

« به نام خدای رنگین کمان »

مستندات فاز اول پروژه رایانش ابری

اعضای گروه : بهاره کیوانی - یاسمین زارعی

پرسش ها

(۱) چه قابلیت‌هایی از kernel امکان containerization را به ما می‌دهد؟ توضیح دهید.

containerization شکلی از مجازی‌سازی است که در آن برنامه‌ها در فضاهای کاربری مجزا به نام container اجرا می‌شوند ولی در عین حال از سیستم عامل مشترکی استفاده می‌کنند. container ها محیط‌های محاسباتی کاملاً بسته بندی شده و قابل حمل هستند.

kernel با ایجاد کردن name-space های مجزا به ازای هر container، محیط‌های مجزا و ایزوله را فراهم می‌کند. name-space یکی از ویژگی‌های Linux kernel هستند که منابع هسته را به گونه ای پارتیشن بندی می‌کنند که یک مجموعه از فرایندها یک مجموعه از منابع را می‌بینند در حالی که مجموعه دیگری از فرایندها مجموعه متفاوتی از منابع را می‌بینند. این ویژگی با داشتن name-space یکسان برای مجموعه‌ای از منابع و فرایندها کار می‌کند، اما این name-space ها به منابع مجزا اشاره می‌کنند. منابع ممکن است در چندین فضا وجود داشته باشد. نمونه‌هایی از این منابع عبارتند از شناسه‌های فرایند (PID)، نام میزبان (host name)، شناسه‌های کاربر (UID)، نام فایل‌ها، برخی از نام‌های مرتبط با دسترسی به شبکه و ارتباطات بین فرآیندی (IPC).

یک سیستم لینوکسی با یک name-space واحد از هر نوع شروع می‌شود که توسط همه فرایندها استفاده می‌شود. فرایندها می‌توانند name-space جدید ایجاد کنند و یا به name-space های دیگر بپیوندند. cgroup یا control group، مکانیزم Linux kernel برای محدود کردن و اندازه گیری کل منابع در حال استفاده توسط گروهی از فرایندها در حال اجرا روی سیستم است. برای مثال با استفاده از cgroup می‌توانید منابع سیستم‌های لینوکسی خود مانند CPU، RAM یا IO را کنترل کنید.

در نهایت در پاسخ به پرسش می‌توان گفت kernel با استفاده از قابلیت‌های name-space ها و cgroup ها کانسپت containerization را برای ما فراهم می‌کند.

(۲) چه name-space هایی وجود دارد؟ در مورد هر یک توضیح بدهید.

*** mount :**

برای اینکه فرایندهای داخل یک container به فایل سیستم میزبان دسترسی نداشته باشد، باید بین آن‌ها مرزی ایجاد شود. با استفاده از فضا نام mount می‌توان این قابلیت را برای فرایندها فراهم کرد. mount، نقاط نصب یا mount point ها را مدیریت و ایزوله سازی می‌کند. پس از ایجاد mount ها از name-space مربوط به mount فعلی در name-space جدید کپی می‌شوند، اما نقاط mount ایجاد شده پس از آن بین name-space ها منتشر نمی‌شوند (البته با استفاده از زیردرخت‌های مشترک، امکان انتشار mount point ها بین name-space وجود دارد که در ادامه به آن اشاره خواهیم کرد).

با دو روش clone و یا unshare با پرچم CLONE_NEWNS می‌توان name-space mount جدید ایجاد کرد.

① clone : در صورتی که از این طریق mount name-space جدید را ایجاد کنیم، لیست mount های آن یک کپی از لیست mount های پدر خودش خواهد بود.

② unshare : اگر از این unshare برای ایجاد mount name-space جدید استفاده کنیم، لیست mount های آن یک کپی از لیست mount های آخرین mount name-space در لیست caller خواهد بود.

mount ها به طور پیش فرض به علت ویژگی shared subtree در kernel قابلیت پخش دارند. ای موضوع باعث می شود تا هر mount point نوع انتشار خاص خودش را داشته باشد. در این صورت تعیین می شود که mount های جدید تحت یک مسیر مشخص به سایر mount point ها منتشر می شوند یا خیر.

یک peer group به عنوان گروهی از mount ها تعریف می شود که رویدادها را به یکدیگر منتشر می کنند. حالت mount تعیین می کند که آیا عضوی از یک peer group می تواند رویداد را دریافت کند یا خیر. در حال حاضر پنج حالت mount وجود دارد:

① shared :

نقطه mount به peer group تعلق دارد. هر تغییری ایجاد شود، در کل mount های حاضر در peer group منتشر خواهد شد.

② slave (انتشار یک طرفه):

انتشار یک طرفه نقطه mount اصلی (master) رویدادها را به یک slave منتشر می کند، اما master هیچ عملی را که slave انجام می دهد، نخواهد دید.

