VERSI 1.0

APRIL, 2023

PRAKTIKUM SISTEM OPERASI

Tuning System Performance

TIM PENYUSUN:

MAHAR FAIQURAHMAN, S.KOM., M.T.

MUHAMMAD RIDHA AGAM

SYAHRUL PANGESTU

MADE WITH PRIDE BY: LAB. INFORMATIKA

UNIVERSITAS MUHAMMADIYAH MALANG

MODUL 3 SISTEM OPERASI – TUNING SYSTEM PERFORMANCE

CAPAIAN PEMBELAJARAN MATA KULIAH

- Mengontrol dan menghentikan suatu proses yang tidak berhubungan dengan *shell*, mengakhiri *session* dan proses *user* secara paksa.
- Mendeskripsikan apa itu *load average* dan menentukan proses yang bertanggung jawab atas penggunaan sumber daya *server/pc* yang tinggi.
- Mengoptimasi performa sistem dengan memilih *profil tuning* yang dikelola oleh *tuned daemon*.
- Memprioritaskan atau de-prioritas proses secara spesifik, dengan *command* yang halus dan kasar.

KEBUTUHAN HARDWARE & SOFTWARE

- Laptop/PC
- Virtual Machine (VMware, Virtual Box, VPC bila ekonomi diatas rata-rata :")
- Sistem Operasi CentOS, download image OVA (Wajib)

MATERI PRAKTIKUM

- Killing Process
- Monitoring Process Activity
- Adjusting Tuning Profiles
- Influencing Process Scheduling

MATERI PRAKTIKUM

1. Killing Process

Signal adalah interupsi perangkat lunak yang dikirim ke proses. Signal melaporkan event ke sebuah executing program. Event yang menghasilkan *signal* dapat menjadi *error*, *external* event (*request* I/O atau *timer* yang kedaluwarsa), atau dengan perintah pengiriman sinyal atau rangkaian keyboard secara eksplisit.

Tabel berikut mencantumkan fundamental signal yang digunakan oleh sistem administrator untuk manajemen proses secara rutin. Mengacu pada sinyal (HUP) atau (SIGHUP).

Fundamental Process Management Signal:

Signal Number	Short Name	Definisi	Tujuan
1	SIGHUP	Hangup	Digunakan untuk melaporkan pemberhentian (*termination*) proses controlling terminal. Digunakan juga untuk melakukan request proses *re-initialization* (*configuration reload*) tanpa pemberhentian (*termination*).

2	SIGINT	Keyboard Interrupt	Menyebabkan pemberhentian (*termination*) program. dapat diblokir atau di-handle. Signal dikirimkan dengan press **INT** key sequence (Ctrl+c).
3	SIGQUIT	Keyboard Quit	Hampir sama dengan Signal INT; menambahkan proses *dump* pada pemberhentian (*termination*). *Signal* dikirimkan dengan menekan key sequence (Ctrl+\).
9	SIGKILL	Kill, unblockable	Menyebabkan pemberhentian (*termination*) program secara tiba-tiba. Tidak dapat diblokir, diabaikan, atau di-handle.
15 default	SIGTERM	Terminate	Menyebabkan pemberhentian (*termination*) program. Berbeda dengan Signal **KILL**, **TERM** dapat diblokir, diabaikan, atau ditangani. Cara yang baik atau sopan untuk meminta program berhenti.
18	SIGCONT	Continue	Dikirim ke proses untuk melanjutkan kembali proses, jika suatu proses dihentikan. Tidak dapat diblokir.
19	SIGSTOP	Stop, unblockable	Menangguhkan proses. tidak dapat diblokir atau di-handle.
20	SIGTSTP	Keyboard stop	Berbeda dengan Signal STOP, Signal TSTP dapat diblokir, diabaikan, ataupun dihandle. *Signal* dikirimkan dengan menekan **SUSP** key sequence **(Ctrl + z)**

Setiap signal mempunyai action default. beberapa diantaranya seperti berikut:

- Term - Menyebabkan pemberhentian (*terminate*) program (*exit*) sekali.
- Core - Menyebabkan program untuk menyimpan sebuah *image memory* (*core dump*), untuk nantinya diberhentikan.
- Stop - Menyebabkan program berhenti dieksekusi (*suspend*) dan menunggu untuk lanjut (*resume*).

Program dapat disiapkan untuk mereaksi event signal yang diharapkan dengan mengimplementasikan handler routines untuk mengabaikan, mengganti, atau memperpanjang default action milik signal.

Command untuk mengirim Signal dengan Explicit Request

Kita dapat mengirimkan *signal* untuk program yang berjalan pada *foreground* saat ini dengan menekan keyboard *control sequence* untuk *suspend* **(Ctrl+z)**, *kill* **(Ctrl+c)**, atau *core dump* prosesnya **(Ctrl+\)**. Selain itu, kita akan menggunakan *signal-sending command* untuk mengirim *signal* ke sebuah *background process* atau untuk memproses pada *session* yang berbeda.

Signal dapat ditentukan sebagai opsi, entah dengan nama (contohnya -HUP atau -SIGHUP) atau dengan nomor (yang berhubungan dengan -1). *User* mungkin melakukan *kill* terhadap proses yang miliki, tetapi *privilege root* dibutuhkan untuk melakukan *kill process* milik orang lain.

