

# Meningkatkan Kinerja Multimedia dengan Tuning Kernel

Sering pada kombinasi *multitasking* tertentu, kernel Linux seperti “kedodoran” padahal dari spesifikasi *hardware* masih mencukupi. Hal ini terutama terjadi pada pengguna rumahan yang mania multimedia. Lalu, bagaimana cara *tuning*-nya?

Anda punya komputer bagus, Athlon 3 GHz, memori 512 MB, video card GeForceFX 5900, harddisk SCSI. *Welah-welah*, mestinya Anda bisa jadi lupa waktu (asal jangan lupa kerja...hehehe. “*Bejibun*” game, “selongsong” MP3, VCD Matrix Reloaded, coba-coba konversi MP3 ke OGG, *render* gambar, dan seabrek rencana lain udah membayang. Lalu ada boot Linux Anda di komputer baru. Tapi setelah beberapa job tadi dijalankan (mungkin lebih tepat disebut “bom nuklir”) anda berpikir “baru 5 task multimedia, kok udah keok, gimana nih? Saya kan maunya sambil *render* gambar, dengar MP3, plus *compile program*”. Tipe orang begini (dan saya juga) mungkin tipe *benchmarker* sejati. Inginnya *benchmark* dan *benchmark* dan tidak pernah puas dengan *tweaking*.

Kira-kira dengan asal usul yang sama plus karena sedang mulai coba-coba memprogram di kernel, saya mulai eksplorasi mengenai cara meningkatkan kinerja kernel. Dasar pemikirannya begini. Kernel adalah jantung dari sistem operasi. Bisa juga dikatakan sebagai pengatur utama interaksi pengguna dan *hardware* komputer. Kernel yang diprogram secara baik akan menghasilkan kinerja sistem yang baik pula. Jadi biarpun *hardware* bagusnya selangit, kalau kernelnya sedang-sedang, sistem tidak akan bekerja optimal.

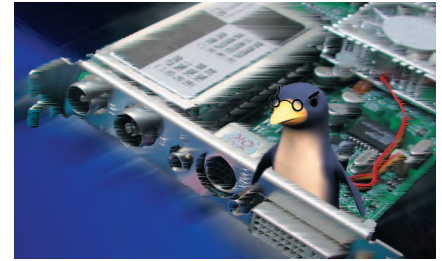
Berbicara mengenai optimasi sistem, sebenarnya banyak aspek yang terlibat. Kalau kita persempit pada aspek multimedia, saya simpulkan faktor-faktor yang terlibat adalah waktu respons interrupt, kecepatan kerja scheduler, dan kecepatan akses I/O. Faktor-faktor ini akan dijelaskan satu per satu.

Hardware komputer bekerja dengan cara menerima instruksi berupa *interrupt*. Bagi

yang pernah sedikit belajar *assembly*, Anda akan tahu bahwa interrupt ini ada banyak jumlahnya. Salah satunya, interrupt 10h (hexa) berkaitan dengan manipulasi *display* ke monitor. Pada sistem operasi multitasking seperti Linux, pengiriman interrupt akan masuk ke dalam antrian (*queue*). Ini disebabkan mungkin saja dalam waktu yang sama, ada permintaan interrupt lain (misal event keyboard pada interrupt 16h). Antrian interrupt ini akan dikerjakan satu per satu berdasar urutan masuk (*First In First Out*) kecuali pada program *real time*. Jika program yang meminta akses interrupt tidak segera dilayani, maka program tersebut akan dihentikan sementara (*sleep*) dan dibangun lagi (*wake up*) jika interrupt telah dilayani.

Dari paragraf di atas, terlihat ada kaitan antara waktu pelayanan interrupt dengan mekanisme *scheduler*. Secara umum, scheduler bisa dianalogikan seperti polisi lalu lintas yang bertugas pada perempatan jalan. Polisi (scheduler) akan menyetop mobil dari 3 ruas (ruas A,B,C) dan memperbolehkan mobil-mobil dari 1 ruas sisa (ruas D) berjalan. Setelah (anggap saja) 1 menit, giliran ruas A diperbolehkan berjalan, dan ruas B C D berhenti. Demikian seterusnya. Kecerdikan dan kecekatan polisi (scheduler) akan menentukan apakah antrian mobil bisa segera dikosongkan dengan segera sambil melihat antrian baru dari segala ruas. Bisa saja ruas C terus menerus mendapat antrian, tapi ruas A B dan D cukup sepi.

