Linux Professional Institute

LPIC-1

جزوه‌ی چهارم: مدیریت پراسس‌ها و آشنایی با سخت‌افزارها

By: The Albatross

thealbatross@yandex.com
https://github.com/TheAlbatrossCodes/Linux-In-Persian

فهرست مطالب

مقدمه

جلسه‌ی قبل به صورت خیلی ابتدایی با ابزار sed کار کردیم، کارت شبکه‌ی سیستم خود را تنظیم کردیم و همچنین یک سری فایل از طریق SFTP به سیستم خود منتقل کردیم. پس از آن به صورت مفصل در مورد روش‌های متفاوت نصب نرم‌افزار روی لینوکس صحبت کردیم و چگونگی بررسی یکپارچگی فایل‌های نصب شده را یاد گرفتیم. در این جلسه در مورد مدیریت پراسس‌ها در سیستم صحبت می‌کنیم و پس از آن، کمی در مورد سخت‌افزارهای سیستم و چگونگی مدیریت آنها در لینوکس صحبت می‌کنیم.

آشنایی با وبسایت DistroWatch

اگر به خاطر داشته باشید، گفتیم که برای لینوکس، توزیع‌های متفاوتی وجود دارد. اما از کجا می‌توانیم در مورد این توزیع‌های متفاوت اطلاعات به دست آوریم؟ از کجا می‌توانیم بفهمیم کدام توزیع معروف‌تر است، یا اصلا از کجا بدانیم که هر توزیع، بر پایه‌ی چه توزیع دیگر ایجاد شده است؟

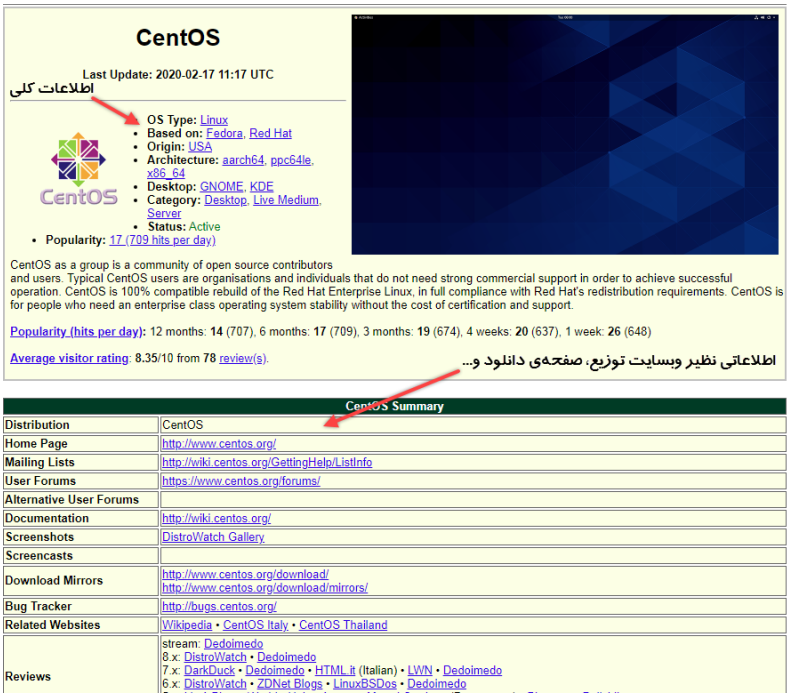
پاسخ همه‌ای این سوال‌ها در وبسایت DistroWatch می‌باشد. صفحه‌ی اصلی DistroWatch، نمایی شبیه تصویر ۱ دارد:

تصویر ۱ - صفحه‌ی اصلی وبسایت DistroWatch

همانطور که می‌بینید، سایت DistroWatch، اخبار مربوط به منتشر شدن یا آپدیت شدن یک توزیع، لیستی از معروف‌ترین توزیع‌ها و... را به ما نمایش می‌دهد. ما می‌توانیم نام توزیع مورد نظر خود را در این سایت جستجو کنیم و اطلاعات متفاوتی نظیر:

- نوع سیستم عامل (لینوکس، BSD و...)
- توزیع بر پایه‌ی چه توزیعی می‌باشد (Red Hat، Debian و...)
- کشور ایجاد کننده
- موارد استفاده (سرور، دسک‌تاپ و...)
- و...

نمونه‌ای از اطلاعاتی که DistroWatch در مورد یک توزیع به ما می‌دهد را در تصویر ۲ می‌بینید:



تصویر ۲- کسب اطلاعات در مورد یک توزیع

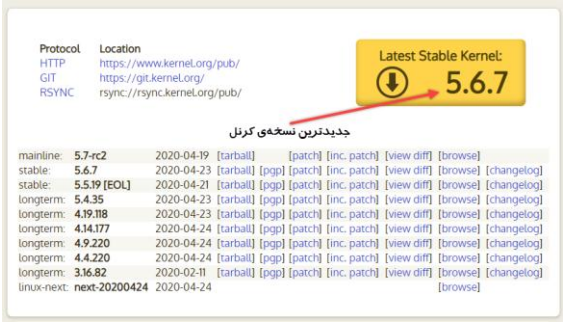
سایت کرنل لینوکس

همانطور که می‌دانید، لینوکس به تنهایی یک سیستم‌عامل نیست، بلکه یک کرنل است که در کنار سایر ابزارها، تبدیل به یک سیستم‌عامل می‌شود. اگر به خاطر داشته باشید، با دستور uname، می‌توانستیم اطلاعات متفاوتی در مورد سیستم خود به دست آوریم. یکی از این اطلاعات، نسخه‌ی کرنل لینوکس می‌باشد. برای مشاهده‌ی نسخه‌ی کرنل سیستم، از آپشن r - استفاده می‌کنیم:

```
[root@localhost ~]# uname -r
3.10.0-1062.el7.x86_64
```

همانطور که می‌بینید، نسخه‌ی کرنل نصب شده روی سیستم ما، ۳,۱۰ می‌باشد. شاید برایتان سوال باشد که از کجا می‌توانید اطلاعاتی در مورد نسخه‌های کرنل به دست آورید، یا حتی کرنل را برای خودتان دانلود کنید.

سایت kernel.org، اطلاعاتی در مورد آخرین نسخه‌های کرنل را در اختیار ما قرار می‌دهد و ما می‌توانیم از طریق این وبسایت، سورس کد کرنل را به صورت کامل دانلود کنیم. صفحه‌ای اصلی این وبسایت نمایی نظیر تصویر ۳ دارد:



تصویر ۳- صفحه‌ی اصلی سایت kernel.org

همانطور که می‌بینید، آخرین نسخه‌ی کرنل، نسخه‌ی ۵،۶،۷ می‌باشد. اما روی سیستم ما که CentOS 7 می‌باشد، نسخه‌ی ۳،۱۰ کرنل نصب شده است. این که چه نسخه‌ای از کرنل در یک توزیع استفاده شود، بستگی به تصمیم توسعه‌دهندگان آن دارد. مثلا نسخه‌ی کرنل CentOS 7، هیچ وقت از ۳ بالاتر نمی‌رود. ما با اجرای دستور `yum update` و آپدیت کردن کرنل، فقط minor version کرنل را آپدیت می‌کنیم، اما نمی‌توانیم major version کرنل را آپدیت کنیم (منظور از major version، اولین رقمی که در عدد نسخه‌ی کرنل می‌بینید می‌باشد). یکی از دلایل این امر، stability می‌باشد. همانطور که می‌دانید در سرورها، stability مهم‌ترین فاکتور می‌باشد و آپدیت کردن کرنل می‌تواند روی سرویس‌ها و نرم‌افزارهایی که روی سیستم اجرا می‌کنیم، تاثیر بگذارد. یعنی ممکن است یک سروری که به صورت کاملا صحیح کار می‌کند، پس از آپدیت کرنل به آخرین نسخه، کاملا از کار بیافتد و مشکلات compatibility و... به وجود آورد.

نکته: آپدیت کردن کرنل از طریق `yum` معمولا برای سیستم مشکلی ایجاد نمی‌کند، اما کامپایل کردن کرنل و آپدیت آن به صورت دستی، نه تنها کار دشواری می‌باشد، بلکه به احتمال زیاد سرور شما را از کار خواهد انداخت.

کرنل لینوکس طبیعتی ماژولار دارد و ما می‌توانیم یک سری از ماژول‌ها را به آن اضافه کنیم یا یک سری ماژول را از آن حذف کنیم. مثلا اگر به یک فایل‌سیستم خاص احتیاج نداشته باشیم، می‌توانیم ماژول آن را از روی کرنل حذف کنیم، یا مثلا اگر بخواهیم از طریق لینوکس به سیستم‌های دیگر تونل GRE بزنیم، باید ماژول آن را به کرنل اضافه کنیم. بعدا با این مسئله بیشتر آشنا می‌شویم.

نوع ترمینال‌های لینوکس

هر وقت یک سیستم لینوکسی را روشن می‌کنیم، برای استفاده از Command Line آن، باید به یک ترمینال متصل شویم. در واقع دلیلی که ما می‌توانیم در لینوکس دستورهای متفاوت را تایپ کنیم این است که هر وقت سیستم خود را روشن می‌کنیم، یا حتی به سیستم خود SSH می‌زنیم، به یک ترمینال متصل می‌شویم. اما چه تفاوتی بین استفاده‌ی از ترمینال در صفحه‌ی ماشین مجازی یا استفاده از ترمینال با اتصال SSH می‌باشد؟
اگر مستقیما از طریق ماشین مجازی به ترمینال متصل شویم، از طریق TTY به ترمینال متصل شده‌ایم. اگر از طریق SSH، xterm یا هر برنامه‌ی دیگری به ترمینال سیستم متصل شویم، از طریق pty به سیستم متصل شده‌ایم.
با استفاده از دستور who، می‌توانیم طریقه‌ی اتصال به ترمینال را بفهمیم. مثلا اگر مستقیما در ماشین مجازی این دستور را تایپ کنیم، با خروجی زیر مواجه می‌شویم:

```
[root@localhost ~]# who
root     tty1         2020-04-27 23:37
```

همانطور که می‌بینید، سیستم به ما می‌گوید که ما از طریق tty1 به ترمینال متصل شده‌ایم. این امر به ما نشان می‌دهد که به صورت مستقیم در حال تایپ در ترمینال هستیم یا به عبارت دیگر، انگار لینوکس را روی یک سیستم فیزیکی نصب کرده‌ایم و با کیبورد و مانیتور در حال تعامل با آن هستیم. اما منظور از عدد 1 پس از tty چیست؟

ما می‌توانیم در لینوکس به صورت همزمان، از ۶ عدد ترمینال tty متفاوت استفاده کنیم. برای رفتن به یک tty دیگر، مثل tty2، کافی است داخل ماشین مجازی خود، دکمه‌ی Ctrl+Shift+F2 را بزنید. در این حالت، شما وارد tty2 می‌شوید و می‌توانید در این ترمینال بار دیگر login کرده و از سیستم استفاده کنید.

بیایید بار دیگر دستور who را اجرا کنیم:

```
[root@localhost ~]# who
root     tty1         2020-04-27 23:37
root     tty2         2020-04-27 23:41
```

همانطور که می‌بینید، اکنون دو ترمینال متفاوت در حال استفاده از سیستم می‌باشند. در CentOS، ما می‌توانیم ۶ ترمینال tty به صورت همزمان داشته باشیم.

نکته: برای رفتن به سایر ترمینال‌های tty یا جابه‌جایی بین آنها، کافی است دکمه‌ی Ctrl + Shift و دکمه‌های F1 تا F6 را به صورت همزمان بزنید. دکمه‌های F1 تا F6، به ترتیب نشان دهنده‌ی tty1 تا tty6 می‌باشند.

اما tty تنها روش اتصال به ترمینال لینوکس نیست. ما می‌توانیم به با استفاده از یک برنامه‌ی جانبی، ترمینال را emulate کنیم. برنامه‌هایی نظیر ssh، xterm (در سیستم‌های لینوکسی که GUI دارند) و... یک ترمینال را روی سیستم emulate می‌کنند. اگر به سیستم SSH بزنیم و سپس دستور who را اجرا کنیم، با چنین خروجی مواجه می‌شویم:

```
[root@localhost ~]# who
root     tty1         2020-04-27 23:37
root     pts/0        2020-04-27 23:44 (192.168.1.108)
root     tty2         2020-04-27 23:41
```

همانطور که می‌بینید، یک ترمینال جدید با نام pts/0 به این لیست اضافه شده است. هر وقت از طریق SSH، telnet یا مثلا رابط گرافیکی به سیستم متصل شویم، ارتباط ما با ترمینال از طریق pts خواهد بود. برای این که بدانیم چه تعداد ترمینال pts می‌توانیم داشته باشیم، می‌توانیم به فایل زیر نگاه کنیم:

```
[root@localhost ~]# cat /proc/sys/kernel/pty/max
4096
```

همانطور که می‌بینید، ما می‌توانیم ۴۰۹۶ ترمینال pts به صورت همزمان داشته باشیم. البته این مقدار را می‌توان تغییر داد ولی ما فعلا به آن نمی‌پردازیم.

مشاهده‌ی پراسس‌های سیستم با استفاده از ps

در هر لحظه، ده‌ها برنامه روی لینوکس در حال اجرا می‌باشند. در لینوکس، ما به هر برنامه‌ی در حال اجرا، پراسس (Process) می‌گوییم. لینوکس به هر پراسس یک شماره به نام PID (Process ID) اختصاص می‌دهد و هر پراسس را با توجه به PID آن، مدیریت می‌کند.