Command kill mengirim signal ke sebuah proses dengan nomor PID. Meskipun demikian, *kill command* dapat digunakan untuk mengirim *signal* apa pun, bukan hanya menghentikan program saja. Kita dapat menggunakan *command* **kill -l** untuk menampilkan nama dan nomor dari semua *signal* yang tersedia.

```
[user@host ~]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
...output omitted...
[user@host ~]$ ps aux | grep job
5194  0.0  0.1 222448  2980 pts/1    S    16:39   0:00 /bin/bash /home/user/bin/
control job1
5199  0.0  0.1 222448  3132 pts/1    S    16:39   0:00 /bin/bash /home/user/bin/
control job2
5205  0.0  0.1 222448  3124 pts/1    S    16:39   0:00 /bin/bash /home/user/bin/
control job3
5430  0.0  0.0 221860  1096 pts/1    S+   16:41   0:00 grep --color=auto job
[user@host ~]$ kill 5194
[user@host ~]$ ps aux | grep job
user   5199  0.0  0.1 222448  3132 pts/1    S    16:39   0:00 /bin/bash /home/
user/bin/control job2
user   5205  0.0  0.1 222448  3124 pts/1    S    16:39   0:00 /bin/bash /home/
user/bin/control job3
user   5783  0.0  0.0 221860   964 pts/1    S+   16:43   0:00 grep --color=auto
 job
[1]   Terminated              control job1
[user@host ~]$ kill -9 5199
[user@host ~]$ ps aux | grep job
user   5205  0.0  0.1 222448  3124 pts/1    S    16:39   0:00 /bin/bash /home/
user/bin/control job3
user   5930  0.0  0.0 221860  1048 pts/1    S+   16:44   0:00 grep --color=auto
 job
[2]-  Killed                  control job2
[user@host ~]$ kill -SIGTERM 5205
user   5986  0.0  0.0 221860  1048 pts/1    S+   16:45   0:00 grep --color=auto
 job
[3]+  Terminated              control job3
```

Command **killall** dapat memberikan *signal* ke banyak proses, berdasarkan nama *command*-nya.

```
[user@host ~]$ ps aux | grep job
5194  0.0  0.1 222448  2980 pts/1    S    16:39   0:00 /bin/bash /home/user/bin/
control job1
5199  0.0  0.1 222448  3132 pts/1    S    16:39   0:00 /bin/bash /home/user/bin/
control job2
5205  0.0  0.1 222448  3124 pts/1    S    16:39   0:00 /bin/bash /home/user/bin/
control job3
5430  0.0  0.0 221860  1096 pts/1    S+   16:41   0:00 grep --color=auto job
[user@host ~]$ killall control
[1]   Terminated              control job1
[2]-  Terminated              control job2
[3]+  Terminated              control job3
[user@host ~]$
```

Gunakan **pkill** untuk mengirim *signal* ke salah satu atau lebih proses yang cocok dengan kriteria yang dipilih. Memilih kriteria sendiri dapat menggunakan sebuah nama pada *command*, sebuah proses dimiliki oleh secara spesifik oleh seorang user, atau semua proses secara keseluruhan. *command* **pkill** menyediakan *advanced selection criteria*:

- **Command** : proses dengan *pattern-matched* nama *command*.
- **UID** : Proses yang dimiliki oleh pemilik akun linux.
- **GID** : Proses yang dimiliki oleh *group* akun linux.
- **Parent** : Proses *child* dari proses *parent* tertentu.
- **Terminal** : Proses yang berjalan pada *controlling* terminal tertentu.

```
[user@host ~]$ ps aux | grep pkill
user   5992  0.0  0.1 222448  3040 pts/1    S    16:59   0:00 /bin/bash /home/
user/bin/control pkill1
user   5996  0.0  0.1 222448  3048 pts/1    S    16:59   0:00 /bin/bash /home/
user/bin/control pkill2
user   6004  0.0  0.1 222448  3048 pts/1    S    16:59   0:00 /bin/bash /home/
user/bin/control pkill3
[user@host ~]$ pkill control
[1]   Terminated              control pkill1
[2]-  Terminated              control pkill2
[user@host ~]$ ps aux | grep pkill
user   6219  0.0  0.0 221860  1052 pts/1    S+   17:00   0:00 grep --color=auto
 pkill
[3]+  Terminated              control pkill3
[user@host ~]$ ps aux | grep test
user   6281  0.0  0.1 222448  3012 pts/1    S    17:04   0:00 /bin/bash /home/
user/bin/control test1
```

```
user   6285  0.0  0.1 222448  3128 pts/1    S    17:04   0:00 /bin/bash /home/
user/bin/control test2
user   6292  0.0  0.1 222448  3064 pts/1    S    17:04   0:00 /bin/bash /home/
user/bin/control test3
user   6318  0.0  0.0 221860  1080 pts/1    S+   17:04   0:00 grep --color=auto
 test
[user@host ~]$ pkill -U user
[user@host ~]$ ps aux | grep test
user   6870  0.0  0.0 221860  1048 pts/0    S+   17:07   0:00 grep --color=auto
 test
[user@host ~]$
```

Logging Users out Administratively

Temukan berapa lama pengguna yang telah berada di dalam sistem untuk melihat waktu login session. Untuk setiap session, *resource* CPU yang digunakan oleh *jobs* saat ini, *jobs* pada *background* dan *child* proses dapat dilihat di kolom JCPU. Konsumsi CPU yang dikonsumsi proses pada *foreground* saat ini berada di kolom PCPU.

Proses dan sesi dapat ditandai secara individual atau kolektif. Untuk menghentikan semua proses yang dijalankan oleh seorang pengguna, dapat menggunakan perintah **pkill**. Karena initial proses dalam login session dirancang untuk menghandle permintaan penghentian *session* dan mengabaikan keyboard *signal* yang tidak diinginkan. Sedangkan untuk menghentikan semua proses dan *login shell* dapat menggunakan **Signal KILL**.

