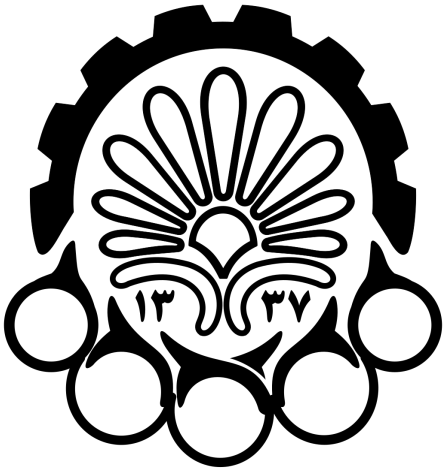
سیستم‌های عامل

دکتر جوادی



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین سری اول

۴ آبان ۱۴۰۲

سیستم‌های عامل

تمرین سری اول

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

دانشکده مهندسی کامپیوتر

به سوالات زیر پاسخ دهید

الف) هدف از DMA چیست؟

پاسخ

هدف اصلی استفاده از DMA افزایش سرعت انتقال داده ها بین دستگاه‌های I/O و حافظه سیستم است. با استفاده از DMA می‌توان مستقیما و بدون نیاز به مداخله CPU به I/O و حافظه‌ها دسترسی پیدا کرد و دیتا را به‌صورت مستقیم انتقال داد.

ب) چگونه می توان سیستمی طراحی کرد که اجازه ی انتخاب یک سیستم عامل از چند سیستم عامل را هنگام بوت شدن به کاربر بدهد؟ برنامه‌ی Bootstrap برای این منظور چه کاری باید انجام دهد؟

پاسخ

برای این کار باید برنامه ای نوشته شود که در زمان Boot سیستم اجرا شود. این برنامه باید توانایی نمایش رابط کاربری مناسب (مانند منو انتخاب سیستم‌عامل) را داشته باشد و بتواند با ارتباط با I/O های سیستم ورودی کاربر که OS انتخابی آن است را دریافت کند.
همچنین این برنامه باید قابلیت این را داشته باشد که پس از دریافت سیستم‌عامل انتخابی توسط کاربر، بتواند فایل ها و داده های مورد نیاز برای اجرای OS را بارگذاری کند و پس از اجرای OS باید برنامه Bootstrap کنترل سیستم را به سیستم‌عامل بدهد.

پ) توضیح دهید که تفاوت بین حالت کرنل و حالت کاربر چگونه به حفاظت و امنیت سیستم کمک میکند. کدام یک از دستورات زیر باید در حالت کرنل اجرا شوند؟

A) Set value of timer

B) Read the clock

C) Clear memory

D) Issue a trap instruction

E) Turn off interrupts

F) Modify entries in device-status table

G) Switch from user to kernel mode

H) Access I/O device

پاسخ

حالت کرنل و حالت کاربر دو حالت اجرایی در OS هستند که تفاوت های مهمی در امنیت و حفاظت سیستم دارند. در حالت کرنل، برنامه‌ها و سرویس‌های سیستم با دسترسی کامل به منابع سخت‌افزاری و سیستم عامل اجرا می‌شوند و در حالت کاربر، برنامه‌ها تنها با دسترسی محدود به منابع سیستم عامل اجرا می‌شوند.
حالت کرنل به دلایل زیر به حافظت و امنیت سیستم کمک می‌کند:

۱. **محدودیت دسترسی:** در حالت کاربر، برنامه‌ها دسترسی محدودتری به منابع سیستم عامل دارند و نمی‌توانند به منابع حساس مثل حافظه سیستم یا دستگاه‌های I/O مستقیماً دسترسی داشته باشند. در حالت کرنل، سرویس‌ها و برنامه‌های سیستم عامل با دسترسی کامل به منابع سیستم عامل اجرا می‌شوند، اما این دسترسی برای برنامه‌های کاربردی محدود می‌شود.

۲. **جدا بودن فضای آدرسی:** در حالت کرنل و کاربر، فضای آدرسی برای برنامه‌ها جداگانه تعیین می‌شود. در حالت کاربر، برنامه‌ها تنها به فضای آدرسی خودشان دسترسی دارند و نمی‌توانند به فضای آدرسی برنامه‌های دیگر یا سیستم عامل دسترسی داشته باشند. این از خطرات نفوذ و دسترسی غیرمجاز جلوگیری می‌کند.

۳. **محدودیت دسترسی به سخت‌افزار:** در حالت کاربر، برنامه‌ها نمی‌توانند به دستگاه‌های سخت‌افزاری مستقیماً دسترسی داشته باشند و باید از طریق واسط‌های سیستم عامل از آن‌ها استفاده کنند. در حالت کرنل، سرویس‌ها و برنامه‌های سیستم عامل می‌توانند به طور مستقیم با دستگاه‌های سخت‌افزاری ارتباط برقرار کنند

دستوراتی که مستقیماً با منابع سیستم عامل یا سخت‌افزار ارتباط برقرار می‌کنند، مانند تنظیم تایمر، پاکسازی حافظه، صدا زدن دستور توقف (trap) و دسترسی به دستگاه‌های I/O، در حالت کرنل باید اجرا شوند. این دستورات نیاز به دسترسی به منابع حساس سیستم دارند که در حالت کاربر محدود می‌شود. دستوراتی که مستقیماً با منابع کاربردی برنامه‌ها ارتباط برقرار می‌کنند، مانند خواندن ساعت، تغییر I/O و مدیریت حافظه، در حالت کاربر باید اجرا شوند. این دستورات معمولاً نیاز به دسترسی مستقیم به منابع کاربران دارند و در حالت کرنل اجرا نمی‌شوند.

