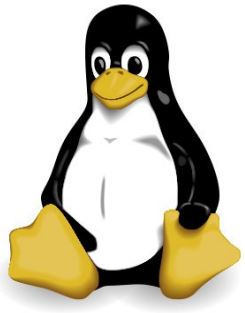

LINUX SİSTEM YÖNETİMİ

Murat Demirten

Table of Contents

Linux Sistemi Yönetimi

Bu kitapta Linux tabanlı sistemlerde sistem yönetimi üst başlığında, öncelike yeterli Türkçe doküman bulunmayan veya görece yeni geliştirilen özelliklerle ilgili bilgilere yer vereceğiz.

Kitabın belirli bir sıraya göre takip edilmesi gerekli değildir. Bağımsız konularda yazdığımız belgeler bir araya getirilerek oluşturulmaktadır.

Control Groups (cgroups)

Control Groups (Cgroups) Linux çekirdeğinde yer alan, proseslerin gruplara ayrılarak işlemci, bellek, disk, network kaynak kullanımlarına ilişkin hiyerarşik kurallar tanımlanabilmesine sağlayan gelişmiş bir kontrol sistemidir.

2.6.24 versiyonundan itibaren Linux çekirdeğinde bulunmakla birlikte son zamanlarda önemli bir yeniden dizayn çalışması yapılmış olduğundan **3.16** versiyonu veya daha yeni bir çekirdek kullanılması önerilir.

Cgroup mekanizmasıyla sadece normal prosesler değil, kernel thread'lerini dahi belirli işlemciler üzerine kaydırmak mümkün olabilmektedir.

Cpuset Kullanımı

Cgroups sistematiği içerisindeki en önemli kalemlerden biri cpu ve bellek kaynaklarının hiyerarşik yönetimidir.

Bunun için cgroup sanal dosya sistemi mount edilmişse, dosya sistemi arayüzü üzerinden dizin açma, dosyaların içerisine değer atama yöntemleriyle konfigürasyon yapmak mümkündür.

Örnek olarak cgroup sistemi `/sys/fs/cgroup` dizini altına mount edilmiş ise, **cpuset** işlemleri için `/sys/fs/cgroup/cpuset` dizini kullanılır.

```
# mkdir /sys/fs/cgroup/cpuset/egitim
```

şeklinde bir komut çalıştırdığımızda otomatik olarak yeni bir grup oluşturulmuş olur.

Bu dizin içerisine baktığımızda aşağıdaki sanal dosyaların olduğunu görürüz:

```
# ls /sys/fs/cgroup/cpuset/egitim
cgroup.clone_children
cpuset.cpus
cpuset.memory_migrate
cpuset.memory_spread_slab
cpuset.sched_relax_domain_level
cgroup.procs
cpuset.mem_exclusive
cpuset.memory_pressure
cpuset.mems
notify_on_release
cpuset.cpu_exclusive
cpuset.mem_hardwall
cpuset.memory_spread_page
cpuset.sched_load_balance
tasks
```

Örnek olarak 4 çekirdekli bir sistemde, 2 ve 3 nolu çekirdekleri **egitim** adını verdiğimiz cpu kümesine dahil etmek istersek aşağıdaki komutu vermemiz yeterlidir:

```
# echo 2,3 > /sys/fs/cgroup/cpuset/egitim/cpuset.cpus
```

Bir NUMA bellek node'unu atamak istersek bu defa komutumuz aşağıdaki gibi olacaktır:

```
# echo 0 > /sys/fs/cgroup/cpuset/egitim/cpuset.mems
```

Çalışan bir uygulamanın PID (proses numarası) değerini ilgili cpuset dizini altındaki `tasks` dosyasına yazdığımızda, uygulamanın dahil olduğu cpu kümesi değiştirilmiş olur. Örnek olarak 12345 PID değerine sahip uygulamayı `egitim` adındaki cpu kümemize aktaracak olursak aşağıdaki komutu kullanabiliriz:

```
# echo 12345 > /sys/fs/cgroup/cpuset/egitim/tasks
```