Faktor lain yang bermain adalah kecepatan akses I/O. I/O disini bisa bermacam-macam, bisa ke arah disk (floppy, harddisk, USB storage, CD-ROM drive, dan sebagainya), video card, memory, bahkan network card. Pada saat CPU meminta akses ke (misalkan) harddisk, maka perintah



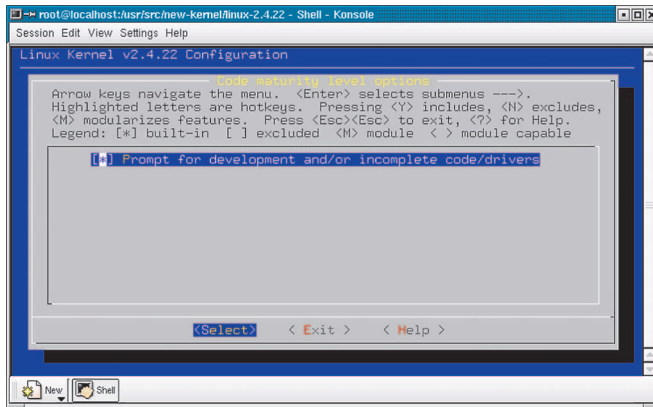
CPU akan melewati “bus” (bukan bus yang mampir di halte) dengan kecepatan tertentu menuju harddisk melewati controller EIDE (jika harddisk Anda IDE). Tapi jangan lupa, perintah akses I/O ini (lagi-lagi) melewati lapisan sistem operasi. Jadi selain kecepatan bus, seberapa cepat sistem operasi melayani akses I/O ini juga sangat berpengaruh pada kecepatan akses.

Terlihat dari penjelasan singkat di atas, keterlambatan (*delay* atau *latency*) adalah sesuatu yang tidak terelakkan. Benar, sekali lagi, tidak terelakkan. Bahkan pada sistem yang mengklaim dirinya *real time* sekalipun. Yang bisa kita lakukan adalah mereduksi latency ini di bawah suatu batasan sehingga kita menganggap sistem sudah responsif. Soal kata “responsif” ini, sangat relatif. Sebuah server dengan CPU 200 MHz mungkin cukup untuk *mail server* dengan 100 *account*, tapi tidak cukup untuk mengolah video sekaligus *streaming* ke Internet. Jadi tergantung kita sendiri, bagaimana yang kita anggap cukup.

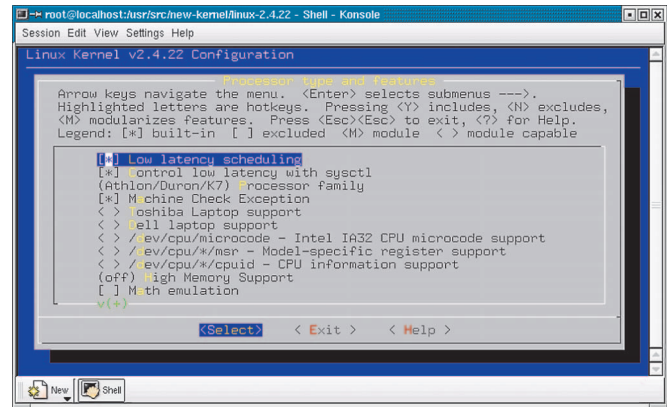
Sekarang pertanyaannya, “OK, sekarang saya mulai paham letak keterlambatan OS saya, lalu bagaimana cara *tuning*-nya?” Karena konteks pembicaraan kita pada OS, maka ada beberapa patch yang bisa membantu yaitu:

1. *Low latency patch* buatan **Andrew Morton**.
2. *Preemptible kernel patch* buatan **Robert M Love**.
3. *Variable HZ patch* buatan **Robert M Love**.

Untuk tiga patch di atas, Anda membutuhkan source kernel “vanilla”. Maksud “vanilla” di sini bukan seperti rasa es krim, tapi istilah untuk menyatakan source yang asli belum ditambah patch apapun. Penulis sarankan Anda memakai kernel 2.4.22 dari



▲ Gambar 1. Code maturity level options.



▲ Gambar 2. Processor type - Low latency.