ما می‌توانیم با استفاده از دستور ps، پراسس‌هایی که در حال حاضر روی سیستم اجرا هستند را ببینیم:

```
[root@localhost ~]# ps
  PID TTY          TIME CMD
 1271 pts/0    00:00:00 bash
 1286 pts/0    00:00:00 ps
```

دستور ps به صورت پیش‌فرض، فقط پراسس‌هایی که توسط شِل کنونی اجرا شده‌اند را به ما نشان می‌دهد. همانطور که می‌بینید ما روی شل کنونی، فقط برنامه یا پراسس bash و ps را اجرا کرده‌ایم.

اگر به خروجی دقت کنید، می‌بینید که خروجی به ٤ ستون تقیسم شده است:

- **PID**: نشان دهنده‌ی شماره‌ی اختصاص داده شده به پراسس می‌باشد.
- **TTY**: نشان دهنده‌ی نوع ترمینالی (مثلا tty، pts و...) است که آن پراسس را استارت زده است.

- **TIME**: نشان دهنده‌ی CPU Time مصرف شده توسط آن پراسس می‌باشد.
- **CMD**: نشان دهنده‌ی نام آن دستور یا پراسس می‌باشد.

آپشن‌های ps

دستور ps می‌تواند سه نوع آپشن از ما دریافت کند:

- آپشن‌های یونیکسی که با - شروع می‌شوند.
- آپشن‌های BSD که نیازی به - ندارند.
- آپشن‌های GNU که با -- شروع می‌شوند.

این بدین معناست که ما می‌توانیم آپشن‌های خیلی زیادی به دستور ps بدهیم. بسیاری از ادمین‌های لینوکس آپشن‌های مورد علاقه‌ی خود را گلچین می‌کنند و همیشه از آن استفاده می‌کنند. پیشنهاد می‌شود برای آشنایی با آپشن‌های ps، صفحه‌ی manpage آن را مطالعه کنید.

برای مشاهده‌ی همه‌ی پراسس‌هایی که روی سیستم در حال اجرا هستند، می‌توانیم از آپشن یونیکسی -ef استفاده کنیم:

```
[root@localhost ~]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
...
root           2       0  0 23:50 ?        00:00:00 [kthreadd]
root           3       2  0 23:50 ?        00:00:00 [kworker/0:0]
root           4       2  0 23:50 ?        00:00:00 [kworker/0:0H]
root           5       2  0 23:50 ?        00:00:00 [kworker/u256:0]
root           6       2  0 23:50 ?        00:00:00 [ksoftirqd/0]
...
postfix     1180    1162  0 23:51 ?        00:00:00 qmgr -l -t unix -u
root        1272    1010  0 23:52 ?        00:00:00 sshd: root@pts/0
root        1276    1272  0 23:52 pts/0    00:00:00 -bash
root        1300    1276  0 23:56 pts/0    00:00:00 ps -ef
```

این آپشن اطلاعات مناسبی را در مورد پراسس‌های سیستم به ما می‌دهد:

- **UID**: کاربری که این پراسس را اجرا کرده است.
- **PID**: شماره پراسس.
- **PPID**: شماره‌ی پراسسی که این پراسس را استارت زده است. (این عدد، در صورتی که پراسس توسط یک پراسس دیگر ایجاد شده باشد وجود خواهد داشت.)
- **C**: میزان استفاده از CPU در طول عمر پراسس
- **STIME**: زمان شروع پراسس
- **TTY**: ترمینالی که این پراسس را استارت زده است.

> اگر دقت کنید می‌بینید که مقدار tty برخی از پراسس‌ها، ? می‌باشد. این بدین معنی است که این پراسس، وابسته به هیچ ترمینالی نیست. اهمیت این امر زمانی مشخص می‌شود که بدانید با بسته شدن یک ترمینال، کلیه‌ی پراسس‌هایی که درون آن ترمینال استارت شده باشند، از بین خواهند رفت. حال فرض کنید شما با SSH به سیستم متصل شده‌اید، یک وب سرور را کانفیگ کرده‌اید و سپس آن را در محیط کاری اعمال کرده‌اید. اگر پراسس مربوط به وب سرور به محض قطع کردن ارتباط SSH از بین برود، شما به مشکل بر میخورید. به همین

دلیل، برخی از پراسس‌های سیستمی، مقدار TTY برابر با علامت ? خواهند داشت تا از وقوع همچین مشکلی جلوگیری کنند.

- **CMD**: نام دستور یا برنامه‌ای که این پراسس را استارت زده است. به عبارت دیگر، نام پراسس می‌باشد.

یکی دیگر از آپشن‌های معروف ps، آپشن aux می‌باشد. این این آپشن بسیار شبیه به آپشن ef- می‌باشد، با این تفاوت که اطلاعات بیشتری نظیر میزان استفاده از RAM و... را نیز به ما نشان می‌دهد:

```
[root@localhost ~]# ps aux
 USER  PID  %CPU  %MEM  VSZ  RSS  TTY  STAT  START  TIME  COMMAND
 ...
 root   2   0.0   0.0   0    0    ?    S     23:12  0:00  [kthreadd]
 root   3   0.0   0.0   0    0    ?    S<    23:12  0:00  [kworker/0:0H]
 root   4   0.0   0.0   0    0    ?    S     23:12  0:00  [kworker/u256:0]
 root   5   0.0   0.0   0    0    ?    S     23:12  0:00  [ksoftirqd/0]
 ...
```

همانطور که می‌بینید، خروجی این دستور بسیار شبیه به خروجی ps -ef می‌باشد، البته چند ستون جدید دارد که به شرح زیر می‌باشند:

- **%CPU**: مقدار CPU مصرفی توسط پراسس.
- **%MEM**: مقدار RAM مصرفی توسط پراسس.
- **VSZ**: مقدار virtual memory مصرفی توسط پراسس (در واحد بایت).
- **RSS**: مقدار RAM واقعی مصرفی (non-swappable) توسط این پراسس.
- **STAT**: وضعیت یا State پراسس.

در نمونه‌ی بالا، مقدار S و S< را در این ستون مشاهده می‌کنید. S یعنی این پراسس در حالت sleep mode قرار گرفته است (یا به شکل صحیح‌تر، پراسس در حالت interruptible sleep می‌باشد. جلوتر در مورد این مفهوم صحبت می‌کنیم.) و منتظر یک اتفاق برای فعال شدن می‌باشد. علامت < جلوی وضعیت، نشان‌دهنده‌ی داشتن اولویت بالا می‌باشد. برای پیدا کردن مفهوم سایر STATها، به manpage دستور ps مراجعه کنید.

با اضافه کردن آپشن --forest-، می‌توانیم پراسس‌ها را به صورت سلسله‌مراتبی ببینیم این امر به ما کمک می‌کند تا بفهمیم چه پراسسی، فرزند کدام پراسس می‌باشد، یا به عبارت دیگر، هر پراسس توسط چه پراسس دیگری اجرا شده است. برای مثال:

```
[root@localhost ~]# ps aux
USER   PID  %CPU  %MEM    VSZ    RSS    TTY    STAT  START  TIME  COMMAND
...
root   1287  0.0   0.5   158932  5608     ?      Ss    23:35  0:00  \_ sshd: root@pts/0
root   1291  0.0   0.2   115584  2132   pts/0    Ss    23:35  0:00    \_ -bash
root   1336  0.0   0.1   155508  1936   pts/0    R+    23:35  0:00      \_ ps aux --forest
...
```

اگر به ستون COMMAND نگاه کنید، می‌بینید که در یک ساختار سلسله‌مراتبی، به ما گفته شده که پراسس sshd، پراسس bash را استارت زده و پراسس bash هم پراسس ps aux --forest را اجرا کرده است. دلیل این است که ما از طریق SSH به سیستم متصل شده‌ایم، پس به همین دلیل پراسس sshd فعال شده است، سپس از آنجایی که هنگام اتصال SSH نیاز به یک شِل داریم، پراسس sshd، bash را استارت زده و ما هم

درون bash، دستور `ps aux --forest` را وارد کردیم، پس bash، پراسس `ps aux --forest` را استارت زده است.

جستجو میان پراسس‌ها

برای جستجو میان پراسس‌ها هنگام استفاده از ps، می‌توانیم از دوست قدیمی خود، grep، استفاده کنیم. مثلا فرض کنید ما دنبال پراسس‌های مربوط به postfix هستیم. در چنین حالتی، دستور ps را به صورت زیر اجرا می‌کنیم:

```
[root@localhost ~]# ps -ef | grep postfix
root      1149     1    0  00:18     ?     00:00:00  /usr/libexec/postfix/master -w
postfix   1166  1149    0  00:18     ?     00:00:00  pickup -l -t unix -u
postfix   1167  1149    0  00:18     ?     00:00:00  qmgr -l -t unix -u
root      1303  1274    0  00:34  pts/0    00:00:00  grep --color=auto postfix
```

همانطور که می‌بینید، با استفاده از دستور ps و grep، هر پراسسی که مربوط به postfix بود را بیرون کشیدیم. توجه کنید که هنگام استفاده از grep برای جستجو در خروجی ps، پراسس برنامه‌ی grep نیز به شما بازگردانده می‌شود (این نکته را در خط آخر خروجی می‌بینید)، پس وجود برنامه‌ی grep در خروجی را با اجرا بودن پراسس مورد جستجو اشتباه نگیرید.

مفهوم وضعیت پراسس‌ها

پراسس‌هایی که درون virtual memory قرار گرفته باشند، sleeping processes نامیده می‌شوند. کرنل پراسسی که منتظر یک رویداد باشد را در حالت sleep mode قرار می‌دهد.

زمانی که آن رویداد اتفاق بیافتد، کرنل یک سیگنال به پراسس می‌فرستد. اگر پراسس در حالت interruptible sleep باشد، سیگنال را دریافت کرده و اصطلاحا بیدار می‌شود. اگر پراسس در حالت uninterruptible sleep قرار داشته باشد، فقط در اثر یک رویداد خارجی، مثل اتصال یک سخت‌افزار جدید به سیستم، بیدار می‌شود. در این حالت هر سیگنال دیگری که به پراسس ارسال شده باشد ذخیره شده و زمانی که پراسس بیدار شود، به آن سیگنال‌ها پاسخ می‌دهد.

اگر یک پراسس فرزند، پایان یافته باشد اما پراسس والد آن به دلیل رفتن در حالت sleep mode، پایان یافتن آن پراسس را تصدیق نکرده باشد، پراسس فرزند در وضعیت zombie قرار می‌گیرد؛ یعنی که این پراسس در یک برزخ بین اجرا بودن و پایان یافتن قرار می‌گیرد و تا زمانی که پراسس والد پایان یافتن آن را تصدیق نکند، در همین وضعیت باقی می‌ماند.

اگر به خروجی `ps aux` یا `ps -ef` نگاه کنید، می‌بینید که در ستون CMD، برخی از دستورها درون براکت [] قرار دارند. این دستورها، دستورهایی هستند که به دلیل inactive بودن، در حالت sleep mode قرار دارند و لینوکس آنها را از روی RAM، به بخش virtual memory موجود در هارد دیسک منتقل کرده است.

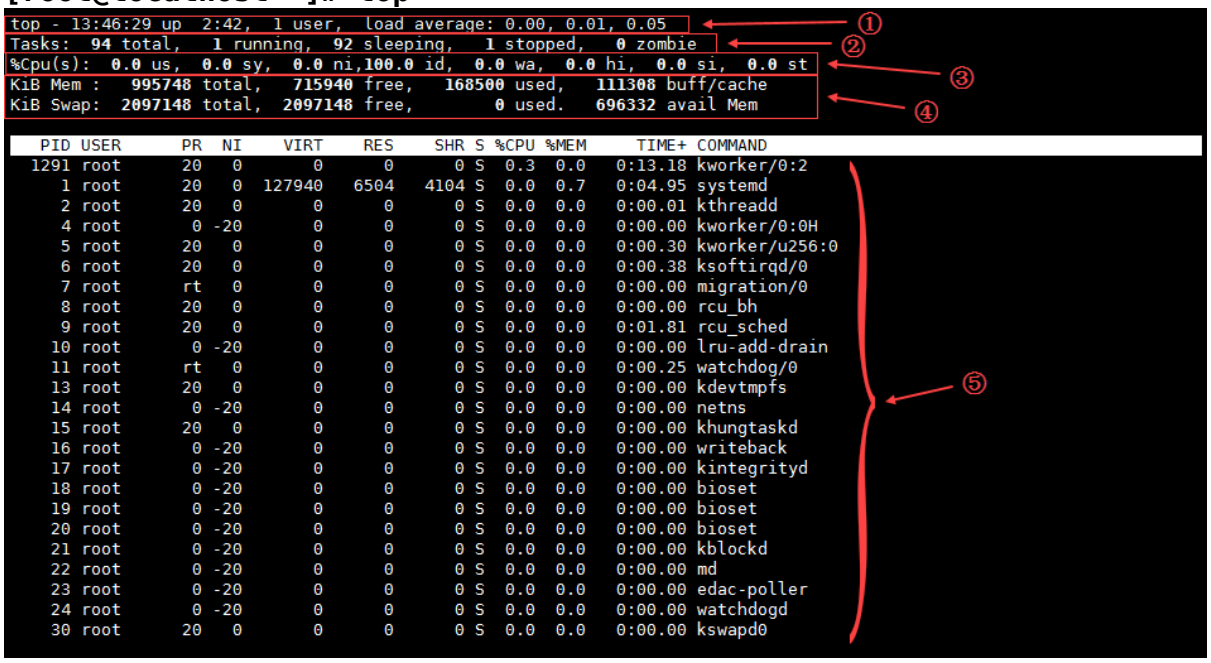
نکته: دستور ps، فقط یک snapshot از پراسس‌های در حال اجرا به ما نشان می‌دهد. به عبارت دیگر، پراسس‌ها را به صورت زنده به ما نشان نمی‌دهد (بر خلاف Task Manager ویندوز). برای مشاهده‌ی پراسس‌ها به صورت زنده، باید به سراغ دستورهای دیگر برویم.

