

Memahami Proses di Linux

Linux adalah sistem operasi *multi tasking* yang sangat sukses. Pengaturan proses yang optimal adalah salah satu kunci keberhasilannya. Dengan memahami proses di Linux, kita akan dapat memahami Linux lebih baik lagi.

Masih ingat MS DOS? Ketika Anda menjalankan satu program, word star misalnya, maka Anda hanya bisa menjalankan program tersebut (*single task*). Tidak ada mekanisme resmi untuk menjalankan dua program atau lebih sekaligus. Program *resident* tidak dimasukkan dalam kategori multitasking karena *mem-bypass* sistem untuk dapat berjalan di latar belakang.

Oleh karena itulah, maka MS DOS seringkali dikatakan sangat stabil. Tentu saja karena hanya menangani satu program user dalam satu waktu. Masih teringat di benak penulis ketika Windows 95 muncul dan ketahuan senang sekali hang, orang-orang suka bersungut-sungut dan mengatakan MS DOS 6 lebih stabil.

Perbandingan tersebut tidaklah relevan karena Windows 95 adalah sistem operasi yang multitasking. Artinya, dalam satu waktu, bisa banyak program yang berjalan. Anda bisa mendengarkan lagu sambil mengetik di Microsoft Word misalnya. Hal tersebut dimungkinkan dengan berbagi waktu dengan alokasi waktu yang singkat dalam interval yang singkat untuk masing-masing proses.

Misalnya, katakanlah aplikasi pemutar lagu kita sebut sebagai aplikasi A1. Microsoft Word kita sebut sebagai aplikasi A2. Untuk melayani kedua program tersebut, Windows akan berpindah ke A1 dan melayani A1 untuk – katakanlah – 0.001 detik (time slice). Kemudian, berpindah lagi ke A2 dan melayani A2 – katakanlah – juga 0.001 detik. Setelah itu kembali lagi ke A1 dan seterusnya. Di mata pengguna, A1 dan A2 tampak jalan berbarengan karena interval 0.001 detik sangat sudah diukur. Dengan mekanisme serupa, kita bisa mengatakan kalau ada 1000 aplikasi, maka Windows akan mengunjungi dari A1 sampai A1000.

Sayangnya, sistem tidaklah sederhana itu. Sistem sebenarnya memang tidak

melayani dalam satuan seperseribu detik. Masih jauh lebih kecil. Namun, dengan perumpamaan kita sebelumnya, maka jika ada 1000 aplikasi, maka untuk melayani aplikasi A1 lagi setelah berkeliling, maka akan membutuhkan waktu 1 detik. Lama sekali. Bagaimana jika A1 adalah aplikasi pemutar lagu dan setiap 1 detik lagu Anda akan berhenti berputar, berputar lagi, lalu berhenti lagi? Baiklah. Jadikan time slice 1/10000 detik. Tetap saja akan kerepotan.

Hal tersebut belum termasuk ada proses yang memiliki prioritas berbeda. Ada yang minta dilayani agak lama. Ada yang lebih egois lagi. Dan sebagainya. Dan, yang penting, bagaimana kalau ada program yang harus mengakses perangkat keras seperti printer misalnya, dan ngambek menunggu printer yang tidak mau mencetak? Apakah sistem harus menunggu? Bagaimana kalau program tersebut memiliki prioritas tinggi dan sistem kebetulan menunggu? Jadilah kita sebut komputer kita hang.

Sistem yang sebenarnya memang tidak sesederhana itu. Tentunya ada mekanisme yang lebih unggul yang membuat pergantian melayani proses menjadi jauh lebih efisien. Tapi, kondisi sederhana tersebut bisa kita gunakan untuk mengetahui mengapa suatu sistem operasi lebih mudah dan sering ‘hang’ dibanding yang lainnya.

Coba lihat sistem operasi - sistem operasi besar kelas enterprise dengan harga selangit yang melayani bank yang sibuk misalnya. Dalam satu waktu, mungkin terdapat lebih dari 1000 proses berjalan secara konkuren. Atau, coba amati server AOL atau Yahoo! Dalam satu waktu, bisa-bisa terdapat lebih dari 10.000 proses yang berjalan. Tidak bisa dipastikan karena jumlah pelanggan mereka sangat besar. Dan pengguna adalah raja. Oleh karena itu, berbagai cara harus dilakukan agar pelayanan tetap dapat dilakukan. Sesibuk apapun juga.