[www.kernel.org](http://www.kernel.org). Patch nomor satu bisa anda ambil dari <http://www.zip.com.au/~akpm/linux/schedlat.html>. Patch nomor dua dan tiga bisa Anda download dari <http://www.tech9.net/rml/linux/>. Untuk kernel 2.4.22 ambil patch sebagai berikut:

- <http://www.zip.com.au/~akpm/linux/2.4.22-low-latency.patch.gz>
- <http://www.kernel.org/pub/linux/kernel/people/rml/preempt-kernel/v2.4/preempt-kernel-rml-2.4.22-1.patch>
- <http://www.kernel.org/pub/linux/kernel/people/rml/variable-HZ/v2.4/>

Sebenarnya bagaimana cara kerja tiga patch tadi? Berikut ringkasannya:

Low latency patch memodifikasi kode kernel yang memakan waktu lama agar dalam bagian tersebut kernel OS selama beberapa kali membreak task dan memanggil scheduler sehingga kernel mempunyai waktu menjalankan tugas yang lain. Untuk ini, Andrew Morton (dan Ingo Molnar, maintainer sebelumnya) melengkapi diri dengan sistem pelacak untuk mengukur waktu di mana saja tepatnya kernel paling sibuk. Lalu secara bertahap ditambahkan kode-kode untuk *rescheduling* sambil tetap mempertahankan konsistensi *task*.

Preemption patch melakukan modifikasi pada kernel sehingga scheduler bekerja secara lebih “rajin dan efisien”. Ini dilakukan dengan cara menambah kesempatan melakukan *rescheduling* jika task yang sekarang dikerjakan kernel bukan merupakan interrupt handler dan tidak task tidak memegang *spin lock* (suatu jenis variabel sistem untuk mengamankan akses ke suatu area memory terhadap kemungkinan korupsi data).

Logikanya, saat kernel mengerjakan interrupt, tidak boleh terjadi perpindahan ke task lain. Kalau ini terjadi, sistem akan bingung karena di satu sisi interrupt sedang dikerjakan, tapi karena tidak sampai selesai, bisa jadi ada akses ke interrupt yang sama sehingga saling menumpuk.

Demikian juga saat ada program yang akan mencoba mengambil atau melepas lock. Bayangkan seperti ini. Ada dua orang A dan B. Mereka akan mengambil sebuah gembok (lock). Pada saat tangan A terjulur mengambil gembok, tiba tiba ada keajaiban alam dan waktu untuk A berhenti (sudah nonton Matrix *kan*? Bayangkan saja seperti Bullet Time, beres). Si B dibangunkan dan juga bergerak mengambil gembok. Sampai dengan B sukses memegang gembok, si A masih “mati suri”, lalu giliran si B di-“stop”. Si A dibangunkan dan melanjutkan “gerakan” mengambil gembok. Tapi apa daya, gembok sudah di tangan B. Dalam kamus komputer, keadaan seperti ini dinamakan *race condition* alias berebut. Mengambil dan melepas lock adalah mekanisme *atomic* (artinya tidak boleh dihentikan oleh proses lain).

Jadi disimpulkan oleh **Robert Love**, setelah kembali dari *interrupt handler* atau lock dalam keadaan bebas, re-scheduling dijadwalkan kembali. Dengan demikian diharapkan respons sistem lebih tinggi.

Variable HZ adalah patch untuk memodifikasi frekuensi dari timer pada kernel. Gambarnya begini, scheduler sistem melakukan proses preemption (perpindahan satu task ke task lain) saat ada interrupt trap dari *timer hardware* (ya, motherboard Anda mempunyai timer). Pada saat inisialisasi kernel, “ditanam” suatu

*handler* untuk timer yang dipanggil pada interval tertentu. Misalkan timer diset tiap 0.01 mikro second, makan tiap 0.01 mikro second scheduler akan berpindah ke task lain (kecuali ada keadaan seperti nomer 2). Mulai paham idenya? Jadi akalinya, kita perkecil interval ini. Kernel Linux biasanya mengambil suatu nilai yang “aman” untuk semua hardware, jadi bisa saja nilai ini terlalu rendah untuk PC Anda. Patch variable HZ mengijinkan Anda mengubah nilai ini saat konfigurasi kernel ke sembarang nilai.

