Orta Düzey Linux Komutları

1. Yönlendirme

Kullanıcı, ekrana yazdığı bir komutun neler yaptığını en rahat şekilde komut tarafından ekrana yönlendirilen bilgilerden anlayabilir. Program, kullanıcıyı bilgilendirme amacıyla mümkün olduğu kadar çok, fakat ortalığı fazla karıştırmamak için de mümkün olduğu kadar az bilgiyi ekrana vermelidir.

1.1 Standart Girdi, Çıktı ve Hata

Linux'ta, programın ekrana yazılan bilgiyi iki sınıf altında toplayabiliriz. Birincisi, olağandışı bir durumu bildiren standart hata, diğeri de her türlü verinin yazıldığı standart çıktı. Program çalıştığı andan itibaren bu iki kanal üzerinden akan bilgiler, programın çalıştığı sanal terminale yazılırlar. Program girdilerini ise standart girdi aracı olan klavyeden alır.

1.1.1 Girdi Yönlendirmesi

Girdi yönlendirmesinde, verilen ***sözcük***'ün yorumlanması ile elde edilen isme sahip dosya, verilmişse dosya tanıtıcı **n** üstünde, aksi takdirde standart girdide (dosya tanıtıcı 0) açılır.

Girdi yönlendirmesi olarak değerlendirilecek sözdizimi:

[n]<*sözcük*

Eğer bu bilgiler bir ekran boyundan (25 satır) fazla tutuyorsa bazı satırlar programcının gözünden kaçabilir. Bunu önlemek amacıyla standart çıktı ve hata bir dosyaya yazılacak şekilde ayarlanabilir. Yönlendirme olarak da bilinen bu işlem UNIX altında (DOS'ta olduğu gibi) > karakteri ile gerçekleştirilir.

Örnek olarak o an bulunduğunuz dizinde yeralan dosyaları ekrana getirin :

```
$ ls -al
```

Bu komut, standart çıktı olarak dosyaların bilgilerini ekrana getirecektir. Bu çıktıyı, bir dosyaya yönlendirelim ve dosyanın içeriğine göz atalım:

```
linux:~$ ls -al > liste

linux:~$ cat liste

total 16

drwxr-xr-x  5 gorkem   users     1024 Feb 13 13:10 .

drwxr-xr-x  4 root     root      1024 Jan  7  1980 ..

-rw-r--r--  1 gorkem   users     390 Feb 13 12:56 .Xdefaults

-rw-r--r--  1 gorkem   ftpadm  2016 Feb 13 13:09 .bash_history

-rw-r--r--  1 gorkem   users      1 Feb 13 12:57 .bashrc

-rw-r--r--  1 gorkem   users      163 Nov 24  1993 .kermrc

-rw-r--r--  1 gorkem   users     34 Nov 24  1993 .less

-rw-r--r--  1 gorkem   users     114 Nov 24  1993 .lessrc

drwxr-xr-x  2 gorkem   users     1024 Jan  7  1980 .term

-rw-r--r--  1 gorkem   users     87 Feb 13 12:56 .xinitrc

-rw-r--r--  1 gorkem   users     2795 Feb 13 13:06 adres

-rw-r--r--  1 gorkem   users     0 Feb 13 13:10 liste

drwxr-xr-x   2 gorkem   users    1024 Feb 13 12:54 mail

drwxr-xr-x   2 gorkem   users    1024 Feb 13 12:54 perl

-rw-r--r--  1 gorkem   users     0 Feb 13 13:10 typescript

linux:~$
```

- karakteri standart hatayı dosyaya yönlendirmez. Bu işlem için 2> kullanılır. Ama hatayı görebilmek için, hata yaratan bir komut yazmalıyız, öyle değil mi ?

```
$ ls /deneme

/deneme : No such file or directory

$ ls /deneme 2> hata

$ cat hata

/deneme : No such file or directory
```

Aşağıdaki komutun işletilmesinin ardından standart çıktı oku1 dosyasına, standart hata ise oku2 dosyasına yazılacaktır. Bu dosyaları komutu çalıştırdıktan sonra incelemek suretiyle neler olup bittiğini anlamak mümkün olur.

```
$ mkdir ~/deneme            (deneme isimli bir dizin yarat)

$ touch ~/deneme/gecici     (gecici isimli bir dosya yarat)

$ cat ~/deneme 2>oku2 >oku1
```