Dari sisi *hardware* sudah pasti. Namun, dari sisi *software* pun harus kuat. Maka, umumnya mereka ada mempergunakan sistem operasi besar kelas *enterprise* dengan harga selangit tersebut.

Dulu, Linux belum mampu sekelas sistem operasi enterprise. Namun, sejak kernel 2.6 lahir, banyak hal yang mampu menjadikan Linux memasuki pasar enterprise dengan harga yang sangat masuk akal. Beberapa catatan misalnya. Jumlah user dari 64K (sekitar 65.000) menjadi lebih dari 4 juta (16 bit ke 32 bit). Dan untuk kasus proses, batas PID tidak lagi 32000, namun menjadi lebih dari 1 juta.

Hal ini berarti, teorinya, memungkinkan Linux melayani mendekati hampir 1 juta proses. Dnekan kemampuan memasuki enterprise ini, wajar apabila Red Hat mulai lebih fokus. Novell membeli SUSE dan lain sebagainya.

Mau dijual mahal pun, server enterprise Linux masih akan tampak lebih masuk akal. Kita, tentu saja tidak bisa mengatakan Red Hat, yang menjual Linux seharga 10.000 USD misalnya, keterlaluan. Kontribusi Red Hat dan SUSE pada *source code* kernel turut menjadikan kernel Linux jauh lebih baik.

Di artikel ini, kita akan membahas bagaimana memahami proses di linux. Pembahasan akan dilakukan mulai dari sisi *user*, *sysadmin* dan *developer*. Ketiga kategori pengguna dilibatkan untuk contoh dan pembahasan yang lebih luas.

Simulasi sistem multi tasking

Sebelum kita memasuki pembahasan proses, ada baiknya kalau kita sedikit melakukan simulasi bagaimana sistem bekerja. Kita akan membuat sistem yang melayani dua program yang sedang berjalan: A dan B.

Berikut ini adalah source codenya dalam bahasa C. Penjelasan dan output dibahas setelahnya.

```
#include <stdio.h>

int main(void) {
    unsigned long int counter=1;
    unsigned long int limit=9;
    while (1) {
        counter++;
        if ( (counter % limit) ==
            0) {
            printf("melayani
                proses B\n");
            counter = 1;
        } else {
            printf("melayani
                proses A\n");
        }
    }
    return 0;
}
```

Penjelasan kode:

- Perulangan akan dilakukan terus menerus (*while* (1))
- Setiap perulangan dilakukan, counter akan ditambah satu (*counter++*)
- Pemeriksaan *variabel* counter akan dilakukan. Apabila sisa bagi counter terhadap limit adalah 0, maka saatnya melayani proses B. Apabila sisa bagi bukan 0, maka proses A masih terus dilayani.

Apabila program tersebut dijalankan, maka tulisan melayani proses A akan tercetak beberapa kali, setelah itu tulisan melayani proses B akan tercetak sekali. Setelah itu, melayani proses A dicetak lagi selama beberapa kali, diikuti dicetaknya tulisan melayani proses B satu kali dan seterusnya. Beberapa kali tersebut tentunya dapat diatur di variabel limit.

Kondisi ini mensimulasikan sistem multitasking untuk dua proses. A di sini lebih dominan dari B (prioritas lebih tinggi). Tentu saja, dengan mudah kita bisa mengubah nilai prioritas dengan mengubah limit dan

atau hasil modulus. Termasuk untuk lebih banyak proses.

Di MS DOS, dimulasi ini dapat diterapkan secara sederhana pada beberapa permainan seperti *arkanoid* yang sementara bola berjatuhan, kita masih dapat menggerakkan *paddle*. Tentu saja, menggerakkan *paddle* memiliki prioritas lebih tinggi.

Linux dan sistem operasi multitasking lain tentu jauh lebih kompleks. Tidak hanya sekedar menggunakan counter. Teknik sistem operasi terus berkembang. Di Linux, apabila ada hal yang tidak efisien, maka bisa-bisa ditulis ulang dari awal. Contoh kasus paling mengejutkan adalah digantinya Virtual Memory (VM) Linux ke sistem milik Andrea Arcangeli (SUSE) dari sistem VM milik Rik van Riel.

Program, proses, thread

Kita sering mendengar istilah ini. Banyak pembuat program yang menyatakan program saya *multithreading*, loh! Jadi, pasti lebih baik. Atau, tak jarang kita mendengar, thread di Java canggih sekali, yang lain kalah. Itu tentang thread.