بنابراین فقط مورد B است که باید در حالت کاربر اجرا شود و بقیه موارد همگی در حالت کرنل اجرا می‌شوند.

د) در یک محیط Multi programming و Time sharing چند کاربر به صورت همزمان سیستم را به اشتراک می‌گذارند و این وضعیت می‌تواند منجر به مشکلات امنیتی مختلف شود. ۲ مورد از این مشکلات را نام ببرید.

پاسخ

۱. **کلاهبرداری از داده‌ها:** وجود چند کاربر در یک سیستم به اشتراک گذاشته شده ممکن است باعث افزایش ریسک کلاهبرداری از داده‌ها شود. اگر یک کاربر از طریق آسیب‌پذیری‌های امنیتی در سیستم، به داده‌های دیگری که توسط کاربران دیگر در حافظه سیستم قرار دارد، دسترسی پیدا کند، می‌تواند اطلاعات حساس را بدون اجازه و به طور غیرمجاز به دست آورد. این مشکل می‌تواند منجر به فاش شدن اطلاعات شخصی، رمزهای عبور، داده‌های حساس کسب و کار و سایر اطلاعات محرمانه شود.

۲. **تداخل در حافظه و منابع سیستم:** وجود چند کاربر در یک سیستم به اشتراکزمان، ممکن است منجر به تداخل‌های حافظه و منابع سیستم شود. زمانی که چند کاربر به صورت همزمان در حال اجرای برنامه‌ها و پردازش‌های مختلف هستند، ممکن است منابع سیستم مانند حافظه، پردازنده و I/O به طور ناهمزمان و نامتعادل مورد استفاده قرار گیرند. این موضوع می‌تواند منجر به افزایش زمان پاسخ و کندی عملکرد برنامه‌ها شود. همچنین، در مواردی که هر کاربر به منابع سیستم با دسترسی محدود دسترسی دارد، تداخل‌ها می‌توانند باعث کاهش کارایی و عملکرد کاربران شود.

به سوالات زیر پاسخ دهید

الف) وقفه چیست؟ وقفه‌های سنکرون و آسنکرون را باهم مقایسه کنید.

پاسخ

وقفه (Interrupt) در محیط برنامه‌نویسی به وقوع پیوستن یک رویداد ناگهانی در حین اجرای برنامه گفته می‌شود که عملکرد طبیعی برنامه را متوقف می‌کند و برنامه‌ای را به اجرای یک کد خاص یا روند دیگر تغییر می‌دهد. وقفه‌ها معمولاً توسط سخت‌افزار و سیستم عامل به منظور پاسخگویی به رویدادهای مهم مانند درخواست‌های I/O، خطاها، تایمرها و سایر رویدادها ایجاد می‌شوند.
از نظر زمان وقوع، وقفه‌ها به دو دسته سنکرون و آسنکرون تقسیم می‌شوند:

۱. وقفه‌های سنکرون:

(آ) وقوع وقفه در زمانی قرار دارد که برنامه در یک نقطه مشخص خود را در حالت انتظار قرار می‌دهد و منتظر وقوع وقفه است.

(ب) برنامه‌ای که با وقوع وقفه مواجه می‌شود، به طور مستقیم و بلافاصله وارد روند وقفه می‌شود و ادامه اجرای برنامه بعد از پایان وقفه ادامه می‌یابد.

(ج) معمولاً وقفه‌های سنکرون توسط سخت‌افزار ایجاد می‌شوند، مانند درخواست‌های ورودی کاربر، تقاضای دستگاه‌های جانبی و غیره.

۲. وقفه‌های آسنکرون:

(آ) وقوع وقفه در زمانی قرار دارد که برنامه در حال اجرا است و به طور غیرمنتظره با یک رویداد ناگهانی مواجه می‌شود.

(ب) وقفه آسنکرون می‌تواند در هر نقطه‌ای از اجرای برنامه رخ دهد و برنامه را به وقفه‌هایی مانند خطاها، سیگنال‌های سیستم عامل، تقاضای دیگر برنامه‌ها و غیره وصل می‌کند.

(ج) وقفه‌های آسنکرون برنامه را از جریان اصلی آن جدا کرده و به روند وقفه منتقل می‌کنند. بعد از پایان وقفه، برنامه از جایی که متوقف شده بود، ادامه می‌یابد.

(د) معمولاً وقفه‌های آسنکرون توسط سخت‌افزار (مانند خطاهای سخت‌افزاری) یا سیستم عامل (مانند سیگنال‌های سیستم عامل) ایجاد می‌شوند.

ب) تفاوت‌های بین Interrupt و Trap را توضیح دهید.

پاسخ

وقفه و Trap هردو پدیده‌هایی در محیط برنامه‌نویسی هستند که به وقوع پیوستن رویدادهای ناگهانی در حین اجرای برنامه را مشخص می‌کنند. اما تفاوت‌هایی بین این دو وجود دارد:

۱. Interrupt

(آ) وقفه‌ها معمولاً توسط سخت‌افزار یا سیستم عامل ایجاد می‌شوند و می‌توانند در هر زمانی و در هر نقطه‌ای از اجرای برنامه رخ دهند.

(ب) هدف اصلی وقفه‌ها، متوقف کردن عادی برنامه و پاسخگویی به رویدادهای مهم است. مثال‌هایی از وقفه‌ها شامل درخواست‌های ورودی کاربر، تقاضای دستگاه‌های جانبی، خطاها و تایمرها می‌شوند.