مشاهده‌ی پراسس‌های سیستم با استفاده از top

برای مشاهده و مدیریت کلیه‌ی پراسس‌های در حال اجرا روی سیستم به صورت زنده، از دستور top استفاده می‌کنیم. مشاهده‌ی وضعیت پراسس‌ها به صورت زنده، به ما کمک می‌کند که پراسس‌ها را بهتر مدیریت کنیم. مثلا اگر سیستم کند شده باشد و بخواهیم ببینیم کدام پراسس رفتاری غیر معمول از خود نشان می‌دهد، باید از دستور top استفاده کنیم، چرا که مشاهده‌ی رفتار واقعی یک پراسس با استفاده از دستور ps، کار دشواری می‌باشد.

به محض وارد کردن این دستور، با نمایی نظیر تصویر ٤ مواجه می‌شویم:

```
[root@localhost ~]# top
```



تصویر ٤- مشاهده‌ی پراسس‌ها با استفاده از دستور top

top اطلاعات زیادی در مورد پراسس‌های سیستم به ما می‌دهد، پس بیایید مفهوم تک‌تک قسمت‌هایی که در تصویر ٤ مشخص کرده‌ایم را بررسی کنیم:

- ①: این بخش ساعت کنونی سیستم، مدت زمان روشن بودن سیستم، تعداد کاربرانی که در سیستم login کرده‌اند و متوسط بار (load average) روی سیستم را نشان می‌دهد.

  اگر دقت کنید، جلوی load average سه عدد می‌بینید. این سه عدد، به ترتیب نشان‌دهنده‌ی متوسط بار روی سیستم در ۱ دقیقه، ۵ دقیقه و ۱۵ دقیقه‌ی اخیر می‌باشند. مقدار متوسط بار در ۱ دقیقه‌ی اخیر ممکن است گاهی از اوقات زیاد باشد، که این امر طبیعی است؛ اما اگر مقدار متوسط بار در ۱۵ دقیقه‌ی اخیر زیاد باشد، به احتمال زیاد سیستم مشکلی دارد.

- ②: این بخش نشان دهنده‌ی اطلاعات کلی در مورد پراسس‌ها می‌باشد. در این بخش، به ترتیب تعداد پراسس‌ها، تعداد پراسس‌های در حال اجرا، تعداد پراسس‌های در وضعیت sleeping و تعداد پراسس‌های zombie به ما نشان داده می‌شود.

  منظور از task، همان پراسس‌ها می‌باشد.

- ③: این بخش نشان دهنده‌ی اطلاعات کلی در مورد CPU در یک مدت زمان خاص می‌باشد. top، اطلاعاتی که در مورد CPU می‌دهد را به چند بخش تقسیم می‌کند:
  - us: میزان CPU مصرفی برای اجرای فرمان‌های کاربر (یعنی هر پراسسی که متعلق به کرنل نباشد، مثلا وب‌سرورها، دیتابیس‌ها و...)
  - sy: میزان CPU مصرفی برای اجرای کرنل و پراسس‌های مربوط به آن.
  - ni: میزان CPU مصرفی برای اجرای پراسس‌هایی که اولویتشان توسط کاربر دستکاری شده است.
  - id: میزان idle بودن CPU را به ما نشان می‌دهد. در تصویر ٤، عدد ۱۰۰ را در مقابل این شاخص می‌بینید. این یعنی CPU در چند لحظه‌ی گذشته، در ۱۰۰٪ اوقات در حالت idle به سر می‌برده است.
  - wa: میزان زمانی که CPU در انتظار تمام شدن یک عملیات توسط یک دستگاه I/O (مثل هارد دیسک) بوده است را نشان می‌دهد.

    > سرعت انجام عملیات دستگاه‌های I/O سیستم، مثل هارد دیسک، در مقایسه با سرعت CPU بسیار آهسته می‌باشد و بعضا وقتی CPU دستور انجام یک عمل را به یکی از این دستگاه‌ها می‌دهد، مجبور است منتظر اتمام آن عمل توسط آن دستگاه بماند. در واقع wa، مدت زمانی که CPU در این حالت به سر می‌برد را نشان می‌دهد. اگر جلوی wa به مدت طولانی عدد ۱۰۰ را ببینید، به احتمال زیاد هارد دیسک شما یک مشکلی دارد یا از نظر سرعت، مناسب کار شما نیست.
  - hi: میزان زمانی که CPU برای handle کردن interruptهای سخت‌افزاری صرف کرده است را نشان می‌دهد.

    > هر وقت یک سخت‌افزار به CPU نیاز داشته باشد، با ارسال یک اینتراپت به CPU، از CPU می‌خواهد که عملیات کنونی را رها کرده و به اینتراپت اعمال شده پاسخ دهد.
  - si: میزان زمانی که CPU برای پاسخ به اینتراپت‌های نرم‌افزاری صرف کرده است را نشان می‌دهد.
  - st: این شاخص در محیط‌های مجازی‌سازی شده معنی دارد. این شاخص، نشان دهنده‌ی میزان زمانی است که CPU واقعی در اختیار ماشین مجازی کنونی نبوده است.
- ④: در این بخش اطلاعات مربوط به حافظه‌ی RAM سیستم به ما داده می‌شود. در خط اول، به ترتیب اطلاعاتی در مورد حجم کل RAM، میزان حجم خالی در RAM، میزان حجم استفاده شده از RAM و در نهایت، میزان RAM مصرفی توسط buffer و cache به ما نشان داده می‌شود. در خط دوم، اطلاعاتی در مورد Swap Memory سیستم به ما داده می‌شود.

اگر بخواهیم خیلی ساده در مورد آن صحبت کنیم، می‌توانیم بگوییم که swap بخشی از هارددیسک است که سیستم از آن به عنوان RAM استفاده می‌کند. البته سرعت حافظه‌ی swap بسیار کمتر از RAM می‌باشد.

- ⑤: در این بخش اطلاعات جزئی در مورد همه‌ی پراسس‌ها به ما نشان داده می‌شود. معنی هر کدام از ستون‌ها به صورت زیر می‌باشد:
  - **PID**: شماره‌ی پراسس.
  - **USER**: کاربری که پراسس را استارت زده است.
  - **PR**: اولویت پراسس.
  - **NI**: مقدار nice پراسس.

nice در سیستم‌های لینوکسی برنامه‌ای است که می‌تواند اولویت یک پراسس را تغییر دهد. تفاوت بین PR و NI در این است که PR اولویت حقیقی پراسس را نشان می‌دهد و NI چیزی است که به کرنل می‌گوید این پراسس، باید چه اولویتی داشته باشد. مقدار nice، مقدار PR را تغییر می‌دهد. بعدا با nice و مفهوم اولویت بیشتر آشنا می‌شویم.

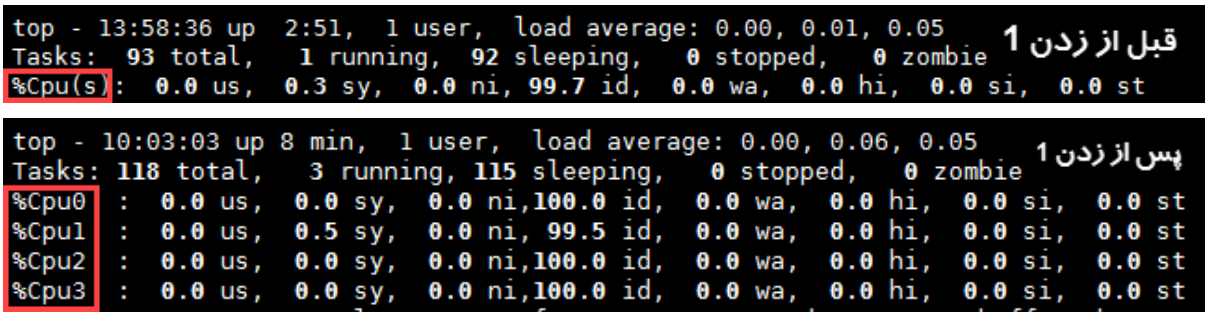
  - **VIRT**: میزان حافظه‌ی مجازی استفاده شده توسط این پراسس.
  - **RES**: میزان حافظه‌ی RAM در حال استفاده توسط پراسس.
  - **SHR**: میزان حافظه‌ای که پراسس با سایر پراسس‌ها به اشتراک گذاشته است.
  - **S**: وضعیت (status) پراسس
    - D: پراسس در حالت interruptible sleep قرار دارد.
    - I: پراسس در حالت idle قرار دارد.
    - R: پراسس در حال اجرا (running) است.
    - S: پراسس در حالت sleep قرار دارد.
    - T: پراسس stop یا trace شده است.
    - Z: پراسس در حالت زامبی قرار دارد 😨.
  - **%CPU**: میزان CPU مصرفی توسط پراسس.
  - **%MEM**: میزان حافظه‌ی RAM مصرفی توسط پراسس.
  - **TIME+**: کل میزان CPU Time مصرفی توسط پراسس از زمان استارت شدن.
  - COMMAND: دستوری که این پراسس را استارت زده یا همان نام پراسس.

نکته: top به صورت پیش‌فرض، پراسس‌ها را بر حسب میزان CPU% در جدول خود مرتب می‌کند.

دستورهای Interactive برنامه‌ی *top*

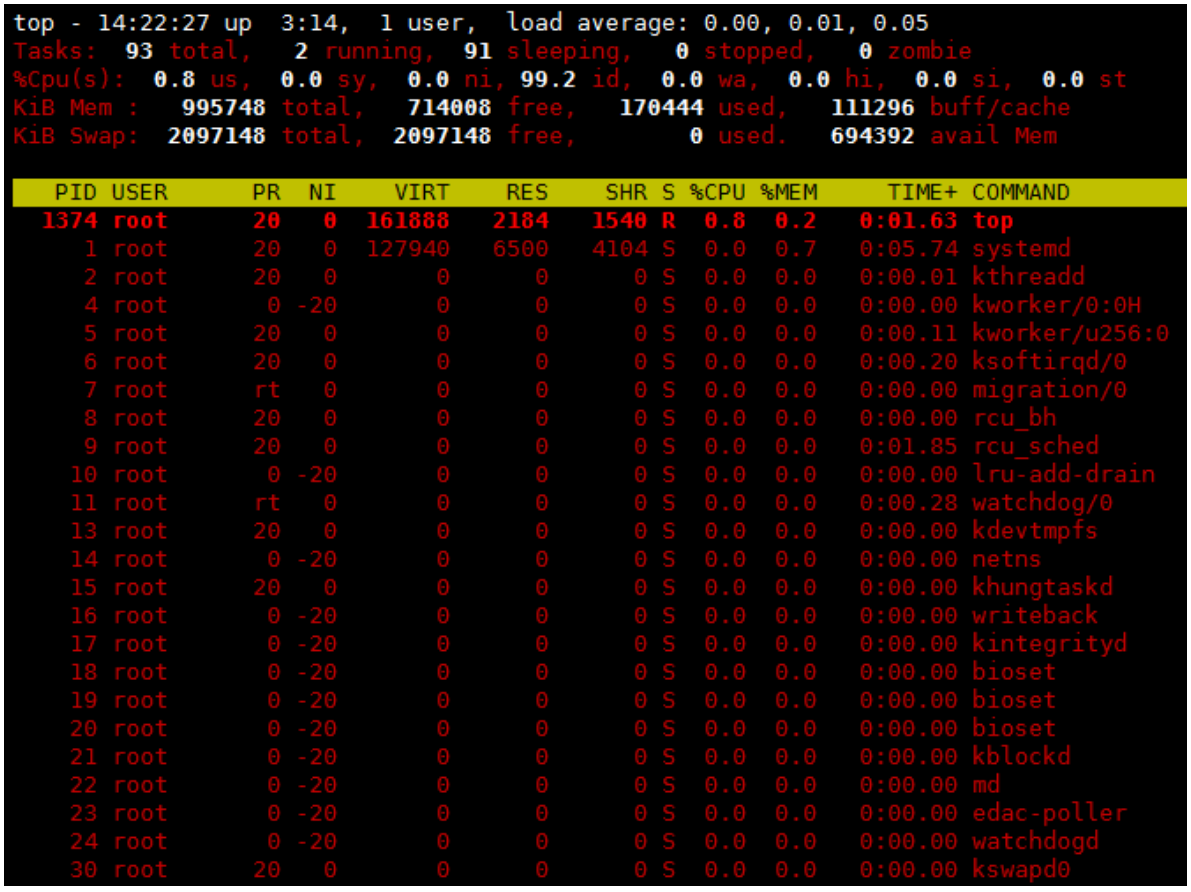
top یک برنامه‌ی interactive می‌باشد، یعنی با زدن دکمه‌های کیبورد، می‌توانیم به آن دستورهایی بدهیم. در این بخش با برخی از این دکمه‌ها و عملکرد آنها آشنا می‌شویم، اما برای داشتن لیست کاملی از این دکمه‌ها و عملکردشان، بهتر است به manpage این برنامه مراجعه کنید.

با زدن دکمه‌ی 1 روی کیبورد، می‌توانیم تعداد Coreهای CPU و میزان استفاده از هر هسته را مشاهده کنیم. همانطور که در تصویر ۵ می‌بینید، ابتدا اطلاعات CPU به صورت %CPU(s) نمایش داده شده بود و پس از زدن دکمه‌ی 1، اطلاعات CPU به تفکیک هر هسته نشان داده شد.