Kalau tentang proses. Proses saya sudah ribuan, dengan proses A menggunakan resource sistem lebih dari 40%, misalnya. Atau, di sistem saya, banyak sekali proses yang tidur. Macam-macam.

Bagi kalangan *developer*, tak jarang ada mengatakan, jangan pakai *fork()*, tidak jalan di windows, Windows tidak mendukung pembuatan anak proses. Dan bermacam-macam alasan lainnya.

Dari sisi user. Di sistem saya, ada sekitar 100 program yang sedang berjalan. Dan, sistem saya tidak hang sama sekali!

Apakah program, proses, dan thread itu, dan apa pula hubungan diantara mereka?

Secara sederhana, proses adalah program yang berjalan. Program yang tidak dijalankan tidak akan mendaftarkan dirinya sebagai suatu proses. Tapi, harap diperhatikan, suatu proses tidak selalu harus berjalan. Ada kalanya suatu proses tidur, berhenti, menunggu dan mati (dan masih terdaftar).

Cobalah buka program top dan amatilah tulisan bagian atas program ini. Anda akan melihat tulisan *running*, *sleeping*, *stopped* dan *zombie*. Semua ada status proses. Dengan perumpamaan kita sebelumnya, suatu sistem harus melayani proses-proses yang

Selera bisa saja Beda

Linux

semuanya
bebas dan aman



SuSE LINUX 9.2
Professional



Mandrake Linux 10.1
PowerPack+

Dapatkan semua Linux di:
GudangLinux
The Open Source Destination
T: (021) 5793-4060
F: (021) 5793-5557
info@gudanglinux.net
<http://www.gudanglinux.com>

Di Linux, setiap proses memiliki atribut seperti halnya file. Proses memiliki ID proses. Ini akan membedakan suatu proses dengan proses lain secara unik. Proses tentu punya nama, *resource* yang digunakan, pemilik proses dan lain sebagainya. Cobalah berikan perintah berikut ini untuk melihat proses sistem:

- PID, *Process ID*.
- PPID, *Parent Process ID*
- *Real User ID*.
- *Effective User ID*
- *Real Group ID*
- *Effective Group ID*
- Informasi user
- *resource* yang digunakan proses seperti *wall clock time* (waktu yang dipergu-

Thread sendiri bukanlah isu yang sederhana. Di Linux sendiri, beberapa distro telah menerapkan pustaka thread yang lebih baik. Di SUSE 9.1 atau SLES 9 misalnya, pustaka thread baru NPTL (*Native Posix Thread*

```
pstree
```



Program ini akan memvisualisasikan hirarki proses dalam sebuah *tree*.

Catatan lain tentang proses adalah adanya sesi proses. Ketika Anda membuka xterm di X dan menjalankan suatu program, maka ada beberapa proses yang terkelompok dalam suatu sesi bersama. Ketika suatu sesi diakhiri (xterm diterminasi misalnya) maka seluruh proses juga akan diterminasi.

Berikut ini adalah contoh program C untuk menghasilkan anak sesuai dengan permintaan user. Program akan meminta input jumlah anak yang akan dibuat dan setelah itu, program akan menampilkan PID anak-anaknya beserta orang tuanya (PPID). Hirarki proses kemudian dapat diamati dari keluaran program tersebut (PPID orang harus sama tentunya).

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
    pid_t temp_id;
    int child_amount;
    int i;

    printf("Masukkan jumlah
anak [max 5]: ");
    scanf("%d", &child_
amount);

    if (child_amount > 5) {

        printf("terlalu banyak
anak yang akan dibuat\
n");

        return 1;
    }

    printf("PARENT: PID orang
tua: %d\n", getpid());

    for (i=0; i< child_
amount; i++) {

        if ((temp_id =
fork()) == -1) {

            printf("ERROR: terjadi kesalahan
pada pengulangan ke %d\n", i+1);
            exit(1);

        } else if (temp_
```

```
id == 0) {

    printf("\tCHILD: sudah dibuat
dengan PID: %d, dan PPID: %d\n",
getpid(), getppid());

    exit(0);

}

}

return 0;