Pertama identifikasi nomor PID yang ingin di kill menggunakan **pgrep**, yang mana **pgrep** beroperasi seperti **pkill**, termasuk menggunakan *option* yang sama, kecuali perintah **pgrep** yang menjabarkan proses dibanding mengkilling nya.

```
[root@host ~]# pgrep -l -u bob
6964 bash
6998 sleep
6999 sleep
7000 sleep
[root@host ~]# pkill -SIGKILL -u bob
[root@host ~]# pgrep -l -u bob
[root@host ~]#
```

Terminate dengan menggunakan *prosess selective* yang sama juga dapat menggunakan hubungan *parent* dan *child*. Menggunakan *command* **pstree** untuk melihat proses *tree* dari sistem maupun *single user*. Gunakan PID process milik *parent* untuk *kill* semua *childern* proses yang telah dibuat. Kali ini, *parent* bash *login shell* dapat bertahan karena *signal* diarahkan hanya pada *child* prosesnya.

```
[root@host ~]# pstree -p bob
bash(8391)─┬─sleep(8425)
           ├─sleep(8426)
           └─sleep(8427)
[root@host ~]# pkill -P 8391
[root@host ~]# pgrep -l -u bob
bash(8391)
[root@host ~]# pkill -SIGKILL -P 8391
[root@host ~]# pgrep -l -u bob
bash(8391)
[root@host ~]#
```

2. Monitoring Process Activity

Describing Load Average

Load average merupakan alat ukur yang disediakan oleh Linux Kernel yang mana ini merupakan langkah sederhana untuk merepresentasikan beban sistem dari waktu ke waktu. Ini dapat digunakan sebagai *rough gauge* dari banyaknya *request* ke system resource yang pending, dan untuk memastikan apakah beban sistem meningkat atau menurun dari waktu ke waktu.

Setiap lima detik, kernel mengumpulkan *load number* saat ini berdasarkan jumlah proses di dalam *runnable* dan *interruptible state*. Jumlah ini diakumulasi dan dilaporkan sebagai pergerakan rata-rata secara eksponensial selama 1, 5, dan 15 menit.

Understanding the Linux Load Average Calculation

Load Average merepresentasikan beban sistem yang dibebankan selama periode waktu tertentu. Linux menanggapi hal ini dengan melaporkan banyaknya jumlah proses yang siap dijalankan oleh CPU, dan berapa banyaknya proses yang menunggu *disk* atau I/O jaringan selesai.

- Jumlah *load* merupakan jumlah rata-rata proses yang sudah berjalan (pada state proses R) atau menunggu untuk I/O selesai.
- Beberapa sistem UNIX hanya memikirkan pemanfaatan CPU atau menjalankan panjang antrian untuk merepresentasikan system load. Linux juga memuat pemanfaatan disk atau jaringan, karena sangat berdampak terhadap performa sistem sebagai CPU load. Ketika mengalami rata-rata load yang tinggi dengan aktifitas CPU yang minim, periksa disk dan aktifitas jaringan.

Load average merupakan pengukur yang buruk, karena dari banyaknya *request* saat ini yang menunggu agar selesai sebelum dapat melakukan hal lain. Ini terjadi karena request mungkin membutuhkan waktu CPU untuk menjalankan proses. Alhasil, request menjadi *critical* untuk diselesaikan dalam operasi *disk I/O*, dan proses tidak dapat dijalankan pada CPU sebelum selesai, meskipun ketika kondisi CPU idle atau nganggur :”. Meskipun demikian, sistem *load* berpengaruh dan sistem menunjukan agar berjalan dengan pelan karena proses menunggu untuk dijalankan.

Interpreting Displayed Load Average Values

Command **uptime** merupakan satu satunya cara untuk menampilkan load average saat ini. **uptime** menampilkan waktu saat ini, seberapa lama mesin dinyalakan, banyaknya session user yang berjalan, dan load average saat ini.

```
[user@host ~]$ uptime
 15:29:03 up 14 min,  2 users,  load average: 2.92, 4.48, 5.20
```

Tiga nilai *load average* merepresentasikan load selama 1, 5, dan 15 menit terakhir. Sekilas menunjukan sistem load mana yang menunjukan kenaikan atau penurunan.

Jika kontribusi utama untuk *load average* berasal dari proses menunggu CPU, kita dapat menghitung perkiraan *load value* per CPU untuk menunjukan sistem mana yang mengalami penungguan yang signifikan.

Command **lscpu** dapat membantu menentukan berapa banyak CPU yang dimiliki sistem.

Pada contoh berikut, sistem memiliki dual-core single socket dengan dua *hyper threads* per core. Secara kasar, linux akan memperlakukan ini sebagai empat sistem CPU untuk tujuan penjadwalan.

```
[user@host ~]$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  2
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
...output omitted...
```

Sejenak, bayangkan satu satunya kontribusi ke jumlah load adalah dari proses yang membutuhkan waktu CPU. Maka kita dapat membagi load average yang ditampilkan dengan jumlah dari logical CPU di dalam sistem. Nilai 1 di bawah menunjukkan penggunaan sumber daya yang memuaskan dan waktu tunggu yang minimal. Nilai 1 di atas menunjukkan saturasi sumber daya dan sejumlah penundaan pemrosesan.

```
# From lscpu, the system has four logical CPUs, so divide by 4:
#                               load average: 2.92, 4.48, 5.20
#          divide by number of logical CPUs:    4     4     4
#                                             ----  ----  ----
#                     per-CPU load average: 0.73  1.12  1.30
#
# This system's load average appears to be decreasing.
# With a load average of 2.92 on four CPUs, all CPUs were in use ~73% of the time.
# During the last 5 minutes, the system was overloaded by ~12%.
# During the last 15 minutes, the system was overloaded by ~30%.
```

Sebuah CPU yang idle mempunyai load number 0. Setiap proses yang menunggu CPU menambahkan hitungan 1 ke nomor beban. Jika satu proses berjalan pada CPU, maka load number-nya adalah satu dan *resource* (CPU) sedang digunakan, tetapi tidak ada permintaan yang menunggu. Jika proses itu berjalan selama satu menit penuh, kontribusi terhadap *load average* satu menit adalah 1.