(ج) وقفه‌ها معمولاً باعث تغییر جریان اجرای برنامه می‌شوند. برنامه به طور مستقیم و بلافاصله به یک روند وقفه منتقل می‌شود و پس از پایان وقفه، به جریان اصلی خود بازگشت می‌کند.

۲. Trap

(آ) ترپ‌ها معمولاً توسط خود برنامه نوشته شده و قابلیت اجرای آنها وجود دارد. معمولاً در نقاط خاصی از برنامه قرار داده می‌شوند تا در صورت بروز شرایط خاص، عملیات خاصی انجام دهند.

(ب) هدف اصلی ترپ‌ها، نیاز به یک رفتار خاص در برنامه است و معمولاً برای انجام عملیات‌های خاص (مانند خطاها، استثناها و غیره) استفاده می‌شوند.

(ج) ترپ‌ها برای تعامل با سیستم عامل یا سخت‌افزار می‌توانند استفاده شوند. به عنوان مثال، یک برنامه می‌تواند ترپی برای درخواست سیستم عامل برای اختصاص حافظه یا فایل‌ها داشته باشد.

(د) ترپ‌ها معمولاً توسط برنامه بررسی می‌شوند و در صورت بروز شرایط، برنامه به طور دستوری به روند ترپ منتقل می‌شود. پس از انجام عملیات ترپ، برنامه به جریان اصلی خود بازگشت می‌کند.

به طور خلاصه، وقفه‌ها معمولاً توسط سخت‌افزار یا سیستم عامل ایجاد می‌شوند و هدف اصلی آنها پاسخگویی به رویدادهای مهم است. ترپ‌ها به طور کلی توسط برنامه نوشته شده و هدف اصلی آنها انجام عملیات خاص در برنامه است.

پ) فرآیند مدیریت یک وقفه از لحظه ایجاد شدن تا اتمام آن را توضیح دهید. فرض کنید وقفه متعدد نداریم و CPU مشغول انجام برنامه کاربر است.

پاسخ

فرآیند مدیریت یک وقفه از لحظه ایجاد شدن تا اتمام آن عموماً توسط سیستم عامل و سخت‌افزار انجام می‌شود. در زیر، مراحل اصلی مدیریت یک وقفه را توضیح می‌دهم:

۱. **شناسایی وقفه:** سیستم عامل و سخت‌افزار در هر لحظه وقفه‌ها را بررسی می‌کنند. این بررسی ممکن است توسط سخت‌افزار (مانند تایمرها، درخواست‌های دستگاه‌های جانبی و ...) یا سیستم عامل (مانند درخواست‌های ورودی کاربر و ...) صورت گیرد.

۲. **ذخیره وضعیت فعلی:** سیستم عامل اطلاعات مربوط به وضعیت فعلی برنامه را در یک مکان مناسب ذخیره می‌کند. این کار از طریق استفاده از مکانیزمی به نام Context Switch انجام می‌شود.

۳. **اجرای روند وقفه:** پس از ذخیره وضعیت فعلی، سیستم عامل به Interrupt Handler منتقل می‌شود. روند وقفه کدی است که توسط سیستم عامل تعریف شده است و وظیفه پاسخگویی به وقفه را دارد. در این مرحله، عملیات مربوط به وقفه انجام می‌شود (مانند پاسخ به درخواست کاربر، خواندن داده از دستگاه جانبی و ...)

۴. **بازگشت به برنامه اصلی:** بعد از اتمام عملیات وقفه، سیستم عامل وضعیت قبلی را بازیابی می‌کند. این شامل بازگشت به وضعیت قبلی ثبات‌های CPU، شمارنده‌ها و سایر اطلاعات مربوطه است.

۵. **ادامه اجرای برنامه:** سیستم عامل بعد از بازگشت به وضعیت قبلی، اجرای برنامه را از جایی که قبل از وقفه متوقف شده بود، ادامه می‌دهد. در این مرحله، جریان اجرای برنامه به همان نقطه قبل از وقفه برمی‌گردد و برنامه از همان جایی که متوقف شده بود ادامه می‌یابد.

این فرآیند مدیریت وقفه معمولاً به صورت خودکار توسط سیستم عامل و سخت‌افزار انجام می‌شود و برنامه نویس نیازی به دخالت مستقیم در این فرآیند ندارد.

در هر یک از موارد زیر، پردازنده از چه حالتی به چه حالت دیگری تغییر وضعیت می‌دهد؟ (منظور از حالت، وضعیت های مختلف پردازنده شامل Waiting، Ready، Terminated، New، Running است)

الف) در حین اجرای پردازنده، کاربر کلیدی را فشار داده و وقفه ای با اولویت بالا تر در سیستم اتفاق می‌افتد.
ب) پردازه در حین اجرای کد خود، به جایی می‌رسد که نیاز به دریافت داده‌ها از طریق شبکه دارد
پ) رویداد مربوط به یکی از دستگاه های ورودی/خروجی به پایان رسیده و داده‌های مورد نیاز پردازنده آماده می‌شود.
ت) برنامه ای به زبان C تابع `()exit` را فراخوانی می‌کند.
ث) زمان‌بندی سیستم‌عامل، پردازنده ای را از صف انتظار خارج کرده و به آن اجازه اجرا می‌دهد.

پاسخ

در هر یک از موارد زیر، پردازه از یک حالت به حالت دیگر تغییر وضعیت می‌دهد:

۱. **وقفه با اولویت بالا:**

(آ) وضعیت قبلی: Running

(ب) وضعیت جدید: Interrupted/Waiting

وقفه با اولویت بالا می‌تواند پردازه‌ای که در حال اجرا است را متوقف کند و به حالت وقفه (Interrupted) یا انتظار (Waiting) برود.

