

Membaca Proses dengan Shell Script

Linux menyediakan banyak informasi proses. Dengan *shell script*, kita bisa membacanya dan menampilkannya dengan mudah.

Linux adalah sistem operasi yang benar-benar ramah dan *user friendly*. Bagaimana tidak? Keadaan sistem dengan transparan dipetakan di dalam pseudo file system `/proc`. Kondisi memory, memory yang digunakan oleh proses, status sistem, dan berbagai informasi lainnya dipetakan dalam berbagai file di dalam direktori `/proc`.

Bagi yang menyenangi *shell script*, Anda bisa mendapatkan banyak keuntungan dengan membaca `/proc` tersebut. Sebagai contoh, dengan mudah, kita bisa mengetahui sudah berapa lama sistem berjalan. Cukup dengan membaca `/proc/uptime`, informasi lama sistem berjalan bisa didapatkan dengan mudah. Atau, dengan membaca `/proc/cpuinfo`, kita dapat mengetahui dengan lengkap informasi processor yang digunakan. Semua itu dapat dilakukan tanpa harus menggunakan pemrograman *low level* seperti *assembly* atau bahasa C. Bayangkan ketika Anda harus membaca informasi processor di Windows.

Di artikel ini, kita akan memfokuskan diri untuk membaca status proses. Proses adalah hal yang supermenarik di Linux. Kehebatan Linux dalam menangani proses dan alokasi sumber daya menjadikannya sangat stabil. Di linux, semua proses memiliki sebuah id unik, yang lazimnya disebut sebagai PID (Process ID). Kita cukup mengetahui ID sebuah proses untuk melakukan berbagai hal pada suatu proses.

Di Linux, kita mengenai sebuah proses nenek moyang, yaitu proses `init`. Proses ini memiliki PID 1. Semua proses adalah anak, cucu, cicit atau turunan lebih lanjut dari proses `init` ini.

Berikanlah perintah `'ps ax'` untuk melihat proses-proses di sistem. Anda akan melihat sejumlah PID di layar. Sekarang, tampilkanlah isi `/proc`, Anda akan mendapatkan banyak

direktori berupa angka-angka. Direktori tersebut mencerminkan ID proses. Apabila masuk ke sebuah direktori ID proses tersebut, Anda akan melihat banyak file. File-file tersebut memetakan informasi detail setiap proses. Dengan demikian, sebenarnya, dengan sebuah *shell script* pun, dengan mudah kita bisa membuat program `ps` sendiri. Setuju?

Berbagai tool siap pakai

Shell script bukanlah bahasa pemrograman lengkap seperti C. Oleh karena itu, shell script tidak menyediakan berbagai pustaka siap pakai untuk mengakses sistem. Shell script berfungsi sebagai “file batch” yang akan memroses perintah-perintah yang kita berikan di dalamnya. Shell script menyediakan berbagai fungsi dasar pemrograman seperti perulangan, seleksi, array, dan lain sebagainya.

Untuk menggantikan fungsi-fungsi seperti fungsi pembacaan file, kita menggunakan program yang telah disediakan. Sebagai contoh, kita menggunakan program `cat` untuk membaca isi sebuah file. Atau, kita menggunakan program `grep` untuk penggunaan *regular expression*.

Dengan demikian, kita sangat bergantung kepada ketersediaan program eksternal untuk menghasilkan shell script yang memadai. Untungnya, program-program standar seperti `cat`, `grep`, dan lain sebagainya selalu tersedia di hampir setiap distro Linux.

Untuk memperlancar pembuatan shell script, kita perlu sering berlatih agar lebih menguasai penggunaan berbagai program. Dengan penguasaan program seperti `grep` atau `tr` misalnya, dengan mudah kita bisa menggunakan *regular expression* di shell script.

Di artikel ini, kita akan banyak menggunakan program `cat`, `tr`, `cut`, `grep`, `sort`, `uniq`, dan berbagai tool lain.