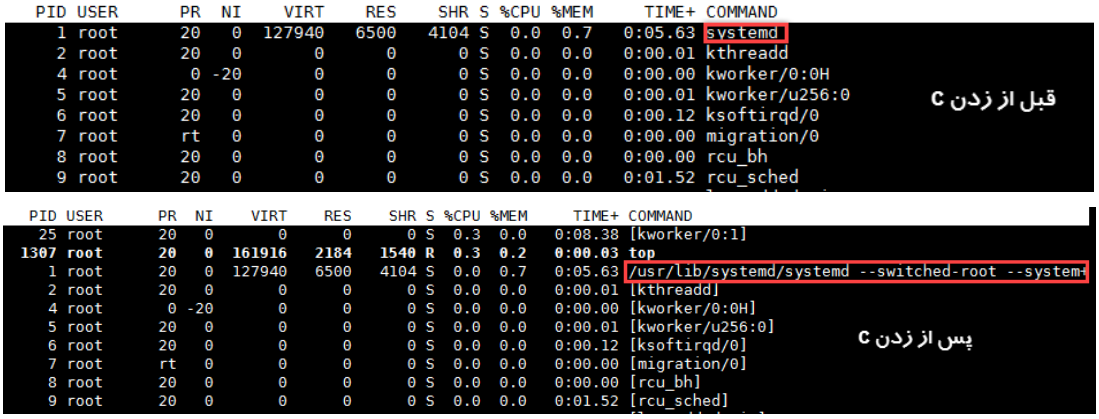
تصویر ۵- اطلاعات ردیف CPU قبل و بعد از زدن دکمه‌ی 1

با زدن دکمه‌ی Z، خروجی top رنگ‌آمیزی می‌شود:



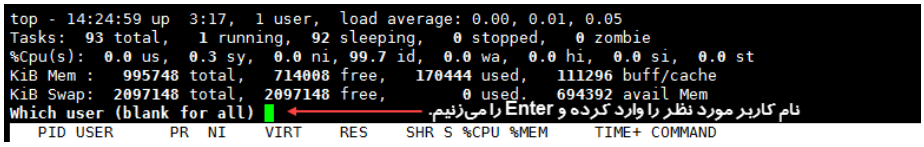
تصویر ۶- خروجی top پس از زدن دکمه‌ی Z

با زدن دکمه‌ی C، می‌توانیم نام کامل و آپشن‌های دستوری که پراسس را استارت زده را در ستون COMMAND، مشاهده کنیم. مثلا در تصویر ۷ اگر به محتویات ستون COMMAND نگاه کنید، می‌بینید که قبل از زدن دکمه‌ی C، فقط نام systemd در ستون COMMAND نوشته شده بود، اما پس از زدن دکمه‌ی C، آدرس دقیق آن و آپشن‌هایی که پراسس systemd با آن استارت شده بود به ما نمایش داده شد.

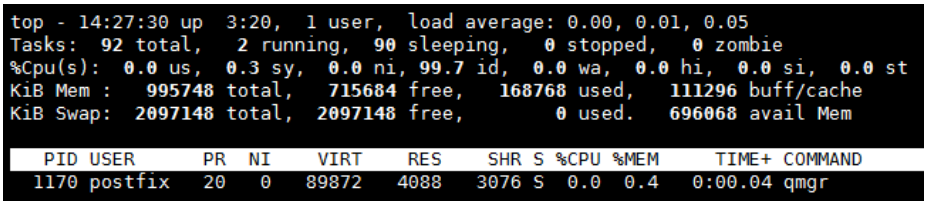


تصویر ۷- محتویات ستون COMMAND قبل و بعد از زدن دکمه‌ی C

با زدن دکمه‌ی u، می‌توانیم نام کاربر مورد نظر را وارد کرده تا فقط پراسس‌هایی که توسط آن کاربر استارت شده را مشاهده کنیم:

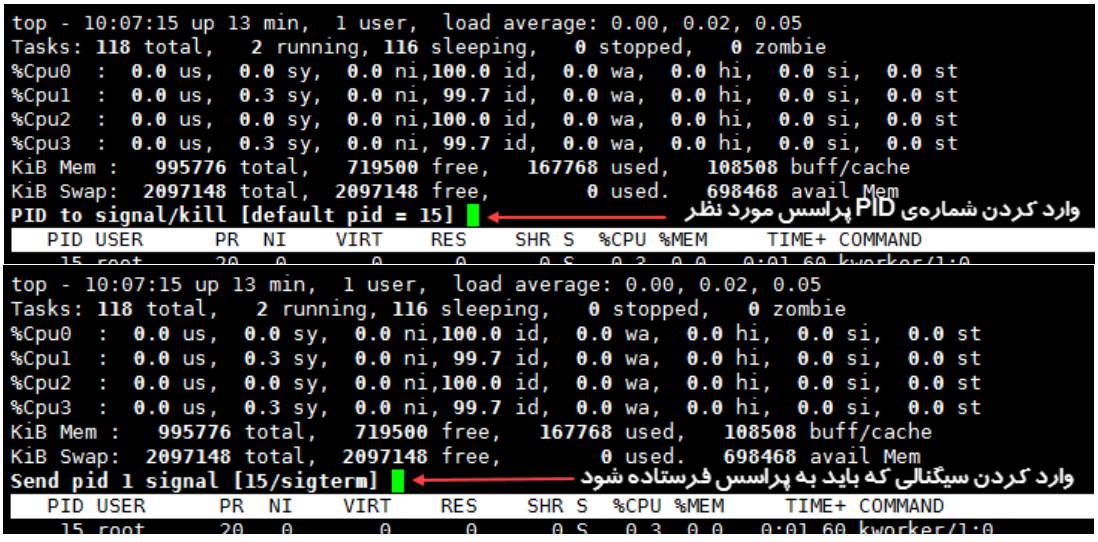


تصویر ۸- زدن دکمه‌ی u جهت مشاهده‌ی پراسس‌های ایجاد شده توسط یک کاربر خاص



تصویر ۹- با زدن دکمه‌ی u و جستجو برای کاربر postfix، پراسس‌های ایجاد شده توسط این کاربر (سرویس) را می‌بینیم.

با زدن دکمه‌ی k، می‌توانیم با وارد کردن شماره‌ی PID پراسس مورد نظر، به آن پراسس سیگنالی بفرستیم. بعدا در مورد سیگنال‌ها به طور مفصل صحبت می‌کنیم؛ اما فعلا به طور کلی می‌توانیم بگوییم که می‌توانیم از سیگنال‌ها برای kill یا به عبارت دیگر، بستن یک پراسس استفاده کنیم. توجه کنید که کاربرهای غیر از root، فقط پراسس‌هایی که خودشان استارت زده باشند را می‌توانند ببندند.



تصویر ۱۰- با زدن دکمه‌ی k می‌توانیم پراسس مورد نظر را پایان دهیم.

مشاهده‌ی پراسس‌های سیستم با استفاده از glances

glances یکی از برنامه‌های جدیدی می‌باشد که از آن برای مشاهده و مدیریت پراسس‌های سیستم استفاده می‌شود. این برنامه عملکردی نظیر top دارد، یعنی پراسس‌ها و وضعیت آنها را به صورت زنده به ما نشان می‌دهد. این برنامه به صورت پیش‌فرض روی سیستم نصب نیست و باید خودمان آن را نصب کنیم.

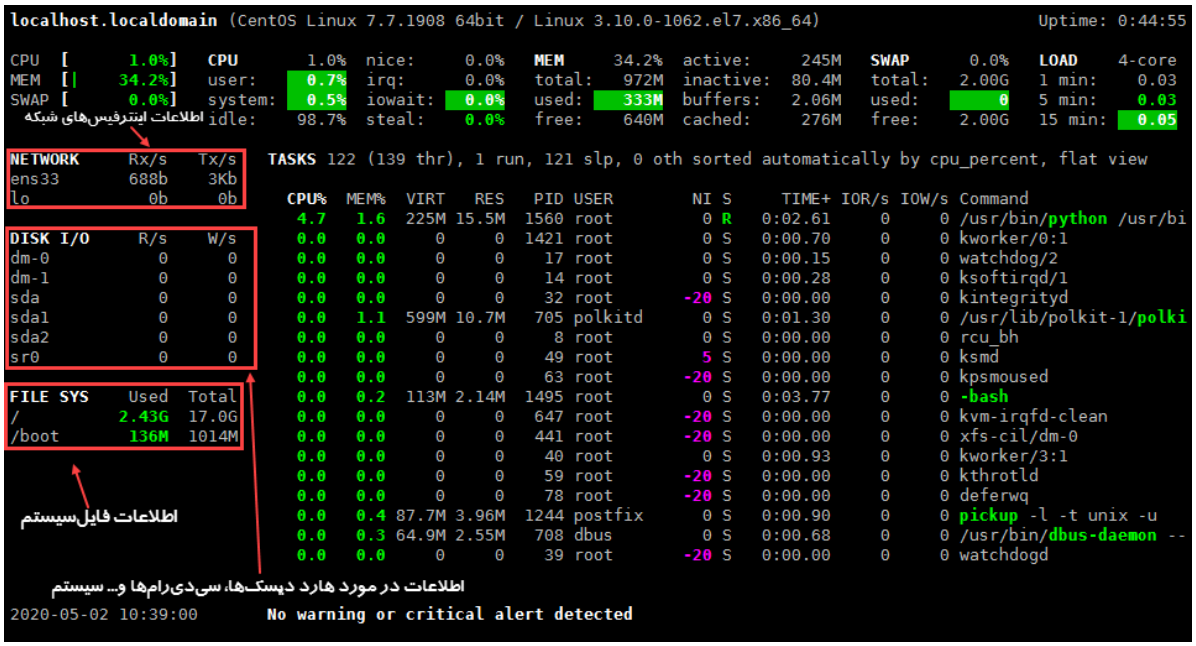
> توجه کنید که برای نصب این برنامه از طریق yum، باید ریپازیتوری EPEL را نصب کرده باشید.

```
[root@localhost ~]# yum install glances
...
Complete!
```

پس از نصب برنامه، می‌توانیم با استفاده از دستور glances، وارد این برنامه شویم:

```
[root@localhost ~]# glances
```

به محض وارد کردن این دستور، با نمایی نظیر تصویر ۱۱ مواجه می‌شویم. همانطور که می‌بینید این برنامه علاوه بر اطلاعات معمول در مورد CPU، RAM و پراسس‌ها، اطلاعاتی در مورد اینترفیس‌های شبکه، دیسک‌های موجود در سیستم (هارد دیسک، سی‌دی‌رام و...) و همچنین فایل سیستم نیز به ما می‌دهد.



تصویر ۱۱- نمایی از محیط برنامه‌ی glances

برنامه‌ی glances بسیار قدرتمند می‌باشد و در صورت تنظیم صحیح، می‌توانند نمودار میزان مصرف CPU و... را نیز ایجاد کند. البته ما به توضیح این قابلیت‌های برنامه نمی‌پردازیم. پیشنهاد می‌شود که در صورت تمایل manpage این دستور را مطالعه کنید.

تست سخت‌افزار با استفاده از stress

با استفاده از برنامه‌ی stress، می‌توانیم روی CPU، RAM، دستگاه‌های ورودی‌خروجی (I/O) و دیسک‌های موجود در سیستم، یک مقدار قابل تنظیم workload بیاندازیم. این برنامه به صورت پیش‌فرض روی سیستم نصب نیست و باید آن را با استفاده از yum روی سیستم نصب کنیم.

> توجه کنید که برای نصب این برنامه از طریق yum، باید ریپازیتوری EPEL را نصب کرده باشید.

```
[root@localhost ~]# yum install stress
...
Complete!
```

برای این که روی CPUهای سیستم workload بیاندازیم، دستور stress را با آپشن c- و تعداد هسته‌هایی که می‌خواهیم روی آن workload بیاندازیم، وارد می‌کنیم:

```
[root@localhost ~]# stress -c 2
stress: info: [1528] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
```

به محض وارد کردن این دستور، ما روی دو تا از هسته‌های CPU، استرس (یا همان workload) می‌اندازیم. بیایید با استفاده از یک TTY دیگر، نگاهی به اطلاعات CPU در برنامه‌ی top بیاندازیم:

```
top - 12:52:33 up 21 min,  1 user,  load average: 1.70, 0.93, 0.44
Tasks: 127 total,   3 running, 124 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 99.7 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu2  : 99.7 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni, 99.7 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
```

تصویر ۱۲- میزان استفاده از هر هسته‌ی CPU هنگام اعمال استرس

همانطور که می‌بینید، برنامه‌ی stress دو تا از هسته‌های CPU ما را کاملا پر کرده است.

> برنامه‌ی stress به صورت پیش‌فرض، به مدت زمان بی‌نهایت روی CPU استرس می‌اندازد. برای این که از این برنامه خارج شوید، باید کمه‌ی Ctrl + C را بزنید.

ما می‌توانیم با استفاده از آپشن t-، به stress بگوییم که به مدت چند ثانیه ایجاد استرس کند. مثلا اگر بخواهیم stress به مدت ۱۲۰ ثانیه روی CPU ایجاد استرس کند، دستور stress را به صورت زیر وارد می‌کنیم:

```
[root@localhost ~]# stress -c 2 -t 120
stress: info: [1528] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
```

با وارد کردن این دستور، stress پس از ۱۲۰ ثانیه به صورت اتوماتیک کار خود را متوقف می‌کند و پیام زیر را به ما نشان می‌دهد:

```
stress: info: [1582] successful run completed in 120s
```

ما می‌توانیم روی هارددیسک‌ها نیز استرس بیاندازیم. برای این کار، از آپشن d- استفاده می‌کنیم:

```
[root@localhost ~]# stress -d 1 -t 60
stress: info: [1360] dispatching hogs: 0 cpu, 0 io, 0 vm, 1 hdd
```

به محض وارد کردن این دستور، استرس شروع به نوشتن روی هارددیسک می‌کند. معمولا با وارد کردن این دستور، سیستم شما بسیار کند می‌شود. اگر به top نگاهی بیاندازیم، می‌بینیم که مقدار wa هر هسته‌ی CPU دائما پر و خالی می‌شود:

```
top - 13:25:51 up 54 min,  1 user,  load average: 1.48, 1.20, 0.71
Tasks: 145 total,   1 running, 128 sleeping,  16 stopped,   0 zombie
%Cpu0  :  0.0 us,  1.0 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  3.4 sy,  0.0 ni, 13.7 id, 82.8 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  1.4 sy,  0.0 ni, 84.9 id, 12.9 wa,  0.0 hi,  0.7 si,  0.0 st
KiB Mem :   995776 total,    66800 free,   486548 used,   442428 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.   358144 avail Mem
```

تصویر ۱۳- مشاهده‌ی تاثیر اعمال استرس روی هارد دیسک بر CPU

همانطور که گفتیم، مقدار wa اشاره‌ی مستقیم به دستگاه‌های I/O سیستم (مخصوصا هارددیسک) دارد. اگر روی یک سرور کار می‌کنید که بسیار آهسته می‌باشد، یکی از کارهایی که برای پیدا کردن دلیل کندی آن می‌توانید بکنید، چک کردن مقدار wa در top می‌باشد.