OK, sudah mulai penasaran? Segera download kernel plus patch dan ikuti langkah-langkah berikut:

1. “Su” menjadi root
2. Unpack source kernel yang baru, misal di /usr/src

```
# cp linux-2.4.22.tar.bz2 /usr/src/
# cd /usr/src
# bunzip2 linux-2.4.22.tar.bz2 && tar
xvf linux-2.4.22.tar
```

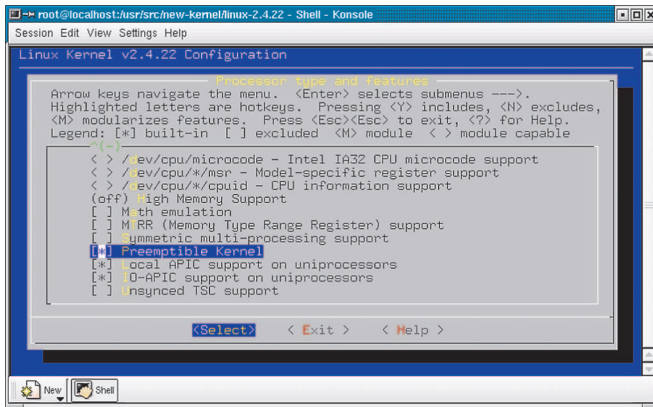
Akan tercipta direktori linux-2.4.22 di bawah /usr/src

3. Kumpulkan patch di /usr/src/

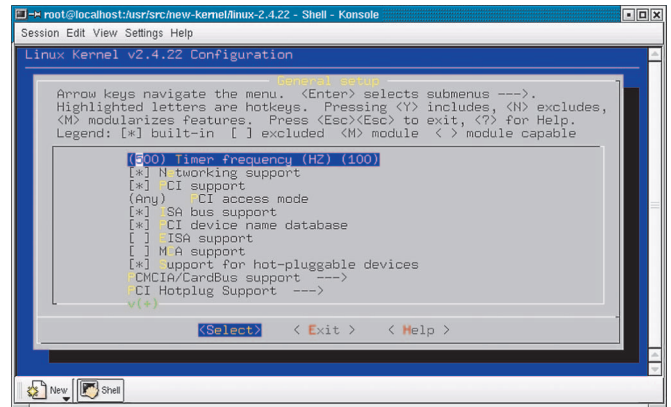
```
# cp 2.4.22-low-latency.patch.gz /
usr/src
# cp vhz-j64-2.4.22.patch /usr/src
# cd /usr/src && gunzip 2.4.22-low-
latency.patch.gz
```

4. Apply patch ke kernel

```
# cd /usr/src/linux-2.4.22
# patch -p1 < ../preempt-kernel-rml-
2.4.22-1.patch
# patch -p1 < ../2.4.22-low-
latency.patch
# patch -p1 < ../vhz-j64-2.4.22.patch
```



▲ Gambar 3. Processor type - Preemptible Kernel.



▲ Gambar 4. General setup - Timer frequency.

Pastikan tidak ada kata-kata *rejected* saat *patching*. Jika ada, ada kemungkinan source kernel atau patch yang anda download tidak sesuai. Cek ulang versi patch dan kernel yang sesuai.

5. Mulai persiapan kompilasi kernel

```
# make mrproper
# make menuconfig
```

(Penulis biasa memakai menu curses) atau "make config" (text based) atau make xconfig (X based).

6. Pada bagian "code maturity level options" aktifkan "Prompt for development and/or incomplete code/drivers" (lihat Gambar-1).
7. Pada bagian "processor type and features" aktifkan "Low Latency scheduling", "Control low latency with sysctl", "Preemptible kernel". Sesuaikan juga entry "Processor family" dengan jenis CPU Anda. (Gambar 2 dan 3).
8. Pada bagian "general setup", isikan angka 500 pada "Timer frequency". Cukup sorot bagian "Timer Frequency", tekan Enter, ketik angka yang Anda inginkan, lalu tekan Enter lagi. Bisa juga dicoba nilai 1000 (Gambar 4).
9. Pada bagian "Character Devices" aktifkan pilihan 'Enhanced Real Time Clock Support'. Anda bisa pilih sebagai internal kernel <\*> atau module <M>. Saya sarankan sebagai internal kernel. (Gambar 5).
10. Pada bagian "Processor type and features", non aktifkan pilihan "Symmetric Multi processing support". (Gambar 6) Untuk diketahui, jika Anda aktifkan dukungan SMP pada kernel namun

Anda hanya memiliki satu processor, maka kernel akan tidak efektif karena pada sistem SMP ada tambahan mekanisme *locking* untuk mencegah *race condition* antar-processor dalam mengakses suatu *resource* (misal memory) maupun sinkronisasi lock.