③ shared و slave :

نشان می دهد که نقطه mount یک master دارد، اما peer group خود را نیز دارد. master از تغییرات در یک نقطه نصب مطلع نخواهد شد، اما هر یک از اعضای peer group پایین دستی مطلع خواهد شد.

④ private :

هیچ رویداد انتشاری را دریافت یا ارسال نمی کند.

⑤ unbindable :

هیچ رویداد انتشاری را دریافت یا ارسال نمی کند و نمی توان آن را متصل کرد.

* PID یا process ID :

PID name-space فضای شماره شناسه فرآیند را ایزوله می کند، به این معنی که فرآیندها در فضاهای نام مختلف می توانند PID یکسانی داشته باشند. فضاهای نام PID به container ها اجزای می دهد تا عملکردهایی مانند تعلیق و یا از سرگیری مجموعه فرآیندها در container و انتقال container به میزبان جدید را ارائه دهند در حالی که فرآیندهای داخل container همان PID ها را حفظ می کنند. PID ها در یک PID name-space جدید از 1 شروع می شوند، تا حدودی شبیه به یک سیستم مستقل و با فراخوانی توابعی چون vfork ، fork و یا clone فرآیندهایی با PID هایی تولید می کند که در name-space منحصر به فرد هستند.

* network :

network name-space جداسازی منابع سیستم مرتبط با شبکه را فراهم می کند. برای مثال دستگاه های شبکه، پشته های پروتکل IPv4 و IPv6، جداول مسیریابی IP، قوانین firewall، شماره پورت (سوکت ها) و غیره. علاوه بر این name-space شبکه، UNIX domain abstract socket

namespace را ایزوله می کند. فضای نام انتزاعی یک ویژگی خاص لینوکس است که به ما اجازه می دهد یک سوکت دامنه یونیکس را به یک نام متصل کنیم بدون اینکه آن نام در سیستم فایل ایجاد شود.

یک دستگاه شبکه فیزیکی می تواند دقیقاً در یک network name-space زندگی کند. هنگامی که یک network name-space آزاد می شود (یعنی زمانی که آخرین فرآیند در name-space پایان می یابد)، دستگاه های شبکه فیزیکی آن به فضای نام شبکه اولیه (نه به والد فرآیند) برمی گردند. برای ایجاد کردن یک network name-space جدید به صورت زیر عمل می کنیم :

```
$ sudo unshare -net /bin/bash
```

برای مشاهده جزئیات name-space ایجاد شده از دستور زیر استفاده می کنیم :

```
$ lsns -t net
```

جزئیات بیشتر در رابطه با پیکربندی name-space شبکه ایجاد شده را پرسش سوم به همراه اسکرین شات مشاهده خواهید کرد.

* inter process communication یا IPC :

در سیستم عامل لینوکس دو پرده می توانند از طریق حافظه ی مشترک یا صف پیام مشترک (message queue) با یکدیگر پیام تبادل کنند. برای اینکار این دو پرده باید عضو یک فضای نام IPC یکسان باشند. از طرفی به طور پیش فرض ما انتظار داریم که پرده های موجود در container های مختلف نتوانند به حافظه ی مشترک یکدیگر دسترسی داشته باشند برای همین برای آن ها name-space های IPC مجزا ایجاد می کنیم. برای مثال به صورت پیش فرض کانتینرها در داکر دارای فضای نام ipc اختصاصی هستند. برای آزمایش IPC name-space ابتدا یک حافظه ی مشترک ایجاد می کنیم سپس با دستور ipcs فهرست IPC های موجود را بررسی می کنیم:

```
$ ipcmk -M 1000  
$ ipcs
```

سپس با استفاده از unshare یک فضای نام IPC ایجاد می کنیم و با دستور ipcs فهرست IPC هارا بررسی می کنیم.

```
$ sudo unshare --ipc /bin/bash
```

تصویر مربوط به خروجی این کامند ها در kernel در صفحه بعدی می باشد. (تصویر ۱ و تصویر ۲)

```
bahareh@spark: ~  
bahareh@spark:~$ ipcs  
  
----- Message Queues -----  
key          msqid      owner      perms      used-bytes   messages  
  
----- Shared Memory Segments -----  
key          shmid      owner      perms      bytes       nattch     status  
0x00000000   5          bahareh    600        1048576     2          dest  
0x00000000   28         bahareh    600        163840      2          dest  
0x00000000   29         bahareh    600        163840      2          dest  
0x00000000   30         bahareh    600        20480       2          dest  
0x00000000   31         bahareh    600        20480       2          dest  
0x00000000   34         bahareh    600        106496      2          dest  
0x00000000   35         bahareh    600        106496      2          dest  
0x00000000   36         bahareh    600        356352      2          dest  
0x00000000   37         bahareh    600        356352      2          dest  
0x00000000   40         bahareh    600        86016       2          dest  
0x00000000   41         bahareh    600        86016       2          dest  
0x6d860f5a   42         bahareh    644        1000        0  
  
----- Semaphore Arrays -----  
key          semid      owner      perms      nsens  
  
bahareh@spark:~$
```