نظارت دستگاه‌های I/O با *iotop*

برای نظارت دستگاه‌های I/O نظیر هارددیسک، می‌توانیم از برنامه‌ی iotop استفاده کنیم. این برنامه به صورت پیش‌فرض روی سیستم نصب نیست و باید خودمان آن را با استفاده از yum نصب کنیم:

```
[root@localhost ~]# yum install iotop
...
Complete!
```

بیایید بار دیگر روی هارددیسک استرس بیاندازیم و با استفاده از iotop هارددیسک را مانیتور کنیم:

```
[root@localhost ~]# stress -d 1 -t 60
stress: info: [1360] dispatching hogs: 0 cpu, 0 io, 0 vm, 1 hdd
[root@localhost ~]# iotop
```

```
Total DISK READ :       0.00 B/s | Total DISK WRITE :      32.45 M/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:      31.43 M/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 1561 be/4 root        0.00 B/s   32.45 M/s  0.00 % 98.00 % stress -d 1 -t 60
  323 be/4 root        0.00 B/s    0.00 B/s  0.00 % 76.15 % [kworker/u256:3]
 1507 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.15 % [kworker/1:2]
 1536 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % python /usr/sbin/iotop
    1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % systemd --switched-root --system --deserialize 22
    2 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
    3 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/0:0]
    4 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/0:0H]
    6 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/0]
    7 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [migration/0]
```

تصویر ۱۴- استفاده از iotop برای نظارت بر روی تاثیر پراسس‌ها بر هارددیسک

همانطور که می‌بینید، با استفاده از iotop می‌توانیم ببینیم که چه پراسس‌هایی در حال استفاده از I/O سیستم می‌باشند و همچنین می‌توانیم ببینیم که چه میزان استفاده از هارددیسک می‌تواند سیستم را از کار بیاندازد یا کُند کند.

نکته: استفاده از دستور stress -d می‌تواند سیستم شما (هم ماشین مجازی و هم خود سیستم) را به شدت کُند کرده و تقریبا آن را غیر قابل استفاده کند، پس هنگام استفاده از این دستور، مراقب باشید.

آشنایی با مفهوم پراسس‌های Foreground و Background

اجرای برخی از برنامه‌ها، ممکن است مدت زیادی به طول انجامد و ما در اکثر اوقات، نمی‌خواهیم که شِل به طور کامل در اختیار یک برنامه قرار گیرد. برای رفع این مشکل، باید پراسسی که شل را در اختیار خود گرفته را در Background اجرا کنیم. اجرای یک برنامه در بک‌گراند، کار نسبتا ساده‌ای می‌باشد. در این بخش، با روش‌های متفاوت این کار آشنا می‌شویم.

علامت امپرسند (&)

ساده‌ترین روش برای اجرای یک برنامه در بک‌گراند، قرار دادن یک علامت امپرسند (&) پس از نوشتن دستور می‌باشد. بیایید این کار را روی دستور sleep امتحان کنیم. از دستور sleep در شِل‌اسکریپت‌ها برای ایجاد توقف (pause)، استفاده می‌کنند. بیایید اول عملکرد این دستور را بررسی کنیم.

```
[root@localhost ~]# sleep 5
```

همانطور که می‌بینید، به محض وارد کردن این دستور، سیستم به مدت ۵ ثانیه کنترل شِل را از دست شما می‌گیرد و شما در این ۵ ثانیه نمی‌توانید از شِل استفاده کنید و دستوری در آن اجرا کنید. به محض پایان ۵ ثانیه، سیستم کنترل شِل را به شما پس می‌دهد و می‌توانید مانند قبل با آن کار کنید.

برای این که کاری کنیم که هم sleep اجرا شود و هم ما بتوانیم همچنان از شِل استفاده کنیم، باید sleep را به بک‌گراند بفرستیم. همانطور که گفتیم، برای این کار کافی است پس از دستور مورد نظر، یک علامت & قرار دهیم.

```
[root@localhost ~]# sleep 30 &
[1] 1464
```

همانطور که می‌بینید، این بار دستور sleep کنترل شِل را از دست ما نگرفت و در بک‌گراند اجرا شد (می‌توانید صحت این امر را با نگاه‌کردن به خروجی برنامه‌ی ps یا top بررسی کنید). اما بیایید به خروجی این دستور کمی نگاه کنیم. در خروجی، عدد [1] و سپس 1464 را می‌بینید. عدد درون براکت، نشان دهنده‌ی job number این دستور می‌باشد و عدد خارج از براکت، نشان دهنده‌ی PID آن دستور می‌باشد.

اما منظور از job number چیست؟ هر پراسسی را که در بک‌گراند اجرا کنیم، یک job number خواهد گرفت. چون sleep 30 & اولین پراسسی بود که در بک‌گراند قرار دادیم، job number شماره 1 به آن اختصاص یافت. اگر یک پراسس دیگر را در این حین در بک‌گراند قرار دهیم، job number شماره 2 به آن اختصاص می‌یابد.

برای مشاهده‌ی پراسس‌هایی که در بک‌گراند قرار داده‌ایم، از دستور jobs استفاده می‌کنیم. این دستور، کلیه‌ی پراسس‌هایی که توسط کاربر کنونی در بک‌گراند قرار گرفته را نشان می‌دهد. بیایید نگاهی به این دستور بیاندازیم:

```
[root@localhost ~]# sleep 50 &
[1] 1370
[root@localhost ~]# sleep 30 &
[2] 1371
[root@localhost ~]# jobs
[1]-   Running                 sleep 50 &
[2]+   Running                 sleep 30 &
```

همانطور که می‌بینید jobs پراسس‌هایی که در بک‌گراند قرار داده‌ایم و همچنین وضعیت آنها را به ما نشان می‌دهد. اگر بخواهیم jobs به ما PID هر پراسس را نشان دهد، از آپشن l- استفاده میکنیم، اما امتحان چگونگی عملکرد آن را به شما می‌سپاریم.

اگر به خروجی دقت کنید، خواهید دید که پس از [1]، یک علامت - و پس از [2] یک علامت + قرار دارد. علامت +، یعنی این پراسس، آخرین پراسسی است که در بک‌گراند قرار گرفته است و علامت -، پراسس یکی‌مانده‌به‌آخر را نشان می‌دهد.

پس از پایان این پراسس‌ها در بک‌گراند، در اولین اینترکشن خود با شِل، با پیامی نظیر زیر مواجه می‌شویم:

```
[1]   Done                    sleep 50
[2]   Done                    sleep 30
```

اگر بخواهیم پراسسی که در بک‌گراند قرار داده‌ایم را در Foreground بیاوریم، باید از دستور fg استفاده کنیم. چگونگی انجام این کار را در صفحه‌ی بعد می‌بینید.

```
[root@localhost ~]# sleep 50&
[1] 1416
[root@localhost ~]# sleep 30&
[2] 1417
[root@localhost ~]# jobs
[1]-  Running                 sleep 50 &
[2]+  Running                 sleep 30 &
[root@localhost ~]# fg %1
sleep 50
```

همانطور که می‌بینید با وارد کردن دستور fg و قرار دادن job number مورد نظر پس از یک علامت %، پراسس مورد نظر به فورگراند آورده می‌شود. بدی قرار دادن یک پراسس در فورگراند این است که حال مجبور هستید تا پایان یافتن این پراسس، شِل را در اختیار این پراسس قرار دهید.

قرار دادن یک برنامه‌ی در حال اجرا در بک‌گراند

گاهی اوقات ممکن است یک پراسس را اجرا کنیم و متوجه شویم که این پراسس زمان زیادی از ما و شِل می‌گیرد. در چنین حالتی، می‌توانیم این پراسس در حال اجرا را به بک‌گراند بفرستیم. برای این کار، باید ابتدا پراسس را با استفاده از دکمه‌ی Ctrl + Z، متوقف یا pause کنیم.

> ما قبلا گفته بودیم که می‌توانیم از دکمه‌ی Ctrl + Z برای خروج از برنامه‌ها استفاده کنیم؛ اما چیزی که نگفته بودیم، این است که ما با استفاده از دکمه‌ی Ctrl + Z، فقط آن پراسس را pause می‌کنیم، اما آن پراسس هنوز در جدول پراسس‌ها قابل مشاهده می‌باشد؛ در واقع در چنین حالتی، هنوز از برنامه خارج نشده‌ایم.

بیایید این کار را امتحان کنیم. فرض کنید می‌خواهیم با استفاده از دستور yum، نرم‌افزار mariadb را روی سیستم نصب کنیم:

```
[root@localhost ~]# yum -y install mariadb
```

> **نکته:** استفاده از آپشن y- در yum، باعث می‌شود که yum پس از پیدا کردن برنامه‌ها درون ریپازیتوری، دیگر از ما درخواست تایید نصب نکند و مستقیما به سراغ نصب برنامه برود.

پس از اجرای این دستور، می‌بینیم که آپدیت‌کردن ریپازیتوری‌ها کمی زمان می‌برد. به همین دلیل، می‌خواهیم yum در بک‌گراند اجرا شود. برای این کار، حین اجرای این دستور، دکمه‌ی Ctrl + Z را می‌زنیم:

```
^Z
[1]+  Stopped                 yum -y install mariadb
```

همانطور که می‌بینید، با زدن این دکمه، عملکرد برنامه‌ی yum را موفق کردیم. اگر نگاهی به خروجی jobs بیاندازیم، با چنین چیزی مواجه می‌شویم:

```
[root@localhost ~]# jobs
[1]+  Stopped                 yum -y install mariadb
```

همانطور که گفتیم این پراسس در حال حاضر متوقف است. برای اجرای مجدد این پراسس در بک‌گراند، کافی است دستور bg و job number دستور متوقف شده را به صورت زیر وارد کنیم:

```
[root@localhost ~]# bg %1
[1]+ yum -y install mariadb &
```

با اجرای این دستور، yum دقیقا از جایی که متوقف شده بود شروع به کار می‌کند و در بک‌گراند کار خود را

انجام می‌دهد. اما اگر این دستور را وارد کنید، خواهید دید که yum، کلیه‌ی خروجی‌های خود را روی شِل (یا به طور صحیح‌تر، STDOUT) می‌ریزد. شما در این حالت با زدن دکمه‌ی Enter می‌توانید از شِل استفاده کنید و دستورهای مورد نظر را وارد کنید، اما شِل دائما کار شما را با نمایش خروجی yum، مختل می‌کند. یکی از راه‌های جلوگیری از این مشکل، استفاده از redirectorها برای ریختن STDOUT در یک فایل می‌باشد.

مشکل دیگری که استفاده از علامت & و Ctrl + Z برای فرستادن یک پراسس به بک‌گراند ایجاد می‌کند، این است که پراسسی که به بک‌گراند رفته، به محض خروج شما از session کنونی (قطع ارتباط SSH، وارد کردن دستور exit در یک tty و...)، متوقف می‌شود. این امر می‌تواند بسیار مشکل‌ساز باشد، چون بسیاری از اوقات ممکن است اسکریپت یا دستوری را اجرا کنیم که برای انجام وظیفه‌ی خود، چندین ساعت زمان لازم داشته باشد و طبیعتا ما نمی‌توانیم در طول این مدت، به سیستم متصل باشیم.

نکته: همانطور که گفتیم Ctrl + Z پراسس را نمی‌بندد (kill نمی‌کند)، بلکه آنرا متوقف یا pause می‌کند. برای این که یک پراسس در حال اجرا را کاملا ببندیم، می‌توانیم از کلیدهای Ctrl + C استفاده کنیم.

استفاده از *nohup* برای فرستادن یک دستور به بک‌گراند

در بخش قبل، دیدیم که استفاده از روش & و Ctrl + Z برای فرستادن یک پراسس به بک‌گراند، هم در اکثر اوقات STDOUT را اشغال می‌کند و هم به محض پایان session، پراسسی که در بک‌گراند قرار داده بودیم را از بین می‌برد.