Opsi-opsi lain bisa Anda sesuaikan sendiri misal untuk dukungan APM. Penulis sendiri memilih mengaktifkan opsi "Magic SysRq key" pada bagian "Kernel Hacking" (gambar 7). Dengan opsi ini, akan di-*enable* suatu "tombol backdoor" yang memungkinkan kita melakukan aksi tertentu pada OS. Ini sangat berguna dalam keadaan darurat. Sebagai contoh untuk mendump isi register lakukan sebagai berikut (setelah kernel baru Anda *build*).

```
# echo "1" > /proc/sys/kernel/sysrq
```

Tekan Alt-Print Screen-P (huruf P) (note: coba "O" untuk shutdown).

Setelah selesai pilih "Exit". Pada pertanyaan "Do you wish to save your new kernel configuration" jawab "Yes". Pada direktori kernel source Anda dibuat file .config, simpan baik-baik file ini. Sekarang saatnya kompilasi kernel.

- `cd /usr/src/linux-2.4.22`
- `make dep && make bzImage`
- `make modules`
- Sebelum Anda menginstal module, ada baiknya Anda back-up dulu modul-modul kernel Anda sekarang, apalagi jika versi kernel yang digunakan sama-sama 2.4.22, caranya:
  1. Pindah ke direktori /lib/modules/
  2. `tar -f ./2.4.18-backup.tar --preserve`

```
-c --exclude build ./2.4.18-3/ &&
gzip ./2.4.18-backup.tar
```

(Ganti argument pada opsi -f dengan sembarang nama file yang diinginkan. Argumen terakhir pada tar adalah direktori yang akan di-backup, jadi arahkan ke direktori <versi-kernel-anda>. Opsi --exclude untuk mengabaikan direktori build karena ini adalah symbolic link).

- `cd /usr/src/linux-2.4.22 && make modules_install`
- `cp arch/i386/boot/bzImage /boot/linux-2.4.22-varhz`
- `cp System.map /boot/System.map-2.4.22-varhz`

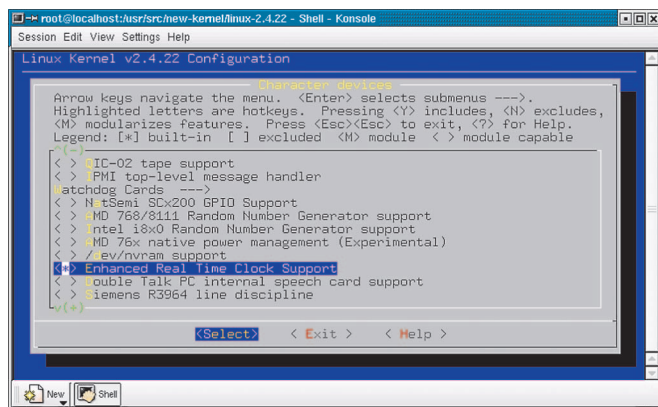
Edit loader Anda, di sini dicontohkan untuk LILO tambahkan entry sebagai berikut:

```
image = /boot/linux-2.4.22-varhz
label = linux-varhz
read-only
root = /dev/hda2
```

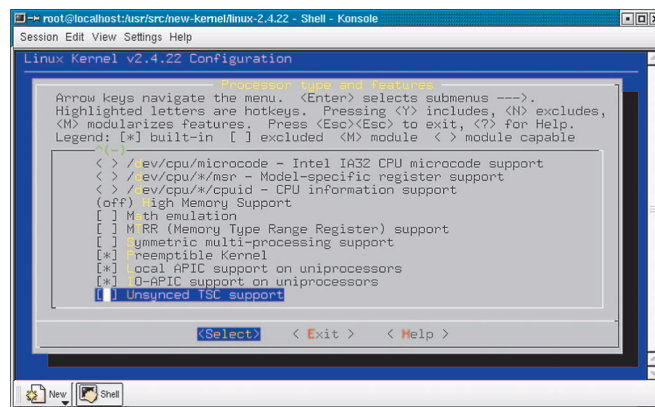
(Ganti /dev/hda2 dengan partisi tempat root file system Anda).

Ketikkan "lilo" untuk me-load entry baru. Sampai disini kompilasi sudah sukses. Tapi tunggu, sebelum kompilasi ada baiknya anda lakukan *benchmark* sehingga Anda yakin ada peningkatan *performance system*. Perlu diingat, Anda akan mengukur sesuatu yang satuannya menggunakan milisecond bahkan microsecond, jadi sangat sulit diukur secara kasat mata.

Tambahkan juga baris-baris berikut di file /etc/rc.d/rc.local.



▲ Gambar 5. Character devices.



▲ Gambar 6. Processor type - Unsynchronized TSC.