Bu kullanım modeli pek pratik olmadığından, işleri bizim için kolaylaştıran **cset** uygulamasını kullanmanız önerilir. Uygulamayı aşağıdaki komutla sisteminize kurabilirsiniz:

```
$ sudo apt-get install cpuset
```

> **NOT:** cset uygulaması henüz yeni 3.X serisi çekirdeklerdeki isimlendirme değişiklikleriyle düzgün çalışamamaktadır. Bu nedenle hiç cpuset yaratılmamış ise, `No such file or directory: '/cpusets//cpus'` şeklinde hata mesajlarıyla sonlanmaktadır. Geçici çözüm için ilk cpuset'ini yukarıda anlatıldığı şekliyle `mkdir` komutuyla dizin oluşturarak yapmamız yeterlidir.

Örnek senaryomuzda şunları yapalım:

- 4 çekirdekli bir sistemimiz olduğunu düşünelim
- `system` adında bir cpuset oluşturup 0 ve 1 nolu çekirdekleri bu kümeye atalım
- `producer` adında bir cpuset oluşturup 2 nolu çekirdeği atayalım, aynı sistem üzerindeki bir başka yazılımımıza yoğun hızda paket gönderimi yapacak bir uygulamayı bu kümeye dahil edelim
- `consumer` adında bir cpuset oluşturup 3 nolu çekirdeği atayalım, producer yazılımından yoğun olarak gelen paketleri aynı hızda işleyebilmek için hazırladığımız server yazılımını bu kümeye dahil edelim

İşlemlere başlarken ilk kümeyi **cset** bug'ı nedeniyle `mkdir` ile oluşturuyoruz:

```
# mkdir /sys/fs/cgroup/cpuset/system
```

Ardından mevcut cpuset'lerin listesini alalım:

```
# cset set --list
cset:
         Name       CPUs-X    MEMs-X Tasks Subs Path
 ------------ ---------- - ------- - ----- ---- ----------
         root        0-3 y       0 y   492    1 /
       system      ***** n   ***** n     0    0 /system
```

Sonrasında 0 ve 1 nolu çekirdekleri `system` kümesine dahil edelim ve son durumu listeleyelim:

```
cset set -c 0,1 -m 0 -s system
cset: --> modified cpuset "system"

# cset set --list
cset:
         Name       CPUs-X    MEMs-X Tasks Subs Path
 ------------ ---------- - ------- - ----- ---- ----------
         root        0-3 y       0 y   493    1 /
       system        0-1 n       0 n     0    0 /system
```

Devamında 2 ve 3 nolu çekirdekleri sırasıyla `producer` ve `consumer` kümelerini oluşturarak ilişkilendirelim:

```
# cset set -c 2 -s producer
cset: --> created cpuset "producer"

# cset set -c 3 -s consumer
cset: --> created cpuset "consumer"

# cset set --list
cset:
         Name       CPUs-X    MEMs-X Tasks Subs Path
 ------------ ---------- - ------- - ----- ---- ----------
         root        0-3 y       0 y   475    3 /
```

```
      consumer          3 n       0 n     0    0 /consumer
        system        0-1 n       0 n     0    0 /system
      producer          2 n       0 n     0    0 /producer
```

Çıktıda görüleceği üzere `root` cpuset'i sistemde öntanımlı olarak bulunmakta ve tüm eski prosesler ile yeni başlayan prosesler bu kümeye girmektedir. `root` kümesi içerisinde tüm işlemci çekirdek dahil olduğu için (0-3), öncelikle `root` altındaki tüm prosesleri yeni oluşturduğumuz `system` adındaki kümeye taşımak için aşağıdaki komutu kullanalım:

```
# cset proc -m -f root -t system
cset: moving all tasks from root to /system
cset: moving 390 userspace tasks to /system
[==================================================]%
cset: done

# cset set --list
cset:
         Name       CPUs-X    MEMs-X Tasks Subs Path
 ------------ ---------- - ------- - ----- ---- ----------
         root        0-3 y       0 y    80    3 /
     consumer          3 n       0 n     0    0 /consumer
       system        0-1 n       0 n   390    0 /system
     producer          2 n       0 n     0    0 /producer
```