تصویر ۱

```
root@spark: /home/bahareh  
bahareh@spark:~$ sudo unshare --ipc /bin/bash  
root@spark:/home/bahareh# ipcs  
  
----- Message Queues -----  
key          msqid      owner      perms      used-bytes   messages  
  
----- Shared Memory Segments -----  
key          shmid      owner      perms      bytes       nattch     status  
  
----- Semaphore Arrays -----  
key          semid      owner      perms      nsens  
  
root@spark:/home/bahareh#
```

تصویر ۲

* Unix time-sharing system یا UTS :

هر ماشین دارای یک شناسه است که به آن hostname گفته می‌شود. با قرار دادن یک پرده در یک uts name-space اختصاصی، پرده صاحب hostname و domain name اختصاصی خواهد شد و می‌توان مقادیر دلخواهی برای hostname و domain name برای آن فرآیند تنظیم کرد که این تغییر تأثیری در مقادیر مربوطه در ماشین میزبان نخواهد داشت. تکنولوژی‌های containerization (مانند Docker) به هر container یک شناسه‌ی تصادفی اختصاص می‌دهند که به صورت پیش‌فرض همین شناسه به عنوان hostname در آن container استفاده می‌شود. برای بررسی این موضوع با استفاده از دستور زیر می‌توان یک کانتینر ایجاد کرد و مقدار hostname داخل کانتینر را مشاهده نمود.

```
$ hostname
```

با استفاده از unshare می‌توانیم یک uts name-space جدید ایجاد کنیم. برای اینکار باید unshare را با دسترسی کاربر root اجرا کنیم. در زمان فراخوانی unshare نوع name-space و نام برنامه‌ای که می‌خواهیم اجرا کنیم را وارد می‌کنیم. در صورتی که نام برنامه وارد نشود به صورت پیش‌فرض از مقدار {SHELL} استفاده خواهد شد.

```
$ sudo unshare --uts /bin/bash
```

این کار باعث شده که bash داخل یک پرده‌ی جدید که فضا نام uts مختص به خودش را دارد اجرا شود. هر برنامه‌ای که در این shell اجرا شود نیز این uts name-space را به ارث خواهد برد و برای اینکه hostname متفاوت باشد، باید آن را به طور دستی بعد از ایجاد کردن uts name-space تغییر بدهیم.

* user ID یا user :

uts name-space شناسه‌ها و ویژگی‌های مرتبط با امنیت (به طور ویژه شناسه‌های کاربر و شناسه‌های گروه)، دایرکتوری root و کلیدها را جدا می‌کنند. شناسه‌های کاربر و گروه یک فرآیند می‌تواند در داخل و خارج از uts name-space متفاوت باشند. به طور خاص، یک فرآیند می‌تواند یک شناسه کاربر عادی غیرمجاز در خارج از فضای نام کاربری داشته باشد در حالی که در همان زمان شناسه کاربری 0 در داخل uts name-space داشته باشد. به عبارت دیگر، فرآیند دارای امتیازات کامل (privileges) برای عملیات داخل uts name-space است، اما در خارج از آن uts name-space فاقد امتیاز (unprivileged) است.

مهم‌ترین مزیتی که این سطح از ایزوله کردن فراهم می‌کند این است که در داخل container می‌توان کاربر یک root داشت که خارج از container یک کاربر غیر root باشد. یکی از چالش‌های امنیتی container ها دور زدن مکانیزم‌های امنیتی و دسترسی به میزبان است. از منظر امنیت این یک ویژگی کلیدی است که اجازه می‌دهد فرآیندهای داخل container که وابسته به کاربر root isjkn اجرا شود ولی حتی اگر به هر نحوی فرآیند بتواند از محیط ایزوله خارج شود سطح دسترسی آن در میزبان کم‌تر از کاربر root می‌باشد.

برای ایجاد یک فضا نام user از کامند زیر استفاده می‌کنیم. توجه کنید که در user ساخته شده کاربر دارای شناسه‌ی nobody است و همین‌طور هیچ capability نیز به آن تخصیص نیافته است.

```
$ unshare --user bash
```

تصویر مربوط به خروجی کامند بالا در kernel در ادامه آورده شده است. (تصویر ۳)

```
nobody@spark: ~  
bahareh@spark:~$ unshare --user bash  
nobody@spark:~$
```

تصویر ۳

* time :

با استفاده از این name-space فرآیند ها می توانند مقادیر دلخواهی برای CLOCK_MONOTONIC و CLOCK_BOOTTIME داشته باشد. بنابراین زمان جاری و مقدار uptime میزبان می تواند با پرده‌ی داخل این name-space متفاوت باشد. دستور uptime ، وضعیت فعلی سیستم در کل مدت زمانی که سیستم در حال اجرا بوده است ، تعداد کاربران (تعداد کاربران فعلی وارد شده) و بار سیستم در ۱.۵ تا ۱۵ دقیقه را نمایش می دهد. می توان برای محیط های متفاوت timezone های متفاوتی در نظر گرفت و در این صورت می تواند خروجی uptime در داخل یک container و میزبان دریافت کرد.