Kabuk, standart çıktı ve standart girdi için sırayla 2 ve 1 numaralarının kullanımına izin verir. Yukarıda yeralan son komutta, standart hata mesajları için 2 kullanılarak hataların oku2 dosyasına yazılmıştır. Aşağıda, çekirdek derlemek için sürekli olarak kullanılan program yer alıyor. Yönlendirme sayesinde ekrana gelmesi gereken mesajlar kullanılmayan 9. sanal konsola yönlendiriliyor.

```
# make config

# make dep > /dev/tty9

# make clean > /dev/tty9

# time make zImage > /dev/tty9
```

Son satırdaki time komutu, kendinden sonra gelen komutun ne kadar zaman içinde çalıştırıldığını gösterir. Çekirdek derlemede geçen zaman, makinanın gücü hakkında bir fikir verebilir. Peki ne hata, ne de çıktıyı ekranda görmek istemiyorsam ne yapmalıyım? Bunun için standart çıktı ve hatayı biraraya getirerek yönlendirilen her çıktının kaybolduğu '**kara deliğe**' atmak yeterlidir:

```
$ ls /deneme > /dev/null 2>&1
```

Yukarıdaki komutun yazılış sırasına dikkat edin.

Standart çıktı ya da *standart hata*yı yönlendirirken, > işareti kullanırsanız, dosya yoksa, oluşturulur ve komutun çıktısı dosyaya yazılır. Dosya varsa, içeriği yok olur ve komutun çıktısı dosyanın yeni içeriği olur. Var olan bir dosyayının eski içeriğini tamamen silmek değil de komutun çıktısını dosyaya eklemek istiyorsanız >> kullanmalısınız. Bu durumda dosya varsa komutun çıktısı dosyanın eski içeriği korunarak sonuna eklenir, dosya yoksa oluşturulur ve komutun çıktısı dosyaya yazılır.

Örneğin:

```
$ echo deneme1 >>deneme.txt

$ cat deneme.txt

deneme1

$ echo deneme2 >>deneme.txt

$ cat deneme.txt

deneme1

deneme2

$
```

Örnekte görüldügü gibi ilk komut deneme.txt dosyasını oluşturdu. İkincisi ise oluşan dosyanın içeriğini koruyarak ikinci komutun çıktısını bu dosyanın sonuna ekliyor.

Standart hata ve çıktıya ek olarak UNIX'in desteklediği bir yönlendirme daha vardır: Standart girdi sayesinde bir dosyayı oluşturan satırlar, bir komut veya programa yönlendirilebilir. Daha önce bir metin editor kullanarak hazırlamış olduğumuz raporu patrona kısa yoldan göndermek için,

```
$ mail -s "rapor" patron < rapor.txt
```

Dosyanın içeriği, mail komutuna girdi olmuş ve rapor.txt dosyası patron kullanıcısına '**rapor**' konu başlığıyla e-posta ile gönderilmiştir.

Bazı yaygın kabuk sembolleri

Sembol	İşlerliği
ENTER	Komut satırını çalıştır
;	Aynı satırda farklı komutları ayır
[ ]	Dosya isimlerinde olası karakter dizilerini benzeştir
****	Takip eden karaktere izin verir
&	Geri planda bir komutu çalıştırır
!	history komutunun sembol şekli olup verilmiş komutları listeler
*****	Dosya isimlerinde herhangi karakter setini benzeştir
?	Dosya isimlerinde herhangi bir karakteri benzeştir
>	Bir dosya veya aygıta standart çıktıyı yönlendir, bu esnada eğer dosya mevcut değilse yarat, eğer mevcut ise üzerine yaz
>!	Eğer dosya halen mevcut ise üzerine yazma işlemine zorlar bunu noclobber seçeneğinin üzerine yazar
<	Bir dosya veya aygıttan bir programa standart giriş işlemini yönlendir
>>	Bir dosya veya aygıta standart çıktıyı yönlendir, çıktıyı dosyanın sonuna ilave et