Satu hal yang penting adalah, kita memiliki banyak cara untuk mencapai tujuan ketika kita menggunakan shell script. Seperti pepatah, “banyak jalan menuju roma”. Contoh yang Anda lihat mungkin bertentangan dengan gaya Anda. Anda berhak untuk memiliki gaya sendiri. Hanya, sebaiknya gunakanlah gaya yang efisien dan pastikan script Anda mudah terbaca untuk memudahkan *debugging* dan penyempurnaan oleh pihak lain. Tidak ada gunanya membuat shell script yang pendek, namun hanya dimengerti oleh Anda.

Membaca status proses

Ada sebuah file yang menarik di direktori ID proses di dalam `/proc`. File tersebut adalah file status. File tersebut memetakan informasi *real time* setiap proses seperti nama, status proses, memory yang digunakan, pemilik proses, *mask signal*, dan lain sebagainya. Kita akan banyak memanfaatkan file ini nantinya.

Cobalah untuk memberikan perintah berikut ini untuk mendapatkan informasi dari proses `init`:

```
cat /proc/1/status
```

Berikut ini adalah contoh keluaran di sistem yang penulis gunakan untuk menulis artikel ini:

Name:	init
State:	S (sleeping)
Tgid:	1
Pid:	1
PPid:	0
TracerPid:	0
Uid:	0000
Gid:	0000
FDSize:	32
Groups:	
VmSize:	620 kB
VmLck:	0 kB

```

VmRSS:      72 kB
VmData:     136 kB
VmStk:       8 kB
VmExe:      464 kB
VmLib:       0 kB
SigPnd:0000000000000000
SigBlk:0000000000000000
SigIgn:fffffffd770d8fc
SigCgt:00000000288b2603
CapInh:0000000000000000
CapPrm:00000000ffffff
CapEff:00000000ffffff

```

Dengan program grep, kita dapat menyaring informasi dengan mudah. Sebagai contoh, kita akan mendapatkan nama proses dari proses dengan ID 1 ini.

```
cat /proc/1/status | grep Name:
```

Yang tampil adalah:

```
Name: init
```

Apabila ingin mendapatkan string `init` saja, kita perlu membuang tulisan `Name:`. Cara yang penulis gunakan adalah dengan memanfaatkan program `cut` untuk memecah string dengan pemecah berupa tanda titik dua. Berikut ini adalah contohnya:

```
cat /proc/1/status | grep Name:
| cut -d: -f2
```

Perintah tersebut akan menampilkan:

```
init
```

Apabila ingin membuang spasi di depan `init`, Anda bisa menggunakan bantuan program `tr` untuk membuang segala spasi kosong:

```
cat /proc/1/status | grep Name:
| cut -d: -f2 | tr -d
'[:space:]'
```

Perintah tersebut akan menghasilkan tulisan `init`.

Dengan demikian, kita bisa melihat betapa mudahnya mengambil informasi dari file status ini menggunakan shell script dan berbagai tool.

Berikutnya, kita akan melihat berbagai contoh lebih lanjut.

Contoh 1: prunstatus

Kita akan melihat bagaimana menghitung jumlah proses yang sedang berjalan, sedang tidur ataupun sedang dihentikan. Hal ini

umumnya kita temukan ketika kita menjalankan program `top`. Linux adalah program *multitasking* sehingga dalam satu waktu, kita bisa menemukan banyak proses. Beberapa di antaranya sedang berjalan, sedang tidur, ataupun sedang dihentikan.

Kita menyebut kondisi sedang berjalan, sedang tidur, ataupun sedang dihentikan tersebut sebagai status proses. Dalam *time-slice* (potongan waktu tertentu, umumnya dalam hitungan nanosecond atau milisecond) tertentu, sebuah proses dapat berubah status. Sistem membagi waktu dalam berbagai *timeslice*. Oleh karena itu, dalam satu waktu tertentu, kita dapat melihat beberapa proses berjalan sekaligus.

Perhatikanlah *State* pada file status di `/proc/<ID>`. Kita akan membaca semua file status di `/proc/<ID>`, kemudian menghitung berapa saja yang sedang dalam status *running*, *sleeping*, atau *stopped*.