* control groups یا cgroups :

در سیستم عامل منابعی مانند پردازنده (CPU)، حافظه اصلی (RAM)، پهنای بند شبکه و ... به صورت مشترک بین پرده‌ها استفاده می شود. در این شرایط یک پردازش می توان به صورت ناعادلانه بخش زیادی از این منابع را تصرف کرد. برای محدود کردن میزان مصرف این منابع توسط پرده‌ها در سیستم عامل لینوکس قابلیت به نام گروه های کنترل یا به اختصار cgroups وجود دارد. با بهره برداری از cgroups مدیر سیستم می تواند به صورت دقیق و بهینه بر روی ویژگی ها و محدود کردن (Resource limiting)، اولویت بندی (Prioritization)، مدیریت (Control) و حساسی (Accounting) منابع (زیر سیستم ها) کنترل داشته باشید.

برای مشاهده cgroup های موجود از دستور زیر استفاده می کنیم :

```
$ cat /proc/cgroups
```

برای ایجاد کردن یک cgroup جدید وارد دایرکتوری sys/fs/cgroup می شویم و در kernel با استفاده از دستور mkdir یک گروه جدید ایجاد میکنیم.

```
$ mkdir new_cgroup_name
```

برای دیدن فرآیند های داخل این گروه تازه ایجاد شده سپس به صورت زیر عمل میکنیم.

```
$ cat new_cgroup_name/cgroup.procs
```

همینطور می توانیم روی فرآیند های داخل یک گروه محدودیت اعمال کنیم. برای مثال اینکه نهایتاً ۳ فرآیند می تواند در این گروه باشد یا فقط یک درصد مشخصی از CPU به فرآیند های گروه خاصی تخصیص داده بشود.

۳) Network name-space علاوه بر isolation در cloud چه کاربردی دارد؟ یک Network name-space بسازید و آن را پینگ کنید. سپس Network name-space دومی ساخته و ارتباط آن هارا با هم برقرار کنید به نحوی که از هر name-space بتوان دیگری را پینگ کرد.

علاوه بر ایزوله سازی ، name-space های شبکه برای مسیریابی و فوروارد مجازی (Virtual routing and forwarding) نیز استفاده می شوند. مسیریابی و فوروارد مجازی یک فناوری IP است که به چندین نمونه از یک جدول مسیریابی اجازه می دهد تا همزمان در یک روتر وجود داشته باشند که مبنای کار ابزار OpenStack می باشد.

```
bahareh@spark: ~  
bahareh@spark:~$ sudo ip link add veth0 type veth peer name veth1  
bahareh@spark:~$ sudo ip netns add namespace0  
bahareh@spark:~$ sudo ip link set veth0 netns namespace0  
bahareh@spark:~$ sudo ip netns exec namespace0 ip addr add 10.0.1.1/24 dev veth0  
Command "10.0.1.1/24" is unknown, try "ip address help".  
bahareh@spark:~$ sudo ip netns exec namespace0 ip addr add 10.0.1.1/24 dev veth0  
bahareh@spark:~$ sudo ip netns exec namespace0 ip link set veth0 up  
bahareh@spark:~$  
bahareh@spark:~$ sudo ip netns add namespace1  
bahareh@spark:~$ sudo ip link set veth1 netns namespace1  
bahareh@spark:~$ sudo ip netns exec namespace1 ip addr add 10.0.2.1/24 dev veth1  
bahareh@spark:~$ sudo ip netns exec namespace1 ip link set veth1 up  
bahareh@spark:~$  
bahareh@spark:~$ sudo ip netns exec namespace0 ip route add 10.0.2.0/24 via 10.0.2.1  
Error: Nexthop has invalid gateway.  
bahareh@spark:~$ sudo ip netns exec namespace1 ip route add 10.0.1.0/24 via 10.0.1.1  
Error: Nexthop has invalid gateway.  
bahareh@spark:~$ sudo ip netns exec namespace0 ip route add 10.0.2.0/24 via 10.0.1.1  
bahareh@spark:~$ sudo ip netns exec namespace1 ip route add 10.0.1.0/24 via 10.0.2.1  
bahareh@spark:~$  
bahareh@spark:~$ sudo ip netns exec namespace0 ping -c 4 10.0.2.1  
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data:  
64 bytes from 10.0.2.1: icmp_seq=1 ttl=64 time=0.054 ms  
64 bytes from 10.0.2.1: icmp_seq=2 ttl=64 time=0.094 ms  
64 bytes from 10.0.2.1: icmp_seq=3 ttl=64 time=0.092 ms  
64 bytes from 10.0.2.1: icmp_seq=4 ttl=64 time=0.087 ms  
  
--- 10.0.2.1 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3059ms  
rtt min/avg/max/mdev = 0.054/0.081/0.094/0.016 ms  
bahareh@spark:~$  
bahareh@spark:~$ sudo ip netns exec namespace1 ping -c 4 10.0.1.1  
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data:  
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.034 ms  
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.080 ms  
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.090 ms  
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.083 ms  
  
--- 10.0.1.1 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3055ms  
rtt min/avg/max/mdev = 0.034/0.071/0.090/0.022 ms  
bahareh@spark:~$
```

