MANAJEMEN PROSES LINUX UBUNTU

1.MANAJEMEN PROSES

Secara umum, manajemen proses di dalam OS (*operating system*) Linux Ubuntu memiliki cara dan mekanisme yang hampir sama dengan manajemen proses dalam OS turunan UNIX (dan Linux) lainnya. Oleh karenanya dalam Bab ini, akan diberikan penjelasan singkat mengenai manajemen proses di dalam OS berbasis Linux pada umumnya, kemudian dilanjutkan dengan pembahasan beberapa hal khusus yang dijumpai di Linux Ubuntu.

1.1 MANAJEMEN PROSES LINUX

1.1.1 TIPE PROSES

Terdapat beberapa tipe proses yang dikenal dalam OS berbasis Linux pada umumnya, antara lain:

Interactive : proses yang dimulai (dan dikontrol oleh) **shell**[1]. Bisa tampak di luar (*foreground*) ataupun hanya di dalam (*background*).

Batch : proses yang tidak berhubungan dengan terminal, tetapi menunggu untuk dieksekusi secara berurutan (sekuensial).

Daemon : proses yang dimulai ketika Linux *booting* dan berjalan secara *background*. Proses ini menunggu permintaan dari proses lainnya, bila tidak ada *request*, maka berada dalam keadaan '*idle*'.

Dalam Linux, sifat-sifat proses dibagi menjadi tiga bagian, yakni: Identitas Proses, Lingkungan, dan Konteks.

1.1.2 IDENTITAS PROSES

Identitas proses memuat beberapa hal penting berikut:

Process ID (PID) → pengenal unik untuk proses; digunakan untuk menentukan proses-proses mana yang dibawa ke dalam OS saat suatu aplikasi membuat *system call*[2] untuk mengirim sinyal, mengubah, atau menunggu proses lainnya. PID adalah 32-bit bilangan yang mengidentifikasikan setiap proses dengan unik. Linux membatasi PID sekitar 0-32767 untuk menjamin kompatibilitas dengan sistem UNIX tradisional.

Mandat (*Credentials*) → setiap proses harus memiliki sebuah *user ID* dan satu atau lebih *group ID* yang menentukan hak proses untuk mengakses sumber daya sistem dan file.

Personality → tidak ditemukan dalam sistem UNIX, namun dalam Linux setiap proses memiliki sebuah pengenal pribadi (*personality*) yang dapat (sedikit) mengubah *system call* tertentu secara *semantic*. Terutama digunakan oleh *library*[3] *emulation* agar *system call* dapat kompatibel dengan bentuk tertentu UNIX.

1.1.3 LINGKUNGAN PROSES

Lingkungan proses diturunkan dari orang tuanya dan terdiri atas dua vektor *null-terminated*[4] sebagai berikut:

- **Vektor argument** berisi daftar argument *command-line* yang digunakan untuk memanggil program yang berjalan; secara konvensional dimulai dengan nama programnya sendiri.
- **Vektor lingkungan** merupakan sebuah daftar pasangan "NAME=VALUE" yang menghubungkan nama variabel lingkungan dengan nilai tekstual tertentu.

1.1.4 KONTEKS PROSES

Yang dimaksud dengan konteks proses adalah keadaan (perubahan konstan) dari suatu program yang berjalan pada setiap titik dalam satuan waktu. Konteks proses terdiri atas konteks penjadwalan, *accounting*, tabel file, konteks *file-system*, tabel penanganan sinyal, dan konteks *virtual-memory*.

Konteks penjadwalan adalah bagian terpenting dari proses konteks; berupa informasi yang dibutuhkan oleh penjadwal (*scheduler*) untuk menghentikan sementara (*suspend*) dan menjalankan kembali (*restart*) proses tersebut.

Kernel menyimpan informasi statistik (***accounting***) mengenai sumber daya yang digunakan saat ini oleh tiap proses dan total sumber daya yang digunakan oleh proses tersebut sepanjang hidupnya (selama dieksekusi).

Tabel file (***file table***) adalah sebuah larik pointer yang merujuk pada struktur file kernel. Saat membuat *system call* file I/O, proses merujuk pada file berdasarkan indeksnya dalam tabel ini.

Bila tabel file berisi daftar file terbuka yang ada, ***file-system context*** digunakan untuk meminta pembukaan file baru. Root saat ini dan default direktori yang akan digunakan untuk file baru disimpan di sini.

Tabel penanganan sinyal (***signal-handler table***) mendefinisikan rutin dalam ruang alamat proses yang akan dipanggil saat sinyal tertentu tiba.

Konteks virtual-memory (***virtual-memory context***) dari sebuah proses menggambarkan seluruh isi dari ruang alamat pribadinya.