Berikut adalah kode dan penjelasannya:

```

#!/bin/sh

#(c) Noprianto, June 2004.

TEMP=temp

for i in `ls /proc | grep -e
'[0-9]'`
do
    cat /proc/$i/status 2>/dev/
null | grep State: >> $TEMP
done

echo `cat $TEMP | wc -l` total
echo `cat $TEMP | grep sleeping
| wc -l` sleeping
echo `cat $TEMP | grep stopped
| wc -l` stopped
echo `cat $TEMP | grep running
| wc -l` running
rm -f $TEMP

```

Pertama-tama, kita mendapatkan direktori angka-angka di `/proc`. Penulis menggunakan bantuan program `grep` untuk mendapatkannya. Penulis menggunakan regular expression dengan pola `[0-9]` untuk menyaring direktori berupa angka dari keluaran program `ls`.

Setelah itu, dalam perulangan, kita akan membaca setiap file status dan menyaring

State: untuk kemudian ditambahkan ke dalam file temp. Setiap perulangan akan menghasilkan satu baris entri di dalam file temp.

Pada akhirnya, kita akan menghitung jumlah baris dengan bantuan program `wc`. Untuk mendapatkan total, kita hanya akan membaca jumlah baris tanpa melakukan penyaringan apapun. Untuk mendapatkan jumlah proses yang *sleeping*, kita akan menyaring kata *sleeping*. kemudian menghitung jumlah baris yang dihasilkan. Begitu pun dengan *stopped* dan *running*.

Berikut adalah keluaran *script* ini di komputer yang penulis gunakan:

```

222 total
221 sleeping
0 stopped
1 running

```

Contoh 2: pproglib

Setiap proses akan memanfaatkan berbagai file, program, atau pustaka sistem. Dengan mudah kita bisa mendapatkan informasi tersebut. Bedanya, kita tidak lagi membaca file status untuk mendapatkan informasi tersebut. file status hanya kita perlukan untuk mendapatkan nama proses.

Sebagai gantinya, kita akan membaca file `maps`. Kita tidak akan mengambil semua informasi di dalam file ini. Yang kita ambil adalah nama file, pustaka, atau program, yang ciri khasnya adalah diawali dengan karakter `/`.

Berikut adalah kode dan penjelasannya:

```

#!/bin/sh

[ $# -ne 1 ] && echo `basename
$0` \<pid> && exit 1

cat /proc/$1/status | grep
Name:
cat /proc/$1/maps | grep /

```

Program ini harus menerima satu parameter. Apabila jumlah parameter tidak sama dengan satu, kita akan menampilkan cara penggunaan kemudian keluar dengan exit code 1. program `basename` kita gunakan untuk memperindah tampilan. Apabila pengguna tidak memberikan parameter, maka program akan menampilkan tulisan berikut ini:

```
pproglib <pid>
```

Perhatikan bahwa untuk menampilkan karakter <, kita menggunakan cara <.

Setelah itu, kita akan mendapatkan nama proses dengan menyaring Name: dari file /proc/\$1/status. Kemudian, kita menyaring berbagai file yang terdaftar di /proc/\$1/maps. tanda \$1 menunjukkan parameter pertama yang dilewatkan pada program.

Berikut ini adalah contoh keluaran dari program dengan parameter 1321 pada komputer yang penulis gunakan:

```
Name: resmrgd
08048000-0804f000 r-xp 00000000
03:01 51281 /sbin/resmrgd
0804f000-08050000 rw-p 00007000
03:01 51281 /sbin/resmrgd
40000000-40018000 r-xp 00000000
03:01 6654 /lib/ld-
2.3.2.so
40018000-40019000 rw-p 00017000
03:01 6654 /lib/ld-
2.3.2.so
40029000-4002c000 r-xp 00000000
03:01 51278 /lib/
libresmgr.so.0.1
4002c000-4002d000 rw-p 00002000
03:01 51278 /lib/
libresmgr.so.0.1
4002d000-40159000 r-xp 00000000
03:01 6651 /lib/i686/
libc.so.6
40159000-4015e000 rw-p 0012c000
03:01 6651 /lib/i686/
libc.so.6
40160000-40167000 r-xp 00000000
03:01 6667 /lib/
libnss_compat.so.2
40167000-40168000 rw-p 00006000
03:01 6667 /lib/
libnss_compat.so.2
40168000-4017a000 r-xp 00000000
03:01 6666 /lib/
libnsl.so.1
4017a000-4017b000 rw-p 00011000
```

```
03:01 6666 /lib/
libnsl.so.1
4017d000-40185000 r-xp 00000000
03:01 6671 /lib/
libnss_nis.so.2
40185000-40186000 rw-p 00007000
03:01 6671 /lib/
libnss_nis.so.2
40186000-4018f000 r-xp 00000000
03:01 6669 /lib/
libnss_files.so.2
4018f000-40190000 rw-p 00008000
03:01 6669 /lib/
libnss_files.so.2
```