Namun, proses tidur tanpa henti untuk *critical I/O* disebabkan karena *disk* yang sibuk atau *resource* jaringan juga termasuk dalam hitungan dan meningkatkan rata-rata beban. Meskipun bukan merupakan indikasi penggunaan CPU, proses ini ditambahkan ke hitungan antrian karena mereka menunggu sumber daya dan tidak dapat berjalan pada CPU sampai mereka mendapatkannya. Ini masih *system load*, mengingat keterbatasan sumber daya lah yang menyebabkan proses tidak berjalan.

nomor tools tambahan melaporkan *load average* termasuk **w** dan **top**.

Real-time Process monitoring

Program top merupakan tampilan dinamis dari proses sistem, menampilkan ringkasan *header* yang diikuti dengan daftar proses atau *thread* yang mirip dengan informasi **ps**. Tidak seperti *output* **ps** statis, **top** terus-menerus menyegarkan pada interval yang dapat diatur, dan menyediakan kemampuan untuk menyusun ulang, menyortir, dan menyorot kolom. Konfigurasi user dapat disimpan dan dibuat secara persisten.

Default output program top yang ditampilkan memuat beberapa resource lain seperti:

- Process ID (PID)
- Nama user (USER) merupakan pemilik proses.
- Virtual Memory merupakan semua proses memori yang digunakan, termasuk resident set, shared libraries, dan halaman memori apa pun yang dipetakan atau di-swap. (Berlabel VSZ di *command* ps.)
- Resident memory (RES) merupakan memori fisik yang digunakan oleh proses, termasuk setiap resident shared objects. (Berlabel RSS di *command* ps.)
- Proses state (S) ditampilkan sebagai:
    - D = Uninterruptible Sleeping
    - R = Running or runnable
    - S = Sleeping
    - T = Stopped or Traced
    - Z = Zombie
- CPU time (TIME) merupakan jumlah total waktu pemrosesan dari awal dimulai. Dapat dialihkan untuk menampung waktu kumulatif dari semua children sebelumnya.
- Nama proses *command* (*COMMAND*)

Fundamental Keystroke pada top

Key	Tujuan
? atau h	Membantu interaktif keystrokes
l, t, m	Toggle agar dapat beralih ke load, threads, dan memory header lines
1	Toggle agar dapat beralih menampilkan CPU milik individu atau merangkum semua CPU di header.
s(1)	Mengubah kecepatan refresh (layar), dalam detik (mis., 0, 5, 1, 5).

b	Toggle agar dapat beralih membalikan highlight untuk running proses.
shift+b	Memungkinkan untuk menggunakan bold pada display, di dalam header, dan untuk menjalankan proses.
u, shift+h	Toggle threads; menampilkan rangkuman proses atau threads milik perseorangan.
shift+u	Memfilter nama user apa pun.
shift+m	Memilah daftar proses berdasarkan penggunaan memori, dalam urutan tinggi ke rendah.
shift+p	Memilah daftar proses berdasarkan pemanfaatan proses, dalam urutan tinggi ke rendah.
k(1)	Kill sebuah proses. Ketika digunakan masukan PID, dan selanjutnya masukan signal.
r(1)	Renice sebuah proses. Ketika digunakan masukan PID, dan selanjutnya masukan signal.
shift+w	Menulis konfigurasi tampilan saat ini untuk nantinya digunakan top restart selanjutnya.
q	Keluar (Quit)
f	Mengelola kolom dengan memungkinkan atau melarang field. Juga mengizinkan kita untuk mengatur urutan field top.
Note:	(1) Tidak tersedia jika top dimulai ketika berada pada secure mode.

3. Adjusting Tuning Profiles

Tuning Systems

Sistem administrator dapat mengoptimasi performa sebuah sistem dengan menyesuaikan pengaturan pada beberapa macam perangkat berdasarkan beban kerja yang dibebankan. **Tuned daemon** menyesuaikan *tuning* baik secara statis maupun dinamis, dengan menggunakan profil *tuning* yang merefleksikan kebutuhan beban kerja tertentu.

Configuring Static Tuning

Tuned daemon menerapkan pengaturan sistem ketika layanan dimulai atau waktu memilih profil *tuning* baru. *Static tuning* mengkonfigurasi parameter kernel pada profil yang menerapkan *tuning* pada runtimenya. Dengan *static tuning*, parameter kernel mengatur performa keseluruhan yang diinginkan dan bukan menyesuaikan perubahan pada level aktivitas.

Configuring Dynamic Tuning

Dengan *dynamic tuning*, *tuned daemon* memantau aktifitas sistem dan menyesuaikan pengaturan berdasarkan perubahan sifat *runtime*. *Dynamic tuning* selalu menyesuaikan penyetelan agar

sesuai dengan beban kerja saat ini secara terus menerus, dimulai dengan pengaturan awal yang dideklarasikan di dalam profil tuning yang dipilih.

Contohnya, perangkat penyimpanan mengalami peningkatan saat digunakan startup dan login, namun mempunyai aktivitas yang rendah ketika beban kerja user yang terdiri dari penggunaan web browser dan email *client*. Hampir sama dengan CPU dan perangkat jaringan mengalami peningkatan aktivitas saat berada pada penggunaan tertinggi sepanjang waktu. *Tuned daemon* memantau aktifitas komponen ini dan menyesuaikan pengaturan parameter agar memaksimalkan performa ketika berada pada saat jumlah waktu aktivitas yang padat dan mengurangi pengaturan ketika aktivitas rendah. *Tune daemon* menggunakan performa parameter yang disediakan di dalam *tuning profile*.