Adapun beberapa status proses yang dikenal dalam Linux, antara lain sebagai berikut:

Task running : proses sedang ataupun siap dieksekusi oleh CPU

Task interruptible : proses sedang menunggu sebuah kondisi. Interupsi, sinyal, ataupun pelepasan sumber daya akan membangunkan proses

Task uninterruptible : proses sedang tidur dan tidak dapat dibangunkan oleh suatu sinyal

Task stopped : proses dihentikan, misalnya oleh sebuah *debugger*

Task zombie : proses telah berhenti, namun masih memiliki struktur data **task_struct** di task vektor dan masih memegang sumber daya yang sudah tidak digunakan lagi

1.1.5 PROSES DAN THREAD

Linux menggunakan representasi internal yang sama untuk proses dan *thread*; sederhananya sebuah **thread** adalah sebuah proses baru yang membagi ruang alamat yang sama dengan orang tuanya.

Perbedaan hanya terlihat pada saat sebuah *thread* baru diciptakan dengan *system call* **clone**.

Fork menciptakan sebuah proses baru dengan proses konteks-nya sendiri yang baru. Sementara **clone** menciptakan sebuah proses baru dengan identitasnya sendiri, namun diizinkan untuk membagi struktur data orang tuanya. **Clone** memberikan suatu aplikasi control yang baik tentang apa yang di-*share* di antara dua *thread*.

Empat buah **argumen clone** yang umum dijumpai adalah sebagai berikut:

fn : fungsi yang akan dieksekusi oleh *thread*

arg : pointer ke data yang dibawa oleh *fn*

flags : sinyal yang dikirim ke orang tua ketika anak berakhir dan pembagian sumber daya antara anak dan orang tua

child_stack : pointer *stack* untuk proses anak

1.2 MANAJEMEN PROSES LINUX UBUNTU

Dalam subbab ini akan dijelaskan beberapa layanan dan komponen yang berperan dalam manajemen proses di OS berbasis Linux Ubuntu, meliputi: menjalankan layanan saat *bootup*, menjalankan proses *boot loading*, menjalankan dan menghentikan layanan secara manual, tugas-tugas penjadwalan, dan *shell* di Ubuntu.

1.2.1 MENJALANKAN LAYANAN SAAT BOOTUP

Meskipun kebanyakan orang pada umumnya hanya melihat keadaan komputer sedang hidup atau mati, dalam Ubuntu tidak hanya terdapat dua keadaan - hidup atau mati - namun ada beberapa keadaan di antaranya. Istilah yang

digunakan adalah *runlevel*, yang mengendalikan layanan-layanan sistem apa yang dimulai pada saat *bootup*[5]. Layanan ini sederhananya adalah aplikasi yang berjalan secara *background* yang menyediakan beberapa fungsi yang diperlukan ke dalam sistem, seperti menangkap informasi dari *mouse* dan menampilkannya ke layar monitor, dan sebagainya.

Layanan-layanan pada umumnya di-*load* dan dijalankan (dimulai) selama proses *boot*, sebagaimana seperti layanan pada Microsoft Windows. Namun secara internal, Ubuntu menggunakan sebuah sistem yang dikenal dengan **Upstart** untuk *fast booting*, yang umum dikenal oleh para veteran Linux.

Anda dapat mengatur hampir seluruh aspek dalam komputer Anda dan mengatur bagaimana perilakunya setelah *booting* dengan mengkonfigurasi *boot scripts*, atau dengan menggunakan berbagai alat *system administration* yang disediakan dalam Ubuntu. Pembahasan lebih lanjut akan diberikan pada subbab berikutnya.

1.2.2 MENJALANKAN PROSES BOOT LOADING

Mekanisme *boot loading* diselesaikan melalui **Basic Input Output System** atau **BIOS**. BIOS adalah sebuah aplikasi yang disimpan dalam suatu chip di *motherboard* yang menginisialisasi perangkat keras *motherboard*. BIOS akan menyiapkan sistem agar siap me-*load* dan menjalankan *software* yang dikenal sebagai sistem operasi (OS).

Sebagai langkah terakhir, kode BIOS akan mencari suatu program khusus yang dikenal sebagai *boot loader* atau kode *boot*. Perintah-perintah dalam kode kecil ini akan memberitahu BIOS letak kernel Linux, bagaimana cara untuk me-*load* kernel tersebut ke dalam memori, dan bagaimana untuk mulai menjalankannya.

1.2.2.1 Loading Kernel Linux

Sebagaimana yang diketahui, kernel-lah yang mengatur sumber daya sistem. Di dalam Linux, setiap aplikasi adalah sebuah proses, dan kernel memberikan setiap proses sebuah bilangan yang disebut sebagai ***process ID*** (**PID**).

Pertama, kernel Linux akan me-*load* dan menjalankan sebuah proses bernama **init**, yang juga dikenal sebagai “ayah dari semua proses” karena ia yang memulai semua proses lainnya.

Langkah berikutnya dari proses *boot* dimulai dengan sebuah pesan bahwa kernel Linux sedang *loading*, dan sekumpulan pesan yang memberikan informasi tentang status setiap perintah akan ditampilkan di layar.

