آزمایشگاه سیستم عامل

دستورکار ۱۰: فرآیندها در لینوکس

فرآیند موجودیتی است که از آن برای نشان دادن یک برنامه در حال اجرا استفاده می شود. به فرآیندها برنامه‌های در حال اجرا نیز می‌گویند. به هر فرآیند در سیستم یک شناسه یکتا نسبت داده می‌شود که به آن شناسه فرآیند یا PID می‌گوییم. PID یک فرآیند موجود قابل تغییر دادن نیست. اما وقتی فرآیندی خاتمه یافت، سیستم عامل می تواند از شناسه‌ی آن برای مشخص کردن فرآیندی دیگر استفاده کند.

بخش اول: مدیریت فرآیندها

- **دستور ps**: پردازش‌های در حال اجرا را لیست می‌کند.

-f	اطلاعات بیشتری را نمایش می‌دهد.
-a	پردازش‌های تمام کاربران را نمایش می‌دهد.
-u	نام کاربران را نمایش می‌دهد.
-e	پردازش‌های سرویس‌های در حال اجرا را هم نمایش می‌دهد.
-ax	همه پردازش‌ها چه آن‌هایی که در محیط ترمینال اجرا شده یا نشده‌اند را نمایش می‌دهد.

انواع فرآیندهای لینوکس

وقتی شما برنامه‌ای را اجرا می‌نمایید، این برنامه توسط سیستم عامل به دو صورت، می تواند اجرا گردد: فرایندهای Foreground و فرآیندهای Background.

فرآیندهایForeground

وقتی که از یک ترمینال در لینوکس، فرمانی را اجرا می‌کنید، به طور پیش‌فرض این برنامه موقع اجرا در صورتی که نیاز به ورودی خاصی داشته باشد، منتظر ورود اطلاعات توسط کاربر شده و سپس نتیجه را در خروجی نمایش داده و سپس به حالت پرامپت یا علامت اعلان بازمی‌گردد. به این نوع فرآیندها، فرآیندهای Foreground می‌گویند.

نکته‌ی قابل ذکر این است که فرآیندهای Foreground تا موقعی که در حال اجرا می‌باشند، شما قادر به اجرای فرمان دیگری در آن ترمینال نخواهید بود. وقتی که مراحل فرآیند به اتمام رسید و اعلان سیستم به نمایش در آمد، شما می‌توانید فرمان‌های بعدی را اجرا نمایید. به عنوان مثال، با وارد نمودن دستور ls سیستم عامل این برنامه را در حافظه قرار داده و با اختصاص دادن یک شماره‌ی فرآیند اختصاصی به آن، فرایند شروع به کار نموده و نتیجه در خروجی نمایش داده می‌شود. پس از اتمام کار فرآیند، مجدد منابع اختصاصی فرآیند آزاد شده و حالت اعلان، نمایش داده می‌شود.

فرآیندهای Background

فرآیندهایی که به صورت Background اجرا می‌شوند، دیگر ترمینال را در انتظار اجرا قرار نمی‌دهند. شما می‌توانید با اجرای برنامه‌ای که در حالت Background اجرا می‌شود، فرمان‌های دیگر را نیز، اجرا نمایید. برای این‌که بتوانیم فرآیندی را در حالت Background اجرانماییم، باید برای اجرای دستور یا برنامه‌ی مربوطه یک & را با آخر فرمان اضافه نماییم.

دیمون‌ها(Daemons)

دیمون‌ها فرآیندهایی هستند که معمولا با سطح دسترسی کاربر root اجرا می‌گردند. دیمون‌ها معمولا سرویسی را به فرآیندهای دیگر ارائه می دهند. دیمون‌ها به صورت Background اجرا می‌شوند و معمولا منتظر اتفاق خاصی می‌مانند تا به آن واکنش نشان دهند. مثلا یک دیمون وب سرور منتظر درخواست ارتباط از طریق پروتکل http می‌ماند. به محض دریافت درخواست از طرف کلاینت سرویس لازم را به آن ارائه می‌دهد. دیمون‌ها معمولا با بوت شدن سیستم عامل، اجرا شده و تا آخر، باقی می‌مانند.