Installing and enabling tuned

Image centos yang disediakan sudah memuat dan menyediakan *tuned package* secara default. Untuk menginstall dapat dilakukan sebagai berikut:

```
[root@host ~]$ yum install tuned
[root@host ~]$ systemctl enable --now tuned
Created symlink /etc/systemd/system/multi-user.target.wants/tuned.service → /usr/
lib/systemd/system/tuned.service.
```

Selecting a Tuning Profile

Tuned application menyediakan *profile* yang dibagi menjadi beberapa kategori:

- Power-saving profiles
- Performance-boosting profiles

Profil performance-boosting memuat profil yang berfokus pada aspek berikut ini:

- Latency yang rendah untuk storage dan jaringan.
- Throughput tinggi untuk storage dan jaringan.
- Virtual machine performance
- Virtualisasi host performance

Tuning Profiles Distributed with Red Hat Enterprise Linux 8

Tuned Profile	Tujuan
balanced	Ideal untuk sistem yang membutuhkan compromise antara penghematan sumber daya dan performa.
desktop	Diturunkan dari profil balance. Menyediakan respon yang lebih cepat untuk aplikasi yang interaktif.
throughput-performance	Setelan sistem untuk throughput yang maksimal.
latency-performance	Ideal untuk sistem server yang membutuhkan latency yang rendah pada pengeluaran konsumsi sumber daya.

network-latency	Diturunkan dari profil latency-performance. Memungkinkan tuning parameter jaringan tambahan agar dapat menyediakan latensi jaringan yang rendah.
network-throughput	Diturunkan dari profil throughput-performance. Tuning parameter jaringan tambahan digunakan untuk throughput yang maksimal.
powersave	Setelan sistem penghematan daya secara maksimal.
oracle	Mengoptimasi load Oracle database berdasarkan profil throughput-performance.
virtual-guest	Setelan sistem agar memaksimalkan performa jika dijalankan pada virtual machine.
virtual-host	Setelan sistem agar memaksimalkan performa jika berperilaku sebagai hos dari virtual machine.

Managing profiles from the command line

Command **tuned-adm** digunakan untuk merubah pengaturan *tuned daemon*. *Command* **tuned-adm** dapat meng-*query* pengaturan saat ini, daftar profil yang tersedia, rekomendasi *tuning profile* untuk sistem, merubah profile secara langsung, atau mematikan *tuning*.

Seorang *sistem administrator* mengidentifikasi *tuning profile* yang aktif saat ini dengan *tuned-adm active*.

```
[root@host ~]# tuned-adm active
Current active profile: virtual-guest
```

Command **tuned-adm** list menampilkan daftar semua *tuning profile* yang tersedia, termasuk *built-in profile* dan *custom tuning profile* yang dibuat oleh sistem administrator.

```
[root@host ~]# tuned-adm list
Available profiles:
- balanced
- desktop
- latency-performance
- network-latency
- network-throughput
- powersave
- sap
- throughput-performance
- virtual-guest
- virtual-host
Current active profile: virtual-guest
```

Gunakan **tune-adm profile profilename** untuk mengganti profil aktif ke profil lain yang lebih cocok dengan persyaratan tuning sistem saat ini.

```
[root@host ~]$ tuned-adm profile throughput-performance
[root@host ~]$ tuned-adm active
Current active profile: throughput-performance
```

Command **tuned-adm** dapat merekomendasikan tuning profile untuk sistem. Mekanisme ini digunakan untuk menentukan default profil dari sistem pasca instalasi.

```
[root@host ~]$ tuned-adm recommend
virtual-guest
```

Untuk memulihkan perubahan pengaturan yang dibuat *profile* saat ini, antara mengganti ke profil lain atau menonaktifkan *tuned daemon*. Mematikan aktifitas tuned tuning dengan tuned-adm off.

```
[root@host ~]$ tuned-adm off
[root@host ~]$ tuned-adm active
No current active profile.
```

4. Influencing Process Scheduling

Linux Process Scheduling and Multitasking

Sistem komputer modern memiliki rentang yang luas, dimulai dari sistem *low-end* yang hanya memiliki satu unit pemrosesan pusat (CPU) yang hanya dapat mengeksekusi satu instruksi pada satu waktu, hingga superkomputer berkinerja tinggi yang memiliki ratusan CPU dan puluhan hingga ratusan inti pemrosesan pada setiap CPU, yang memungkinkan eksekusi sejumlah besar instruksi secara bersamaan. Namun, semua sistem ini memiliki satu kesamaan, yaitu kebutuhan untuk menjalankan lebih banyak proses daripada yang bisa dilakukan oleh CPU.
Linux dan sistem operasi lainnya, menjalankan lebih banyak proses daripada processing unit yang tersedia dengan menggunakan teknik yang disebut *time-slicing* atau *multitasking*. *Process scheduler* di dalam sistem operasi beralih dengan cepat pada *single core*, yang mana ini memberikan kesan bahwa ada beberapa proses yang berjalan pada saat yang sama.

Relative Priorities

Proses yang berbeda memiliki level kepentingan yang berbeda. *Process scheduler* dapat dikonfigurasi untuk menggunakan *scheduling policies* yang berbeda pada proses yang berbeda. *Scheduling policy* digunakan hampir semua proses yang berjalan pada regular sistem yang disebut **SCHED_OTHER**.

Karena tidak semua proses sama pentingnya, proses yang berjalan dengan kebijakan **SCHED_NORMAL** dapat diberi prioritas yang relatif. Prioritas ini disebut *nice value* dari suatu proses, yang diatur sebagai 40 tingkat *niceness* yang berbeda untuk setiap proses.

Nilai *nice level* berkisar dari -20 (prioritas tertinggi) hingga 19 (prioritas terendah). Secara default, proses mewarisi *nice level* dari induknya, yang biasanya 0. Semakin tinggi *nice level* menunjukkan prioritas yang lebih rendah (proses dengan mudah menghentikan penggunaan CPU-nya), sementara *nice level* yang lebih rendah menunjukkan prioritas yang lebih tinggi (proses cenderung tidak memberikan meningkatkan CPU). Jika tidak ada perebutan sumber daya, misalnya, ketika ada lebih sedikit proses aktif daripada core CPU yang tersedia, bahkan proses dengan nice level

yang tinggi masih akan menggunakan semua sumber daya CPU sebanyak banyaknya. Namun, ketika ada lebih banyak proses yang meminta waktu CPU daripada *core* yang tersedia, proses dengan *nice level* yang lebih tinggi akan menerima lebih sedikit waktu CPU daripada proses dengan *nice level* yang lebih rendah.