Komut 2> dosya_ismi	Standart hatayı (Bourne kabuğunda) bir dosyaya veya aygıta yönlendir
komut 2>> dosya_ismi	Standart hatayı (Bourne kabuğunda) bir dosya veya aygıta yönlendir ve sonuna ekle
komut 2>&1	Standart hatayı (Bourne kabuğunda) standart çıktıya yönlendir
Komut >& dosya_ismi	Standart hatayı (C kabuğunda) bir dosyaya veya aygıta yönlendir

1.1.2 Standart Çıktı ve Standart Hatanın Yönlendirmesi

Bash, hem standart çıktı (dosya tanıtıcı 1) hem de standart hatayı (dosya tanıtıcı 2), *sözcük*'ün yorumlanması ile elde edilen isme sahip dosyaya yönlendirebilir.

Standart çıktı ve standart hatanın yönlendirilmesine konu iki sözdizimi vardır:

&>*sözcük* ve

>&*sözcük*

Bu iki biçemden birincisinin kullanılması önerilir. Her ikisi de aşağıdakine eşdeğerdir:

```
>sözcük 2>&1
```

1.1.3 Çıkış Durumu

Kabuğun amaçlarına uygun olarak, sıfır çıkış durumu ile çıkan bir komut başarılıdır. Sıfırdan farklı bir çıkış durumu ise başarısızlık göstergesidir. Böylece sayılardan oluşan hem başarı durumunu hem de başarısızlık nedenlerinin ifade edilebildiği bir yapı sağlanır. Bir komut, numarası N olan bir ölümcül sinyal ile sonlandırılırsa, Bash, çıkış durumu olarak 128+N değerini kullanır.

Bir komut bulunamamışsa, bir çocuk süreç oluşturulur ve bu süreç 127 çıkış durumu ile döner. Bir komut bulunmuş ancak çalıştırılabilir değilse, 126 dönüş durumu döner. Yönlendirme ya da yorumlama sırasında bir hatadan dolayı komutun çalıştırılması başarısız olursa, sıfırdan büyük bir çıkış durumu döner.

Bash yerleşiklerinin de başarı durumunda sıfır ve başarısızlık durumunda sıfırdan farklı bir çıkış durumu ile dönerek, çıkış durumlarının koşul ve liste yapılarında kullanılabilmesi sağlanmıştır. Tüm yerleşikler yanlış kullanım halinde 2 çıkış durumu ile döner.

1.2- Boru (pipe) İşlemleri

Bazı durumlarda, bir komutun çıktısı diğer bir komuta yönlendirilebilir. Başka bir deyişle, komutun standart çıktısını bir dosyaya değil, bu çıktıyı işleyecek başka bir komuta yönlendirmek istiyorsunuz. Bu amaçla UNIX altında (yine DOS'ta olduğu gibi) boru (|) karakteri kullanılır. Bu karakter, kendinden önce gelen komut veya komut serisinin çıktısını,

kendinden sonra gelen komuta gönderir. Örneğin bir dizinde yeralan tüm dosyaları yazıcıya aktarmak için,

```
$ komut | komut

$ ls -al | lpr
```

komutları kullanılabilir. Artık **ls -al** komutunun ekrana vermesi gereken tüm bilgiler, **lpr** komutu aracılığıyla yazıcıya gönderilmiştir. İlk komutun standart çıktısı, ikinci komuta standart girdi olarak atanır. Diğer bir örnekte, README dosyasında kaç satır olduğu bulunuyor. Bir dosyadaki veya komut çıktısındaki satır, karakter ve kelime sayılarını bulmak için **wc** komutunu kullanabilirsiniz. Aşağıdaki komut, **dosya.txt** dosyasının içinde kaç satır olduğunu ekrana yazar :

```
$ wc -l dosya.txt

40
```

wc komutunun parametreleri şunlardır:

-l : satır sayısını bulur

-w : kelime sayısını bulur

-c : karakter sayısını bulur

Yönlendirme ve boru işlemleri bazen insanın kafasını karıştırabilir. Yönlendirme, bir programdan bir dosyaya yapılabilir, fakat bir komut'dan başka bir komut'a yönlendirme yapamazsınız. Benzer şekilde, iki dosyanın arasında boru işlemi uygulamak mümkün olmaz.