در ادامه هر قسمت از کامند هایی که در بالا مشاهده می کنید ، روی شکل توضیح داده شده اند.

```

bahareh@spark:~$ sudo ip link add veth0 type veth peer name veth1
bahareh@spark:~$ sudo ip netns add namespace0
bahareh@spark:~$ sudo ip link set veth0 netns namespace0
Current: namespace0: veth0: ip netns exec namespace0 ip netns exec namespace0 ip link set veth0 up
bahareh@spark:~$ sudo ip netns exec namespace0 ip link set veth0 up
bahareh@spark:~$

bahareh@spark:~$ sudo ip netns add namespace1
bahareh@spark:~$ sudo ip link set veth1 netns namespace1
bahareh@spark:~$ sudo ip netns exec namespace1 ip addr add 10.0.2.1/24 dev veth1
bahareh@spark:~$ sudo ip netns exec namespace1 ip link set veth1 up
bahareh@spark:~$

bahareh@spark:~$ sudo ip netns exec namespace0 ip route add 10.0.2.0/24 via 10.0.2.1
Error: Nexthop has invalid gateway.
bahareh@spark:~$ sudo ip netns exec namespace1 ip route add 10.0.1.0/24 via 10.0.1.1
Error: Nexthop has invalid gateway.
bahareh@spark:~$ sudo ip netns exec namespace0 ip route add 10.0.2.0/24 via 10.0.1.1
bahareh@spark:~$ sudo ip netns exec namespace1 ip route add 10.0.1.0/24 via 10.0.2.1
bahareh@spark:~$

bahareh@spark:~$ sudo ip netns exec namespace0 ping -c 4 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data:
64 bytes from 10.0.2.1: icmp_seq=1 ttl=64 time=0.054 ms
64 bytes from 10.0.2.1: icmp_seq=2 ttl=64 time=0.094 ms
64 bytes from 10.0.2.1: icmp_seq=3 ttl=64 time=0.092 ms
64 bytes from 10.0.2.1: icmp_seq=4 ttl=64 time=0.087 ms

--- 10.0.2.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3059ms
rtt min/avg/max/mdev = 0.054/0.081/0.094/0.016 ms
bahareh@spark:~$

bahareh@spark:~$ sudo ip netns exec namespace1 ping -c 4 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data:
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.034 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.080 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.090 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.083 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3055ms
rtt min/avg/max/mdev = 0.034/0.071/0.090/0.022 ms
bahareh@spark:~$

```

ساخت namespace اول و ست کردن تنظیمات پیکر بندی آن

ساخت namespace دوم و ست کردن تنظیمات پیکر بندی آن

در این قسمت جا به جا مسیر روتر هارو آدرس داده شده به همین دلیل کرنل ارور داده که gateway معتبر نیست. قسمت بعدی مسیر درست گذاشته شده و مشکل حل میشه.

مدیریت مسیر روتر های مجاری ایجاد شده در قسمت های قبلی

فضا ایجاد شده اولیه شبکه مربوط به فضای ایجاد شده دومی را به درستی پینگ میکند.

فضا ایجاد شده دوم، شبکه مربوط به فضای ایجاد شده اول را به درستی پینگ میکند.

۴) فیچر cgroup در cloud چه کاربرد هایی می تواند داشته باشد؟

یکی از کاربردهای این فیچر کنترل و مدیریت منابع در سیستم های ابری است. برای مثال، با استفاده از Cgroup می توانیم منابع پردازشی، حافظه و شبکه را برای پروسه های مختلف در محیط ابری مدیریت کنیم. با این کار، می توانیم بهبود عملکرد سیستم و کاهش زمان انتظار برای کاربران را تضمین کنیم. در یک محیط ابری، می توانیم برای هر سرویس یا برنامه ای که در آن محیط اجرا می شود، یک Cgroup جداگانه ایجاد کنیم و منابع مورد نیاز آن را به آن اختصاص دهیم. این کار می تواند کاهش هزینه های مربوط به سخت افزار و بهبود عملکرد و پایداری سیستم را در محیط ابری تضمین کند.