Untuk menjalankan sistem dengan baik, layanan sistem harus dimulai terlebih dahulu. Layanan-layanan tersebut adalah aplikasi-aplikasi yang memungkinkan pengguna untuk dapat berinteraksi dengan sistem komputer.

1.2.2.2 Layanan Sistem dan Runlevel

Perintah **init** mengarahkan Ubuntu pada suatu keadaan sistem tertentu, umumnya dikenal sebagai **runlevel**.

Runlevel akan menentukan layanan sistem mana yang akan dijalankan di antara layanan yang tersedia, dan dalam urutan yang benar. Sebuah *runlevel* khusus digunakan untuk menghentikan sistem, dan *runlevel* khusus lainnya digunakan untuk *maintenance* (menjaga performa) sistem. Jadi, setiap *runlevel* memiliki tujuannya masing-masing. Anda dapat menggunakan *runlevel* untuk mengatur layanan sistem yang berjalan di komputer Anda.

1.2.2.3 Definisi Runlevel

Untuk sistem Ubuntu, *runlevel-runlevel* Ubuntu didefinisikan dalam **/etc/init.d**.

Setiap *runlevel* memberitahu perintah **init** tentang layanan mana yang dimulai dan dihentikan. Meskipun setiap *runlevel* dapat memiliki definisinya sendiri, Ubuntu mengadopsi beberapa standar *runlevel* sebagai berikut:

> **Runlevel 0** – dikenal sebagai "**halt**", *runlevel* ini digunakan untuk menghentikan sistem.
>
> **Runlevel 1** – adalah *runlevel* khusus, didefinisikan sebagai "**single**", yang akan mem-*boot* Ubuntu ke *shell prompt* akses *root*, di mana hanya pengguna *root* yang dapat *log in*.
>
> **Runlevel 2** – adalah *runlevel* default Ubuntu.
>
> **Runlevel 3-5** – *runlevel-runlevel* ini tidak digunakan di Ubuntu, namun banyak digunakan pada distro Linux lainnya.
>
> **Runlevel 6** – *runlevel* ini digunakan untuk *reboot* sistem.

Runlevel 1 (dikenal juga sebagai mode *single-user* atau *maintenance mode*) banyak digunakan untuk memperbaiki *file system* dan mengubah *password root* pada sebuah sistem bila *password*-nya terlupakan.

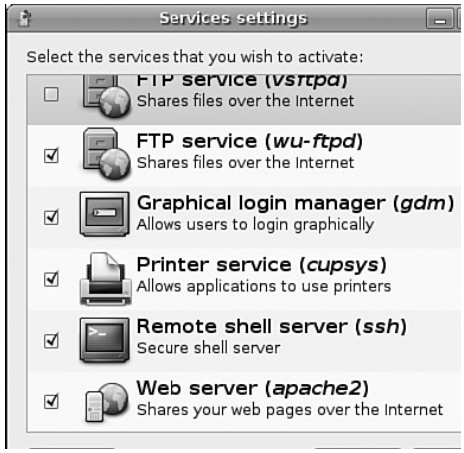
1.2.2.4 Booting dengan Default Runlevel

Ubuntu masuk ke dalam *runlevel 2* secara *default*, yang berarti memulai sistem secara normal dan meninggalkan Anda dalam *X Window System*[6] mencari *Gnome login prompt*. Ubuntu mengetahui apa yang butuh di-*load* oleh *runlevel 2* dengan mencari di dalam direktori **rc*.d** dalam **/etc**.

1.2.2.5 Menentukan Layanan yang Dijalankan pada Saat *Boot* dengan *Administrative Tools*

Dalam kotak dialog Layanan (gambar 1), Ubuntu mendaftarkan seluruh layanan yang dapat Anda aktifkan secara otomatis pada saat *booting*. Umumnya, layanan-layanan tersebut telah di-*set* ***enable*** (yang berarti dijalankan otomatis) secara *default*, namun dengan mudah Anda dapat menghilangkan tanda centang untuk layanan yang tidak Anda inginkan dan selanjutnya klik **OK**. Sebagai catatan, Anda tidak direkomendasikan untuk menonaktifkan layanan secara acak

dengan tujuan untuk “membuat sistem berjalan lebih cepat”. Beberapa layanan vital bagi operasi-operasi dalam komputer Anda, seperti *graphical login manager* dan *system communication bus*.



Gambar 1. Kotak dialog Layanan

1.2.2.6 Mengubah Runlevel

Setelah membuat perubahan-perubahan dalam layanan sistem dan *runlevel*, Anda dapat menggunakan perintah **telinit** untuk mengubah *runlevel* secara langsung pada sebuah sistem Ubuntu yang sedang berjalan. Dengan cara ini, sistem administrator dapat mengubah bagian tertentu dari sistem yang berjalan guna menerapkan perubahan pada layanan atau untuk memberi efek yang telah dibuat (seperti meminta kembali alamat jaringan untuk suatu *interface* jaringan).