Setting Nice Levels and Permissions

Semenjak mengatur *nice level* rendah terhadap proses yang membutuhkan banyak CPU dapat berdampak negatif pada kinerja proses lain yang berjalan pada sistem yang sama, hanya pengguna *root* yang dapat mengurangi proses *nice level*.

unprivileged user hanya diizinkan untuk meningkatkan nice level pada proses mereka sendiri. Mereka tidak dapat menurunkan *nice level* pada proses mereka, juga tidak dapat mengubah *nice level* dari pengguna lain.

Reporting Nice Levels

Beberapa tools menampilkan *nice level* dari proses yang sedang berjalan. Tools manajemen proses seperti **top**, menampilkan *nice level* secara *default*. *Tools* lain seperti perintah **ps**, menampilkan *nice level* ketika menggunakan opsi yang tepat.

Displaying Nice Levels from the Command Line

Command **ps** menampilkan *proses nice level*, tetapi hanya dengan memuat opsi format yang benar.

Command **ps** berikut ini menampilkan daftar proses dengan PID, nama proses, *nice level*, dan *scheduling class*, diurutkan secara *descending* berdasarkan *nice level*. Proses yang menampilkan **TS** di dalam kolom CLS *scheduling class*, dijalankan di bawah aturan *scheduling* **SCHED_NORMAL**. Proses dengan dash (-) sebagai nice levelnya, menjalankan dibawah aturan *scheduling* lain menginterpretasikan sebagai prioritas yang lebih tinggi oleh *scheduler*. Detail kebijakan *scheduling* tambahan berada di luar cakupan modul ini.

```
[user@host ~]$ ps axo pid,comm,nice,cls --sort=-nice
  PID COMMAND         NI CLS
   30 khugepaged      19  TS
   29 ksmd             5  TS
    1 systemd          0  TS
    2 kthreadd         0  TS
    9 ksoftirqd/0      0  TS
   10 rcu_sched        0  TS
   11 migration/0      -  FF
   12 watchdog/0       -  FF
...output omitted...
```

Start Processed with Different Nice Levels

Selama pembuatan proses, sebuah proses menurunkan *parent* milik *nice level*. Ketika sebuah proses dimulai dari *command line*, ini akan menurunkan *nice level* dari proses *shell* ketika dimulai. Secara khusus, hasil dari proses baru ini berjalan dengan *nice level* 0.

Berikut ini merupakan contoh memulai sebuah proses dari *shell*, dan menampilkan *nice value* milik proses. Perlu diingat bahwa penggunaan PID di dalam **ps** untuk menentukan *output* yang diminta.

```
[user@host ~]$ sha1sum /dev/zero &
[1] 3480
[user@host ~]$ ps -o pid,comm,nice 3480
  PID COMMAND          NI
 3480 sha1sum           0
```

Command **nice** dapat digunakan oleh semua user untuk memulai *command* dengan *nice level* secara *default* atau lebih tinggi. Tanpa *option*, *command* **nice** memulai sebuah proses dengan default *nice level* 10.
Berikut merupakan contoh memulai command **sha1sum** sebagai *background job* dengan *default nice level* dan menampilkan proses milik *nice level*:

```
[user@host ~]$ nice sha1sum /dev/zero &
[1] 3517
[user@host ~]$ ps -o pid,comm,nice 3517
  PID COMMAND          NI
 3517 sha1sum          10
```

Gunakan opsi **-n** untuk menerapkan *user-defined nice level* untuk memulai proses. Default merupakan untuk menambah 10 pada proses *nice level* saat ini. Berikut ini merupakan contoh *command* sebagai *background job* dengan *user-defined nice value* dan menampilkan *nice level* milik proses.

```
[user@host ~]$ nice -n 15 sha1sum &
[1] 3521
[user@host ~]$ ps -o pid,comm,nice 3521
  PID COMMAND          NI
 3521 sha1sum          15
```

Changing the Nice Level of an Existing Process

Nice level dari sebuah proses yang ada dapat diubah dengan menggunakan *command* **renice**. Contoh berikut ini menggunakan PID *identifier* dari contoh sebelumnya untuk mengubah dari *nice level* 15 saat ini menjadi *nice level* 19 yang diinginkan.

```
[user@host ~]$ renice -n 19 3521
3521 (process ID) old priority 15, new priority 19
```

command **top** juga dapat digunakan untuk merubah *nice level* pada sebuah proses. Dari dalam interaktif *interface* **top**, tekan opsi **r** untuk mengakses *command* **renice**, diikuti dengan PID yang ingin diubah dan *nice level* yang baru.

LEMBAR KERJA

KEGIATAN 1

Killing Process

1. Pada *image* CentOS yang dijalankan, buka dua *terminal* dan *split* kanan kiri. Pada tahap ini, kedua *terminal* disebut sebagai *terminal* kanan dan kiri. Masuk sebagai *user* **Student**.
2. Pada *terminal* kiri, buat sebuah direktori baru yang dinamakan **bin. path** secara spesifik terletak di **/home/student/bin**. Di dalam direktori yang baru, buat *script shell* yang disebut **killing**. Jangan lupa berikan *permission* agar file **killing** dapat dieksekusi.
    - Gunakan *command* **mkdir** untuk membuat direktori baru **bin /home/student/bin**
    - Gunakan *command* **vim** atau **nano** untuk membuat sebuah *file script* yang dinamakan **killing** di dalam direktori **/home/student/bin**. Jika menggunakan **vim**, tekan **key i** untuk masuk mode interaktif **vim**. Gunakan *command* **:wq** untuk menyimpan dan keluar **vim**. Karena saya baik maka saya sediakan scriptnya :)

```
#!/bin/bash
while true; do
        echo -n "$@" >> ~/killing_outfile
        sleep 5
done
```