Cgroup در کاربردهای مختلف می تواند به شکل زیر مفید باشد:

1. **مدیریت منابع سیستم:** با استفاده از Cgroup، می توانید میزان منابع سیستمی مانند CPU، حافظه، شبکه و I/O را به صورت جداگانه برای هر پروسه و برنامه مدیریت کنید. این به شما اجازه می دهد تا منابع سیستم را بهینه تر به صورت دقیق تر و به شکل مناسب به برنامه ها و پروسه ها تخصیص دهید.
2. **محدود کردن منابع:** با استفاده از Cgroup، می توانید میزان منابع سیستم مانند CPU، حافظه و I/O را برای هر پروسه و برنامه محدود کنید. این می تواند به شما کمک کند تا پروسه هایی که بیش از حد منابع را مصرف می کنند را محدود کنید و از تأثیرات منفی آنها بر روی عملکرد سیستم جلوگیری کنید.
3. **محافظت از منابع:** با استفاده از Cgroup، می توانید میزان منابع سیستم مانند CPU، حافظه و I/O را برای پروسه ها و برنامه ها محدود کنید تا جلوی سوء استفاده از منابع را بگیرید و از تأثیرات منفی آنها بر روی عملکرد سیستم جلوگیری کنید.

4. محدود کردن دسترسی: با استفاده از Cgroup، می‌توانید دسترسی پروسه‌ها و برنامه‌ها به منابع سیستمی مختلف را محدود کنید. این می‌تواند به شما کمک کند تا به اطلاعات حساس دسترسی ناآوانمردانه داده نشود و از تأثیرات منفی آنها بر روی عملکرد سیستم جلوگیری کنید.

۵) نحوه ایجاد cgroup را توضیح دهید و یکی درست کنید.

Cgroup یا Control Group، یک ویژگی در سیستم عامل لینوکس است که به کاربران امکان مدیریت منابع سیستم را در سطح پروسه‌ها و گروه پروسه‌ها می‌دهد. با استفاده از Cgroup می‌توانید منابع مانند پردازش‌ها، حافظه، شبکه، دستگاه‌های ورودی و خروجی، اسکچرهای صوتی و تصویری و ... را محدود کنید.

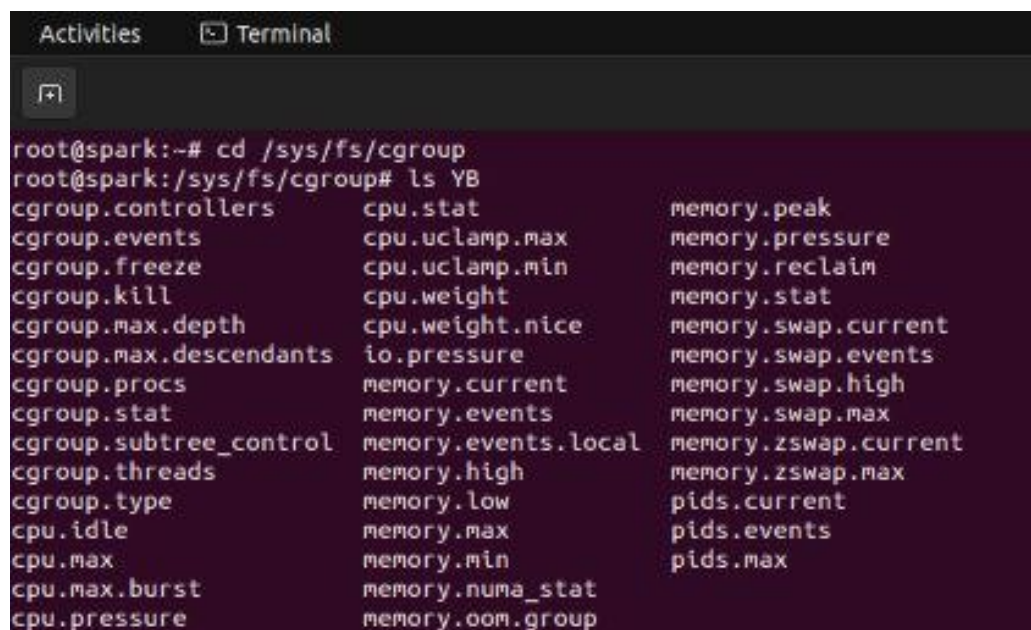
برای ایجاد یک Cgroup در لینوکس، می‌توانید از دستورات ترمینال استفاده کنید. این دستورات به شما اجازه می‌دهند تا Cgroup جدیدی ایجاد کنید و آن را به یک پروسه یا گروه پروسه‌ها اختصاص دهید.