۲. **درخواست دریافت داده‌ها از شبکه:**

(آ) وضعیت قبلی: Running

(ب) وضعیت جدید: Waiting

پردازه در حال اجرا برای دریافت داده‌ها از شبکه نیاز به انتظار (Waiting) دارد تا داده‌های مورد نیاز آماده شوند.

۳. **پایان رسیدن رویداد دستگاه ورودی/خروجی:**

(آ) وضعیت قبلی: Waiting

(ب) وضعیت جدید: Ready

رویداد مربوط به دستگاه ورودی/خروجی به پایان رسیده و داده‌های مورد نیاز پردازنده آماده می‌شود، در نتیجه پردازه از حالت انتظار (Waiting) به حالت آماده (Ready) تغییر وضعیت می‌دهد.

۴. **فراخوانی تابع `()exit`:**

(آ) وضعیت قبلی: Running

(ب) وضعیت جدید: Terminated

هنگامی که برنامه C تابع ()exit را فراخوانی می‌کند، پردازه به حالت پایانی (Terminated) می‌رود و اجرای آن به پایان می‌رسد.

۵. پردازنده در هنگام اجرای کد خود، به جایی می‌رسد که نیاز به دریافت داده‌ها از شبکه دارد.

(آ) وضعیت قبلی: Ready

(ب) وضعیت جدید: Waiting

خروجی قطعه کدهای داده شده چیست؟ درخت پردازه‌های هر برنامه را رسم و راه‌حل خود را توضیح دهید.

Listing 1: Some Code

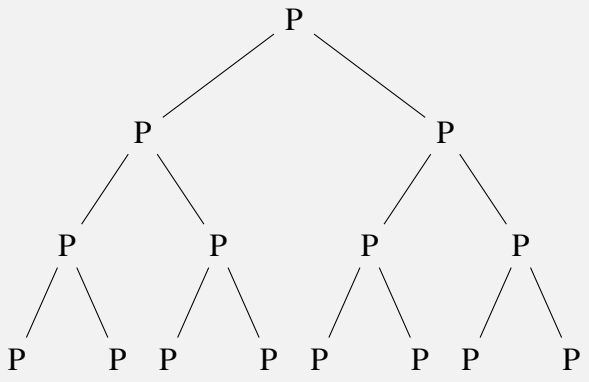
```
int(main)
{
        int i;
        for(i = 0; i < 3; ++i)
                fork();
        printf("Hello\n");
        return0;
}
```

پاسخ

خروجی کد به صورت زیر است:

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

در این کد، یک حلقه for وجود دارد که سه بار تکرار می‌شود. در هر تکرار، تابع fork() فراخوانی می‌شود. تابع fork() یک پردازه جدید را ایجاد می‌کند که از پردازه فعلی تولید شده است، به طوری که حالت اجرای برنامه پس از این نقطه در هر دو پردازه ادامه می‌یابد. بنابراین، در هر تکرار حلقه، تعداد پردازه‌ها دو برابر می‌شود.
در نتیجه، در این برنامه اصلی، ابتدا یک پردازه ایجاد می‌شود. در تکرار اول حلقه، یک پردازه دیگر ایجاد می‌شود و در نتیجه تعداد پردازه‌ها دو می‌شود. در تکرار دوم حلقه، هر یک از دو پردازه قبلی یک پردازه دیگر ایجاد می‌کنند و تعداد پردازه‌ها چهار می‌شود. در تکرار سوم حلقه، هر یک از چهار پردازه قبلی یک پردازه دیگر ایجاد می‌کنند و تعداد پردازه‌ها هشت می‌شود. پس از اتمام حلقه for، هر یک از هشت پردازه تولید شده عبارت "Hello" را چاپ می‌کند. بنابراین، در خروجی نهایی، عبارت "Hello" ۸ بار چاپ می‌شود.
درخت پردازش برای این برنامه به صورت زیر است:



Listing 2: Some Code

```
int(main)
{
        if(fork() && fork())
        {
                fork();
                fork();
        }
        printf("A");
        return0;
}
```

پاسخ

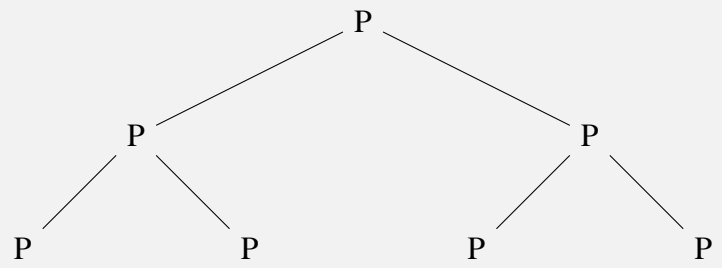
خروجی کد به صورت زیر است:

AAAA

در این برنامه، تابع fork() برای ایجاد پردازه‌های جدید استفاده می‌شود. با استفاده از این تابع، پردازه‌های فرزند از پردازه فعلی تولید می‌شوند. حلقه شرطی

if(fork() && fork())

باعث ایجاد چهار پردازه فرزند می‌شود. در اینجا، ابتدا یک پردازه فرزند ایجاد می‌شود و سپس دو پردازه فرزند از هر یک از پردازه‌های فرزند قبلی تولید می‌شوند. بنابراین، در نهایت، تعداد کل پردازه‌ها به چهار می‌رسد. سپس در داخل حلقه دوباره fork() فراخوانی می‌شود که باعث تولید چهار پردازه فرزند دیگر می‌شود. در نتیجه، تعداد کل پردازه‌ها به هشت می‌رسد. در نهایت، در همه‌ی پردازه‌ها، عبارت "A" چاپ می‌شود. بنابراین، خروجی نهایی برنامه "AAAA" خواهد بود. درخت پردازش برای این برنامه به صورت زیر است:



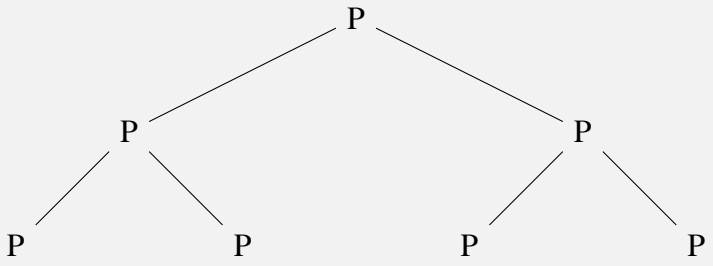
Listing 3: Some Code

```
int(main)
{
        int i = 2;
        while(i > 0)
        {
                i--;
                if(fork())
                {
                        fork();
                        printf("B");
                }
```

```
        }
        return0;
}
```

پاسخ

در این برنامه حلقه بی‌نهایت وجود دارد و خروجی برنامه شامل بی‌نهایت عبارت B است. به دلیل این که هر بار که `fork()` فراخوانی می‌شود، تعداد پردازه‌ها دو برابر می‌شود، درخت پردازه‌ها بسیار پیچیده و حجیم خواهد شد. درخت ممکن است به طور نمایی رشد کند و نمی‌توان آن را به صورت کامل رسم کرد.
در شکل زیر درخت پردازه را صرفا برای دور اول حلقه آورده ایم:



Listing 4: Some Code

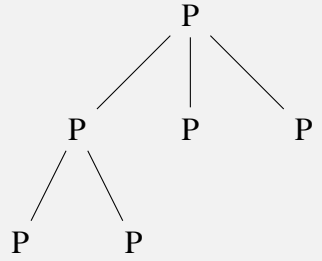
```
int(main)
{
        if(fork() || fork() && fork())
        {
                fork();
                print("C")
        }
        return0;
}
```

پاسخ

خروجی کد به صورت زیر است:

```
CCCC
```

با توجه به شرط در عبارت شرطی، اگر حداقل یکی از دو `fork()` اولیه موفق باشد (یعنی یک فرآیند فرزند ایجاد شود)، عبارت داخل `if` اجرا خواهد شد.
درخت پردازش برای این برنامه به صورت زیر است:



به سوالات زیر پاسخ کامل دهید

الف) قطعه کد زیر را درنظر بگیرید:

Listing 5: Some Code

```
int(main)
{
        int count = 0;
        pid_t ret = fork;
        if(ret == 0)
                printf("count in child = %d \n", count)
        else
                count = 1;

        return0;
}
```

اگر والد دستور `count = 1` را قبل از اجرای فرزند برای اولین بار اجرا می‌کند، مشخص کنید مقدار چاپ شده توسط کد بالا چقدر است؟ توضیح دهید.

پاسخ

با توجه به قطعه کد داده شده، اگر والد دستور `count = 1` را قبل از اجرای فرزند برای اولین بار اجرا کند، مقدار چاپ شده توسط کد بال به صورت `count in child = 0` خواهد بود.
در ابتدا، متغیر `count` به مقدار 0 مقداردهی می‌شود. سپس با استفاده از `()fork`، یک فرآیند فرزند ایجاد می‌شود و ارزش بازگشتی `()fork` به متغیر ret اختصاص داده می‌شود. اگر `ret` برابر با 0 باشد، به این معنی است که کد در حال اجرا در فرآیند فرزند است. در این حالت، دستور

```
printf("count in child = %d \n", count)
```

اجرا می‌شود و مقدار `count` که در فرآیند فرزند است هنوز تغییر نکرده و برابر با 0 است.
اما اگر `ret` مقدار دیگری غیر از 0 داشته باشد، به این معنی است که کد در حال اجرا در فرآیند والد است. در این حالت، دستور `count = 1` اجرا می‌شود و مقدار `count` تغییر کرده و برابر با 1 می‌شود. اما هیچ دستوری برای چاپ مقدار `count` در فرآیند والد وجود ندارد.

ب) باتوجه به پردازه‌های Zombie و Orphan به سوالات زیر پاسخ دهید:

۱. توضیح دهید که هرکدام از این پردازه ها چگونه ایجاد می‌شوند و تفاوت آنها با یکدیگر چیست؟

پاسخ

(آ) **Orphan:**

i. وقتی یک فرآیند والد خود را قبل از اتمام یا خروج از برنامه ببندد، فرزندانش به عنوان پردازه‌های orphan به شمار می‌روند. پردازه‌های orphan در واقع پردازه‌هایی هستند که والدشان قبل از اتمام خود زنده نمی‌ماند و آن‌ها بدون والد باقی می‌مانند.

ii. در این حالت، سیستم عامل پردازه‌های orphan را به عهده می‌گیرد و آن‌ها را به یک پردازه والد جدید انتقال می‌دهد. پردازه جدید به عنوان والد جایگزین برای آن‌ها عمل می‌کند.