Aşağıda **/etc** dizini içinde yeralan alt dizinlerin tamamı listelenmiştir.

```
$ ls -F /etc | grep -/-

acct/

cron.d/

default/
```

İki komuttan daha fazla komutu içeren bir boru işlemi oluşturulabilir:

```
$ head -10 dante | tail -3 | lp

request id is printerA-177  (standart input)
```

Şimdi de biraz karışık bir örnek:

```
# ps -aux|grep inetd|grep -v grep|awk '{print $2}'|xargs kill
-1
```

Yukarıdaki örnek zorlama bir örnek değil, bir Linux sistem yöneticisinin her an kullanması gerekebilecek türden. Bu örneği burada ayrıntılı olarak açıklamayacağız, sadece *pipe* kullanarak ne kadar fazla sayıda komutun birbirine bağlanabildiğini göstemek amacıyla verilmiştir. Bu ve benzeri komutların ayrıntılı açıklamaların kabuk programlamayla ilgili bölümde bulabilirsiniz.

Standart çıktıyı mevcut bir dosyaya yönlendirmek önceki dosya içeriğinin kaybına yol açacak olan üzerine yazma işlemini gerçekleştirir. Mevcut veri üzerine yeni veri yazma işlemi 'clobbering' olarak olarak adlandırılır. Bunun gerçekleşmesini engellemek için kabuk 'noclobber' opsiyonunu sunar. Aşağıda kabuk içinde **noclobber** opsiyonu **set** komutu yardımı ile aktive edilmiştir.

```
$ set -o noclobber

$ set -o | grep noclobber

noclobber    on

$ ps -ef > file_new

$ cat /etc/passwd > file_new

ksh: file_new: file already exists

$
```

Geçici olarak **noclobber** opsiyonunu de-aktivate etmek istersek **>|** kombinasyonunu kullanmamız gerekmektedir.

```
$ ls -l >| file_new
```

2- Çok Görevlilik

UNIX'in en büyük silahlarından biri süreçlerdir. Her süreç sistemde bağımsız çalışan, birbirini etkilemeyen ve herbiri kendi kaynağını kullanan programdır. Süreçler arka planda veya kabuğun kontrolünde çalışabilir. Çekirdek, her sürecin kullandığı sistem kaynağından haberdar olur ve bu kaynakların süreçler arasında eşit bir şekilde paylaşılmasından

sorumludur. Bir süreç, aksi belirtilmedikçe çalıştığı süre içinde klavyeden bilgi alır ve ekrana bilgi verir.

Kullanıcıların haberi bile olmadan çalışan süreçler, Linux makinasındaki G/Ç işlemlerini gerçekleştirebilmek için sürekli faaliyet içinde bulunurlar. Onlarca süreçten bazıları kullanıcıların sisteme girmesini sağlarken (getty) bazıları da WWW ve FTP gibi İnternet tabanlı istekleri yerine getirir (httpd, ftpd).

UNIX/Linux altında çalışan her program en az bir süreçten oluşur. İşletim sistemleri teorisine göre her süreç diğer süreçlerden bağımsızdır. Bilgisayar kaynaklarının paylaşımı istenmediği sürece her süreç belleğe alınır ve CPU bu sürece tahsis edilir. Bir sürece ayrılan bellek alanını başka bir süreç kullanmak istediği zaman buna teşebbüs eden süreç öldürülür. Burada öldürme terimi hiçbir sürece hiçbir kaynağın tahsis edilmeyerek sürecin pasif hale bırakılması halidir.

UNIX/Linux tipi işletim sistemlerinin kararlı biçimde çalışması bu yaklaşımın neticesidir. Yani bir kullanıcıya ait bir uygulama ne diğer kullanıcı programlarına, ne de işletim sistemine etki eder.

Buradaki anlamı ile süreç, icra edilen bir programdır. Bu program icra esnasında, işletim sistemi tarafından durdurulabilir ve bekletildikten sonra tekrar çalıştırılabilir.

İşletim sisteminde her süreçle ilgili bilgileri saklayan bir süreç tablosu vardır. Bu süreç tablosu,

-Her süreç ile ilişkili bilgileri kaydeder

-Program kodu ile ilişkili veriyi kaydeder

-Program sayacı(program counter), yığıt göstergesi (stack pointer) ve diğer yazmalar

ile ilişkili diğer bilgileri saklar.