}
```

Penjelasan program:

- Pertama-tama, user diminta untuk memasukkan jumlah anak yang akan dibuat. Lakukan validasi dengan maksimal 5 anak.
- Mencetak PID untuk memudahkan pengecekan lebih lanjut bagi user.
- Mengulang sebanyak jumlah anak.
- Di dalam perulangan, akan memanggil system call `fork()` untuk membuat anak proses. Apabila `fork()` mengembalikan -1 yang artinya gagal, maka kita mencetak pesan kegagalan.
- Harap memperhatikan benar-benar sifat `fork()` yang asinkron. Pada pembuatan anak proses yang berhasil, `fork()` akan mengembalikan nol untuk sesi anak proses dan mengembalikan pid anak untuk sesi orang tua. Kita tidak bisa bergantung pada kode yang memastikan kapan anak akan dibuat dan kemudian melakukan kode-kode tertentu di sana. Bisa menyebabkan race condition.

Berikut adalah contoh keluaran program:

```
$ ./fork_test
Masukkan jumlah anak [max 5]: 5
PARENT: PID orang tua: 4243
    CHILD: sudah dibuat
dengan PID: 4244, dan PPID: 4243
    CHILD: sudah dibuat
dengan PID: 4245, dan PPID: 4243
    CHILD: sudah dibuat
dengan PID: 4246, dan PPID: 4243
    CHILD: sudah dibuat
dengan PID: 4247, dan PPID: 4243
```

```
CHILD: sudah dibuat
dengan PID: 4248, dan PPID: 4243
```

Kita bisa melihat di sini bahwa terdapat lima anak yang dibuat (sesuai permintaan) dan PPID setiap anak adalah sama, yang sama pula dengan PID program `fork_test`.

Pada download accellerator yang memanfaatkan forking, cara kerjanya bisa diasumsikan sebagai berikut:

- Mengambil ukuran file yang akan di-download.
- Membagi sama rata sesuai jumlah anak proses.
- Membuat anak proses dan memberikan tugas untuk mendownload sesuai pada posisi tertentu.
- Menggabungkan file yang telah berhasil didownload oleh anak-anak proses tersebut.

Umumnya, orang tua tidak terlalu banyak bekerja. Lebih banyak anak-anaknya. Orang tua melakukan satu atau dua tugas, lalu memonitor anak-anaknya dan kemudian melakukan *finishing*.

Pembahasan akan kita lanjutkan ke daemon. Daemon adalah hal yang menarik untuk dicermati di Linux. Banyak sekali daemon di Linux. Anda bisa mengetahuinya dengan melihat akhiran *d* yang umumnya digunakan pada nama suatu program. Sebagai contoh `httpd`, `ftpd`, `sshd` dan lain sebagainya. Akhiran *d* tersebut menunjukkan *daemon*.

Sebenarnya, apakah daemon itu? Secara sederhana, daemon dapat diartikan sebagai program yang berjalan di latar belakang, atau tidak memiliki terminal control. Umumnya, daemon digunakan pada aplikasi jaringan dan menunggu pada port tertentu.

Lebih teknis lagi, daemon adalah proses yang egois (dalam pengistilahan oleh umat manusia). Kenapa? Karena, daemon terbentuk dari suatu proses orang tua yang membuat anak proses, setelah itu membunuh dirinya. Jadi, anak-anaknya akan tumbuh tanpa orang tua dan menjadi daemon. Orang tua sebenarnya telah berkorban untuk menjadikan anaknya sebagai daemon. Orang tua daemon setelah itu adalah `init`. Anak yang menjadi daemon itupun kemudian tumbuh membentuk sesi sendiri.