(ب) **Zombie:**

i. وقتی یک فرآیند فرزند خود را قبل از والدش ببندد و اجرای خود را به پایان برساند، به عنوان یک پردازه zombie باقی می‌ماند.

ii. در این وضعیت، پردازه فرزند اجرا خود را به پایان رسانده است ولی والد هنوز آن را با استفاده از سیستم `wait()` یا `waitpid()` به صورت صحیح تمام نکرده است.

iii. پس از اتمام عملیات والد، سیستم عامل پردازه zombie را از جدول پردازه‌ها حذف می‌کند و منابع آن را آزاد می‌کند.

۲. باتوجه به قطعه کدهای زیر برای هر مورد مشخص کنید که بین Orphan و Zambie چه پردازه ای ایجاد می‌شود و جواب خود را به طور کامل توضیح دهید.

Listing 6: Some Code

```
int(main)
{
        pid_t pid = fork;
        if(pid == 0)
        {
                printf("child process with pid %d ", getpid());
                exit(0);
        }
        else
        {
                printf("parent process with pid %d ", getpid());
                sleep(60);
        }

        return0;
}
```

پاسخ

۱. **Orphan:**

(آ) در این قطعه کد، پردازه فرزند `pid == 0` اجرا می‌شود و پردازه والد `pid != 0` منتظر می‌ماند. به عبارت دیگر، در صورتی که پردازه والد قبل از اتمام پردازه فرزند بسته شود، پردازه فرزند به عنوان یک پردازه orphan باقی می‌ماند.

(ب) در این حالت، پردازه فرزند پیام `child" process with pid "[pid]` را چاپ می‌کند و سپس با استفاده از `exit(0)` به پایان می‌رسد.

۲. **Zombie:**

(آ) در این قطعه کد، پردازه والد `pid != 0` در حالت انتظار `sleep(60)` قرار دارد و پردازه فرزند `pid == 0` اجرا می‌شود.

(ب) پس از اجرای پردازه فرزند، پیام `child" process with pid "[pid]` چاپ می‌شود.

(ج) والد در حال انتظار است و در این مدت، پردازه فرزند به پایان می‌رسد و به عنوان یک پردازه `zombie` باقی می‌ماند.

(د) پس از گذشت زمان `sleep(60)`، والد همچنان اجرا می‌شود و پردازه `zombie` نهایتاً توسط سیستم عامل از جدول پردازه‌ها حذف می‌شود.

Listing 7: Some Code

```
int(main)
{
        pid_t pid = fork;
        if(pid > 0)
                printf("parent process");
        else
        {
                sleep(60);
                printf("child process");
        }

        return0;
}
```

پاسخ

۱. **Orphan:**

(آ) در این قطعه کد، پردازه والد `pid > 0` پیام `parent process` را چاپ می‌کند و پردازه فرزند `pid == 0` به قسمت else وارد می‌شود. پس از ورود به قسمت `else`، پردازه فرزند از طریق `sleep(60)` به مدت 60 ثانیه منتظر می‌ماند و سپس پیام `child process` را چاپ می‌کند.

(ب) در این حالت، پردازه والد به عنوان یک پردازه زنده باقی می‌ماند و پردازه فرزند نیز در زمان اجرا زنده است. بنابراین، هیچ پردازه `orphan` ایجاد نمی‌شود.

۲. **Zombie:**

(آ) در این قطعه کد، پردازه والد `pid > 0` پیام `parent process` را چاپ می‌کند و پردازه فرزند `pid == 0` به قسمت `else` وارد می‌شود.

(ب) پس از ورود به قسمت `else`، پردازه فرزند با استفاده از `sleep(60)` به مدت 60 ثانیه منتظر می‌ماند و سپس پیام `child prcess` را چاپ می‌کند.

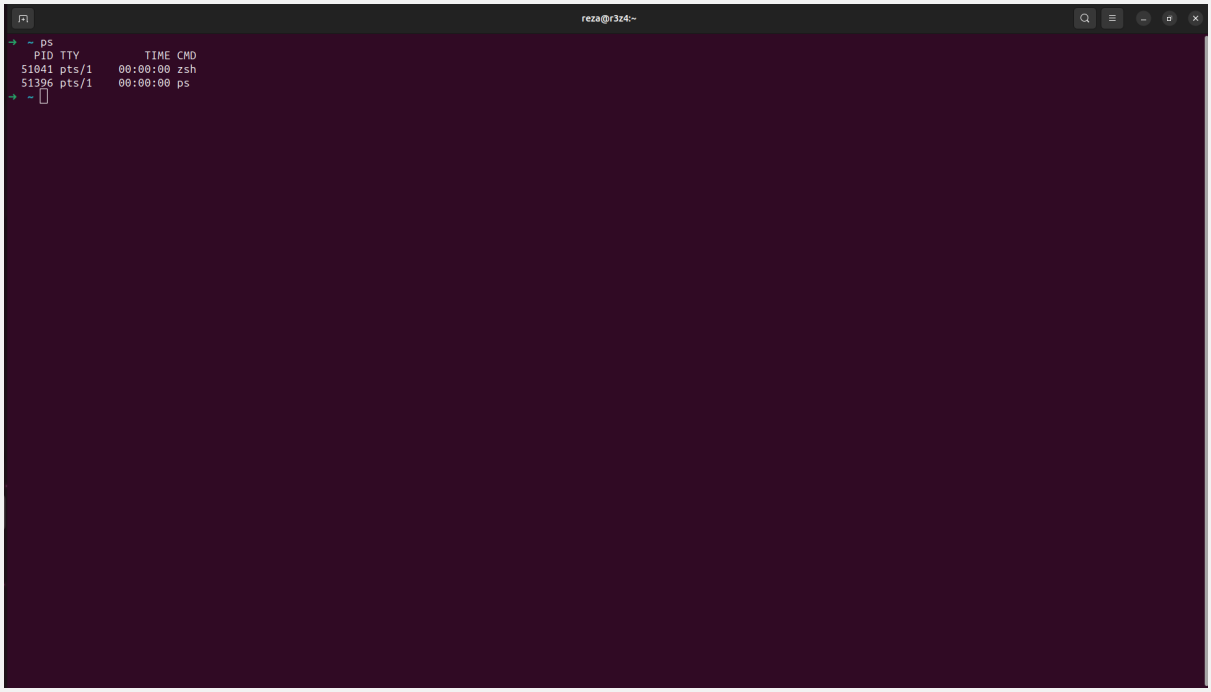
(ج) پس از اجرای پردازه فرزند، والد `pid > 0` همچنان در حال اجراست اما هیچ فعالیتی بر روی پردازه فرزند ندارد. اگر پردازه فرزند قبل از اتمام والد بسته شود، پردازه فرزند به عنوان یک پردازه `zombie` باقی می‌ماند.