Her sürecin ebeveyn süreci (parent process) ve çocuk süreci (child process) olabilir. Ebeveyn süreç, söz konusu sürece referans verir. Çocuk süreç ise söz konusu süreç tarafından referans verilen süreçtir.

Her süreci diğerlerinden ayırtan tek bir (unique) kod vardır; buna süreç belirleyici (process identifier PID) adı verilir.

Örneğin kullandığınız WWW tarayıcınız bir süreçtir.

3- Arka Planda Çalıştırma

Bir komutu arka planda çalıştırmak için, komutun sonuna **&** karakteri getirilir. Komut girildikten sonra tekrar kabuk istemcisine düşecek ve kaldığınız yerden devam edebileceksiniz. Program, arka planda diğer süreçlerle çakışmadan bir süre çalışır ve işi bittiğinde durur.

```
$ sort buyuk_dosya > buyuk_dosya.sirali &

[1] 772
```

Komutun arka plana atılmasından sonra ekranda yeralan 1, sürecin sıra numarasını, 772 sayısı ise süreç kimliğini (Process ID) gösterir. Her program, sistem kaynaklarını biraz daha azalttığından genel anlamda makina *yavaşlar*.

ps komutu, sistemde o esnada mevcut süreçler hakkında, durum bilgisi, boyut, isim, sahiplik, CPU zamanı, duvar saati zamanı vb. bilgileri listeleyen bir komuttur. **ps** komutunun pek çok parametresi mevcuttur. Bunalar arasında en önemli olanlar aşağıda tabloda listelenmiştir:

PARAMETRE	ANLAMI
-a	Belirli bir terminal kontrolündeki listele (sadece aktif kullanıcıların süreçleri değil)

-r	Sadece çalışmakta olan (running) süreçleri listele
-x	Bir terminal kontrolünde olamayan süreçleri listele
-u	Süreç sahiplerini listele
-f	Süreçler arasındaki ebeveyn çocuk ilişkilerini göster
-l	Uzun formatlı bir liste üret
-w	Bir sürecin komut satırı parametresini göster (kısa biçim)
-ww	Bir sürecin komut satırı parametresini göster (uzun biçim)

ps komutu ile birlikte kullanılan en sık parametre kümesi **-auxww** şeklindedir. Böylece tüm süreçler, bunların sahipleri, tüm süreçlerin komut satırı parametreleri listelenir. Aşağıda,

```
# ps -auxww
```

komutu sonucu elde edilen bir liste görülmektedir.

Sütün Başlığı **Anlamı**

USER	Sürecin sahibi olan kullanıcı
PID	Süreç belirleme numarası
%CPU	CPU'nun toplam kullanımının yüzde kaçının o süreç tarafından kullanıldığı bilgisi
%MEM	Süreç tarafından belleğin yüzde kaçının kullanıldığı bilgisi
VSZ	Sürecin kullandığı görüntü bellek miktarı
TTY	Süreci kontrol eden terminal. Bu sütunda bir ? sembolünün olması, sürecin artık terminal kontrolünde olmadığını gösterir
STAT	Sürecin o anki durumu
START	Sürecin başlangıç zamanı
TIME	Sürecin CPU da harcadığı süre

COMMAND	Sürecin adı ve komut satırı parametreleri

Aşağıdaki tablo STAT parametresinin alabileceği değerleri göstermektedir :

S	Süreç uyuyor. İcra edilmeye hazır olan süreçlerin tümü uykudadır.
R	Süreç CPU da işlem görüyor.
D	Süreç araya girilemez biçimde uykuda.
T	Süreç ya bir hata ayıklayıcıda inceleniyor yada durdurulmuştur. (kullanıcı tarafından **ctrl -Z** vb. şekilde)
Z	Süreç 'zombie' durumundadır. Bunun anlamı, a) ebeveyn süreç çocuğu olan sürecin öldüğü hakkında bilgi sahibi değildir b) veya ebeveyn süreç kural dışı öldürüldüğü için **init** süreci çocuk süreci yok edemez. Zombie durumundaki süreçler öldürülemez ancak sistem kaynaklarından da yararlanamaz. Bu durum genellikle kötü yazılmış bir yazılımın sonucu oluşur.