Apabila ternyata Anda merasa bahwa keluaran program terlalu kompleks, Anda dapat menyaring hanya nama file-nya saja dengan memisahkan setiap baris dengan program *cut* atau *awk* dengan pemisah berupa spasi. Anda juga dapat menggantikan spasi dengan karakter # misalnya. Harap perhatikan bahwa Anda tidak dapat menggunakan karakter : sebagai pemisah karena telah digunakan pada informasi waktu proses.

Salah satu kekurangan dari program ini adalah tidak ada validasi ketika membaca direktori ID proses. Apabila user memasukkan ID yang tidak valid, maka pesan kesalahan default akan ditampilkan. Untuk mengujinya, Anda bisa menggunakan program test dengan parameter -d yang akan menguji apakah sebuah file ada dan merupakan direktori.

Contoh 3: pname

Apakah Anda sering menjumpai kasus di mana Anda ingin mengetahui PID dari semua proses java di komputer Anda? Misalnya ketika Anda ingin melewatkan semua PID tersebut dengan signal SIGKILL pada program *kill*? Apabila sering mengalaminya, Anda mungkin tertarik pada contoh program yang satu ini.

Contoh yang satu ini lebih mengacu pada latihan pemrograman. Untuk tugas bunuh-membunuh, Anda bisa melihat manual program *kill* ataupun *pkill*.

```
#!/bin/sh
```

```
##(c) Noprianto, June 2004.
```

IKLAN

<http://www.distrolinux.net>

Sedia CD Distro Linux & BSD
Murah, Bergaransi (10Rb/CD)
Email : info@distrolinux.net
HP/SMS : 0812 1876 981

```
[ $# -ne 1 ] && echo 'basename
$0' <\name\> && exit 1

export PIDS=""

for i in `ls /proc | grep -e
'[0-9]'`
do
    TEMP='cat /proc/$i/status 2>/
dev/null | grep $1 | tr -s
[:space:]' ':' | cut -d: -f2'
    test ! -z $TEMP && test $TEMP
= $1 && PIDS="$PIDS $i"
done
echo $PIDS
```

Program ini juga menerima sebuah parameter. Karena tujuan dari program ini adalah mendapatkan semua PID dari nama proses, maka program ini menerima parameter berupa nama. Apabila pengguna tidak melewati parameter, maka pesan kesalahan berikut akan ditampilkan:

```
pname <name>
```

Sama seperti contoh sebelumnya, kita menggunakan `basename` untuk memperindah tampilan. Exit code 1 akan dikembalikan apabila terjadi kesalahan.

Kita memiliki sebuah variabel `PIDS` yang nilainya kosong ketika program dijalankan. Setelah itu, kita melakukan perulangan pada direktori-direktori angka di dalam `/proc` dan mendapatkan PID proses.

Untuk mendapatkan PID proses, kita akan menyaring parameter pertama dalam file status di semua direktori angka. Harap diperhatikan bahwa kita mungkin akan mendapatkan nilai kosong karena tidak semua file status akan berisikan *string* yang ingin kita cari.

Apabila pencarian berhasil, variabel `TEMP` akan berikan string berupa nama proses. Kita telah membuang spasi yang tidak diperlukan dengan program `tr`.

Sebagai contoh, kita menyaring kata `java`. Tentu saja, tidak semua file status akan berisikan kata `java`. Maka, kita perlu memeriksa variabel `TEMP` yang kita gunakan sebagai penampung tersebut.

Apabila `TEMP` kosong, maka kita asumsikan bahwa direktori tersebut bukan direktori yang kita inginkan. Oleh karena itu, perulangan dilanjutkan. Tapi, apabila `TEMP` berisikan nilai yang kita inginkan,

maka kita akan menambahkan direktori tersebut dalam daftar `PIDS`.

