تمرین دوم سیستم‌های عامل

اشکان شکیبا (۹۹۳۱۰۳۰)

سوال اول

الف) APIها یا Application Programming Interfaces برای تعامل با سیستم عامل و دیگر برنامه‌ها بکار میروند. در حالی که system callها به عنوان یک مکانیزم پایه برای برنامه‌نویسی در سیستم عامل‌ها استفاده می‌شوند.

استفاده مستقیم از system callها می‌تواند در برخی موارد بسیار پیچیده و دشوار باشد، به عنوان مثال برای ایجاد یک فایل، شما باید بتوانید بایت‌های داده را به صورت دستی باز کنید و آنها را در فایل بنویسید. در صورت استفاده از APIها، میتوانید از توابعی مانند fopen و fwrite استفاده کنید تا کارهایی از این قبیل را انجام دهید. APIها عموماً از میان لایه‌های بالاتر سیستم عامل استفاده می کنند و در نتیجه از خطاهای کمتری برخوردار هستند و همچنین قابلیت قابل توجهی برای انجام کارهای مختلف دارند، ضمن اینکه به برنامه‌نویسان قابلیت دیباگ آسان‌تر نیز می‌دهند.

علاوه بر این، APIها معمولاً به شکل کتابخانه‌هایی ارائه می‌شوند که می‌توان آنها را به پروژه‌های بزرگ اضافه کرد و از آنها به عنوان یک ماژول قابل استفاده در سایر پروژه‌ها استفاده کرد. به این ترتیب، کاربران به راحتی می‌توانند از امکانات مختلفی که ارائه شده استفاده کنند، بدون اینکه بخواهند خودشان تمام کد مربوط به آن را بنویسند.

در کل، استفاده از APIها به عنوان یک روش انتزاعی برای دسترسی به سیستم عامل و دیگر برنامه‌ها، توسعه برنامه را سریع‌تر و ساده‌تر می‌کند و

خطاهایی را که ممکن است در استفاده مستقیم از system callها بروز دهد، به حداقل می‌رساند.

ب) یک runtime environment معمولاً مجموعه‌ای از ابزار و کتابخانه‌هایی است که برای اجرای برنامه‌ها در یک محیط خاص ایجاد شده است. این runtime environment می‌تواند شامل کامپایلرها، مفسرها، اجراگرها، کتابخانه‌های اجرایی، ماشین‌های مجازی و موارد دیگر باشد.

یکی از وظایف این runtime environment، فراهم کردن واسط بین برنامه و سیستم عامل است. این واسط عموماً شامل مجموعه‌ای از system callها است که به صورت API برای برنامه‌نویسان در دسترس است.

استفاده از system callها بدون واسط runtime environment بسیار پیچیده است، زیرا برنامه‌نویس باید خودش کد سیستمی را بنویسد تا بتواند به منابع سیستم دسترسی داشته باشد. با وجود runtime environment، برنامه‌نویس می‌تواند به راحتی از این سرویس‌های سیستمی استفاده کند و به منابع سیستم دسترسی داشته باشد.

همچنین، runtime environment معمولاً به صورت استانداردی برای سیستم عامل خاصی طراحی شده است و تفاوت‌های سیستم عاملی که برنامه در آن اجرا می‌شود، به‌طور کامل در این واسط محو می‌شود. به عبارت دیگر، برنامه‌نویس نیازی ندارد کد خاصی برای هر سیستم عاملی که برنامه‌اش در آن اجرا می‌شود بنویسد و فقط کافی است که از سرویس‌های موجود در runtime environment استفاده کند.

علاوه بر این موارد، runtime environment با افزودن یک لایه امنیتی از دسترسی‌های مخرب جلوگیری می‌کند.

بنابراین، وجود runtime environment به برنامه‌نویسان کمک می‌کند تا به راحتی از سرویس‌های سیستمی استفاده کنند و کد قابل حمل بنویسند که بدون مشکل در سیستم‌های مختلف اجرا شود.