Bir örnek :

```
$ ps

PID TTY STAT  TIME COMMAND

   76 v02 S     0:00 -bash
```

```
  111 v02 R     0:00 ps

$ ps -aux

USER  PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND

bin   63  0.0  5.5   64  364  ?  S  11:12  0:00
/usr/sbin/rpc.portmap

mehmet   76  0.0  9.7  101  644 v02 S    11:12   0:00 -bash

mehmet   112  0.0  5.0   59  332 v02 R    11:16   0:00 ps -aux

root     1  0.0  5.0   56  332  ?  S    11:12   0:00 init
[5]

root     6  0.0  4.2   35  284  ?  S    11:12   0:00 bdflush
(daemon)

root         7  0.0  4.2   35  284  ?  S    11:12   0:00
update (bdflush)

root        48  0.0  5.1   45  340  ?  S    11:12   0:00
/usr/sbin/crond -l10

root        59  0.0  5.5   53  364  ?  S    11:12   0:00
/usr/sbin/syslogd

root        61  0.0  5.0   44  336  ?  S    11:12   0:00
/usr/sbin/klogd

root        65  0.0  5.5   62  364  ?  S    11:12   0:00
/usr/sbin/inetd

root        67  0.0  5.8   79  388  ?  S    11:12   0:00
/usr/sbin/rpc.mountd

root        69  0.0  6.0   88  400  ?  S    11:12   0:00
/usr/sbin/rpc.nfsd

root        75  0.1  9.9  115  660 v01 S    11:12   0:00 -bash

root        77  0.0  4.6   52  304 v03 S    11:12   0:00
/sbin/agetty 38400 tty3
```

```
root        78  0.0  4.6   52  304 v04 S    11:12   0:00
/sbin/agetty 38400 tty4

root        79  0.0  4.6   52  304 v05 S    11:12   0:00
/sbin/agetty 38400 tty5

root        80  0.0  4.6   52  304 v06 S    11:12   0:00
/sbin/agetty 38400 tty6

root        81  0.0  5.5   42  368  ?  S    11:12   0:00 gpm
-t ms
```

ps komutuna bilgisayar üzerinde çalıştığınız her an ihtiyaç duymanız mümkündür, bu yüzden çeşitli parametrelerle kullanılması iyi anlaşılmalıdır.

4- Klavye Üzerinden Kesinti

Linux (ve UNIX) altında, klavyeden bazı tuş kombinasyonları yardımıyla çalışmakta olan program kesintiye uğratılabilir.

Klavyeden kabuk komut satırına yazılan bir programın uzun sürmesi halinde, eğer daha önceden komutun arkasına **&** işareti koyup arka planda çalışır halde bırakılmamışsa, klavyeden yapılan bir müdahele ile durdurulup arka planda çalışır hale getirilebilir. Örnek vermek gerekirse, uzun sürmesi beklenen bir komutu klavyeden yazalım ve ardından **Ctrl+Z** tuşlarına basalım:

```
linux:/etc/rc.d# find /usr -name "o*" -print

/usr/bin/od

/usr/lib/lilo/doc/other.fig

/usr/lib/lilo/doc/other.tex

/usr/man/man1/od.1.gz

[1]+  Stopped                 find /usr -name "o*" -print

linux:/etc/rc.d#
```

Bu esnada sürecin çalışmasına ara verilmiş, fakat program tamamen durmamıştır. Programın çalışmasını arka planda sürdürmek için **bg** komutunu kullanın.

```
$ bg <süreç ID>
```

Bu komutu tekrar komut satırında çalışacak ve klavyeden bilgi girilecek şekilde terminale bağlamak için **fg** yazın:

```
$ fg
```

Bir programı çalıştırmaya başlattıktan sonra tamamen durdurmaya karar vermişseniz klavyeden **Ctrl+C** tuşlarına basın.

Durdurulup arka planda çalışmaya yönlendirilen süreçlere kısaca *görev* ismi verilir. Tüm görevleri görebilmek için

```
$ jobs
```

yazın. Görevler, birden fazla oldukları zaman sıra numarası ile belirtilirler.

