag question

این کد را به صورتی تکمیل کنید که **دقیقا** اهداف زیر برآورده شود و نتیجه اجرا به صورتی باشد که شرح داده شده است:

میخواهیم کلمه printf را درون ۴ فایل بزرگ جستجو کرده و خطهای حاوی کلمه موردنظر را در ۴ فایل جداگانه ریخته و نهایتا محتوای این ۴ فایل را پس از این که هر ۴ فایل خروجی تکمیل شدند در خروجی استاندارد چاپ کنیم. در این جستجو جهت **راحتی کار**، از grep و **جهت تسریع جستجو** از مالتی پروسسینگ استفاده میکنیم. (فرض کنید تابع cat_file محتوای یک فایل را در خروجی استاندارد چاپ میکند)

```
int main(int argc, char **argv)
{
    pid_t pid[4];
    char fname[4][50];
    char fout[4][50];
    for(int i=0; i<4;i++){
        sprintf(fout[i],"out_%d.txt", i+1);
    }


            execl("/bin/grep", "grep "," printf", fname[i] , " > ", fout[i], NULL);




    for(int i=0; i<4; i++)
        cat_file(fout[i]);

    printf("--end of main program--\n");
    return 0;
}
```

```
//after sprintf loop
    for(int i=0;i<4;i++) //getting file names
        scanf("%s",fname[i]);
    for(int i=0;i<4;i++)
        if((pid[i]=fork())==0)
            execl("/bin/grep", "grep "," printf", fname[i] , " > ", fout[i], NULL);
    for(int i=0;i<4;i++)
        wait();
//before cat_file loop
```

```c
int main(int argc, char **argv)
{
    pid_t pid[4];
    char fname[4][50];
    char fout[4][50];
    for(int i=0; i<4;i++){
        sprintf(fout[i],"out_%d.txt", i+1);
    }
    for(int i=0; i<1;i++){
        scanf("%s", fname[i]);
        pid[i] = fork();
        if (pid[i]==0){
            execl("/bin/grep", "grep "," printf", fname[i],NULL);

        }
    }

    for(int i=0; i<4; i++)
        wait(NULL);
    for(int i=0; i<4; i++)
        cat_file(fout[i]);

    printf("--end of main program--\n");
    return 0;
}
```

اگر پروسسهای اضافه ساخته میشود ۰.۳ کم شده

اگر مقدار i برای هر پروسس فرزند حین اجرای exex صحیح نباشد ۰.۱ شده

چهار waitقبل از چاپ نتیجه نیاز است. در صورتی که کلا wait نگذاشته باشید ۰.۳ و در صورتی که بعد چاپ گذاشته باشید ۰.۲ کم شده است.

Comment:

Finish review