(د) در این حالت، پردازه والد پس از گذشت زمان `sleep(60)` به پایان می‌رسد و پردازه `zombie` توسط سیستم عامل از جدول پردازه‌ها حذف می‌شود.

در این تمرین قصد داریم تا پردازنده‌های سیستم‌عامل لینوکس را مورد بررسی قرار دهیم

اولین دستوری که می‌خواهیم بررسی کنیم، دستور ps است.این دستور اطلاعات پردازه‌ها را نمایش می‌دهد.

خروجی دستور ps



برای درک وضعیت فعلی فرآیندهای درحال اجرای سیستم خود، دستور ps aux بیشترین مقدار اطلاعاتی که یک کاربر معمولا نیاز دارد را نمایش می‌دهد.

در خروجی این دستور وضعیت هر پردازه (Stat) و شماره هر پردازه (PID) و سایر اطلاعات موجود است.

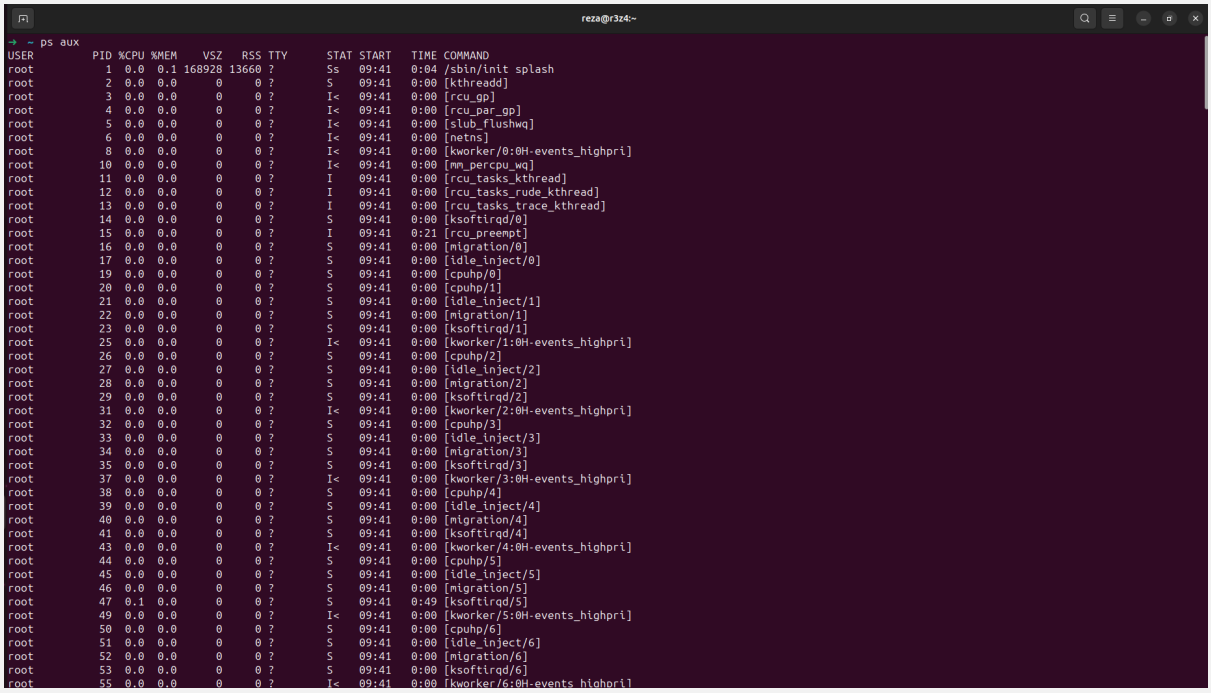
شماره پردازه‌ها در خروجی دستور ps از ۱ شروع شده و می‌دانیم که یک پردازه جدید در سیستم‌عامل با استفاده از دستورات fork و exec ساخته می‌شود. بنابراین هر پردازه ای یک پردازه والد دارد.

برای یافتن شماره پردازه والد، می‌توان از دستور ps -f [pid] استفاده کرد. این دستور PPID که شماره پردازه والد هست را نیز نمایش می‌دهد. از این دستور استفاده کنید و پردازه والد پردازه ۱ را پیدا کنید.