```

top - 11:22:46 up 1:03, 1 user, load average: 0.71, 0.36, 0.21
Tasks: 75 total, 1 running, 73 sleeping, 0 stopped, 1 zombie
Cpus(s): 0.0% us, 2.4% sy, 0.0% ni, 10.3% id, 69.4% us, 0.7% hi, 0.0% si
Mem: 229100k total, 22952k used, 11508k free, 11920k buffers
Swap: 506000k total, 27932k used, 478068k free, 106370k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM     time+  COMMAND
 5239 ntp      15   0 2160 6372 208  B  6.3  3.5   0:00.19  ksmaphot
3296 root      15   0 1446 9412 1336 S  2.0  3.9   0:22.31  X
3494 ntp      15   0 26536 11m 24m  S  1.0  4.9   0:01.66  ksmaphot
3440 ntp      15   0 26024 13m 236  S  0.3  5.6   0:05.72  ksmaphot
3455 ntp      15   0 26788 14m 258  S  0.3  6.1   0:03.79  ksmaphot
3562 ntp      15   0 28124 11m 24m  S  0.3  5.0   0:01.08  ksmaphot
5230 ntp      16   0 1766 716 1540  S  0.1  0.4   0:00.05  top
 1 root      16   0 580 156 444  S  0.0  0.1   0:05.52  init
 2 root      34  19  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
 3 root      5 -10  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
 4 root      5 -10  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
 5 root      5 -10  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
 6 root      5 -10  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
 7 root      15  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
 8 root      15  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
10 root      7 -10  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
 9 root      15  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
105 root     16  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
221 root      5 -10  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
628 root     16 -10  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
1224 root    18   0 5004 1616 4724 S  0.0  0.7   0:00.00  hotplug
1225 root    22   0 1360 380 1280 S  0.0  0.2   0:00.00  logger
1252 root    18   0 5004 1616 4724 S  0.0  0.7   0:00.00  hotplug
1253 root    22   0 1360 380 1280 S  0.0  0.2   0:00.00  logger
1501 root     15  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
1635 root     16  142 424 1280 S  0.0  0.2   0:00.00  ksmaphot
2091 root     17  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
2101 root     17  0 0 0  S  0.0  0.0   0:00.00  ksmaphot
2370 root     16  140 532 1280 S  0.0  0.2   0:00.00  ksmaphot
2381 root     16  0 2312 624 1212 S  0.0  0.3   0:00.10  ksmaphot
2455 bin      16   0 1420 356 1248 S  0.0  0.1   0:00.00  portmap
  
```

Program top.

```

 8 ?      S  0:00 [pdflush]
10 ?      S<  0:00 [aio0]
 7 ?      S  0:00 [ksmaphot]
16 ?      S  0:00 [Userload]
221 ?     S<  0:00 [reiserfs0]
628 ?     S<  0:00 [kcmgpl]
1223 ?     S  0:00 /bin/bash /sbin/hotplug pci
1224 ?     S  0:00 logger -t /sbin/hotplug12021
1225 ?     S  0:00 /bin/bash /etc/hotplug/pci.agent pci
1256 ?     S  0:00 logger -t /etc/hotplug/pci.agent12021
1472 ?     S  0:00 [kcmgpl]
1536 ?     S  0:00 [kcmgpl]
2047 ?     S  0:00 [kcmgpl]
2049 ?     S  0:00 /sbin/sgslogd -a /var/lib/ntp/ntpdc.log
2384 ?     Ss  0:00 /sbin/klogd -c 1 -2
2387 ?     Ss  0:00 /sbin/klogd -c 1 -2
2461 ?     Ss  0:00 /sbin/portmap
2463 ?     Ss  0:00 /sbin/resmgrd
2530 ?     S  0:00 [rcpidd]
2531 ?     S  0:00 [kcmgpl]
2539 ?     Ss  0:00 /usr/sbin/smbd -D -s /etc/samba/smb.conf
2629 ?     Ss  0:00 /usr/sbin/sshd -o PidFile=/var/run/sshd.init.pid
2652 ?     Ss  0:00 /usr/lib/openssh/ssh-keygen -h /dev/null -e /dev/null
2706 ?     S  0:02 /usr/sbin/powersaved -d -e /etc/powersave.conf -a resmgrd -u 3
2962 ?     S  0:00 /usr/sbin/automount /mnt /etc/automnt
3160 ?     Ss  0:00 /usr/sbin/sshd
3176 ?     Ss  0:00 /usr/sbin/cron
3244 ?     S  0:00 /opt/kde3/bin/kde
3249 ?     Ss  0:00 /usr/sbin/smbd -D -s /etc/samba/smb.conf
3297 ?     S  0:47 /usr/X11R6/bin/X -nolisten tcp -br vt7 -auth /var/lib/xdm/authdir/authfiles/a:0-0:0:10:1R
3298 ?     S  0:00 -3
3303 tty1   Ss*  0:00 /sbin/mingetty --noclear tty1
3304 tty2   Ss*  0:00 /sbin/mingetty tty2
3305 tty3   Ss*  0:00 /sbin/mingetty tty3
3306 tty4   Ss*  0:00 /sbin/mingetty tty4
3307 tty5   Ss*  0:00 /sbin/mingetty tty5
3308 tty6   Ss*  0:00 /sbin/mingetty tty6
3365 ?     S  0:00 /bin/sh /opt/kde3/bin/startkde
  
```

Contoh keluaran program ls.

Berikut adalah contoh daemon sederhana. Sebutlah abcd, abc daemon, yang akan membuat log pada /tmp/abcd.log.