اگر بخواهیم پراسسی که در بک‌گراند قرار داده‌ایم پس از پایان session (یا logout کردن از سیستم) بسته نشود و همچنین STDOUT را اشغال نکند، باید از دستور nohup استفاده کنیم. برای مثال، فرض کنید می‌خواهیم با استفاده از yum، ابزارهای مخصوص توسعه‌ی نرم‌افزار را دانلود کنیم. در حالت نرمال، باید از دستور زیر برای نصب این ابزارها استفاده کنیم:

```
[root@localhost ~]# yum groupinstall "Development Tools"
Total size: 60 M
Total download size: 58 M
Is this ok [y/d/N]:
```

همانطور که می‌بینید، برای نصب این ابزارها باید ۵۸ مگابایت فایل دانلود کنیم، پس بهتر است این دستور را در بک‌گراند ایجاد کنیم. از قبل می‌دانیم که استفاده از Ctrl + Z و علامت & به تنهایی به درد ما نمیخورد، پس به سراغ nohup می‌رویم:

```
[root@localhost ~]# nohup yum -y groupinstall "Development Tools" &
[1] 1782
[root@localhost ~]# nohup: ignoring input and appending output to
‘nohup.out’

[root@localhost ~]#
```

همانطور که می‌بینید، با استفاده از nohup در ابتدای دستور و قرار دادن & در انتهای دستور، پراسس مورد نظر را در بک‌گراند قرار دادیم. توجه کنید که دستور nohup، هم STDOUT و هم STDERR پراسس را درون فایل nohup.out در home folder شما قرار می‌دهد. حال که پراسس را با استفاده از nohup در بک‌گراند قرار دادیم، بیایید نگاهی به خروجی jobs بیاندازیم:

```
[root@localhost ~]# jobs
[1]+  Running                 nohup yum -y groupinstall "Development Tools" &
```

همانطور که می‌بینید، yum در حال نصب کردن ابزارهای درخواستی می‌باشد. پس از اتمام کار yum، خروجی jobs شبیه زیر خواهد شد:

```
[root@localhost ~]# jobs
[1]+  Done                    nohup yum -y groupinstall "Development Tools"
```

نکته: با استفاده از yum groupinstall، می‌توانیم مجموعه‌ای از ابزارها را با نوشتن یک دستور دانلود کنیم. برای مشاهده‌ی مجموعه ابزارهایی که می‌توانید به yum groupinstall بدهید، از دستور yum grouplist استفاده کنید. برای به دست آوردن اطلاعات در مورد یک گروه، می‌توانید از yum groupinfo و نام آن گروه استفاده کنید. توجه کنید که نام گروه مورد نظر هم برای نصب و هم برای به دست آوردن اطلاعات، باید میان "" نوشته شود (البته این فقط برای گروه‌هایی که اسمشان از دو کلمه ایجاد شده است صدق می‌کند). تست و بررسی این دستورها و آپشن‌ها را به خودتان می‌سپاریم 😊.

بستن یک پراسس در حال اجرا در بک‌گراند

تا به اینجا زیاد در مورد چگونگی قرار دادن یک پراسس در بک‌گراند صحبت کردیم. اما اگر بخواهیم پراسسی که در بک‌گراند قرار داده‌ایم را ببندیم باید چه کنیم؟

برای این کار کافی است از دستور kill و job number آن پراسس استفاده کنیم. بیایید این کار را با یک مثال انجام دهیم:

```
[root@localhost ~]# sleep 400 &
[1] 14017
[root@localhost ~]# jobs
[1]+  Running                 sleep 400 &
[root@localhost ~]# kill %1
[root@localhost ~]# jobs
[1]+  Terminated              sleep 400
```

همانطور که می‌بینید، ما ابتدا دستور sleep 400 را با استفاده از علامت & در بک‌گراند قرار دادیم. سپس با استفاده از دستور jobs، نگاهی به کلیه‌ی پراسس‌های موجود در بک‌گراند انداختیم. همانطور که می‌بینید، job number دستور sleep 400، عدد 1 می‌باشد.

برای بستن این پراسس، از دستور kill %1 استفاده کردیم. عدد 1، job number پراسسی بود که قصد بستن آن را داشتیم. همانطور که می‌بینید پس از اجرای دستور kill، پراسس sleep 400 بسته یا terminate شد.

مدیریت اولویت پراسس‌ها

گاهی اوقات، ممکن است بخواهیم میزان استفاده‌ی پراسس‌ها از CPU را اولویت‌بندی کنیم. مثلا ممکن است یک اسکریپت داشته باشیم که اجرای کامل آن مدت زمان زیادی طول بکشد و همچنین به CPU زیادی احتیاج داشته باشد. در چنین حالتی، شاید بخواهیم کاری کنیم که آن اسکریپت، از همه‌ی CPU استفاده نکند؛ یا ممکن است یک پراسس داشته باشیم که مهم‌تر از سایر پراسس‌هاست و بخواهیم آن پراسس به منابع CPU دسترسی بیشتری داشته باشد. در هر دو حالت، با مدیریت اولویت‌ها، می‌توانیم چگونگی برخورد سیستم با پراسس‌ها را مشخص کنیم. برای تغییر اولویت یک پراسس، باید مقدار niceness (قشنگی 😊) آن پراسس را تغییر دهیم. در این بخش با چگونگی انجام این کار آشنا می‌شویم.

اجرای یک پراسس با اولویت مورد نظر

دستور nice به ما اجازه می‌دهد که یک پراسس را با میزان niceness مورد نظر، اجرا کنیم. این دستور به شکل زیر می‌باشد:

```
nice -n VALUE COMMAND
```

به طوری که VALUE، عددی بین ۲۰- تا ۱۹ می‌باشد. هر چه این عدد **کمتر** باشد، اولویت پراسس **بالاتر** خواهد بود. به صورت پیش‌فرض، مقدار niceness صفر می‌باشد. COMMAND، پراسسی است که باید با niceness مشخص شده اجرا شود. برای مثال:

```
[root@localhost ~]# nice -n 10 sleep 500
[root@localhost ~]# ps -l 1496
 F   S   UID   PID   C   PRI   NI   ADDR   SZ     WCHAN    TTY    TIME   CMD
 0   S   1496  1487  0   90    10   -      26989  hrtime   pts/1  23:12  sleep 500
```

همانطور که می‌بینید، با استفاده از دستور nice، برنامه‌ی sleep را با مقدار niceness برابر با ۱۰، اجرا کردیم. اگر به خروجی دستور ps نیز نگاه کنید، می‌بینید که مقدار niceness این دستور، برابر با ۱۰ می‌باشد (ستون NI).

نکته: اگر هنگام اجرای nice، مقدار niceness را مشخص نکنیم، به صورت پیش‌فرض niceness برابر با ۱۰ خواهد شد.

تغییر اولویت یک پراسس در حال اجرا

اگر بخواهیم مقدار niceness یک پراسس در حال اجرا را تغییر دهیم، باید از دستور renice استفاده کنیم. این دستور، به شکل زیر می‌باشد:

```
renice PRIORITY [-p PID] [-u USERS] [-g GROUPS]
```

با استفاده از renice، می‌توانیم اولویت یک یا چندین پراسس را با استفاده از PID آنها تغییر دهیم. همچنین می‌توانیم اولویت همه‌ی پراسس‌هایی که توسط یک یا چند کاربر یا یک یا چند گروه ایجاد شده است را تغییر دهیم. بیایید این کار را امتحان کنیم.

همانطور که در خروجی ps می‌بینید، پراسس sleep با PID برابر با ۱۴۸۷ در حال اجرا می‌باشد. اولویت این پراسس در حال حاضر ۸۰ می‌باشد (ستون PRI) و میزان niceness آن برابر با ۰ می‌باشد (ستون NI).

```
[root@localhost ~]# ps -efl
 F  S  UID   PID   PPID  C  PRI  NI  ADDR  SZ     WCHAN   STIME  TTY    TIME  CMD
 0  S  root  1370  1361  0  80   0   -     26989  hrtime  23:31  pts/1  0:00  sleep 3600
```

حال که PID این پراسس را داریم، می‌توانیم با استفاده از برنامه‌ی renice، اولویت این پراسس را تغییر دهیم:

```
[root@localhost ~]# renice 15 -p 1370
1370 (process ID) old priority 0, new priority 15
[root@localhost ~]# ps -efl | grep sleep
 F  S  UID   PID   PPID  C  PRI  NI  ADDR  SZ     WCHAN   STIME  TTY    TIME  CMD
 0  S  root  1370  1361  0  95   15  -     26989  hrtime  23:32  pts/1  0:00  sleep 3600
```

همانطور که می‌بینید، با استفاده از دستور renice، توانستیم میزان niceness و در نتیجه‌ی آن، اولویت یک پراسس در حال اجرا را تغییر دهیم.

نکته: توجه کنید که فقط کاربر root می‌تواند مقدار niceness یک پراسس در حال اجرا را کمتر از ۰ قرار دهد.

ارسال سیگنال به پراسس‌ها

گاهی اوقات ممکن است یک پراسس هنگ کند و احتیاج به یک تلنگر داشته باشد تا به کار خود ادامه دهد یا ممکن است یک پراسس کل CPU را در اختیار خود گیرد و به هیچ پراسس دیگری راه نفس کشیدن ندهد. در این حالات، ما احتیاج به روشی برای تلنگر زدن به پراسس‌ها داریم.

پراسس‌ها در سیستم‌های لینوکسی، با استفاده از سیگنال‌ها با هم ارتباط برقرار می‌کنند. توسعه‌دهندگان یک برنامه، تعیین می‌کنند که آن برنامه چه رفتاری در برابر سیگنال‌ها انجام می‌دهد. اکثر برنامه‌هایی که به صورت صحیح برنامه‌نویسی شده باشند، می‌توانند سیگنال‌های استاندارد یونیکسی را دریافت کنند و با توجه به آن، عملی را انجام دهند. مهم‌ترین سیگنال‌هایی که با آن کار خواهیم کرد، سیگنال ۱، سیگنال ۹ و سیگنال ۱۵ می‌باشند که عملکرد هر کدام به شرح زیر می‌باشد:

- سیگنال ۱ یا SIGHUP، برنامه‌هایی که در foreground هستند را terminate کرده و باعث می‌شود که آن برنامه‌ها، بار دیگر فایل‌های config خود را بخوانند.
- سیگنال ۹ یا SIGKILL، باعث می‌شود پراسس بدون انجام کارهای معمول هنگام خارج شدن، بسته شود.
- سیگنال ۱۵ یا SIGTERM، به پراسس اجازه می‌دهد که کارهای معمول هنگام خارج شدن، نظیر بستن فایل‌های باز و... را انجام داده و سپس بسته شود.

ارسال سیگنال با استفاده از دستور `kill`

قبلا از دستور `kill` برای بستن پراسس‌های موجود در بک‌گراند استفاده کردیم. ما می‌توانیم با استفاده از این دستور، به پراسس‌ها سیگنال نیز ارسال کنیم. برای این کار، کافی است سیگنال مورد نظر و همچنین PID پراسس مورد نظر را به دستور `kill` بدهیم. برای مثال:

```
[root@localhost ~]# ps 1368
   PID TTY      STAT   TIME COMMAND
  1368 pts/1    S+     0:00 sleep 3600
[root@localhost ~]# kill -15 1368
[1]+  Terminated              sleep 3600
```

همانطور که می‌بینید، با استفاده از دستور `kill` و قرار دادن سیگنال مورد نظر پس از علامت -، پراسس دارای PID برابر با ۱۳۶۸ را بستیم.

نکته: اگر به دستور `kill` هیچ شماره‌ی سیگنالی ندهیم، به صورت پیش‌فرض سیگنال ۱۵ را به پراسس ارسال می‌کند.

گاهی اوقات ممکن است پراسس‌ها، به سیگنال ارسالی ما توجهی نکنند و به کار خود ادامه دهند. در چنین شرایطی، باید به سراغ سایر سیگنال‌ها، نظیر سیگنال ۱ یا ۹، برویم. برای مثال:

```
[root@localhost ~]# ps 1372
   PID TTY      STAT   TIME COMMAND
  1372 pts/0    T      0:00 vi
[root@localhost ~]# kill 1372
[root@localhost ~]# ps 1372
   PID TTY      STAT   TIME COMMAND
  1372 pts/0    T      0:00 vi
[root@localhost ~]# kill -SIGHUP 1372
```

```
[root@localhost ~]# ps 1372
   PID TTY      STAT   TIME COMMAND
  1372 pts/0    T      0:00 vi
[root@localhost ~]# kill -9 1372
 [1]+  Killed                  vi
```

همانطور که می‌بینید، این پراسس به سیگنال ۱۵ و سیگنال ۱، پاسخی نداد و به کار خود ادامه داد. به همین دلیل، ما مجبور به استفاده از سیگنال ۹ شدیم. سیگنال ۹، سیگنالی است که بدون هیچ رحمی یک پراسس را می‌بندد.

توجه کنید که برای بستن یک پراسس، بهتر است ابتدا سیگنال ۱۵ را به آن بدهیم. اگر پراسس به آن سیگنال توجهی نکرد، به آن سیگنال ۱ را می‌دهیم و اگر باز هم پراسس به آن سیگنال توجهی نکرد، به آن سیگنال ۹ را می‌دهیم. سیگنال ۱۵ و ۱، باعث می‌شوند که پراسس به صورت صحیح و بدون هیچ گونه عوارض جانبی بسته شود. اما سیگنال ۹، بدون هیچ رحمی پراسس را می‌بندند و ممکن است باعث به وجود آمدن برخی خرابی‌ها در فایل‌ها شود. پس هر وقت واقعا مجبور بودیم به سراغ سیگنال ۹ می‌رویم.

نکته: ما می‌توانیم به جای شماره‌ی سیگنال، نام سیگنال را به دستور `kill` بدهیم. برای این کار کافی است نام سیگنال مورد نظر را پس علامت - قرار دهیم.