Pada akhirnya, kita hanya perlu menampilkan variabel `PIDS` yang telah berisikan daftar PID untuk nama program yang kita lewatkan.

Berikut ini adalah contoh keluaran script pada komputer yang penulis gunakan apabila diberikan parameter `java`:

```
21979 21980 21981 21982 21983
21984 21985 21986 21987 21989
21990 21991 21992 21994 21995
22005 22014 24753 24754 24755
24756 24757 24758 24759 24760
24761 24762 24763 24764 24765
24766 24767 24769 24789 24790
24791 24797 26535 26550 26563
```

Kenapa program ini menampilkan hasil yang sederhana? Tujuannya adalah supaya program ini dapat digunakan bersama program `kill`. Sebagai contoh:

```
kill './pname vi'
```

Kelemahan utama dari program ini adalah lambat. Hal tersebut disebabkan karena kita mencari PID proses secara brute force.

Contoh 4: puser

Linux adalah sistem operasi yang didesain dengan memperhatikan konsep keamanan. Oleh karena itulah, penerapan sisi keamanan dapat ditemukan di hampir semua tempat di Linux. Termasuk proses. Contoh berikut ini akan menampilkan user yang memiliki proses tersebut.

File yang kita gunakan adalah file status. Perhatikanlah `Uid`: di dalam file ini. Kita akan mengambil daftar `Uid` dan melihat ke dalam `/etc/passwd` untuk mendapatkan nama user.

Berikut ini adalah kode dan penjelasan:

```
#!/bin/sh

#(c) Noprianto, June 2004.

[ $# -lt 1 ] && echo 'basename
$0' <\pid\> [-v] && exit 1

export USERS=""

USERS='cat /proc/$1/status 2>/
dev/null | grep Uid: | tr -s
[:space:]' ':' | cut -d: -f2-
5'
```

```
USERS='echo $USERS | tr ':' '
''

for user in $USERS
do
    cat /etc/passwd | grep -w
$user | cut -d: -f1
done

export COUNT='echo $USERS | tr
[:space:]' '\n' | uniq | wc -
l'

test ! -z $2 && test $2 = -v &&
echo -'echo $COUNT | wc -w'
users --
```

program ini dapat menerima dua parameter. Parameter pertama adalah parameter wajib dan parameter kedua adalah parameter opsional. Parameter pertama adalah `pid`, dan parameter kedua adalah *verbose*. Apabila parameter kedua dilewatkan, nama kita akan menampilkan informasi ekstra.

Kemudian, kita memiliki sebuah variabel `USERS` yang bernilai kosong pada saat program dijalankan. variabel ini akan menampilkan user-user yang memiliki proses yang dimasukkan oleh pengguna.

Pertama-tama, kita menyaring terlebih dahulu `Uid` di file status. Kemudian, kita menghapus spasi, menggantikan spasi dengan tanda `:`, memisahkan berdasarkan tanda `:` tersebut dan mengambil field kedua sampai kelima. Setelah itu, kita mengganti kembali pemisah `:` menjadi karakter spasi di variabel `USERS`.

Dengan menggunakan perulangan, kita akan mencari ke dalam `/etc/passwd` dan langsung menampilkannya.

Setelah itu, kita menghitung jumlah user unik yang kita dapatkan. Untuk itu, penulis mencetak isi variabel `USERS`, mengganti spasi dengan karakter newline (`\n`), mendapatkan nilai unik dengan program `uniq` dan mengetahui berapa baris dengan program `wc -l`.

Kita akan memeriksa apabila parameter kedua dilewatkan. caranya, dengan bantuan program `test`, kita menguji apakah isi variabel kedua kosong atau berisikan nilai tertentu. Apabila berisikan nilai tertentu, kita akan memeriksa apakah nilainya sama dengan `-v`. Apabila sama, kita akan mencetak jumlah user unik yang telah kita dapatkan tersebut.

Berikut adalah contoh keluaran program apabila dua parameter dilewatkan:

```
root
root
root
root
- 1 users --
```

Salah satu kelemahan program ini adalah tidak ada pemeriksaan bahwa suatu PID benar-benar valid atau tidak.