#### [[Listing 1]]

```
# --begin--
# untuk secara otomatis mendeteksi patch
low latency dan mengaktifkan fungsi low
latency
[ -x /proc/sys/kernel/lowlatency ] && echo
1 > /proc/sys/kernel/lowlatency

/sbin/hdparm -W0 -m8 -d1 -u1 -c1 -X66 /
dev/hda

#gunakan hdparm -t -T untuk benchmark I/
O hard disk

# --end--
```

Mungkin anda penasaran, "Buat apa pake hdparm?" Singkat saja, untuk meningkatkan performance I/O harddisk. Sebagai catatan, kernel Linux bisa saja tidak optimal dalam menggunakan harddisk Anda dan ini akan berakibat tidak efektifnya harddisk bekerja. Perintah **hdparm** di atas mengerjakan sebagai berikut:

- **W0**—untuk menon aktifkan *write caching*. Ini akan mempercepat proses penulisan sekaligus mencegah data korup.
- **m8**—untuk mengeset sector count menjadi 8 sector.
- **d1**—untuk mengaktifkan penggunaan DMA.
- **u1**—untuk mengaktifkan penggunaan interrupt mask.
- **c1**—untuk mengaktifkan penggunaan EIDE 32 bit I/O.
- **X66**—untuk mengaktifkan mode Ultra DMA 2 pada harddisk.

Ketik 'man hdparm' untuk mempelajari penggunaan hdparm lebih lanjut. Parameter

di atas bisa saja tidak cocok untuk harddisk Anda, jadi Anda harus rajin bereksperimen. Oh ya... jalankan dulu perintah 'hdparm' di atas untuk segera mengoptimalkan harddisk Anda karena setelah ini Anda akan memasuki tahap benchmark. Tambahan tip: gunakan **hdparm -t -T** untuk benchmark I/O hard disk anda. Cek selalu nilai-nilai yang didapat dan ambil parameter yang memberikan nilai paling optimal.

Untungnya untuk benchmark ini sudah ada paket *open source* yang berguna dan siap pakai. Salah satunya adalah Cerberus Suite (<http://people.redhat.com/bbrock/stress-kernel/stress-kernel-1.2.15-16.3.src.rpm>) untuk pembebanan sistem. Dan untuk pengukur latency digunakan Amlat (<http://www.zip.com.au/~akpm/linux/amlat.tar.gz>). Download kedua package ini dan kompilasi dengan cara sebagai berikut:

1. Untuk Cerberus suite, langsung lakukan rebuild untuk membuat paket rpm-nya:
  - **rpm --rebuild ./stress-kernel-1.2.15-16.3.src.rpm**
  - **rpm --Uvh ./stress-kernel-1.2.15-16.3.i386.rpm**
2. Untuk Amlat lakukan ekstrak dan kompilasi sebagai berikut:
  - **cp ./amlat.tar.gz /usr/src/**
  - **cd /usr/src && tar -xvzf ./amlat.tar.gz**
  - **cd amlat && make**

Cerberus Suite tersedia juga dalam versi *package binary* jika Anda ingin langsung mencoba. Catatan lainnya, Cerberus Suite membutuhkan beberapa paket RPM yang

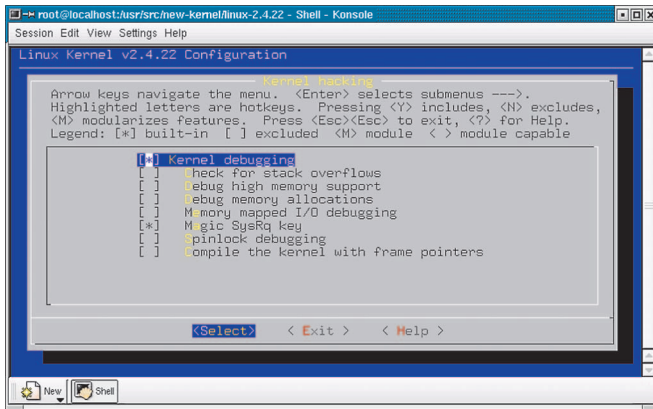
biasanya pada kategori "devel" semisal binutils, glibc, kernel-source, kernel-headers, gcc, g++. Jadi jika Anda menemui pesan error tidak ditemukannya suatu file saat kompilasi Cerberus Suite, Anda perlu menginstal paket-paket *devel* terlebih dahulu. Gunakan command berikut untuk membantu Anda mencari paket yang sesuai.

```
# rpm --filesbypkg --qlp /path/ke/
direktori/RPM_Anda | grep -l
nama_file_yang_diperlukan
```