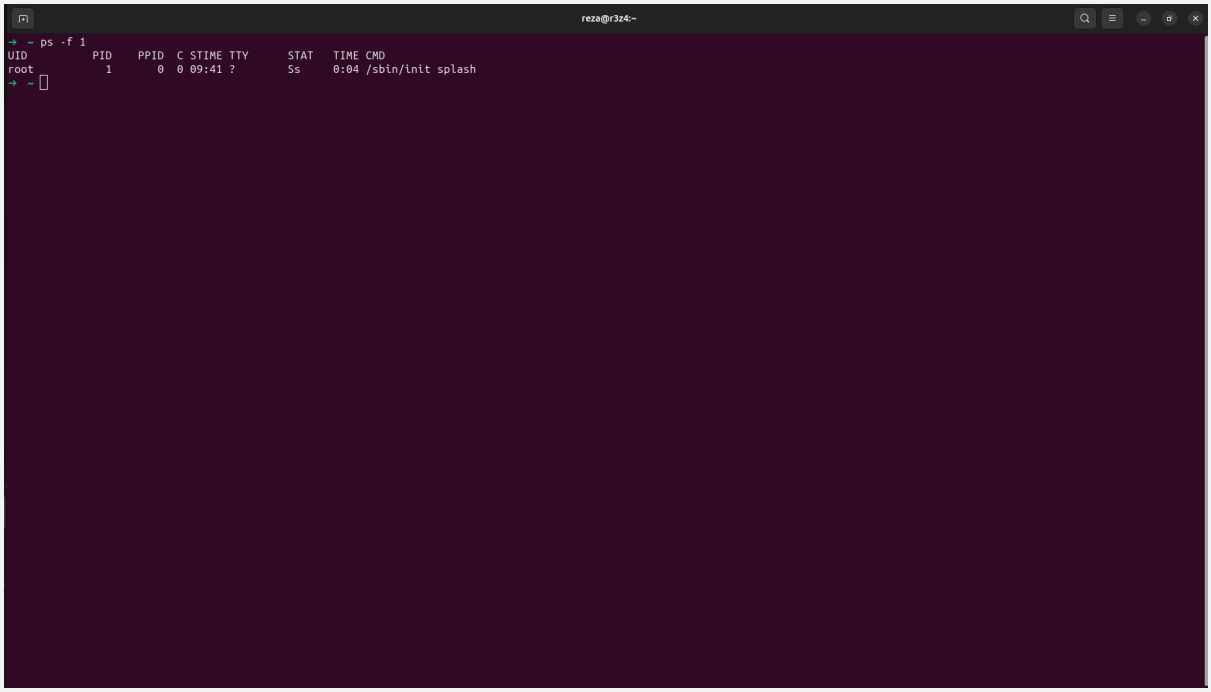
دستور pgrep -P [pid] همه فرزندان یک پردازه را نمایش می‌دهد. از این دستور استفاده کنید و همه پردازه‌هایی که پردازه پدر مشترکی با پردازه ۱ دارند را پیدا کنید.

همچنین با استفاده از دستور pstree می‌توان ارتباط بین پردازه‌ها را نمایش داد.

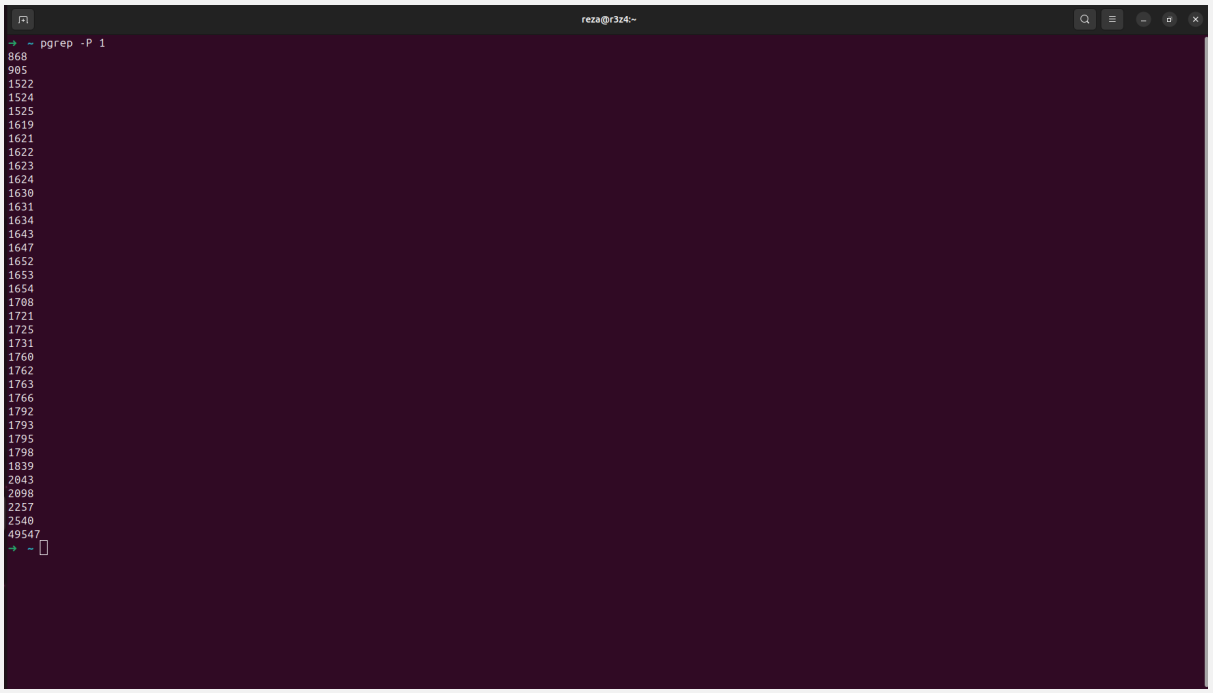
خروجی دستور ps aux



خروجی دستور ps -f [pid]



خروجی دستور pgrep -P [pid]



خروجی دستور pstree