Contoh 5: pchildren

Proses di Linux memiliki hirarki yang jelas. Nenek moyang proses adalah proses init (proses dengan PID 1). Setelah itu, juga terdapat anak-anak proses. Kemudian, ada pula cucu-cucu proses. Umumnya, hubungan orang tua dan anak adalah hubungan yang paling diperhatikan.

Terdapat sebuah program yang sangat berguna untuk melihat hirarki proses dalam bentuk tree. Program tersebut adalah pstree. Dalam banyak kasus, pstree sangat membantu.

Namun, apakah Anda tertarik untuk mendapatkan PID anak-anak suatu proses? Sebagai contoh, pernahkah Anda tertarik untuk membunuh semua anak proses dari sebuah proses? Apabila Ya, mungkin Anda akan tertarik dengan contoh program berikut. Program berikut akan menampilkan PID semua anak proses dari proses tertentu.

Berikut adalah kode dan penjelasannya:

```
#!/bin/sh

#(c) Noprianto, June 2004.

[ $# -lt 1 ] && echo 'basename $0' \<pid\> [-v] && exit 1

export CHILDREN=""

for i in `ls /proc | grep -e '[0-9]'`
do
    TEMP=`cat /proc/$i/status 2>/dev/null | grep PPid: | tr -s '[:space:]' ':' | cut -d: -f2`
    test ! -z $TEMP && test $TEMP
```

```
= $1 && CHILDREN="$CHILDREN $i"
done

echo $CHILDREN

test ! -z $2 && test $2 = -v &&
echo -'echo $CHILDREN | wc -w'
children --
```

Program ini menerima dua parameter. Parameter pertama adalah parameter wajib dan parameter kedua adalah parameter opsional. parameter pertama adalah PID orang tua dan parameter kedua adalah verbose.

Kita akan memiliki variabel CHILDREN yang kosong ketika program dijalankan.

Pertama-tama, kita akan mendapatkan semua direktori angka dan melakukan perulangan di semua direktori tersebut sambil membaca file status dan menyaring kata PPid: Parent Process ID, atau PID milik orang tua proses. karena kita akan mencari anak-anak proses, maka kita akan menyaring proses-proses yang orang tuanya adalah PID yang dilewatkan sebagai parameter pertama.

IKLAN

Kita menghapus spasi dan mengambil hanya PPid proses tersebut. Kita juga memeriksa apakah PPid yang didapatkan sama dengan parameter pertama. Apabila ya, maka kita akan menambahkan direktori tersebut sebagai isi dari variabel CHILDREN.

Setelah itu, kita hanya perlu menampilkan isi dari variabel children. Apabila pengguna memberikan parameter kedua, maka informasi tambahan akan diberikan. Pertama-tama, akan diperiksa dahulu apakah parameter kedua kosong atau tidak. Apabila berisikan nilai, maka akan diperiksa apakah nilainya sama dengan -v. Apabila benar, maka jumlah kata dalam variabel CHILDREN akan dihitung dengan bantuan program wc -w. Jumlah kata yang didapatkan adalah jumlah anak-anak proses.

Berikut ini adalah contoh keluaran dari program ini:

```
1007 1066 12 1321 1373 1402
1439 1440 1491 1597 1607 1689
1833 1834 1835 1836 1838 1870
1941 2 2183 2190 2205 2234 2235
2250 2374 2428 2429 2430 2431
2432 27888 27891 27894 27897
27907 27910 27913 27915 27918
27924 27926 27931 27932 27951 3
30062 31110 31111 3202 373 3918
4 417 5 6 7 723 724 730 8 811 9
992
- 65 children --
```

Contoh 6: mytop

Program top akan melihat proses-proses yang paling banyak memakan *resource* sistem. Program top adalah program yang sangat luar biasa. Kita akan membuat program top sederhana dengan shell script, yang akan menampilkan proses-proses yang paling banyak memakan memory.