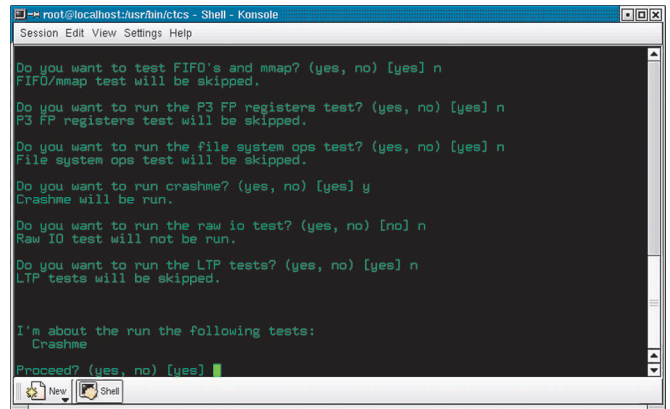
*Nah*, sebelum anda boot kernel baru, kita benchmark dulu sistem Anda yang lama. Caranya mudah, ikuti langkah berikut (ini sekadar ide, Anda tidak harus mengikuti alur ini).

1. Jalankan X Window (bisa KDE, Gnome atau yang lain). Matikan service-service yang tidak dibutuhkan semisal Apache, MySQL, dan seterusnya.
2. Jalankan XMMS (atau media player lain yang bisa play file mp3). Lalu putar beberapa lagu secara *looping*. Jalankan juga visual plug-in semisal OpenGL visualization untuk menambah beban CPU.
3. Jalankan Cerberus Suite, caranya pindah ke direktori `/usr/bin/ctcs`, lalu jalankan skrip `hell-hound.sh`. Untuk pertanyaan "How much extra memory (in MB) beyond the defaults do you want to reserve?" jawab dengan 'O'. Pilih "yes" cukup di bagian "Crash me Test" dan jawab "No" di bagian lain. Jawab pertanyaan "Proceed?" dengan "yes" (Gambar 8).
4. Sambil Cerberus Suite berjalan, buka console lain dan jalankan **realfeel2**.





▲ Gambar 7. Kernel hacking.



▲ Gambar 8. Cerberus suite.

- `cd /usr/src/amlat && ./realfeel2 - samples 1000000 ./normal.hist`
- Tunggu sampai selesai seluruh iterasi, lalu pindah ke console berisi task Cerberus dan tekan "Ctrl-C"

Untuk sementara simpan dahulu file ini dan *reboot* ulang Linux anda dan pilih kernel yang baru Anda buat. Jangan lupa sertakan driver-driver suara untuk nantinya persiapan memutar MP3 (jika Anda akan *benchmark* lagi menggunakan XMMS).

Ok, sudah boot kernel baru? Sekarang jalankan langkah 1 - 4 seperti di atas, tapi kali ini simpan file hasil benchmark *realfeel2* dengan nama lain, misal namakan "lowlat.hist". Sekarang saatnya membuat report dari kedua file ini sekaligus mengecek peningkatan performa Anda bisa gunakan script Perl berikut untuk membuat report.

#### [[Listing 2. Script pengolah hasil realfeel]]

```
#-----begin-----
#!/usr/bin/perl
sub find_percent
{
    my $limit = shift(@);
    my $result = 0;
    for ($i=0;($i<$n) &&
        ($val[$i]<=$limit);$i++)
    {
        $result += $freq[$i];
    }
    $result = $result/$sum_of_freq;
    print "Percentage below and equal to
    $limit milisecond is = $result\n";
}

$sum_of_freq=0;
```

```
$n=0;
$temp = 0;
$average= 0;
while (<STDIN>)
{
    #print $;
    $max = 0;
    $freq_mul_val = 0;
    @this_array=split(/\s+/, $_, 2);
    #print "detik : $this_array[0] frekuensi
    : $this_array[1] \n";
    $sum_of_freq += $this_array[1];
    #cari nilai maximal
    if ($max<$this_array[0])
    {
        $max=$this_array[0];
    }
    #hitung frequency * nilai
    $freq_mul_val[$n]= $this_array[0] *
    $this_array[1];
    $val[$n] = $this_array[0];
    $freq[$n]=$this_array[1];

    $n++;
}

#jumlahkan untuk mencari rata-rata
for ($i=0;$i<$n;$i++)
{
    $temp += $freq_mul_val[$i]
}
$average = $temp / $sum_of_freq;
#mencari nilai standar deviasi
$temp=0;
$stddev_node=0;
for ($i=0;$i<$n;$i++)
{
    $mean_sub_val=$average-$val[$i];
    $stddev_node += $mean_sub_val *
```

```
$mean_sub_val * $freq[$i];
}
$stddev_all=sqrt($stddev_node/
$sum_of_freq);

print "max value is      : $max\n";
print "total sample is   : $sum_of_freq\n";
print "average is        : $average\n";
print "standart deviation : $stddev_all\n";
find_percent(0.1);
find_percent(0.2);
find_percent(0.5);
find_percent(0.7);
find_percent(1);
find_percent(5);
find_percent(10);
#-----end-----
```