فرآیندهای والد و فرزند

در مدیریت فرآیندها در سیستم عامل لینوکس، هر فرآیند توسط فرآیند والد خود ایجاد شده است. هر فرآیند دارای دو شناسه‌ی فرآیند می‌باشد. اولین شناسه‌ی فرآیند یا PID مربوط به خود فرآیند است و دومین شناسه، مربوط به شناسه‌ی فرآیند والد، یا PPID می‌باشد. والد تمام فرآیندها، در بیشتر توزیع‌های لینوکس فرآیند systemd می‌باشد، که به‌جای فرآیند init استفاده می‌شود.

وضعیت‌های یک فرآیند

وضعیتRunning : فرآیند در حال اجرا، و یا آماده برای دریافت زمان از پردازنده برای اجرا

وضعیتWaiting : در این حالت فرآیند منتظر، یک اتفاق، مثلا فشرده شدن یک کلید است.

وضعیتStopped : در این وضعیت، روند اجرای فرآیند، متوقف شده است. مثلا برای حالت debugging.

در مثال زیر دستور sleep به مدت ۱۰۰ ثانیه، اجرا شده است. با فشردن کلید Ctrl_Z روند اجرای برنامه را متوقف می‌سازیم. با دستور jobs می‌توانیم، برنامه‌های در حال اجرا و متوقف شده را ببینیم:

```
sleep 100
Ctrl+Z
jobs
sleep 150 &
jobs
```

وضعیتZombie : در این وضعیت، اجرای فرآیند خاتمه یافته، اما هنوز فرآیند در جدول فرآیندها موجود است.

- **دستورpstree**: لیست پردازش‌های درحال اجرا را به صورت درختی نشان می‌دهد.

```
pstree
```

- **دستور top**: این دستور، گزارشی از وضعیت فرآیندها و همچنین، میزان اختصاص حافظه و پردازنده، ارائه می‌دهد. این دستور، ابزار مفیدی، برای مانیتورینگ وضعیت فرآیندها، در سرورهای شبکه می‌باشد. با کمک این ابزار، می‌توانیم، وضعیت کلی مصرف منابع سیستم، از قبیل حافظه و پردازنده را مشاهده نماییم.

-n	تعداد دفعات به‌روزرسانی را مشخص می کند.
-p	گزارشی از فرآیندی خاص را مشخص می‌کند
-u	گزارشی از فرآیندهای مربوط به کاربر خاص را مشخص می-کند.

مثال)

```
top –n 1
```

دستور top تنها یک‌بار اجرا می‌شود.

```
top –p 1
```

پردازش با شناسه ۱ را کنترل می‌کند.

```
top –u user1
```

پردازش‌های کاربر با نام user1 راکنترل می‌کند.

- **دستور kill**: برای ارسال سیگنال خاتمه به یک پردازش یا گروهی از پردازش‌ها به کار می‌رود.

```
kill {kill-signal} {PID}
kill {kill-signal ID {PID}
```