```

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(void) {
    pid_t pid, sid;
    int fd, len=100;

    printf("PARENT: Pid saya
    adalah %d\n", getpid());

    pid = fork();
    if (pid < 0) {

        printf("gagal
        membuat anak proses\n");
        exit(1);

    } else if (pid > 0) {

        printf("PARENT:
        Saya bunuh diri\n");
        exit(0);

    };

    if (sid = setsid() < 0)
    {

        printf("gagal
        membentuk sesi\n");
  
```

```

    exit(2);

};

    if ( (chdir("/tmp")) <
    0) {

        printf("gagal
        masuk ke area kerja\n");
        exit(3);

    };

    umask (0);

    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);

    /* bekerja sesuai
    fungsinya, abc daemon */

    while (1) {

        char *buf =
        malloc(sizeof(char) * (len+1));

        if ( (fd =
        open("/tmp/abcd.log", O_CREAT |
        O_WRONLY | O_APPEND, 0600)) <
        0) {

            exit(4);

        }
  
```

```

        strncpy(buf,
        "kecap ABC, baterai ABC, mie
        instan ABC, sirup ABC, * ABC",
        len+1);

        write (fd, buf,
        len+1);

        close(fd);

        sleep(60);

    }

    return 0;
}
  
```

Penjelasan program:

- Pertama-tama, orang tua membuat anak proses, lantas membunuh dirinya.
- Anak pun membentuk sesi sendiri
- Setelah itu, anak masuk ke /tmp yang merupakan area kerja.
- Umask kemudian diset ke 0.
- Karena daemon, maka stdin, stdout dan stderr tidak terbuka. Kita menutup ketiga handle file tersebut.
- Kita bekerja dalam perulangan tanpa henti.
- Dalam perulangan, kita membuka file /tmp/abcd.log dan menambahkan isinya apabila file telah ada. Kalau tidak ada, kita akan membuatnya terlebih dahulu.
- Dalam perulangan, kita menuliskan se-

jumlah karakter ke dalam file tersebut /tmp/abcd.log).

- Kita menunda setiap 1 menit untuk menulis kembali.

Sekali dijalankan, abcd akan berjalan terus. Anda dapat mempergunakan program kill untuk membunuh abcd. Sebagai contoh:

```
$ killall abcd
```

Berikut ini adalah contoh keluaran program:

```
$ ./abcd
PARENT: Pid saya adalah 6018
PARENT: Saya bunuh diri
```

Berikut ini adalah contoh log /tmp/abcd.log:

kecap ABC, baterai ABC, mie instan ABC, sirup ABC, * ABCkecap ABC, baterai ABC, mie instan ABC, sirup ABC, * ABCkecap ABC, baterai ABC, mie instan ABC, sirup ABC, * ABC

Kontribusi proses pada /proc

Linux menganut sistem yang transparan. Begitupun dengan proses-proses di dalamnya. Pada file sistem semu /proc, kita dapat melihat direktori-direktori dengan nama direktori berupa angka.

Angka-angka tersebut adalah pid proses. Oleh karena itu, dari waktu ke waktu, angka-angka tersebut bisa berubah-ubah. Manakala sebuah proses diterminasi, maka direktori PID proses tersebut pada /proc akan ikut menghilang pula. Demikian juga ketika terjadi penambahan proses baru.

Cobalah masuk ke dalam salah satu direktori tersebut. Kita akan menemukan beberapa file berikut ini:

- cmdline. File ini bertugas merekam command line yang diberikan ketika menjalankan proses.
- Environ. Nilai-nilai environment variable.
- fd. Direktori yang mengandung semua file descriptor.
- Mem. Memori yang digunakan oleh proses.
- Stat. Status proses.
- Status. Status proses dalam bentuk *human readable*.
- Cwd. Sebuah link yang menunjuk pada direktori aktif proses.

- Exe. Sebuah link kepada *executable proses*.
- Maps. Peta memori.
- Root. Sebuah link yang menunjuk pada *root directory proses*.
- Statm. Status memori.

Berikut ini adalah contoh beberapa isi file untuk proses abcd:

```
Name: abcd
State: S (sleeping)
SleepAVG: 26%
Tgid: 6547
Pid: 6547
PPid: 1
TracerPid: 0
Uid: 1000 1000 1000
1000
Gid: 100 100 100
100
FDSize: 32
Groups: 14 16 17 33 100
VmSize: 1360 kB
VmLck: 0 kB
VmRSS: 360 kB
VmData: 156 kB
```