1.2.3 MENJALANKAN DAN MENGHENTIKAN LAYANAN SECARA MANUAL

Jika Anda mengubah konfigurasi file untuk suatu layanan sistem, biasanya Anda harus menghentikan dan me-*restart* layanan tersebut agar konfigurasi yang baru dapat dibaca. Jika Anda meng-konfigurasi ulang X server, biasanya lebih mudah bila Anda mengubah *runlevel 2* menjadi *runlevel 1* agar *testing* lebih mudah, dan kemudian kembali lagi ke *runlevel 2* untuk mengaktifkan *graphical login*.

Cara tradisional untuk mengatur suatu layanan (sebagai *root*) adalah dengan memanggil nama **/etc/init.d** layanan pada *command line*[7] dengan kata kunci yang tepat, seperti **start**, **status**, atau **stop**. Sebagai contoh, untuk memulai *Apache web server*, panggil *script* **/etc/init.d/apache2** seperti berikut:

```
sudo /etc/init.d/apache2 start
Starting apache 2.2 web server                    [OK]
```

Script di atas akan mengeksekusi program yang diperlukan dan melaporkan statusnya. Menghentikan layanan sama mudahnya, hanya dengan menggunakan kata kunci **stop**.

1.2.4 TUGAS-TUGAS PENJADWALAN

Ada tiga cara untuk mengatur jadwal perintah dalam Ubuntu. Yang pertama adalah perintah **at**, yang menentukan sebuah perintah agar dijalankan pada waktu dan tanggal tertentu dari hari ini. Perintah kedua adalah perintah **batch**, yang tidak lain adalah *script* yang mengarahkan Anda kembali ke perintah **at** dengan beberapa opsi tambahan sehingga perintah Anda berjalan pada saat sistem dalam kondisi "*idle*". Yang terakhir adalah *daemon* **cron**, yang merupakan cara Linux dalam mengeksekusi tugas-tugas pada suatu waktu yang telah ditentukan.

1.2.4.1 Menggunakan 'at' dan 'batch' untuk Menjalankan Tugas yang Akan Dijalankan Kemudian

Jika terdapat suatu tugas dengan waktu tertentu yang ingin Anda jalankan, namun tidak pada saat Anda sedang *log in*, Anda dapat meminta Ubuntu untuk menjalankannya nanti dengan menggunakan perintah **at**. Untuk menggunakan **at**, Anda harus menyatakan waktu yang Anda inginkan untuk dijalankan dan kemudian tekan **Enter**. Anda selanjutnya akan melihat *prompt* baru yang dimulai dengan **at>**, dan apapun yang Anda ketikkan di sana hingga Anda menekan **Ctrl+D** akan menjadi perintah yang dijalankan oleh **at**.

Saat waktu yang ditentukan tiba, **at** akan menjalankan setiap perintah satu per satu dan berurutan, yang berarti perintah berikutnya dapat bergantung pada hasil dari perintah sebelumnya. Perhatikan contoh berikut:

```
[paul@caitlin ~]$ at now + 7 hours
at> wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.10.tar.bz2
at> tar xvfjp linux-2.6.10.tar.bz2
at> <EOT>
job 2 at 2005-01-09 17:01
```

Contoh di atas memperlihatkan bahwa perintah **at** dijalankan setelah pukul 5 p.m., perintah **at** digunakan untuk men-*download* dan meng-ekstrak kernel Linux terakhir pada saat jaringan sedang diam. *+ 7 hours* berarti perintah akan dijalankan sejak perintah **at** diberikan (yakni 5 p.m.) ditambah 7 jam, sehingga *download* dimulai tepat pada tengah malam.

Saat *job* Anda diberikan, **at** akan melaporkan nomor *job*, tanggal, dan waktu *job* tersebut akan dieksekusi; penunjuk antrian; ditambah pemilik *job* (yakni Anda). Nomor *job* dan penunjuk antrian *job* sangatlah penting. Saat Anda menjadwalkan sebuah *job* dengan **at**, *job* tersebut ditempatkan pada antrian "**a**" secara *default*, yang berarti akan berjalan pada waktu yang telah ditentukan dan menggunakan jumlah sumber daya secara normal.

Alternatif perintah lainnya, **batch**, tidak lain adalah *script shell* yang memanggil **at** dengan beberapa opsi tambahan. Opsi-opsi tersebut *(-q b –m now*) menentukan agar **at** berjalan pada antrian **b** (*-q b*), memberikan pemberitahuan pada *user* setelah selesai dijalankan (*-m*), dan berjalan sesegera mungkin (*now*). Karena **batch** selalu menggunakan *now* sebagai penanda waktunya, Anda tidak

perlu lagi menentukan waktunya, **batch** akan segera menjalankan perintahnya pada saat sistem dalam kondisi "*idle*".