Program ini jelas masih kalah jauh dibandingkan dengan program top. Pengertian paling banyak memakan memory juga dilakukan secara sederhana. Berikut ini adalah kode dan penjelasannya:

```
#!/bin/sh

#(c) Noprianto, June 2004

export LONGDATA=""

test -z $1 && COUNT=10 ||
COUNT=$1
```

```
for i in `ls /proc | grep -e
'[0-9]'`
do
    NAME=`cat /proc/$i/status 2>/
dev/null | grep Name: | tr -s
[:space:] ' ' | cut -d: -f2`
    VMSIZE=`cat /proc/$i/status
2>/dev/null | grep VmSize: | tr
-s [:space:] ' ' | cut -d: -
f2`
    DATA="$VMSIZE:$NAME"
    test ! -z $NAME &&
    LONGDATA="$LONGDATA $DATA"
done

LONGDATA=`echo $LONGDATA | tr
[:space:] '\n' | sort -nr`
LONGDATA=`echo $LONGDATA | tr
[:space:] '\n' | uniq `

COUNTER=0
for i in $LONGDATA
do
    let COUNTER=$COUNTER+1
    echo `echo $i | tr ':' '\t'`
    [ $COUNTER -eq $COUNT ] &&
    break
done
```

Program ini dapat menerima satu parameter opsional yang akan diartikan jumlah proses yang akan ditampilkan. Pertama-tama, kita akan memeriksa apabila parameter pertama kosong atau memiliki nilai. Apabila kosong, maka variabel COUNT akan diset secara *default* menjadi 10. Apabila berisikan nilai, maka variabel COUNT akan diset sesuai nilai yang diberikan.

Kita memiliki variabel LONGDATA yang akan menampung seluruh proses dan memory yang digunakan.

Untuk mendapatkan memory proses, kita akan menyaring informasi VmSize. Setelah itu, kita akan membangun struktur sederhana berupa VmSize:Name. Kemudian, apabila Name memiliki nilai, kita akan menggabungkan struktur VmSize:Name tersebut ke dalam variabel LONGDATA.

Setelah LONGDATA berisikan nilai dan perulangan selesai, maka kita akan menurutkan terbalik (dari besar ke kecil) dengan bantuan program sort (yang diberikan opsi -n untuk pengurutan bilangan). Setelah itu,

untuk menghilangkan entri ganda, kita akan meminta bantuan program uniq.

Terakhir, kita akan melakukan perulangan untuk menampilkan isi dari LONGDATA tersebut sesuai isi variabel COUNT. Apabila COUNTER telah mencapai COUNT, maka perulangan dihentikan. Dalam menampilkan isi dari LONGDATA, kita menggantikan : dengan TAB dengan bantuan program tr.

Salah satu kelemahan program ini yang paling terasa adalah lambat. Selain itu, masih ada nama proses yang berganda. Hal ini dapat dihilangkan, namun struktur dalam LONGDATA mungkin harus diubah.

Berikut ini adalah contoh keluaran dari program ini:

```
357432 java
179748 kernel
29364 kdeinit
29356 kdeinit
27908 kdeinit
27116 suseplugger
26860 httpd2-prefork
26676 kdeinit
25916 kdeinit
25628 kdeinit
```

Program *mytop* dapat dijalankan dengan bantuan program *watch* untuk mendapatkan informasi secara periodik.

Demikianlah pengenalan kita dengan informasi-informasi proses di Linux. Contoh-contoh dalam artikel ini lebih berfungsi untuk mempelajari shell script daripada menghasilkan program yang digunakan pada lingkungan kerja produktif.

Satu hal yang perlu diperhatikan secara umum adalah bahwa informasi proses berubah sesuai kondisi sistem. Oleh karena itu, kita tidak dapat mengandalkan pembacaan informasi proses di /proc dalam waktu yang berbeda. Pada contoh terakhir misalnya, *mytop*, kita tidak bisa mengambil semua informasi VmSize, diurutkan, dan barulah mencari nama proses sesuai VmSize. Takutnya, pada saat kita ingin mencari entri yang sesuai dengan VmSize, entri yang bersangkutan telah menggunakan VmSize yang berbeda.

Kita bisa melihat bahwa Linux adalah sistem operasi luar biasa. Masih banyak yang bisa dikerjakan dengan membaca /proc. Selamat mencoba dan sukses!

Noprianto (noprianto@infolinux.co.id)