ارسال سیگنال با استفاده از *killall*

مشکل دستور `kill`، این است که برای ارسال سیگنال به یک پراسس، به PID آن پراسس نیاز داریم. دستور `killall`، می‌تواند به پراسس‌ها با توجه به نامشان (یا دستوری که آن پراسس را ایجاد کرده) سیگنال بفرستد. استفاده از دستور `killall`، بسیار شبیه دستور `kill` می‌باشد، با این تفاوت که این بار نام پراسس را به `killall` می‌دهیم. یعنی:

```
[root@localhost ~]# ps
   PID TTY          TIME CMD
  1322 pts/0    00:00:00 bash
  1577 pts/0    00:00:00 sleep
  1578 pts/0    00:00:00 sleep
  1579 pts/0    00:00:00 sleep
  1580 pts/0    00:00:00 ps
[root@localhost ~]# killall sleep
[1]-  Terminated              sleep 400
[2]-  Terminated              sleep 500
[3]+  Terminated              sleep 600
[root@localhost ~]# ps
   PID TTY          TIME CMD
  1322 pts/0    00:00:00 bash
  1582 pts/0    00:00:00 ps
```

همانطور که می‌بینید، ما چند بار پراسس `sleep` را اجرا کرده بودیم و می‌خواستیم این پراسس‌ها را ببندیم. با استفاده از `killall` و مشخص کردن نام پراسس مورد نظر، به سیستم گفتیم که سیگنال ۱۵ را به هر پراسسی که با دستور `sleep` اجرا شده بود، بفرستد. اگر میخواستیم سیگنال دیگری به این پراسس‌ها بفرستیم، باید آن سیگنال را به صورت دستی، مشخص می‌کردیم (دقیقا مثل دستور `kill`).

چگونگی عملکرد سخت‌افزارها در کامپیوتر

در این بخش، می‌خواهیم به طور کلی در مورد سخت‌افزارهای کامپیوتر و چگونگی عملکرد آن صحبت کنیم. تا به حال فکر کردید که وقتی سیستم خود را روشن می‌کنید، تا زمان بالا آمدن سیستم عامل، چه اتفاقاتی در پشت صحنه می‌افتد؟

امروزه تقریبا همه‌ی کامپیوترها، از یک firmware داخلی برای کنترل چگونگی اجرای سیستم‌عامل استفاده می‌کنند. در سیستم‌های قدیمی‌تر، این firmware با نام Basic Input/Output System یا BIOS شناخته می‌شد. در سیستم‌های جدیدتر، روش جدیدتری به نام Unified Extensible Firmware Interface یا UEFI وظیفه‌ی مدیریت وضعیت سخت‌افزارهای سیستم و اجرای سیستم‌عامل را بر عهده دارد. هر دوی این روش‌ها، در نهایت سیستم‌عامل را اجرا می‌کنند، اما چگونگی اجرای سیستم‌عامل در هر روش متفاوت می‌باشد. ما هر دو روش را به طور خیلی مختصر توضیح می‌دهیم.

BIOS

BIOS که در سیستم‌های قدیمی از آن استفاده می‌شد، محدودیت‌های بسیاری داشت. برای مثال، BIOS فقط می‌توانست تا دو ترابایت هارد دیسک را بشناسد و زیاد سریع نبود. BIOS یک اینترفیس بسیار ساده داشت که از طریق آن می‌توانستیم برخی از سخت‌افزارها را تنظیم کنیم یا به سیستم بگوییم که از چه درایوی بوت شود. یکی از محدودیت‌های BIOS این بود که فقط می‌توانست یک سکتور داده را از روی هارد خوانده و درون RAM قرار دهد. پرواضح است که این مقدار برای اجرای کامل یک سیستم‌عامل کافی نیست.

برای دور زدن این محدودیت، ابتدا BIOS یک برنامه به نام bootloader را اجرا می‌کرد. بوت‌لودر، برنامه‌ای است که می‌تواند همه‌ی سخت‌افزارهای مورد نیاز برای اجرای سیستم‌عامل را راه‌اندازی کند. بوت‌لودر که معمولا در یک پارتیشن جداگانه قرار دارد، یک فایل تنظیمات دارد که در آن، موقعیت سیستم‌عامل مشخص شده است و حتی می‌توانیم با استفاده از آن، یک منو که به کاربر اجازه‌ی انتخاب بین سیستم‌عامل مورد نظر را می‌دهد (dual boot)، ایجاد کنیم.

برای این که BIOS بتواند بوت‌لودر را اجرا کند، باید موقعیت بوت‌لودر در سیستم را پیدا کند. اکثر سیستم‌های BIOS به ما اجازه می‌دهند که بوت‌لودر را از یک هارددیسک اینترنال، هارد اکسترنال، CD یا DVD، فلش مموری، یک فایل ISO و... به آن معرفی کنیم. اگر بوت‌لودر روی یک هارد دیسک قرار داشته باشد، باید به BIOS بگوییم که بوت‌لودر روی کدام هارددیسک و کدام پارتیشن قرار دارد. این کار را با مشخص کردن Master Boot Record یا MBR انجام می‌دهیم. MBR، اولین سکتور موجود بر روی اولین پارتیشن روی هارددیسک می‌باشد. لازم به ذکر است که برای هر سیستم، فقط یک MBR وجود دارد.

پس همانطور که گفتیم، BIOS به دنبال MBR می‌گردد و برنامه‌ی ذخیره شده در MBR را درون RAM قرار می‌دهد. از آنجایی که بوت‌لودر باید در یک سکتور هارد قرار گیرد، نمی‌تواند برنامه‌ی خیلی پیچیده‌ای باشد؛ به همین دلیل، بوت‌لودر معمولا موقعیت کرنل اصلی سیستم‌عامل مورد نظر (که معمولا در یک پارتیشن دیگر قرار دارد) را به BIOS نشان می‌دهد.

UEFI

با این که BIOS محدودیت‌های زیادی داشت، اما سالهای سال در سیستم‌های کامپیوتری به کار برده می‌شد. با پیشرفته‌تر شدن سیستم‌های کامپیوتری، چشم‌پوشی از محدودیت‌های BIOS غیرممکن شد و نیاز به روش

جدیدی برای بوت سیستم حس شد. در سال ۱۹۹۸، شرکت Intel با ایجاد سیستم Extensible Firmware Interrace یا EFI، سعی در رفع برخی از محدودیت‌های BIOS کرد. با این که فرآیند جایگزینی این سیستم جدید با BIOS بسیار آهسته بود، اما در سال ۲۰۰۵، این سیستم با نام United EFI یا UEFI تبدیل به یک استاندارد شد و امروزه تقریبا همه‌ی سیستم‌ها از این روش برای بوت استفاده می‌کنند.

در UEFI به جای استفاده از یک سکتور برای نگهداری بوت‌لودر، از یک پارتیشن مخصوص روی هارددیسک برای نگهداری بوت‌لودر استفاده می‌شود. این پارتیشن خاص که EFI System Partition یا ESP نام دارد، باعث می‌شود که بتوانیم یک بوت‌لودر با هر سایزی و یا چندین بوت‌لودر برای چندین سیستم‌عامل، داشته باشیم.

ESP معمولا از فایل‌سیستم FAT برای ذخیره‌ی بوت‌لودر استفاده می‌کند. در سیستم‌های لینوکس، ESP در موقعیت `/boot/efi` قرار دارد و فایل‌های بوت‌لودر با پسوند efi. در آن قرار گرفته‌اند.

اینترفیس‌های سخت‌افزاری

سیستم‌ها برای ارتباط با هر سخت‌افزاری که به آنها متصل می‌کنیم، از یک سری پروتکل استاندارد استفاده می‌کند. سیستم با استفاده از این پروتکل‌ها به سخت‌افزارها و دستگاه‌های متصل شده به سیستم فرمان می‌دهد. در حال حاضر، ۳ پروتکل معروف برای اتصال سخت‌افزار به سیستم وجود دارد که آنها را به شکل مختصر توضیح می‌دهیم:

- PCI، یکی از استانداردهای موجود برای اتصال سخت‌افزار به مادربورد می‌باشد که در سال ۱۹۹۳ توسعه داده شده است. امروزه تقریبا در تمامی سرورها و کامپیوترها، از نسخه‌ی PCIe یا PCI Express این استاندارد برای اتصال سخت‌افزار به مادربرد استفاده می‌شود. از دستگاه‌هایی که با PCIe به سیستم متصل می‌شوند، می‌توان هارد دیسک‌های اینترنال (اتصال به مادربرد با استفاده از SATA و SCSI از طریق استاندارد PCI)، اینترفیس‌های شبکه، کارت‌های گرافیک و... را نام برد.
- USB، یکی دیگر از استانداردهای بسیار معروف برای اتصال سخت‌افزار و دستگاه‌ها به سیستم می‌باشد. همه‌ی ما با USB آشنایی داریم پس به توضیح بیشتر آن نمی‌پردازیم.
- GPIO یا General Purpose Input/Output استانداردی است که توسط سیستم‌های کوچک و embedded نظیر Raspberry Pi و ... به کار می‌رود و امروزه از محبوبیت زیادی برخوردار شده است. این استاندارد بسیار انعطاف‌پذیر می‌باشد و به ما امکان می‌دهد که اشیاءِ را نیز کنترل کنیم.

دایرکتوری /dev

تا به حال از خود پرسیده‌اید که کرنل لینوکس، پس از شناخت یک سخت‌افزار، از چه طریقی با آن سخت‌افزار ارتباط برقرار می‌کند؟

در لینوکس، ارتباط با سخت‌افزارها از طریق device fileها صورت می‌پذیرد. دیوایس‌فایل‌ها، فایل‌هایی هستند که کرنل در دایرکتوری /dev، جهت دریافت و ارسال اطلاعات به سخت‌افزارها، قرار می‌دهد.

اگر برنامه‌ای بخواهد اطلاعات یک سخت‌افزار را بخواند، کافی است فایل مربوط به آن سخت‌افزار را در دایرکتوری /dev بخواند و همچنین اگر بخواهد چیزی را روی یک سخت‌افزار بنویسد، کافی است اطلاعات مورد نظر را در فایل مربوط به آن سخت‌افزار در دایرکتوری /dev، بنویسد.

به محض اضافه‌کردن سخت‌افزاری جدید به سیستم، مثل یک فلش مموری یا یک هارد دیسک جدید، لینوکس یک فایل جدید که نمایانگر آن سخت‌افزار می‌باشد را در دایرکتوری /dev، ایجاد می‌کند. پس از آن، هر نرم‌افزاری می‌تواند با خواندن فایل نمایانگر هر سخت‌افزار، به آن سخت‌افزار دسترسی پیدا کند.

در لینوکس، دو نوع دیوایس‌فایل وجود دارد که هر نوع، نمایانگر چگونگی انتقال داده به سخت‌افزار می‌باشند:

- دیوایس‌فایل‌های کاراکتری

انتقال داده به صورت تک‌کاراکتری صورت می‌پذیرد. دستگاه‌هایی نظیر فلش‌مموری‌ها، و ترمینال‌ها از این نوع می‌باشند.

- دیوایس‌فایل‌های بلوکی

انتقال داده به صورت بلوکی از داده‌ها صورت می‌پذیرد. دستگاه‌هایی نظیر هارددیسک‌ها و اینترفیس‌های شبکه از این نوع می‌باشند.

دایرکتوری /proc

دایرکتوری /proc یکی از مهم‌ترین ابزارها برای عیب‌یابی مشکلات سخت‌افزاری می‌باشد. این دایرکتوری روی هارددیسک سیستم قرار ندارد و بر روی RAM ذخیره می‌شود. کرنل لینوکس، فایل‌ها و داده‌های موجود در این دایرکتوری را پس از نظارت سخت‌افزارهای موجود در سیستم، آپدیت می‌کند.

برای مشاهده‌ی تنظیمات و وضعیت سخت‌افزارهای سیستم، کافی است فایل‌های موجود در این دایرکتوری را با استفاده از دستوراتی نظیر less، cat و... مشاهده کنیم. فایل‌های موجود در /proc می‌توانند اطلاعاتی در مورد CPU، RAM، پورت‌های I/O، IRQها و... را به ما بدهند. در این بخش به صحبت در مورد فایل مربوط به CPU و RAM می‌پردازیم.

مشاهده‌ی اطلاعات CPU

برای مشاهده‌ی اطلاعات در مورد CPU سیستم، می‌توانیم فایل /proc/cpuinfo را مشاهده کنیم. این فایل، اطلاعاتی نظیر سازنده‌ی CPU، مدل CPU، فرکانس پردازنده، تعداد هسته‌ها و... را درون خود دارد:

```
[root@localhost ~]# cat /proc/cpuinfo
```

```
[root@localhost ~]# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i3-3220 CPU @ 3.30GHz
stepping        : 9
microcode       : 0x12
cpu MHz         : 3300.059
cache size      : 3072 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx
 fxsr sse sse2 ss ht syscall nx rdtscp lm constant_tsc arch_perfmon nopl xtopology tsc_reliable nonst
op_tsc eagerfpu pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer xsav
e avx f16c hypervisor lahf_lm tpr_shadow vnmi ept vpid fsgsbase tsc_adjust smep arat
bogomips        : 6600.11
clflush size    : 64
cache_alignment : 64
address sizes   : 43 bits physical, 48 bits virtual
power management:
```

تصویر ۱۵ - مشاهده‌ی اطلاعات CPU با خواندن فایل /proc/cpuinfo

توجه کنید که ممکن است که در این فایل، چندین CPU مشاهده کنید. هر CPU موجود در این فایل، اشاره به یک هسته از CPU شما دارد.