برای مشاهده لیست سیگنال kill ها را نشان می‌دهد.

```
kill -l
```

```
sheiki@sheikhi:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

kill-signal های مهم عبارتند از:

نام	شماره سیگنال	توضیحات
SIGHUP	۱	پردازش با همان PID قبلی راه‌اندازی مجدد می‌کند.
SIGINT	۲	یک سیگنال وقفه به پردازش مورد نظر ارسال می کند.(CTRL+C این سیگنال را به پردازش مورد نظر ارسال می‌کند.)
SIGTERM	۱۵	سیگنال حذف به برنامه مربوطه می فرستد سپس برنامه پردازش مربوطه را حذف می کند. (سیگنال پیش فرض است)
SIGKILL	۹	پردازش را مستقیماً توسط Kernel حذف می‌کند.

- **دستورnice** : در لینوکس، برنامه‌های زیادی، می‌توانند به صورت هم‌زمان، اجرا شوند. در صورتی که بخواهیم، برای بعضی از فرآیندها، اولویت بیشتری، در استفاده از منابع سیستم، قائل شویم، از دستور nice -n 'Nice Value' process nameبرای اجرای آن فرآیند، استفاده می‌کنیم. دستور nice برای شروع اجرای هر فرآیند اولویت ۱۹- تا ۲۰ را می‌تواند قرار دهد. بیشترین اولویت ۱۹- و کمترین اولویت ۲۰ می باشد. پیش‌فرض اولویت (priority)برای تمام فرایندها، ۰ می‌باشد. در صورتی که یک فرآیند در حال اجرا باشد، برای تغییر مقدار اولویت آن، از دستور renice 'Nice Value' -p PID استفاده می‌نماییم.
- **دستور uptime**: مشخص می‌کند که چه مدت است که سیستم در حال اجرا (up) است. زمان فعلی، میزان زمان فعال بودن سیستم پس از آخرین راه‌اندازی مجدد، تعداد کاربران متصل به سیستم و بارِکاری سیستم را ۱، ۵ و ۱۵دقیقه قبل را نمایش می‌دهد.
- **دستور du**: فضای مصرف شده توسط یک دایرکتوری یا فایل را نمایش می‌دهد.

du {file-name or directory-name}	
-h	اطلاعات را با ذکر واحد اندازه‌گیری و بطورhuman readable نمایش می‌دهد.

- **دستور df**: میزان فضای مصرفی و میزان فضای در دسترس و خالی فایل سیستم را گزارش می‌کند.

-h	اطلاعات را با ذکر واحد اندازه‌گیری و بطورhuman readable نمایش می‌دهد.

- **دستورfree**: با این دستور می توانیم، میزان مصرف و میزان حافظه‌ی قابل استفاده در حافظه‌ی اصلی(RAM)، را مشاهده نماییم.

بخش دوم: بررسی رفتار فرآیندها

همانطور که می‌دانید از طریق فراخوانی سیستمی برنامه‌ها با سیستم عامل ارتباط برقرار می‌کنند. رابط فراخوانی سیستمی شامل تعدادی از توابع می‌شود که سیستم عامل به برنامه ها ارایه می دهد تا روی آن عمل کنند. این توابع اجازه اعمالی مانند بازکردن فایل‌ها، ایجاد ارتباطات شبکه، خواندن و نوشتن فایل‌ها و غیره را می‌دهند.

برای بررسی این منظور چند برنامه به زبان برنامه نویسی c خواهیم نوشت.

برای کامپایل برنامه به زبان c از دستور cc استفاده کنید:

```
cc example.c
```

این دستور خروجی کامپایل را در فایلی قابل اجرا در دایرکتوری جاری به نام a.out ایجاد می کند.
این فایل را به صورت زیر اجرا کنید:

```
./a.out
```

در این مرحله از کامپایلر gcc نیز می‌توانید استفاده کنید. برای نصب آن می‌توانید از دستور زیر استفاده کنید:

```
sudo apt install build-essential
```

هر فرآیند با استفاده از فراخوانی سیستمی getpid می تواند به شناسه خود دسترسی پیدا کند. یک فرآیند به دلایل مختلفی می‌تواند ایجاد شود، مثلا وقتی یک کاربر از طریق یک ترمینال وارد سیستم می‌شود، یک فرآیند جدید برای او ایجاد می‌شود.

همچنین سیستم عامل برای ارائه یک سرویس ممکن است یک فرآیند ایجاد کند؛ در این حال ایجاد یک فرآیند جدید برای ارائه سرویس باعث می شود که کاربر دیگر برای دریافت سرویس نیاز به صبر کردن نداشته باشد (مثلا یک فرآیند برای کنترل عمل چاپ ایجاد می شود.). همچنین یک برنامه کاربر با ایجاد چندین فرآیند می‌تواند از پیمانه ای شدن و فواید موازی سازی استفاده کند.

در لینوکس هر فرآیند جدید به وسیله فرآیندی دیگری (که از قبل موجود می‌باشد) ایجاد می‌شود و همین سبب ایجاد یک رابطه والد-فرزند می‌شود. تنها استثنا در فرآیندی است که دارای شناسه ۰ است. این فرآیند هیچ والدی نداشته و به وسیله سیستم عامل در زمان bootسیستم ایجاد می شود. هر فرآیند می‌تواند PID والد خود را با استفاده از فراخوانی سیستمی getppid بیابد.

یک فرآیند نمی‌تواند والد خود را تغییر داده و یا رابطه والد-فرزندی را بشکند. با این حال تغییر والد یک فرزند ممکن است یک بار طی اجرای یک فرآیند اتفاق بیفتد و این زمانی است که فرآیند والد قبل از فرزندانش خاتمه بیابد. در این گونه مواقع سیستم عامل شناسه والد تمامی فرآیندهای فرزندی که والدشان خاتمه یافته است را به ۱ مقداردهی می‌کند (۱ شناسه فرآیند init است). نحو دو فراخوانی سیستمی که در بالا به آن اشاره شد به صورت زیر است:

```
getpid(); // process ID
getppid(); // parent ID
```

سیستم عامل از یک ساختمان داده‌ی داخلی به نام جدول فرآیند برای نگه‌داری trackهای فرآیندها استفاده می‌کند. این جدول دارای یک مدخل برای هر فرآیند در حال اجرا است.

برنامه‌ها و فرآیندها

یک برنامه شامل مجموعه ای از دستور العمل ها و داده ها است که به یک فرم مشخص سازماندهی شده است و در یک فایل اجرا شدنی بر روی دیسک نگه داری می شود. یک برنامه لینوکس از چندین قطعه تشکیل شده است. code segment شامل دستور العمل به فرمت باینری است.

data segment شامل داده‌های از پیش تعریف شده (به عنوان مثال ثوابت) و داده هایی که مقدار دهی اولیه شده‌اند، می‌باشد. این دو بخش به همراه stack segment که شامل داده‌هایی است که به طور پویا به هنگام اجرای فرآیند تخصیص داده می‌شوند قسمت‌های اساسی یک فرآیند لینوکس هستند.

برای اجرای یک برنامه یک فرآیند جدید ایجاد می شود. از برنامه برای مقدار دهی اولیه دو قسمت اول استفاده می شود و بعد از آن دیگر لینکی بین فرآیند و برنامه وجود نخواهد داشت. داده های سیستمی یک فرآیند شامل اطلاعاتی مانند دایرکتوری جاری، توصیف-گرهای فایل های باز، نوع ترمینال، میزان مدت استفاده از CPU و غیره است.

یک فرآیند نمی‌تواند به طور مستقیم به داده‌های سیستمی خود دسترسی پیدا کند یا آن ها را تغییر دهد، زیرا این داده‌ها خارج از فضای آدرس‌دهی فرآیند قرار گرفته‌اند. با این حال یکسری فراخوانی‌های سیستمی وجود دارد که می‌توان به طور غیر مستقیم به این اطلاعات دسترسی پیدا کرد یا آن‌ها را تغییر داد.

تمامی فرآیندهای سیستم اولاد مستقیم یا غیر مستقیم یک فرآیند خاص هستند که به هنگام شروع به کار سیستم به وسیله دستور init به وجود می آید. وقتی یک کاربر وارد سیستم می شود یک فرآیند به طور اتوماتیک ایجاد می شود. این فرآیند shell یا همان مفسر فرمان مربوط به session کاربر است.

وظیفه این فرآیند تفسیر و اجرای دستوراتی است که کاربر تایپ کرده است. برای ایجاد یک فرآیند جدید می توان از فراخوانی سیستمی fork استفاده کرد. هر زمان که این فراخوانی اجرا شود یک فرآیند جدید مستقل از فرآیندی که fork را فراخوانی کرده است ایجاد می شود و یک PID خودش را دارا خواهد بود.

این دو فرآیند (فرآیندی که فراخوانی سیستمی fork را صدا زده است و فرآیندی که جدیدا ایجاد شده است) همروند هستند یا به عبارتی دیگر از لحاظ اجرا شدن مستقل از همدیگر اجرا می‌شوند. این دو فرآیند وقتی صحبت از محتوا (کد، داده‌ها، پشته، فایل‌های باز و غیره) می‌شود، دو فرآیندی یکسان هستند.

به این طریق fork یک کپی از فرآیند اولیه ایجاد می کند، پس تصویر دو فرآیند در حافظه یکسان است. فرآیندی که دستور fork را فراخوانی کرده است به عنوان فرآیند والد و فرآیند تازه ایجاد شده به عنوان فرآیند فرزند در نظر گرفته می شود. با استفاده از فراخوانی سیستمیwait ، فرآیند والد اجرای خود را معلق می کند و منتظر خاتمه فرآیند فرزند می ماند.

اجرای یک برنامه در فرآیند فرزند، به وسیله فراخوانی سیستمی exit (که یا به طور صریح توسط فرآیند فرزند صدا زده می شود و یا به طور ضمنی توسط سیستم عامل در پایان تابع main فراخوانی می شود) خاتمه می پذیرد. تاثیر فراخوانی سیستمی exit به این صورت می باشد: فرآیند جاری را خاتمه می دهد، یک کد خاتمه را در مدخل جدول فرآیند مربوطه ذخیره می کند و در نهایت فرآیند والد را (که منتظر خاتمه فرآیند فرزند است) از خواب بیدار می کند.

فرآیند سیستمی fork

از فراخوانی سیستمی fork برای ایجاد یک فرآیند جدید استفاده می‌شود. در کرنل، fork عملا به وسیله‌ی یک فراخوانی سیستمی clone پیاده‌سازی می شود. این واسطه به طور موثر سطحی از انتزاع را در مورد اینکه کرنل لینوکس چگونه قادر به ایجاد فرآیندها است را به وجود می‌آورد. clone به شما این اجازه را می دهد که به طور صریح مشخص کنید که کدام قسمت‌ها از فرآیند درون فرآیند جدید کپی شده و چه قسمت‌هایی بین آن‌ها به اشتراک گذاشته شود. فرآیند فرزند که یک کپی از فرآیند والد می باشد، دارای کد مشابه با والد است و اجرای خود را به همان طریقی که فرآیند والد به اجرای خود ادامه می‌دهد شروع می کند. برای اینکه بین برگشت از تابع fork در فرآیند والد و برگشت از تابع fork در فرآیند فرزند تمایز قائل شویم، تابع PID فرآیند فرزند را در حالت اول (در فرآیند والد) و ۰ را در حالت دوم (در فرآیند فرزند) بر می گرداند. وقتی اجرای تابع fork موفقیت آمیز نباشد، ۱- برگردانده می‌شود.

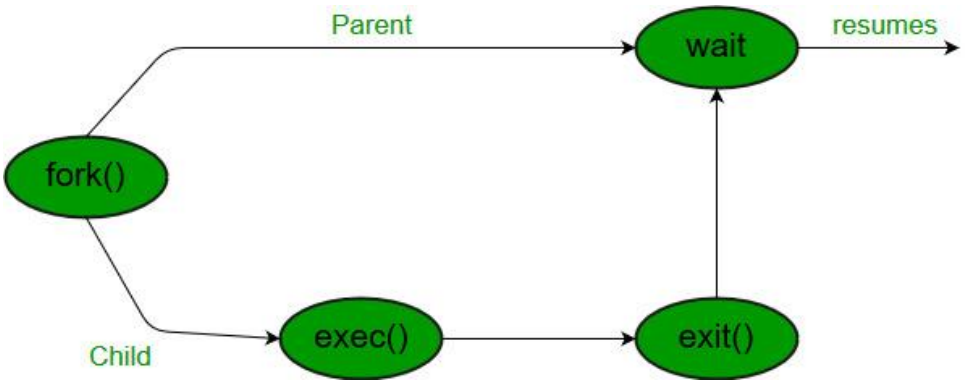
```
pid=fork();
/* source code executed by both processes */
switch (pid){
case -1:
/* Error! Unsuccesful fork! */
case 0 :
/* source code executed only by the child*/
break;
default:
/* source code executed only by the parent*/
}
/*source code executed by both processes*/
```

به این نکته توجه کنید که مثلا اگر تعداد فرآیند های ایجاد شده توسط کاربر و یا تعداد فرآیند هایی که در سیستم می توانند به طور همزمان اجرا شوند به حد مشخصی برسد، ممکن است خطا اتفاق بیافتد. تمامی متغیرهای فرآیند فرزند در ابتدا مقادیر خود را از فرآیند والد به ارث می برند. همچنین تمامی توصیفگرهای فایل مشابه فرآیند والد است. شایان ذکر است که سیگنال های معوق پاک شده و توسط فرزند به ارث برده نمی شوند. همچنین قفل های فایلی که فرآیند والد به دست آورده است به وسیله فرآیند فرزند به ارث برده نمی شود. حافظه فیزیکی که دو فرآیند در آن قرار گرفته اند و منابعی که توسط سیستم عامل به آن ها تخصیص داده می شود متفاوت می باشد، و این یعنی اینکه دو فرآیند مجزا هستند. در لحظه ای که دو فرآیند از فراخوانی fork برمی گردند، فرزند و والد به طور مستقل اجرا شده اند و حال برای به دست گرفتن پردازنده و دیگر منابع موجود با همدیگر به رقابت می پردازند. نمی توان مشخص کرد که کدام یک از این دو فرآیند پس از fork اول اجرا می شوند. تنها گزینه ممکن همان طور که در کد بالا نشان داده شد. این است که با تست کردن مقدار برگشتی توسط fork، اجرا را بین دو فرآیند (فرآیند والد و فرآیند فرزند) جدا کنیم. ایجاد یک فرآیند فرزند یکسان (از لحاظ محتوا) با والدش زمانی منطقی است که بتوانیم بخش کد و داده ی فرآیند ی تازه ایجاد شده را تغییر دهیم، درست مانند این که یک برنامه جدید را بار و اجرا کنیم.

فرآیند سیستمی wait

این فراخونی سیستمی برای همزمان کردن اجرای فرآیند های فرزند و والد استفاده می شود: فرآیند والد تا زمان خاتمه فرآیند فرزند صبر می کند. نحو استفاده از این دستور در زیر آمده است:

```
wait(int* pstatus);
```



در صورت موفقیت، این تابع PID فرزند خاتمه یافته و در صورت بروز خطا ۱- را برمی گرداند. آرگومان pstatus آدرس جایی است که کد خاتمه فرزند در آنجا کپی شده است) . فرزندی که PID آن برگردانده شده است (فرآیند ای که wait را فراخوانی می کند می تواند:

- بلوکه شود. اگر تمامی فرزندان در حال اجرا باشند، این اتفاق می افتد.
- حالت خاتمه فرزند را دریافت کند. اگر حداقل یکی از فرزندان قبل از فراخوانی wait به پایان رسیده باشند، این اتفاق می افتد.
- یک خطا دریافت کند. اگر هیچ فرآیند فرزندی موجود نباشد این اتفاق می افتد.

حالت خاتمه فرآیند فرزند به صورت اکتال کد شده و در آدرسی که به وسیله pstatus مشخص شده ذخیره می شود.

در سه حالت یک فرآیند خاتمه می یابد. ۱- وقتی فرآیند عمدا exit را فراخوانی کند ۲- بعد از دریافت یک سیگنال خاتمه دهنده و یا دریافت سیگنالی که فرآیند قادر به پردازش آن نیست. ۳ -در اثر خرابی سیستم. کد حالتی که به وسیله متغیر pstatus برگردانده می شود، مشخص می کند که کدام یک از دو حالت اول اتفاق افتاده است.

مثال)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
int i=10;
void main()
{
    int pid=fork();
   if(pid==0)
   {
     printf("initial value of i %d \n ",i);
     i+=10;
     printf("value of i %d \n ",i);
     printf("child terminated \n");
   }
  else
  {
    wait(0);
   printf("value of i in parent process %d",i);
  }
}
```