برای ایجاد یک Cgroup جدید، از دستور زیر استفاده کنید:

```
sudo mkdir /sys/fs/cgroup/{ }/{ نام_منبع_نوع_Cgroup }
```

در این دستور، نوع منبع می‌تواند پردازش‌ها، حافظه، شبکه، دستگاه‌های ورودی و خروجی و ... باشد. به عنوان مثال، برای ایجاد یک Cgroup برای محدود کردن پردازش‌ها، می‌توانید از دستور زیر استفاده کنید:

```
sudo mkdir /sys/fs/cgroup/cpu/YB
```



```
Activities Terminal
root@spark:~# cd /sys/fs/cgroup
root@spark:/sys/fs/cgroup# ls YB
cgroup.controllers      cpu.stat                memory.peak
cgroup.events           cpu.uclamp.max          memory.pressure
cgroup.freeze           cpu.uclamp.min          memory.reclaim
cgroup.kill             cpu.weight              memory.stat
cgroup.max.depth        cpu.weight.nice         memory.swap.current
cgroup.max.descendants   io.pressure             memory.swap.events
cgroup.procs            memory.current          memory.swap.high
cgroup.stat             memory.events           memory.swap.max
cgroup.subtree_control  memory.events.local     memory.zswap.current
cgroup.threads          memory.high             memory.zswap.max
cgroup.type             memory.low              pids.current
cpu.idle                memory.max              pids.events
cpu.max                 memory.min              pids.max
cpu.max.burst           memory.numa_stat
cpu.pressure            memory.oom.group
```

۶) یک cgroup جدید ساخته و در آن میزان مصرف cpu را محدود کنید.

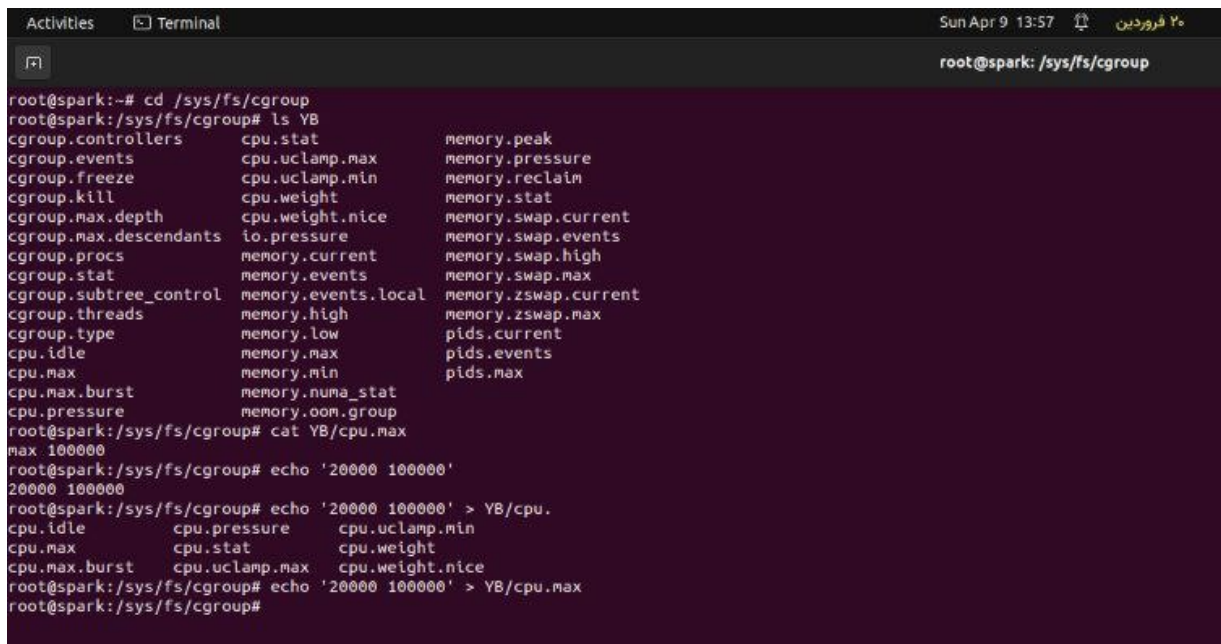
ابتدا مانند قسمت قبل یک Cgroup ایجاد میکنیم.
می‌توانیم با استفاده از دستورات مربوط به Cgroup، منابع را به این گروه اختصاص دهیم و پروسه‌های مورد نظر را در این گروه قرار دهیم. در قسمت cgroup.controllers مشاهده میشود که تنها دسترسی ما فقط به memory pids میباشد.
برای اضافه کردن دسترسی به cpu از کد زیر استفاده میکنیم:

```
echo '+cpu' > /sys/fs/cgroup/cgroup.subtree_controller
```

پس از این دستور مشاهده میکنیم که دستور cpu.max هم به لیست ما اضافه میشود.