مشاهده‌ی اطلاعات RAM

برای مشاهده‌ی اطلاعات RAM، کافی است فایل /proc/meminfo را مشاهده کنیم. در این فایل، اطلاعاتی نظیر کل حجم RAM، میزان RAM مصرف شده، میزان حافظه‌ی swap و... وجود دارد:

```
[root@localhost ~]# less /proc/meminfo
```

```
MemTotal:         995776 kB
MemFree:          711108 kB
MemAvailable:     691176 kB
Buffers:            2108 kB
Cached:            87536 kB
SwapCached:            0 kB
Active:            89012 kB
Inactive:          68452 kB
Active(anon):      68208 kB
Inactive(anon):     7372 kB
Active(file):      20804 kB
Inactive(file):    61080 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:       2097148 kB
SwapFree:        2097148 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:         68012 kB
Mapped:            22496 kB
Shmem:              7760 kB
Slab:              60916 kB
SReclaimable:      21032 kB
SUnreclaim:        39884 kB
KernelStack:        4176 kB
PageTables:         3712 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     2595036 kB
Committed_AS:     282476 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       179524 kB
VmallocChunk:   34359310332 kB
HardwareCorrupted:     0 kB
AnonHugePages:     12288 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       89984 kB
DirectMap2M:      958464 kB
```

تصویر ۱۶ - مشاهده‌ی اطلاعات RAM با خواندن فایل /proc/meminfo

دایرکتوری /sys

یکی دیگر از روش‌ها برای کار با سخت‌افزارها، دایرکتوری /sys می‌باشد. دایرکتوری /sys، دقیقا مانند /proc، یک دایرکتوری مجازی می‌باشد؛ یعنی این دایرکتوری به جای هارددیسک، بر روی RAM قرار می‌گیرد. این دایرکتوری توسط کرنل ایجاد می‌شود و از فایل‌سیستم sysfs استفاده می‌کند.

اطلاعات بسیار زیادی در مورد سخت‌افزارها در دایرکتوری /sys وجود دارد. این اطلاعات، میان دایرکتوری‌های متفاوت تقسیم شده‌اند. بررسی فایل‌های موجود در این دایرکتوری‌ها را به شما می‌سپاریم.

```
[root@localhost ~]# ls -l /sys/
total 0
drwxr-xr-x.   2 root root 0 May 12 22:40 block
drwxr-xr-x.  35 root root 0 May 12 22:40 bus
drwxr-xr-x.  55 root root 0 May 12 22:40 class
```

```
drwxr-xr-x.   4 root root 0 May 12 22:40 dev
drwxr-xr-x.  16 root root 0 May 12 22:40 devices
drwxr-xr-x.   6 root root 0 May 12 22:40 firmware
drwxr-xr-x.   7 root root 0 May 12 22:40 fs
drwxr-xr-x.   2 root root 0 May 12 22:40 hypervisor
drwxr-xr-x.  10 root root 0 May 12 22:40 kernel
drwxr-xr-x. 162 root root 0 May 12 22:40 module
drwxr-xr-x.   2 root root 0 May 12 22:40 power
```

دستورهای موجود برای کارکردن با سخت‌افزارها

لینوکس دستورهای متعددی برای بررسی سخت‌افزارهای موجود در سیستم در اختیار ما قرار می‌دهد. در این بخش با برخی از این دستورها آشنا می‌شویم.

مشاهده‌ی اطلاعات هارددیسک و... با دستور *lsblk*

دستور lsblk کلیه‌ی دستگاه‌های بلوکی موجود در سیستم (قبلا در مورد دیواس‌فایل‌های بلوکی صحبت کردیم) را به ما نشان می‌دهد. این دستور، برای مشاهده‌ی هارددیسک‌های موجود در سیستم بسیار مناسب می‌باشد. برای مثال:

```
[root@localhost ~]# lsblk
NAME            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda               8:0    0   20G  0 disk
├─sda1            8:1    0    1G  0 part /boot
└─sda2            8:2    0   19G  0 part
  ├─centos-root 253:0    0   17G  0 lvm  /
  └─centos-swap 253:1    0    2G  0 lvm  [SWAP]
sr0              11:0    1  942M  0 rom
```

sda نشان‌دهنده‌ی هارددیسک و sr0 نشان‌دهنده‌ی CDROM نصب شده روی سیستم می‌باشد. با استفاده از آپشن S-، می‌توانیم فقط اطلاعات مربوط به دستگاه‌های SCSI نصب شده بر روی سیستم را مشاهده کنیم (مثل هارددیسک‌ها، سی‌دی‌رام‌ها و...):

```
[root@localhost ~]# lsblk -S
NAME HCTL       TYPE VENDOR   MODEL             REV  TRAN
sda  2:0:0:0    disk VMware,  VMware Virtual S  1.0   spi
sr0  1:0:0:0    rom  NECVMWar VMware IDE CDR10  1.00  ata
```

برای کسب اطلاعات بیشتر در مورد آپشن‌های این دستور، بهتر است به manpage آن مراجعه کنید.

مشاهده‌ی اطلاعات دستگاه‌های USB با استفاده از *lsusb*

برای مشاهده‌ی دستگاه‌های USB متصل شده به سیستم، از دستور lsusb استفاده می‌کنیم. توجه کنید که این دستور به صورت پیش‌فرض روی توزیع CentOS نصب نیست و باید آن را از طریق yum روی سیستم نصب کنیم. به صورت زیر:

```
[root@localhost ~]# yum install usbutils
...
Installed:
  usbutils.x86_64 0:007-5.el7

Complete!
```

```
[root@localhost ~]# lsusb
Bus 001 Device 002: ID 1307:0165 Transcend Information, Inc. 2GB/4GB/8GB Flash Drive
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

همانطور که می‌بینید، با استفاده از دستور lsusb، توانستیم کلیه دستگاه‌هایی که از طریق USB به سیستم متصل شده‌اند را مشاهده کنیم.

مشاهده‌ی اطلاعات دستگاه‌های PCI با استفاده از *lspci*

همانطور که گفتیم، امروزه اکثر سخت‌افزارها از طریق استاندارد PCI به سیستم متصل می‌شوند. برای مشاهده‌ی کلیه‌ی سخت‌افزارهایی که از طریق استاندارد PCI به سیسم متصل شده‌اند، از دستور lspci استفاده می‌کنیم. توجه کنید که این دستور به صورت پیش‌فرض روی سیستم نصب نمی‌باشد و باید آن را از طریق yum روی سیستم نصب کنیم. به صورت زیر:

```
[root@localhost ~]# yum install pciutils
...
Installed:
  pciutils.x86_64 0:3.5.1-3.el7

Complete!
[root@localhost ~]# lspci
00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host bridge (rev 01)
00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP bridge (rev 01)
00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08)
00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:07.7 System peripheral: VMware Virtual Machine Communication Interface (rev 10)
00:0f.0 VGA compatible controller: VMware SVGA II Adapter
...
```

دستور lspci، آپشن‌های متفاوتی برای جستجو، فیلترکردن و نمایش سخت‌افزارهای PCI دارد که توضیح آنها از حوصله‌ی ما خارج است 😒. برای اطلاعات بیشتر، می‌توانید به manpage این دستور مراجعه کنید.

ماژول‌های کرنل

کرنل لینوکس برای ارتباط با سخت‌افزارها، احتیاج به یک سری درایور دارد، اما کامپایل کردن درایور برای همه‌ی سخت‌افزارهای موجود، باعث می‌شود که حجم کرنل، بسیار بالا رود. برای جلوگیری از چنین مشکلی، لینوکس از ماژول‌های کرنل استفاده می‌کند. بدین صورت، سیستم می‌تواند فقط درایورهای مورد نیاز برای سیستم شما را درون کرنل قرار دهد.

مازول‌های کرنل به صورت پیش‌فرض در دایرکتوری /lib/modules قرار دارند. ماژول‌ها می‌توانند به صورت سورس‌کد یا باینری‌های از قبل کامپایل شده باشند. کرنل لینوکس موجود در هر توزیع، به همراه خود برخی ماژول‌ها را به صورت پیش‌فرض، دارد. در این بخش با برخی از دستورهای مربوط به ماژول‌های کرنل آشنا می‌شویم.

مشاهده‌ی ماژول‌های لود شده درون کرنل با lsmod

برای مشاهده‌ی ماژول‌های لود شده درون کرنل، از دستور lsmod استفاده می‌کنیم:

```
[root@localhost ~]# lsmod
Module                  Size  Used by
uas                    22445  0
usb_storage            66718  1 uas
ip6t_rpfilter          12595  1
ip6t_REJECT            12625  2
nf_reject_ipv6         13717  1 ip6t_REJECT
ipt_REJECT             12541  2
nf_reject_ipv4         13373  1 ipt_REJECT
xt_conntrack           12760  11
ebtable_nat            12807  1
ip6table_nat           12864  1
nf_conntrack_ipv6      18935  7
nf_defrag_ipv6         35104  1 nf_conntrack_ipv6
nf_nat_ipv6            14131  1 ip6table_nat
ip6table_mangle        12700  1
ip6table_security      12710  1
ip6table_raw           12683  1
iptable_nat            12875  1
nf_conntrack_ipv4      15053  6
nf_defrag_ipv4         12729  1 nf_conntrack_ipv4
nf_nat_ipv4            14115  1 iptable_nat
nf_nat                 26583  2 nf_nat_ipv4,nf_nat_ipv6
...
```

همانطور که می‌بینید، این دستور لیستی از ماژول‌های لود شده درون کرنل را به ما نشان می‌دهد. ستون Module، نام ماژول را به ما نشان می‌دهد، ستون Size نشان‌دهنده‌ی سایز خود ماژول می‌باشد و ستون Used by تعداد برنامه‌های در حال اجرایی که از این ماژول استفاده می‌کنند را به ما نشان می‌دهد و پس از آن، لیستی از ماژول‌هایی که به این ماژول وابسته هستند نمایش داده می‌شود؛ البته این لیست در برخی از اوقات، ناکامل می‌باشد.

برای مشاهده‌ی ماژول‌هایی که توسط یک یا چند برنامه‌ی در حال اجرا استفاده شده‌اند، باید به سراغ grep برویم:

```
[root@localhost ~]# lsmod | grep -Ev  "\s0"
Module                  Size  Used by
binfmt_misc            17468  1
vfat                   17461  1
fat                    65950  1 vfat
usb_storage            66718  2 uas
...
```

همانطور که می‌بینید ما با فیلتر خروجی lsmod با grep، فقط ماژول‌هایی که توسط یک یا چند برنامه‌ی دیگر استفاده شده بودند را نمایش دادیم. ما در اینجا به دنبال خط‌هایی هستیم که در ستون Used by آن، عدد 0 وجود **نداشته** باشد. برای این کار، خروجی lsmod را درون grep پایپ می‌کنیم و الگوی ذکر شده را به آن می‌دهیم. دلیل استفاده از آپشن Ev-، این است که ما می‌خواهیم از Extended grep استفاده کنیم (E-) و همچنین می‌خواهیم خطوطی که الگوی مشخص شده را **ندارند** به ما برگردانده شوند (v-). \s در الگوی ما، به معنای فاصله‌ی خالی یا whitespace می‌باشد و 0 هم مشخص کننده‌ی عدد صفر می‌باشد. ما با این الگو، به

grep می‌گوییم که خطوطی که در آن فاصله‌ی خالی و بلافاصله پس از آن عدد صفر وجود دارد را به ما نشان **ندهد** (به دلیل استفاده از v-). از آنجایی که اعداد موجود در ستون Size، هرگز با صفر شروع نمی‌شوند، انتخاب این الگو، کار منطقی می‌باشد.

اگر هنوز این دستور و این الگو برای شما گُنگ است، می‌توانید به جزوه‌ی جلسه‌ی دوم مراجعه کنید. ما در آن جزوه، به صورت مفصل در مورد دستور grep و همچنین regexها صحبت کردیم.

به دست آوردن اطلاعات در مورد یک ماژول با modinfo

برا به دست آوردن اطلاعات در مورد یک ماژول، از دستور modinfo استفاده می‌کنیم. فرض کنید می‌خواهیم اطلاعاتی نظیر عملکرد، نسخه و... در مورد ماژول vfat به دست آوریم:

```
[root@localhost ~]# modinfo vfat
filename:       /lib/modules/3.10.0-1062.el7.x86_64/kernel/fs/fat/vfat.ko.xz
author:         Gordon Chaffee
description:    VFAT filesystem support
license:        GPL
alias:          fs-vfat
retpoline:      Y
rhelversion:    7.7
srcversion:     A3254796A3CD9815ABDDC94
depends:        fat
intree:         Y
vermagic:       3.10.0-1062.el7.x86_64 SMP mod_unload modversions
signer:         CentOS Linux kernel signing key
sig_key:        51:08:4E:41:88:03:02:BE:5C:B0:74:AC:0D:A3:FE:10:23:3B:7F:1C
sig_hashalgo:   sha256
```

همانطور که می‌بینید، با استفاده از دستور modinfo، اطلاعات به درد بخوری در مورد این ماژول به دست آوردیم. مثلا با نگاه به ردیف description، می‌بینیم که این ماژول به سیستم کمک می‌کند که فایل‌سیستم vfat را بخواند.

اضافه و حذف کردن ماژول به کرنل با استفاده از modprobe

گاهی اوقات ممکن است مجبور باشیم برای اضافه کردن یک قابلیت به لینوکس، ماژول آن را به کرنل اضافه کنیم. برای بارگذاری یک ماژول درون کرنل، از modprobe استفاده می‌کنیم. برای مثال ممکن است بخواهیم بین دو سرور لینوکس، یا یک سرور لینوکس و یک روتر، تونل GRE بزنیم. برای انجام این کار، باید ابتدا ماژول GRE را درون کرنل بارگذاری کنیم:

```
[root@localhost ~]# modprobe ip_gre
```

اگر بارگذاری ماژول در کرنل به صورت صحیح صورت گیرد، modprobe هیچ چیزی در خروجی به شما نشان نمی‌دهد، ولی سیستم شما الان قابلیت ایجاد یک تونل GRE را دارد. صحبت در مورد تونل‌های GRE از حوصله‌ی ما خارج است، پس به آن نمی‌پردازیم؛ ولی فکر نکنید که بلد نیستیم 😉.

برای حذف کردن یا unload کردن یک ماژول، از آپشن r- دستور modprobe استفاده می‌کنیم:

```
[root@localhost ~]# modprobe -r ip-gre
```

اگر این دستور به صورت صحیح اجرا شود، چیزی در خروجی مشاهده نخواهید کرد، اما ماژول مشخص شده از کرنل حذف می‌شود.