    - Gunakan *command* **chmod** untuk membuat *file* **killing** bersifat *executable*. (**boleh numerik atau menggunakan kode**)
3. Pada *terminal* kiri, gunakan *command* **cd** untuk masuk ke dalam direktori **/home/student/bin/**. Jalankan **killing** proses yang sudah dibuat dengan argumen **network**, **interface**, dan **connection**, secara berurutan. Sebelum **kill** mulai ketiga proses yang disebut **network, interface**, dan **connection**. gunakan **ampersand (&)** untuk memulai proses pada *background*. Perlu diingat, setiap proses nanti akan mempunyai *PID number* yang berbeda.
4. Pada *shell terminal* kanan, gunakan *command* **tail** dengan opsi **-f** untuk konfirmasi bahwa ketiga proses yang dijalankan *di-append* ke *file* **/home/student/killing_outfile**.
5. Pada *shell terminal* kiri, gunakan *command* **jobs** untuk me-*list jobs* yang berjalan.
6. Gunakan *signal* untuk *suspend* proses **network**. Pastikan proses **network** telah berhenti. Pada *shell terminal* kanan, proses **network** sudah tidak di-*append* dan ditampilkan ke *file* **~/killing_output**.
    - Gunakan command **kill** dengan opsi **-SIGSTOP** untuk menghentikan proses **network**. Jalankan **jobs** untuk memastikan sudah berhenti.
    - Pada *shell terminal* kanan, lihat output dari *command* **tail**. Pastikan kata **network** sudah tidak di-*append* ke *file* **~/killing_outfile**.
7. Pada *shell terminal* bagian kiri, hentikan proses **interface** menggunakan *signal*. Pastikan proses **interface** sudah hilang. Pada *shell terminal* bagian kanan, pastikan bahwa *output* proses **interface** sudah tidak ter-*append* ke *file* **~/killing_outfile.**
    - Gunakan *command* **kill** dengan opsi **-SIGTERM** untuk menghentikan proses **interface**. Jalankan **jobs** untuk memastikan sudah berhenti.

- Pada *shell terminal* kanan, lihat output dari *command* **tail**. Pastikan kata **interface** sudah tidak di-*append* ke *file* **~/killing_outfile**.

8. Di *shell terminal* kiri, lanjutkan proses **network** menggunakan *signal*. Pastikan bahwa proses **network** sudah berjalan kembali. Di terminal kanan, pastikan bahwa *output* proses **network** sudah ditambahkan kembali ke *file* **~/killing_outfile**.
    - Gunakan *command* **kill** dengan opsi **-SIGCONT** untuk melanjutkan proses **network**. Jalankan *command* **jobs** untuk memastikan sudah berjalan kembali.
    - Pada *shell terminal* kanan, lihat *output* dari *command* **tail**. Pastikan kata **network** sudah di-append ke file **~/killing_outfile** kembali.
9. Pada *shell terminal* kiri, berhentikan (*terminate*) *jobs* sisanya yang masih berjalan. Pastikan tidak ada *jobs* yang berjalan dan *output* sudah berhenti.
    - Gunakan *command* **kill** dengan opsi **-SIGTERM** untuk *terminate proses* **network**. Gunakan *command* yang sama untuk *terminate proses* **connection**.
10. Di *shell terminal* kiri, proses *list* **tail** berjalan pada seluruh *shell terminal* secara terbuka. Hentikan *command* **tail** yang berjalan. Pastikan bahwa proses tidak lagi berjalan.
    - Gunakan *command* **ps** dengan opsi **-ef** untuk list semua proses **tail** yang berjalan. *Filter search* dengan menggunakan *command* **grep.**
    - Gunakan *command* **pkill** dengan **-SIGTERM** untuk *kill* proses **tail**. Gunakan **ps** untuk memastikan **tail** tidak lagi ada.
    - Pada *terminal* bagian kanan, pastikan bahwa *command* **tail** tidak lagi berjalan.

KEGIATAN 2

Monitoring Process Activity

1. Pada *image* CentOS yang dijalankan, buka dua terminal dan *split* kanan kiri. Pada tahap ini, kedua *terminal* disebut sebagai *terminal* kanan dan kiri. Masuk sebagai *user* **Student**.
2. Pada *shell terminal* kiri, buat sebuah direktori yang dinamakan **bin** yang terletak pada *path* **/home/student/bin**. Di dalam file yang baru dibuat, buatlah script shell dengan nama **monitor**, yang mana bertujuan untuk *generate CPU load* buatan. Pastikan script bersifat *executable*.
    - Gunakan **mkdir** untuk membuat direktori **bin**
    - Buat *shell script* bernama **monitor** di dalam direktori **/home/student/bin** dengan isi sebagai berikut:

```
#!/bin.bash
while true; do
        var=1
        while [[var -lt 60000]]; do
                var=$(($var+1))
        done
        sleep 1
done
```

Script **monitor** berjalan hingga diberhentikan. Ini meng-*generate load CPU* buatan dengan melakukan 60000 masalah tambahan. Nantinya *sleep* selama satu detik, berulang ulang terus menerus. seperti si author :".

- Gunakan *command* **chmod** untuk membuat *file* **monitor** agar bersifat *executable*.