5- Süreçlerin Sona Erdirilmesi

Her an çalışan süreçlerden biri veya bir kaçı, beklenmedik döngülere girebilir. Bunun sonucu olarak sistemin kaynaklarını, özellikle hafızayı tüketici bir duruma gelebilirler. Bu tür kısır döngüye giren süreçleri bulup, eğer hayati önem taşımıyorlarsa `öldürmek' gerekir. Süreci öldürmekten kasıt, programı tamamen durdurarak sistemle ilişkisini kesmektir. Bu sayede programın hafızada kapladığı bölge serbest kalacak, çekirdek de hafıza düzenlemesini tekrar yaparak başka süreçlere daha fazla yer ayıracaktır. Bir süreci öldürmek için **kill** komutu kullanılır. Yukarıdaki 67 numaralı sürece ait /usr/sbin/rpc.mountd programını öldürmek için şunları yazın :

```
$ kill 67
```

Birçok süreç sizden bu mesajı aldıktan sonra, dosya sistemi üzerinde yarattığı geçici dosyaları, dosyalar üzerine koyduğu kilitleri temizlemek gibi yapması gereken işlemleri yaptıktan sonra çalışmasına son verecektir. Eğer öldürmeye çalıştığınız süreç herhangi bir nedenle takılmışsa ve bu komuta tepki vermiyorsa aşağıdakini deneyin:

```
$ kill -9 67
```

Artık programın sistemle ilişkisi tamamen kesilmiştir. **kill** komutu, **-9** seçeneğiyle sürece 9 numaralı sinyali gönderir. Bu sinyali alan sürecin yukarıda sözü edilen iki özel durum dışında çalışmayı sürdürmesi olanaksızdır. **-9** seçeneği özellikle sistem süreçleri üzerinde gerekmedikçe kullanılmamalıdır.

#top Komutu :

Bu komut süreçlerin belli bir anındaki durumunu listeledikten sonra her 2-3 saniyede bir bu ekranı yeniler yani en son durumu dinamik bir biçimde aktarır. Bu listeden sürecin önceliğini değiştirebilirisiniz veya onu öldürebilirisiniz. Bu komutun en önemli sakıncası sistemi yavaşlatmasıdır.

Default durum her kullanıcının **top** komutunu kullanabilmesidir. Fakat çok kullanıcı bir sistemde, her kullanıcının **top** komutunu kullanması halinde sistemin ne kadar yavaşlayacağını tahmin edebilirsiniz. O nedenle **top** komutunu kullanma yetkisi sadece sistem yöneticisine bırakmak uygun olabilir.

Bunu gerçekleştirmek için aşağıdaki komutu **root** olarak girebilirsiniz.

```
# chmod 0700 /usr/bin/top
```

6- Takma Adlar (alias)

alias komutu ile bir komut veya komut kümesinin yerine bir isim tahsis edilebilir. İşleyişi bir makroya benzeyen bu komut yardımıyla uzun komutlar, daha kısa komutlarla tanımlanabilir.

Bir **alias** komutu, anahtar kelimeyle başlar, ardından bir eşittir (=) işareti ve yerine kullanılacağı komut yazılır. Arada boşluk bırakılmaz.

```
$ alias dir='ls -al'

$ dir

total 668
```

```
-rw-r--r--  1 serdar   users     1016 Dec  7 13:51 .profile
-rw-r--r--  1 serdar   users     277 Nov 26 13:02 .signature
drwxr-xr-x  4 serdar   users     1024 Dec  3 18:24 .float/
drwxr-xr-x  2 serdar   users     1024 Nov 12 10:58 .spin/
-rw-r--r--  1 serdar   users     231 Nov 23 17:40 .xinitrc
drwxr-xr-x  2 serdar   users     1024 Oct  3 16:17 Mail/
-rw-r--r--  1 serdar   users     26721 Dec  3 14:55 NIS-HOWTO
drwxr-xr-x  2 serdar   users     1024 Nov 17 14:09 News/
```

Daha karmaşık **alias** 'lar da tanımlanabilir:

```
$ alias yedek="cd /var/log; tar -zcvf yedek.tgz cron debug lastlog; cd -"
```

NOT: ITU Bilişim Enstitüsü, Bilişim Ana Bilim Dalı Öğretim Üyesi Dr. M. Serdar ÇELEBİ' nin EST 513B ders notlarından alınmıştır