فرآیند سیستمی exit

این فراخوانی سیستمی باعث خاتمه فرآیند فراخوانی کننده می شود. نحو این دستور به صورت زیر است:

```
void exit(int* status);
```

پارامتر فرستاده شده به تابع exit به عنوان کد خاتمه تفسیر شده و فرآیند والد می تواند برای تعیین روش خاتمه یکی از فرزندانش، از آن استفاده کند. طبق قرار داد کد ۰ به معنی خاتمه نرمال و موفقیت آمیز فرآیند است در حالی که مقدار غیر صفر یک خطا را سیگنال می دهد.برای فرآیند هایی که عضوی از رابطه والد-فرزندی هستند، سه حالت مختلف که مرتبط با فراخوانی exit است، وجود دارد:

- وقتی والد قبل از فرزند خاتمه می یابد. در این حالت به همه فرزندان یک والد جدید نسبت داده می شود. این والد جدید فرآیند init با شناسه ۱ است. با این کار دیگر پروسه ی یتیمی در سیستم وجود نخواهد داشت.
- وقتی فرزند قبل از والد خود پایان می یابد. سیستم عامل بعضی از اطلاعات را راجع به فرآیند خاتمه یافته ذخیره می کند . PID)، دلیل خاتمه و غیره .(والد فرآیند خاتمه یافته به این اطلاعات با استفاده از فراخوانی سیستمی wait دسترسی دارد. حالت zombie به فرآیندی گفته می شود که خاتمه یافته است و والد آن نیز wait را فراخوانی نکرده است. به وسیله دستور ps می توان از فرآیند های zombie اطلاع پیدا کرد. در این موقع در ستون حالت ('S') حرف Z چاپ می شود.
- وقتی فرآیندی که از init ارث بری می کند، خاتمه بیابد. این فرآیند ها وارد حالت zombie نمی شوند به این خاطر که فرآیند init همیشه یکی از فراخوانی های wait یا waitpid را برای فرزندانش فراخوانی می کند. با این مکانیزم از سربار پیدا کردن سیستم با فرآیند های zombie پرهیز می شود.

مثال)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main()
{
        int id=fork();
        if(id==-1)
        {
                printf("cannot create the file");
                exit(1);
        }
        else
                if(id==0)
                {
                        sleep(2);
                        printf("child process");
                }
                else
                {
                        printf("parent process");
                        exit(1);
                }
}
```

تمرین:

۱- برنامه ای بنویسید که از سه دستور fork متوالی استفاده کند. با استفاده از دستورات مربوط به مشخصات فرآیند ها برای هر فرآیند، شماره ID و شماره ID ایجاد کننده آن را به دست آورید.

۲- شماره سه فرآیندی که بیشترین میزان مصرف CPU را داشته‌اند نمایش دهید.