Görüldüğü üzere **390** adet proses `system` kümesine geçti, ancak halen **80** tanesi `root` kümesinde durmaya devam ediyor. Bunlar kernel seviyesinde çalışan kernel thread'leridir. Eğer realtime ihtiyaçları olan uygulamalarımız söz konusu ise, bu kernel thread'lerini de `root` kümeden alıp `system` kümesine taşımak için aşağıdaki komutu verebilirsiniz:

```
# cset proc -k -f root -t system
cset: moving all kernel threads from / to /system
cset: moving 50 kernel threads to: /system
cset: --> not moving 28 threads (not unbound, use --force)
[==================================================]%
cset: **> 32 tasks are not movable, impossible to move
cset: done

# cset set --list
cset:
         Name       CPUs-X    MEMs-X Tasks Subs Path
 ------------ ---------- - ------- - ----- ---- ----------
         root        0-3 y       0 y    60    3 /
     consumer          3 n       0 n     0    0 /consumer
       system        0-1 n       0 n   408    0 /system
     producer          2 n       0 n     0    0 /producer
```

Bazı task'lar daha `system` kümesine taşındı ama diğerlerini taşımak mümkün olmadı. Bu noktada sistem ayarlarını bırakıp uygulamalarımızı ilgili kümelerde çalıştırma işlemine geçelim.

Yapacağımız test şu adımlardan oluşacak:

- `udp_msg_server` adındaki uygulamamızı `consumer` kümesi içerisinde başlatacağız
- `udp_msg_client` uygulamamızı `producer` kümesi içerisinde başlatacağız
- Başka bir konsolda normalde sistem kaynağının tümünü tüketecek olan kernel kaynak kod derleme sürecini `make -j 8` parametresiyle başlatacağız
- Tüm bunlara rağmen uygulamamızın düzgün çalışıp çalışmadığını kontrol edeceğiz

Yeni başlattığımız uygulamaları bir kümeye dahil etmek için normalde uygulamanın PID değerini öğrenip, ilgili küme dizini altındaki `tasks` dosyasına yazmak gereklidir. Bunu elle yapmak zaman kaybı olduğu için `cset` uygulaması üzerinden uygulamamızı başlatacağız ve bizim için PID değerininin yazımı işlemini cset halledecek. Dikkat etmemiz gereken tek

nokta, uygulamamız eğer parametre alıyorsa, uygulamanın parametlerini yazmaya başlamadan önce, konsolda parametre parsing işlemini sonlandıran `--` kısayolunu kullanmak olacaktır.

```
# consumer için
# cset proc -s consumer -e ./udp_msg_server -- -s RR -r 16000000
cset: --> last message, executed args into cpuset "/consumer", new pid is: 3059

# producer için
# cset proc -s producer -e ./udp_msg_client -- -t 1000000
cset: --> last message, executed args into cpuset "/producer", new pid is: 10058
```

Audit Framework

Linux audit framework, sistemde oluşan tanımlı olayların (event) kayıt alınabilmesi için geliştirilmiş bir katmandır.

Bu katmanın amacı sistemde oluşan olayları engelleyebilme olmayıp, sadece olan bitenin kayıt altına alınabilmesi için kernel seviyesinde gereken desteği sağlamaktır.

Aduit Daemon: `auditd`

Linux audit katmanı tarafından oluşturan olay bilgilerinin kullanıcı kipinde bir yazılım tarafından işlenmesi ve opsiyonel ek kontroller sonrası kalıcı bir dosya/veritabanı üzerine yazılması gerekir. Dolayısıyla kernel içerisinde audit desteği bulunsa dahi, olay bilgileri kullanıcı kipinde işlenip saklanmıyorsa herhangi bir loglama da gerçekleşmeyecektir.