3. Pada *shell terminal* bagian kanan, jalankan *top utility*. Sesuaikan *window terminal* setinggi mungkin, karena banyak yang dijalankan. seperti si author :".
4. Pada *shell terminal* bagian kiri gunakan *command* **lscpu** untuk memberikan nomor *logical CPU* pada *virtual machine*.
5. Pada *shell termina*l sebelah kiri, jalankan *single instance* dari *executable* **monitor**. Gunakan **ampersand (&)** untuk menjalankan proses di *background*.
6. Pada *shell terminal* sebelah kanan, amati apa yang ditampilkan **top**. gunakan *single keystrokes* **l, t,** dan **m** untuk *toggle load*, *threads*, dan *memory header lines*. Setelah mengamati perilaku, pastikan semua *header* sudah ditampilkan.
7. Catat ID proses (PID) **monitor**. Lihat persentase CPU untuk proses tersebut, yang diperkirakan berkisar antara 15% hingga 25%.

   Lihat juga *load average*. terlebih berapa jumlah *load average* selama satu menit saat ini.
8. Pada *shell terminal* bagian kiri, jalankan *instance* kedua dari **monitor**. Gunakan **ampersand (&)** untuk menjalankan proses di *background*.
9. Pada *shell terminal* bagian kanan, catat proses ID (PID) proses **monitor** kedua. Lihat persentase CPU untuk proses tersebut, yang diperkirakan berkisar antara 15% hingga 25%. Lihat kembali *load average*. terlebih berapa jumlah *load average* selama satu menit saat ini.
10. Pada *shell terminal* bagian kiri, jalankan *instance* ketiga dari **monitor**. Gunakan **ampersand (&)** untuk menjalankan proses di *background*.
11. Pada *shell terminal* bagian kanan, catat proses ID (PID) proses **monitor** ketiga. Lihat persentase CPU untuk proses tersebut, yang diperkirakan berkisar antara 15% hingga 25%.
12. Ketika selesai mengamati *load average value*, *terminate* setiap proses **monitor** yang ada di dalam **top.**
13. Ulangi step sebelumnya untuk semua instance **monitor**.
14. Pastikan tidak ada proses **monitor** lagi di **top**

KEGIATAN 3

Adjusting Tuning Profiles

1. Masuk ke *image* CentOS dan *login* sebagai Student.
2. Pastikan *tuned package* sudah terinstal, diizinkan untuk berjalan, dan digunakan.
   - Gunakan **yum** untuk memastikan bahwa *tuned package* sudah terinstal.
   - *Command* **systemctl is-enable tuned** menunjukan *service* mana yang diizinkan.
   - *Command* **systemctl is-active tuned** menunjukan *service* mana yang sedang berjalan.
3. *List tuning profile* yang tersedia dan identifikasi *profile* yang aktif. Jika **sudo** mengarahkan *password*, masuk **Student** setelah prompt.
4. Rubah *tuning profile* yang saat ini aktif untuk **powersave**, selanjutnya pastikan hasilnya sudah sesuai kriteria.
   - Ubah *tuning profile* aktif pada saat ini.
   - Pastikan **powersave** merupakan *tuning profile* yang aktif.
5. *Exit* dari *terminal*

KEGIATAN 4

Influencing Process Scheduling

1. Pada *image* CentOS yang dijalankan, buka dua terminal dan *split* kanan kiri. Pada tahap ini, kedua *terminal* disebut sebagai *terminal* kanan dan kiri. Masuk sebagai *user* **Student**.
2. Tentukan jumlah *core* CPU di *image* CentOS dan kemudian mulai dua *instance* **sha1sum /dev/zero &**.
    - Gunakan **grep** untuk *parse* angka *virtual* prosesor dari *file* **/proc/cpuinfo.**
    - Gunakan *command looping* untuk memulai *multiple instance* **sha1sum /dev/zero &**. Mulai dua *instance per virtual* prosesor yang ditemukan di step sebelumnya. (*contoh **for i in $(seq 1 4); do sha1sum /dev/zero & done**)
3. Pastikan *background jobs* berjalan pada setiap proses **sha1sum**.
4. Gunakan *command* **ps** dan **pgrep** untuk menampilkan persentase penggunaan CPU untuk setiap proses **sha1sum**.
5. *Terminate* semua proses **sha1sum**, selanjutnya pastikan tidak ada lagi *jobs* yang berjalan.
    - Gunakan *command* **pkill** untuk *terminate* semua proses yang berjalan dengan *pattern name* **sha1sum**.
    - Pastikan tidak ada *jobs* yang berjalan
6. Mulai *multiple instance* dari **sha1sum /dev/zero &,** selanjutnya mulai satu *instance* tambahan **sha1sum /dev/zero &** dengan *nice level 10*. Mulai setidaknya sebanyak *instance* yang dimiliki sistem *virtual processor*. Minimal menjalankan 3 *instance reguler*, ditambah lagi dengan *nice level* yang lebih tinggi.
    - Gunakan looping untuk memulai tiga instance sha1sum /dev/zero &.
    - Gunakan command nice untuk memulai empat instance dengan *nice level* 10.
7. Gunakan *command* **ps** dan **grep** untuk menampilkan PID, persentase penggunaan CPU, *nice value*, dan nama *executable* setiap proses. *Instance* dengan *nice value* 10 harus menampilkan persentase penggunaan CPU yang lebih rendah daripada *instance* yang lain.
8. Gunakan *command* **sudo renice** untuk *nice level* yang lebih rendah dari proses step sebelumnya (*nice level* 10). Catat PID *value* dari instance dengan *nice level* 10. Gunakan *sudo renice* untuk *nice level* 5 yang lebih rendah.
9. Ulangi *command* **ps** dan **pgrep** untuk kembali menampilkan persentase CPU dan *nice level.*
10. Gunakan *command* **pkill** untuk *terminate* semua proses yang berjalan dengan pattern nama **sha1sum.**

RUBRIK PENILAIAN

Aspek Penilaian	Bobot Penilaian
Ketepatan menjawab semua kegiatan	65%
Pemahaman setiap aspek materi yang dibahas	20%
Quiz pertemuan materi minggu pertama	15%
Total	**100%**