سوال دوم

الف) printf, fgets, strlen, return, fopen, fprintf, fclose

همچنین موارد دیگری چون malloc و free که در پس‌زمینه اجرا می‌شوند.

ب) یکی از روش‌های انتقال پارامترها، ذخیره‌سازی مقادیر آنها در رجیسترهاست.

روش دیگر ذخیره‌سازی آنها در حافظه و آدرس‌دهی به محل ذخیره آنها در رجیسترهاست.

یک روش دیگر ذخیره مقادیر در استک است.

پ) خیر، چون سیستم کال‌های مورد استفاده و نحوه فراخوانی آنها در سیستم‌های مختلف، متفاوت است و کامپایلر هر سیستمی این کد را بسته به ویژگی‌های همان سیستم ترجمه می‌کند؛ از این رو نمی‌توان آن را در هر سیستم دیگری اجرا کرد.

سوال سوم

سیستم کال‌ها را می‌توان به عنوان یک نوع از وقفه‌های software interrupt در نظر گرفت که از نوع synchronous و trap هستند.

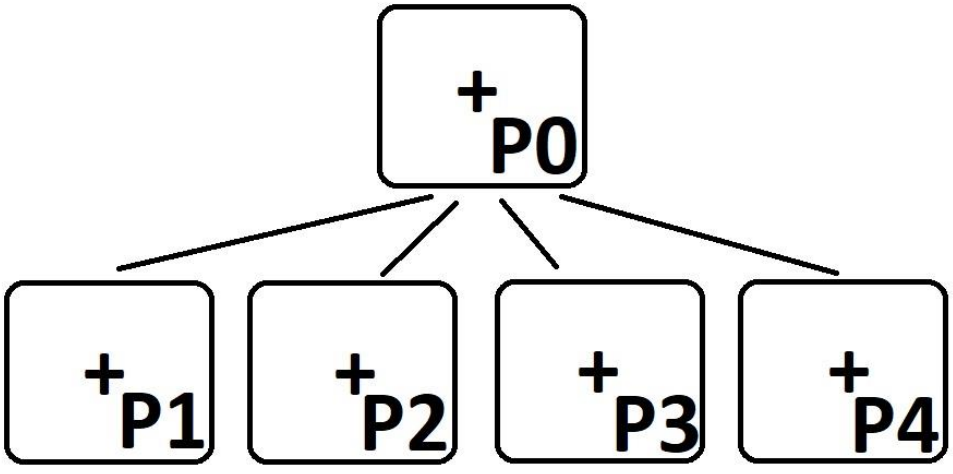
مراحل اجرای آن نیز شامل ذخیره پارامترهای مورد نیاز در رجیسترهای مربوطه و اجرای دستور interrupt است که باعث ایجاد وقفه در روند اجرای برنامه شده و پس از اجرای کد مربوط به سیستم کال، خروجی آن به محل فراخوانی بازگردانده شده و برنامه به ادامه روند پیشین خود می‌پردازد.

سوال چهارم

الف) خروجی:

+++++

درخت پردازه‌ها:

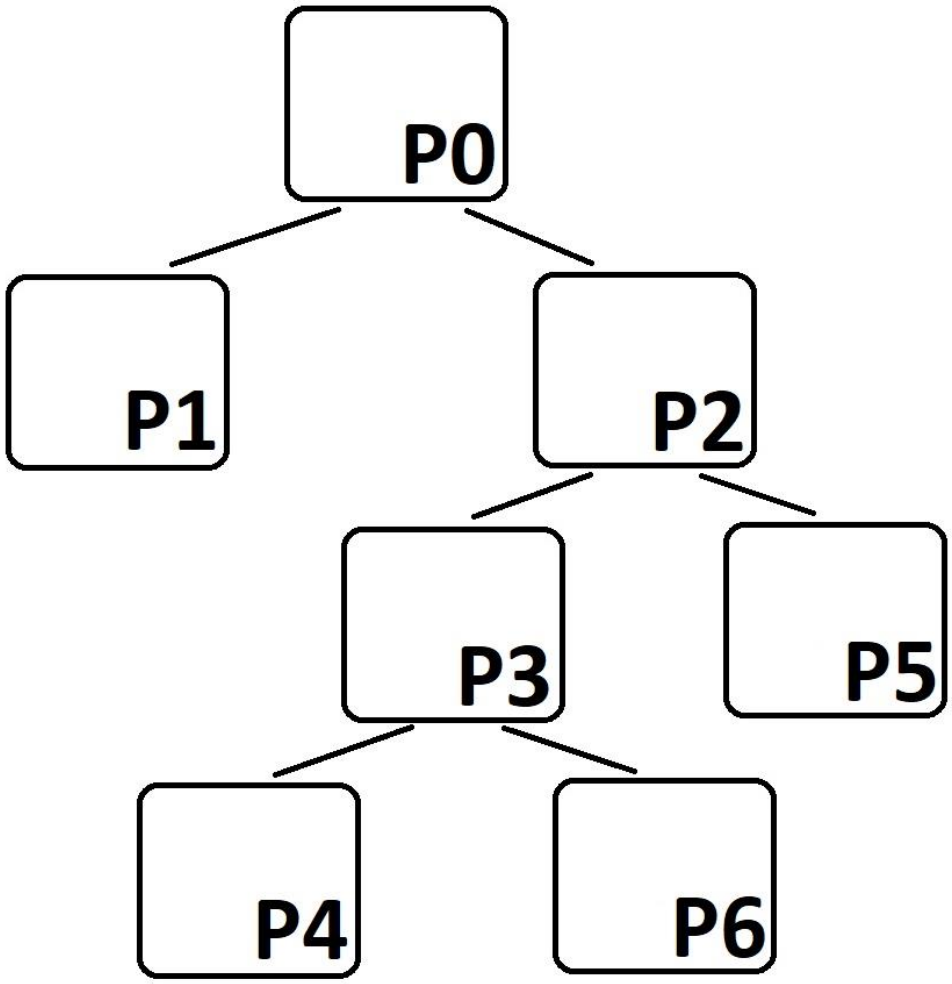


حاصل fork در پردازه والد همواره عددی بزرگ‌تر از صفر است که از نظر منطقی معادل true در نظر گرفته می‌شود، بنابراین چهار پردازه فرزند ساخته می‌شوند و برنامه به خط بعد می‌رود. هر یک از پنج پردازه یک بار + را پرینت می‌کنند.

ب) خروجی:

ندارد! چون هیچ printی نداریم.

درخت پردازه‌ها:

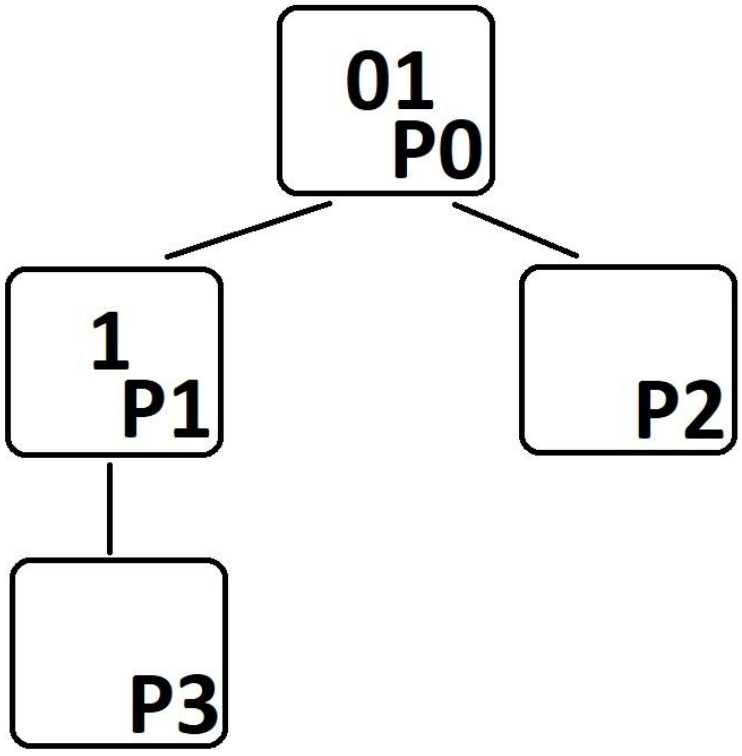


ابتدا با هر یک از forkهای if بیرونی یک پردازه فرزند ساخته می‌شود. یکی از آنها شرط if را نقض می‌کند اما دیگری وارد آن شده و سپس در شرط if درونی دو بار دیگر fork اجرا می‌شود و دو پردازه فرزند می‌سازد که مجددا یکی از آنها شرط if را نقض می‌کند و دیگری مجددا fork می‌شود.

ج) خروجی:

011

درخت پردازه‌ها:



ابتدا یک بار پردازه والد 0 را پرینت می‌کند و سپس یک فرزند مشابه خود می‌سازد. سپس هر دو پردازه با مقدار i=1 وارد حلقه شده و 1 را پرینت می‌کنند و هر یک فرزندی مشابه خود می‌سازند. یک بار دیگر هر چهار پردازه با مقدار i=2 شرط حلقه را نقض می‌کنند و به پایان می‌رسند.

سوال پنجم

سیستم کال exec پردازه در حال اجرا را با پردازه‌ای دیگر که اطلاعات آن پیش‌تر در جایی از حافظه ذخیره شده جایگزین می‌کند، که شامل جایگزینی دستورات برنامه و همچنین حافظه مورد نیاز آن است.

از کاربردهای اصلی این سیستم کال زمانی‌ست که برنامه‌ای بخواهد پردازه‌ای جدید اجرا کند؛ برای این کار ابتدا fork انجام می‌شود که منجر به ساخت فرزندی مشابه والد می‌گردد و سپس با اجرای exec در فرزند (که حاصل fork در آن صفر شده است)، پردازه مشابه والد با پردازه مورد نظر جایگزین می‌شود و به ما امکان ساخت پردازه‌های جدید و متفاوت از والد می‌دهد.

بخش عملی

کد:

```c
##include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();
    if (pid > 0) {
        sleep(60);
    }
    else {
        sleep(0);
    }
    return 0;
}
```

خروجی ps –l:

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    34    33  0  80   0 -  3594 -      tty1     00:00:00 bash
0 S  1000   112     1  0  80   0 -  2670 -      tty1     00:00:00 a.out
0 S  1000   120    34  0  80   0 -  2670 -      tty1     00:00:00 a.out
0 Z  1000   121   120  0  80   0 -     0 -      tty1     00:00:00 a.out <defunct>
0 R  1000   122    34  0  80   0 -  3880 -      tty1     00:00:00 ps
```

چهارمین پردازه که مقدار ستون S آن Z (مخفف zombie) است و آی‌دی ۱۲۱ دارد، یک پردازه زامبی است که والد آن (پردازه ۱۲۰) برای آن wait نکرده است.

توضیح اجزای خروجی:

ستون F فلگ پردازه را مشخص می‌کند.

ستون S وضعیت پردازه را مشخص می‌کند که در حال اجرا، غیرفعال یا زامبی باشد.

ستون PID آی‌دی پردازه و ستون PPID آی‌دی پردازه والد را مشخص می‌کنند.

ستون C مشخص‌کننده این است که پردازه چند درصد از یک هسته پردازنده را استفاده می‌کند.

ستون PRI اولویت پردازه را مشخص می‌کند.

ستون NI مقدار nice value پردازه را نشان می‌دهد.

ستون ADDR محل ذخیره شدن پردازه در حافظه را نمایش می‌دهد.

ستون SZ نشان‌دهنده اندازه فضای اشغال شده از حافظه توسط پردازه است.

ستون WCHAN نشان‌دهنده آدرس تابعی‌ست که کرنل در آن wait می‌کند.

ستون TTY ترمینال مبدا پردازه را مشخص می‌کند.

ستون TIME زمان CPU time پردازه را نمایش می‌دهد.

ستون CMD نشان‌دهنده دستوری‌ست که پردازه را اجرا کرده است.