Konfigurasi *default* untuk **at** dan **batch** mengizinkan setiap *user* untuk menggunakannya, yang tidak selalu merupakan hal yang diinginkan. Pengaturan akses ditentukan melalui dua file: **/etc/at.allow** dan **/etc/at.deny**. Secara *default*, **at.deny** ada namun kosong, sehingga mengizinkan setiap orang untuk menggunakan **at** dan **batch**. Anda dapat memasukkan nama *user* ke dalam **at.deny**, satu per baris, untuk menolak *user-user* tersebut melakukan penjadwalan *job*.

Selain itu, Anda dapat menggunakan file **at.allow**, yang tidak ada secara *default*. Jika Anda memiliki file **at.allow** yang kosong, maka tidak ada seorangpun kecuali *root* yang diizinkan untuk melakukan penjadwalan *job*. Seperti pada **at.deny**, Anda dapat menambahkan nama-nama *user* pada **at.allow**, satu nama per baris, dan *user-user* itu akan memiliki hak untuk melakukan penjadwalan *job*.

Anda harus menggunakan salah satu file **at.deny** atau **at.allow**. Saat seseorang mencoba untuk menjalankan **at** atau **batch**, Ubuntu akan mengecek nama *user* tersebut dalam **at.allow**. Jika namanya tidak ada di sana, atau bila **at.allow** tidak ada, Ubuntu akan mengecek nama *user* tersebut di **at.deny**. Jika namanya ada di sana atau bila **at.deny** tidak ada, maka *user* itu tidak diizinkan untuk melakukan penjadwalan *job*.

1.2.4.2 Menggunakan 'cron' untuk Menjalankan Tugas yang Akan Dijalankan Berulang Kali

Perintah **at** dan **batch** sangat berguna bila Anda hanya ingin mengeksekusi satu tugas pada waktu berikutnya, namun kurang berguna bila Anda ingin menjalankan sebuah tugas yang berulang kali. *Daemon* **cron** digunakan untuk menjalankan tugas-tugas berulang kali berdasarkan permintaan sistem dan *user*. **Cron** memiliki pengaturan izin yang hampir sama dengan **at**: para *user* yang terdaftar dalam file **cron.deny** tidak diizinkan untuk menggunakan **cron**, dan *user* yang terdaftar dalam file **cron.allow** diizinkan untuk menggunakannya. File kosong **cron.deny** – *default* sistem - berarti setiap orang dapat mengatur *job*. File kosong **cron.allow** berarti tidak ada seorangpun kecuali *root* yang memiliki izin untuk mengatur *job*.

Ada dua tipe *job*: *job* sistem dan *job user*. Hanya *root* yang dapat mengedit *job* sistem, sementara setiap *user* yang namanya terdaftar dalam **cron.allow** atau tidak muncul dalam **cron.deny** dapat menjalankan *job user*. *Job* sistem dikendalikan melalui file **/etc/crontab**, yang secara *default* terlihat seperti berikut:

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user command
17 * * * * root run-parts —report /etc/cron.hourly
```