Gunakan editor teks semisal *vi* atau *Kwrite* untuk menyalin script di atas dan simpan ini misal dengan nama "hitung.pl". Lalu jalankan masing-masing dengan input file hasil output *realfeel2*:

```
# chmod a+x ./hitung.pl
# ./hitung.pl < ./normal.hist > ./report-normal.txt
# ./hitung.pl < ./lowlat.hist > ./report-lowlat.txt
```

Sebagai contoh, pada komputer penulis (AMD Athlon XP 1800+, 256 MB RAM), hasil parsing script di atas sebagai berikut:

(Untuk file normal hist)

```
max value is      : 946.5
total sample is   : 999997
average is        : 0.192800978402935
standart deviation : 4.20864151148962
```

```
Percentage below and equal to 0.1
milisecond is = 0.900977702933109
Percentage below and equal to 0.2
milisecond is = 0.921836765510297
Percentage below and equal to 0.5
milisecond is = 0.948988846966541
Percentage below and equal to 0.7
milisecond is = 0.962427887283662
Percentage below and equal to 1
milisecond is = 0.974251922755768
Percentage below and equal to 5
milisecond is = 0.996058988176964
Percentage below and equal to 10
milisecond is = 0.998402995208986
```

(Untuk file lowlat.hist)

```
max value is      : 16.4
total sample is   : 1000000
average is        : 0.0343402
standart deviation : 0.18507212287095
Percentage below and equal to 0.1
milisecond is = 0.940067
Percentage below and equal to 0.2
milisecond is = 0.954387
Percentage below and equal to 0.5
```

```
milisecond is = 0.985009
Percentage below and equal to 0.7
milisecond is = 0.989392
Percentage below and equal to 1
milisecond is = 0.995667
Percentage below and equal to 5
milisecond is = 0.999869
Percentage below and equal to 10
milisecond is = 0.999983
```

Bagaimana kita membaca report ini? Pertama, kita lihat nilai "max value". Terlihat pada kernel normal, latency maksimal adalah 946,5 ms (milisecond). Pada kernel + patch, turun menjadi 16,4 ms. Lumayan *kan*? Indikator sebenarnya terlihat pada average (rata-rata) dan *standart deviation* (nilai yang menyatakan besarnya perbedaan nilai-nilai sampel terhadap nilai rata-rata). Pada kernel normal, rata-rata latency 0.192800978402935 ms dengan standart deviation 4.20864151148962 ms. Pada kernel + patch menjadi rata-rata 0.0343402 ms dengan standart deviation 0.18507212287095 ms. Perhatikan baik-

baik kedua nilai ini. Sistem bisa dikatakan semakin membaik jika nilai rata-rata latency mengecil, juga nilai standart deviation. Kenapa begitu? Katakan saja nilai rata-rata mengecil sampai 50%, tapi standart deviation malah membesar 40%. Ini artinya dalam saat saat tertentu, nilai latency anda sangat berbeda jauh ketimbang nilai rata-rata. Jadi yang ideal, jika didapat penurunan nilai rata-rata latency dan standart deviation.

Sistem Anda membaik? Syukurlah kalau begitu, berarti tidak sia-sia anda mengikuti artikel ini. Atau sistem Anda malah memburuk? Jangan keburu kecewa atau malah memboikot artikel saya. Cobalah bereksperimen lagi, misal dengan nilai timer HZ yang lebih kecil atau lebih besar. Sebenarnya masih banyak teknik penurunan latency, dan karena ini ada kaitannya dengan optimasi sistem operasi maka banyak hal yang perlu dilakukan. Semoga ulasan pada artikel ini bisa membantu Anda men-tuning system sekaligus menambah pengetahuan akan mekanisme kernel Linux.🙏

**Mulyadi Santosa** ([a\\_mulyadi@softhome.net](mailto:a_mulyadi@softhome.net))

# IKLAN