به عنوان مثال، برای محدود کردن پردازش‌های یک برنامه به 20 درصد CPU، می‌توانید از دستور زیر استفاده کنیم:

```
/sys/fs/cgroup# echo '20000 100000' > YB/cpu.max
```



```
Activities  Terminal  Sun Apr 9 13:57  ۲۰ فروردین
root@spark: /sys/fs/cgroup

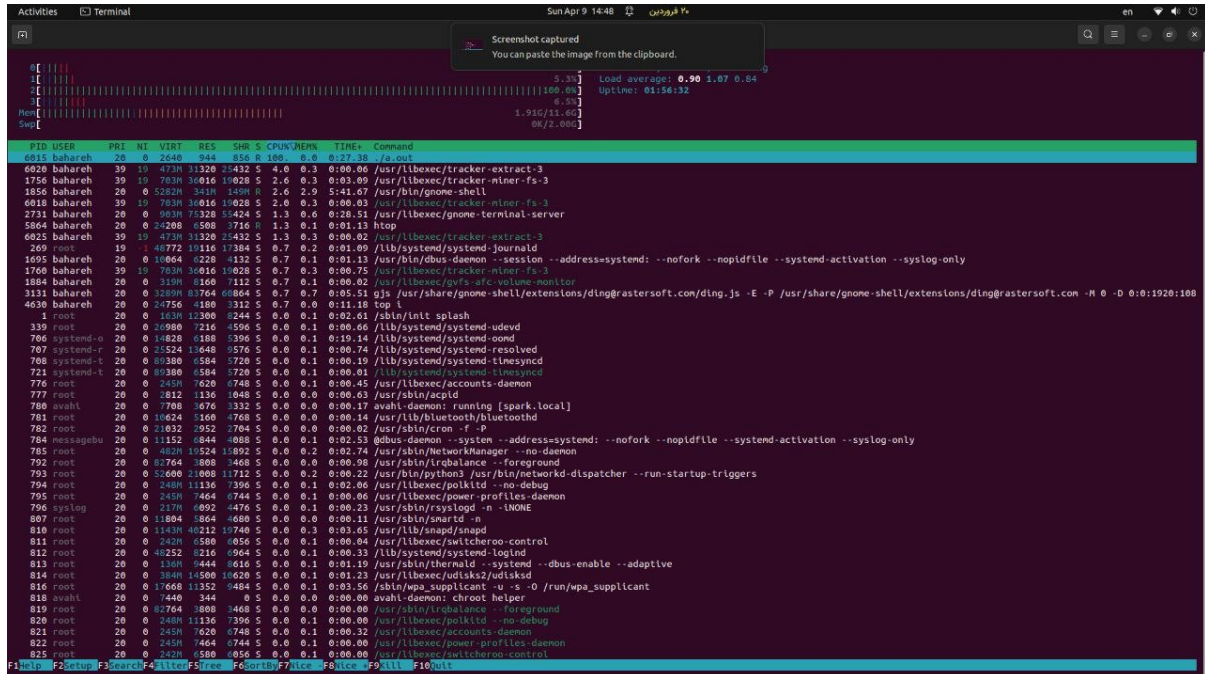
root@spark:~# cd /sys/fs/cgroup
root@spark:/sys/fs/cgroup# ls YB
cgroup.controllers  cpu.stat  memory.peak
cgroup.events       cpu.uclamp.max  memory.pressure
cgroup.freeze       cpu.uclamp.min  memory.reclaim
cgroup.kill         cpu.weight      memory.stat
cgroup.max.depth    cpu.weight.nice  memory.swap.current
cgroup.max.descendants  io.pressure     memory.swap.events
cgroup.procs        memory.current   memory.swap.high
cgroup.stat         memory.events    memory.swap.max
cgroup.subtree_control  memory.events.local  memory.zswap.current
cgroup.threads      memory.high      memory.zswap.max
cgroup.type         memory.low       pids.current
cpu.idle            memory.max       pids.events
cpu.max             memory.min       pids.max
cpu.max.burst       memory.numa_stat
cpu.pressure        memory.oom.group
root@spark:/sys/fs/cgroup# cat YB/cpu.max
max 100000
root@spark:/sys/fs/cgroup# echo '20000 100000'
20000 100000
root@spark:/sys/fs/cgroup# echo '20000 100000' > YB/cpu.
cpu.idle      cpu.pressure  cpu.uclamp.min
cpu.max       cpu.stat     cpu.weight
cpu.max.burst  cpu.uclamp.max  cpu.weight.nice
root@spark:/sys/fs/cgroup# echo '20000 100000' > YB/cpu.max
root@spark:/sys/fs/cgroup#
```

برای مرگیر کردن CPU و مشغول نگه داشتن آن در بیشترین حالت ممکن یک کد C درای حلقه بی نهایت اجرا می‌کنیم. ابتدا این برنامه را در حالت عادی و در گروه اصلی کرنل اجرا میکنیم تا مطمئن شویم که CPU در حالت مورد نظر ما می باشد یا خیر.

برای اینکار از دستور زیر استفاده میکنیم :

```
htop
```

همانطور که تصویر زیر مشاهده می کنید در حال اجرای کد ۱۰۰ درصد CPU مشغول شده است. (در یکی از هسته های پردازشی)



سپس پس از محدود کردن cpu از گروهی که ایجاد کرده بودیم و اجرا کردن همین کد در آن گروه مشاهده می کنیم که در بیش ترین حالت ممکن ۲۰ درصد از cpu درگیر خواهد شد.