```
VmStk: 8 kB
VmExe: 4 kB
VmLib: 1152 kB
Threads: 1
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000000000
SigCgt: 0000000000000000
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
```

Dari file ini, kita dapat mengetahui banyak hal yang berhubungan dengan proses abcd, mulai dari statusnya (*sleeping*), PID dan PPID (harap diperhatikan bahwa orang tua daemon adalah proses init dengan PID 1), resource yang digunakan, pemilik proses, dan informasi lainnya.

Tentunya, kita dapat menggunakan script untuk membaca file-file tersebut untuk keperluan tertentu.

Berkomunikasi dengan proses

Kita, sebagai pengguna, sistem dapat ber-

MORE SPACE RELIABILITY & LESS... TIME & MONEY

LINUX and FreeBSD

Features:

- Unlimited data transfer
- Complete control panels
- POP3 email, FTP access
- SSH, CGI, SQL...
- and much more...
- Start from Rp. 19.500,-/ month
- **Free Setup (*)**
- **2 Months Free (**)**

**Limited Offer :
Dedicated Server
Rp. 1.250.000,-/ mo**

Server Hosting

Features:

- Location NOC Jakarta - Indonesia (IIX)
- Size server : 1 U Rackmount
- Bandwidth : 128 kbps
- IP Address : 8 (max)
- Colocation : Rp. 1.000.000,-/ month

ALSO

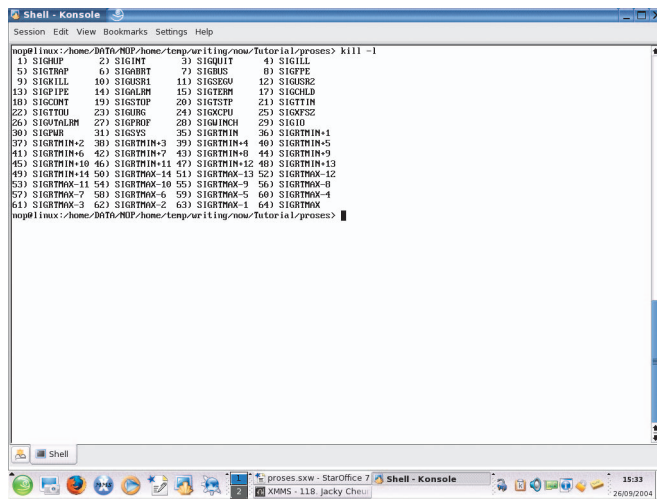
- Colocation & Dedicated Server in USA
- Domain Name Register
- Benefit Reseller Program

*" IT'S NEVER BEEN EASIER
TO TAKE YOUR BUSINESS ONLINE "*

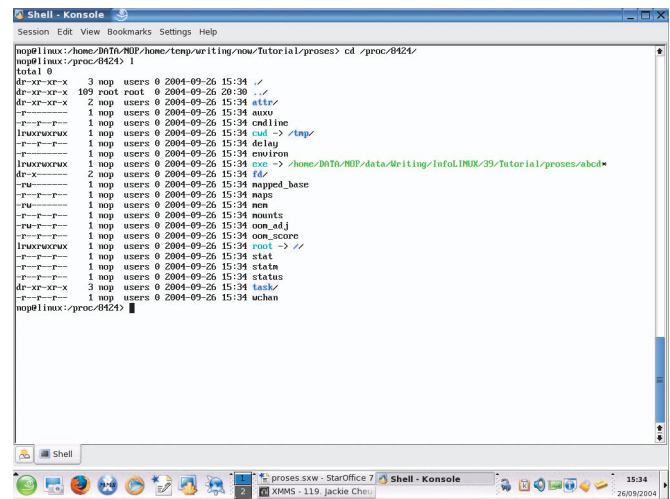
CAKRAWEB
Supporting You to a Web Success

Cyber Building (d/h Elektrindo) 10 th Floor
Jl. Kuningan Barat No. 8 Jakarta Selatan 12710
Phone. (021) 526 8000 Fax. (021) 52 66 444
<http://www.cakraweb.com> - info@cakraweb.com

Note : *) Transfer (restriction apply)
**) 1 year payment



Signal sistem.



Entri direktori proses di /proc.

komunikasi dengan proses. Begitu pula proses A dapat berkomunikasi dengan proses B. Kita atau proses sistem juga dapat berkomunikasi dengan proses daemon.

Salah satu cara tertua komunikasi proses (*Inter Process Communication*, IPC) adalah dengan Signal.

Ketika suatu proses menerima signal, ada tiga tindakan yang mungkin dilakukan oleh suatu proses:

- Mengabaikan *signal*. Namun, ada dua *signal* yang tidak dapat diabaikan, yakni *signal* nomor 9 dan 19.
- Membuat *handler* sendiri untuk *signal*. Ada dua *signal* yang tidak dapat diperlakukan dengan cara demikian, yakni *signal* nomor 9 dan 19.
- Mengikuti *default action signal*.

Signal dapat diberikan dengan perintah `kill`. Walaupun namanya terdengar kejam begitu, sifatnya tidaklah sekejam namanya. Bahkan, pengiriman signal tertentu, umumnya `SIGHUP` pada beberapa daemon menyebabkan daemon tersebut membaca file konfigurasinya dan kemudian mengaplikasikannya.

Untuk melihat signal-signal yang tersedia di sistem berikan perintah berikut ini:

```
kill -1
```

Untuk mengirimkan signal, berikanlah perintah berikut ini:

```
kill -<SIGNAL> <PID>
```

sebagai contoh:

```
kill -KILL 6546
```

Harap diperhatikan bahwa proses juga memiliki informasi hak pemilik. Anda tidak dapat membunuh proses yang bukan milik Anda, misalnya.

Kasus Virus (memori) di Linux

Beberapa analis yang - menurut penulis - agak konyol mengatakan Linux juga akan diserang virus sama seperti halnya Windows dan semua tersebut hanyalah masalah waktu. Mari kita analisa virus di Linux dan hubungannya dengan proses.

Kita tahu bahwa proses di Linux adalah transparan, memiliki skema keamanan seperti file sistem, dan dapat dibatasi dengan *resource limit*. Hal ini adalah fundamental dari sisi proses kenapa virus tidak menyerang di Linux.

Seorang admin yang berhati-hati pada suatu jaringan besar akan menerapkan resource limit pada sistem. Dengan demikian, seorang user hanya boleh menggunakan sekian resource. Dengan program tertentu, proses-proses juga dapat diamati, dan apabila ada proses yang tiba-tiba minta resource besar, sebuah SMS atau mail dapat dikirimkan.

Katakanlah tiba-tiba virus menyerang user xyz di jaringan tersebut. User tersebut adalah pengguna OpenOffice.org dan tidak peduli apapun soal sistem. Virus yang didapatkan dari internet tersebut bermaksud untuk mengacaukan sistem dengan membuat proses sebanyak mungkin dan menguras resource sistem. Semacam stress test.

Ketika virus tersebut berjalan, limit resource xyz akan membatasi kerianya karena

sudah ada pembatasan. Dan, pada limit tertentu, admin akan diberitahu dengan SMS. Admin tersebut, yang ceritanya berdedikasi tinggi, langsung dapat menonaktifkan virus tersebut. Saat ini, secara teknologi, respon cepat (kapan saja, dimana saja, bahkan untuk sistem gerbang masuk yang tidak berfungsi) atas permasalahan sudah sangat memungkinkan.

Dalam konteks tersebut, virus tersebut tidak dapat berbuat apa-apa. Begitupun dengan virus file sistem. Yang terinfeksi hanyalah file-file milik user. Tidak akan berakibat fatal pada sistem.

Tentunya, semua hal tersebut kembali kepada usernya sendiri. Oleh karena itu, jangan menggunakan root dalam penggunaan biasa. Walaupun *by design* Linux aman, keamanan komputer ditentukan lebih dari 75% oleh usernya.

Proses adalah sesuatu yang luar biasa. Salah mengatur proses, maka konsekuensinya besar. Linux telah dikembangkan lebih dari 10 tahun dan terus menerus mengembangkan kemampuan penanganan prosesnya.

Dengan kata lain, manajemen proses adalah hal yang benar-benar menjadi kunci seberapa sebuah sistem operasi bisa dikategorikan serius atau tidak untuk melayani kebutuhan enterprise misalnya. Jadi, penanganan proses bukanlah hal yang sepele dalam sistem operasi, terutama sistem operasi yang didedikasikan khusus sebagai server.

Demikianlah pembahasan kita tentang proses. Selamat mencoba, dan sukses! 🐧
Noprianto (noprianto@infolinux.co.id)