```
25 6 * * * root test –x /usr/sbin/anacron || run-parts —report /etc/cron.daily
47 6 * * 7 root test –x /usr/sbin/anacron || run-parts —report /etc/cron.weekly
52 6 1 * * root test –x /usr/sbin/anacron || run-parts —report /etc/cron.monthly
```

Dua baris pertama menentukan *shell* mana yang harus digunakan untuk mengeksekusi *job* dan mencari *path* untuk eksekusi yang akan digunakan.

Baris berikutnya dimulai dengan tanda (**#**) yang artinya adalah komentar, dan karenanya diabaikan. Empat baris berikutnya adalah yang terpenting karena empat baris itulah yang merupakan *job* itu sendiri.

Setiap *job* dirincikan dalam tujuh *field* yang mendefinisikan waktu *job* dijalankan, pemilik, dan perintah *job*. Lima perintah pertama menentukan waktu eksekusi dalam urutan yang agak aneh: menit (0-59), jam (0-23), hari dalam satu bulan (1-31), bulan dalam satu tahun (1-12), dan hari dalam satu minggu (0-7). Untuk hari dalam seminggu, baik 0 dan 7 merepresentasikan hal yang sama yakni Minggu, selanjutnya 1 adalah Senin, 2 adalah Selasa, 3 adalah Rabu, dan seterusnya. Jika Anda ingin menyatakan setiap waktu (yakni, setiap menit, setiap jam, setiap hari, dan seterusnya) gunakan tanda *asterisk*, *****.

Field selanjutnya menunjukkan nama *user* dari pemilik *job* tersebut. Dan *field* terakhir adalah perintah yang akan dieksekusi.

Jadi, dari contoh di atas dapat disimpulkan bahwa *job* pertama berjalan pada menit 17, setiap jam dari setiap hari setiap bulannya, dan mengeksekusi perintah **run-parts /etc/cron.hourly**.

Sebagaimana dalam *job* sistem, juga terdapat *job user* bagi para *user* yang memiliki izin akses. *Job user* disimpan dalam direktori **/val/spool/cron**, di mana setiap *user* memiliki file dengan namanya sendiri, jika tidak nama *user*-nya – contohnya, **/val/spool/cron/paul** atau **/val/spool/cron/root**. Isi dari file ini terdiri atas *job-job* yang ingin dijalankan oleh *user* dan mengambil format yang sama seperti dalam file **/etc/crontab**, dengan pengecualian bahwa pemilik *job* tidak disebutkan karena akan selalu sama dengan nama file.

1.2.5 UBUNTU SHELL

Ubuntu menyediakan *shell-shell* yang memiliki kapabilitas, fleksibilitas, dan sangat *powerful*. Setiap *shell* berbeda satu sama lainnya, namun memiliki berbagai perintah *built-in* dan *command-line prompt* yang dapat dikonfigurasi, dan dapat menyertakan fitur-fitur seperti *command-line history*, kemampuan untuk memanggil kembali dan menggunakan *command line* sebelumnya, dan kemampuan untuk mengedit *command-line*.

Di antara sekian banyak *shell* yang dapat digunakan, kebanyakan orang tetap memilih menggunakan *shell default*, yakni **bash**. Hal ini dikarenakan **bash** mampu mengerjakan segala sesuatu yang kebanyakan orang perlukan. Anda

disarankan untuk mengganti *shell* Anda hanya jika Anda memang sangat membutuhkannya.

Tabel 1 di bawah memperlihatkan setiap *shell*, bersama dengan deskripsi dan lokasinya, dalam sistem file Ubuntu Anda.

Nama	Deskripsi	Lokasi
bash	the Bourne Again Shell	/bin/bash
ksh	the Korn Shell	/bin/ksh, /usr/bin/ksh
pdksh	a symbolic link to ksh	/usr/bin/pdksh
rsh	the Restricted Shell (for network operation)	/usr/bin/rsh
sh	a symbolic link to bash	/bin/sh
tcsh	a csh-compatible shell	/bin/tcsh
zsh	a compatible csh, ksh, and sh shell	/bin/zsh

Tabel 1. Shell dalam Ubuntu

1.2.5.1 Shell Command Line

Anda dapat memanfaatkan *shell command line* untuk melaksanakan berbagai tugas yang berbeda, meliputi:

- Mencari file atau direktori dengan menggunakan *pattern-matching* (kecocokan pola), atau ekspresi
- Memperoleh data dari dan mengirimkan data ke sebuah file atau perintah, dikenal sebagai pengarahan *input* dan *output*
- Memberi atau menyeleksi *output* sebuah program ke perintah lainnya (disebut menggunakan *pipes*)

Sebuah *shell* juga dapat memiliki perintah kendali *job built-in* untuk menjalankan *command line* sebagai suatu proses *background*, *suspend* (menghentikan sementara) sebuah program yang sedang berjalan, menerima atau *kill* (membunuh) program-program yang sedang berjalan atau dalam keadaan *suspend* secara selektif, dan menjalankan berbagai tipe kendali proses lainnya.

1.2.5.2 Proses Background

Shell mengizinkan Anda untuk memulai sebuah perintah dan kemudian menjalankannya sebagai proses *background* dengan menggunakan tanda

ampersand (**&**) di akhir *command line*. Cara ini biasa digunakan pada *command line* dengan **X** *terminal window* untuk memulai suatu klien dan kembali ke *command line*. Sebagai contoh, untuk menjalankan jendela terminal yang lain dengan menggunakan *xterm client*,

```
$ xterm &
[3] 1437
```

Dari contoh di atas diperlihatkan angka 3 yang menunjukkan nomor *job* atau nomor referensi untuk proses *shell*, dan sebuah nomor PID (1437 dalam kasus ini). Jendela *xterm* dapat ditutup dengan menggunakan perintah *built-in shell* **kill**, bersama dengan **nomor** *job* seperti di bawah ini:

```
$ kill %3
```

Atau proses tersebut dapat di-*kill* dengan menggunakan perintah **kill** bersama dengan **PID** *job*, seperti berikut:

```
$ kill 1437
```

Proses *background* dapat digunakan dalam *shell script* untuk memulai perintah yang memerlukan waktu yang lama, seperti *backup*:

```
# tar –czf /backup/home.tgz /home &
```

1. PENJADWALAN

Yang dimaksud dengan penjadwalan adalah pekerjaan (*job*) untuk mengalokasikan **waktu CPU** atas **tugas-tugas** (*tasks*) yang berbeda dalam sebuah OS.

Umumnya penjadwalan diasosiasikan dengan *running* dan *interrupting* proses; di Linux, penjadwalan juga meliputi *running* dari berbagai tugas kernel.

Tugas kernel yang berjalan meliputi baik tugas-tugas yang diminta oleh sebuah proses yang sedang berjalan maupun tugas-tugas yang berjalan secara internal atas permintaan sebuah *device driver*.

Sebagaimana dalam versi 2.5, algoritma penjadwalan yang baru yang diterapkan dalam Linux adalah ***preemptive, priority-based.***

2.1 PENJADWALAN PROSES LINUX

Linux menggunakan dua **algoritma penjadwalan** proses, yaitu:

1. Algoritma ***time-sharing*** untuk *preemptive scheduling* wajar/ adil (*fairness*) di antara banyak proses
2. Algoritma ***real-time*** untuk tugas-tugas di mana prioritas absolut lebih penting daripada kewajaran (*fairness*)

Kelas penjadwal proses mendefinisikan algoritma mana yang digunakan.

Untuk proses *time-sharing*, Linux menggunakan algoritma berdasarkan nilai **credit, prioritas.**

- Aturan *credit*

  Credit := (credit/2)+priority

  Faktor baik di dalam *history* proses maupun prioritasnya

- Sistem *credit* ini secara otomatis memprioritaskan interaktif atau proses I/O *bound*

Untuk kelas penjadwalan ***real-time,*** Linux menerapkan ***FCFS*** dan ***round-robin***; di kedua kasus, setiap proses memiliki prioritas sebagai tambahan atas kelas penjadwalnya.

- Penjadwal menjalankan proses dengan prioritas tertinggi; untuk proses dengan prioritas sama, penjadwal menjalankan proses yang menunggu paling lama
- Proses FIFO terus berjalan hingga mereka selesai atau di-blok
- Proses *round-robin* akan dipaksa (*preempted*) sementara dan dipindahkan ke akhir antrian, sehingga proses *round-robin* dari prioritas yang sama secara otomatis membagi waktu di antara mereka

2.2 MENGGUNAKAN KENDALI DAN PENJADWALAN PRIORITAS DI LINUX UBUNTU

Tidak ada proses yang dapat menggunakan sumber daya sistem (CPU, memori, akses disk, dan sebagainya) sesuka hatinya. Lagipula, tugas utama kernel adalah untuk mengatur penggunaan sumber daya sistem dengan adil. Caranya adalah dengan memberikan prioritas kepada setiap proses sehingga beberapa proses memiliki akses yang lebih baik atas sumber daya sistem, sementara beberapa proses lainnya mungkin harus menunggu sedikit lebih lama hingga giliran mereka tiba.

Penjadwalan prioritas dapat menjadi alat yang sangat berguna dalam mengatur aplikasi pendukung sistem atau dalam situasi di mana penggunaan CPU dan RAM harus diatur atau dialokasikan untuk tugas tertentu. Dua aplikasi turunan yang disertakan dalam Ubuntu adalah perintah **nice** dan **renice**. (**nice** adalah bagian dari paket GNU[8] sh-utils, sedangkan **renice** diwariskan dari BSD[9] Unix).

Perintah **nice** digunakan dengan opsi –n bersama dengan suatu pernyataan dalam range -20 hingga 19, dalam urutan dari prioritas tertinggi hingga terendah (semakin rendah angkanya, semakin tinggi prioritasnya). Sebagai contoh, untuk menjalankan klien ‘gkrellm’ dengan prioritas rendah, gunakan perintah **nice** seperti berikut:

```
$ nice –n 12 gkrellm &
```

Perintah **nice** pada dasarnya digunakan untuk tugas-tugas *disk* atau CPU yang berat sehingga mungkin dapat menyebabkan penurunan kinerja sistem. Perintah **renice** dapat digunakan untuk me-reset prioritas dari proses yang berjalan atau mengendalikan prioritas dan penjadwalan seluruh proses yang dimiliki oleh seorang *user*.

Administrator sistem juga dapat menggunakan perintah **time** untuk mendapatkan informasi tentang seberapa banyak waktu dan seberapa besar sumber daya sistem yang dibutuhkan untuk sebuah tugas. Perintah ini digunakan bersama dengan perintah (atau *script*) lainnya sebagai argumen seperti contoh berikut:

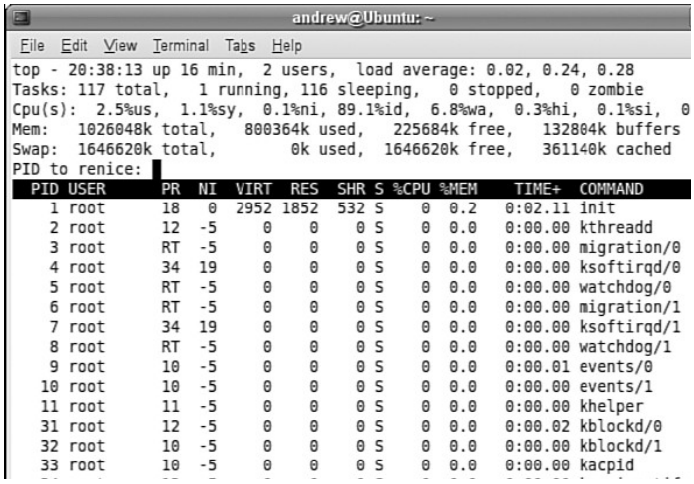
```
# time –p find / -name core –print
/dev/core
/proc/sys/net/core
Real 61.23
User 0.72
Sys 3.33
```

Output perintah di atas memperlihatkan waktu dari awal hingga selesai, bersama dengan waktu *user* dan sistem yang dibutuhkan. Faktor lainnya yang dapat Anda minta adalah informasi statistik memori, penggunaan CPU, dan sistem *input/output* (I/O).

Hampir semua alat monitoring proses grafis mencakup beberapa bentuk kendali atau manajemen proses. Salah satu program monitoring (dan kendali) yang dikenal adalah **top**. Berdasarkan perintah **ps**, perintah **top** memperlihatkan tampilan berbasis teks yang menampilkan *output* berbasis *console* yang terus di-*update* dari proses-proses yang sedang berjalan yang paling banyak menggunakan CPU. Perintahnya sebagai berikut:

```
$ top
```

Setelah Anda menekan **Enter**, Anda akan melihat tampilan seperti yang ditunjukkan pada gambar 2 di bawah. Perintah **top** memiliki beberapa perintah interaktif: menekan **h** menampilkan jendela *help*, menekan **k** meminta Anda untuk memasukkan PID sebuah proses untuk di-*kill*, menekan **n** meminta Anda untuk memasukkan PID proses untuk mengubah *nice value*-nya.



Gambar 2. Layar jendela top

Perintah **top** sedikit banyak menampilkan informasi tentang sistem Anda. Proses-proses dapat diurutkan berdasar PID, usia, penggunaan CPU atau memori, waktu, atau *user*. Perintah ini juga menyediakan layanan manajemen proses, dan administrator sistem dapat menggunakan kata kunci **k** dan **r** untuk membunuh (*kill*) dan menjadwal ulang (*reschedule*) tugas-tugas yang sedang berjalan secara berurutan.

Perintah **top** menggunakan jumlah yang sama dari memori, jadi mungkin Anda ingin bertindak lebih bijaksana dalam pemanfaatannya dan tidak menjalankannya sepanjang waktu. Saat Anda selesai menggunakannya, Anda cukup menekan **q** untuk keluar (*quit*) dari **top**.

2.3 SYMMETRIC MULTIPROCESSING

Linux 2.0 adalah kernel Linux pertama yang mendukung *symmetric multiprocessing* (**SMP)** *hardware*. Hal ini berarti proses atau *thread* yang terpisah dapat berjalan secara paralel pada prosesor yang terpisah.

Untuk mempertahankan persyaratan sinkronisasi kernel yang ***nonpreemptible***, SMP menentukan batasan, melalui suatu ***spinlock*** tunggal kernel, sehingga hanya satu prosesor pada satu waktu dapat menjalankan kode ***mode-kernel***.

Para pengguna distro Ubuntu selalu disarankan untuk meng-*update* versi kernel yang digunakannya. Selain meningkatkan keamanan dan memperbaiki *bug* yang ada pada versi-versi sebelumnya, kernel yang terus di-*update* akan memberikan kinerja dan kemampuan yang lebih baik bagi para penggunanya. Versi kernel terakhir yang digunakan Ubuntu adalah **2.6.28-11** (untuk Ubuntu 9.04). Untuk mengetahui lebih lanjut mengenai informasi kernel dan cara meng-*install* kernel baru dapat Anda lihat di: *https://help.ubuntu.com/community/Kernel/Compile*.

Pustaka:

http://id.wikipedia.org/wiki/GNU

http://id.wikipedia.org/wiki/Berkeley_Software_Distribution

http://ikc.dinus.ac.id/umum/ibam/ibam-os-html/x7914.html

http://www.gregvogl.net/courses/os/glossary.htm

Hudson, Andrew and Paul Hudson. 2008. "*Ubuntu Unleashed 2008 Edition"*. USA: Sams Publishing

ictcenter-purwodadi.net/pustakamaya/download.php?id=448

Silberschatz, Abraham; Peter B. Galvin, and Greg Gagne. 2006. "*Operating System Concepts with Java: Seventh Edition"*. USA: Wiley & Sons. Chp.20: The Linux System

zaki-math.web.ugm.ac.id/download/book/.../PraktikumOS-Bab8.pdf

1 Program yang dapat berkomunikasi dengan pengguna komputer (manusia) dan memungkinkan pengguna untuk berinteraksi dengan komputer.

2 Suatu fungsi OS tingkat rendah, tersedia sebagai sebuah fungsi dalam bahasa tingkat tinggi bagi programmer, berfungsi untuk meminta OS agar menjalankan sebuah layanan seperti alokasi memori atau I/O.

3 Kumpulan yang terdiri atas subrutin dan *class* yang dapat digunakan untuk mengembangkan *software*.

4 Vektor yang terdiri atas *null-terminated strings*, yang berarti string-string yang memiliki nilai *null*.

5 Proses *bootstrap* (teknik di mana suatu program yang sederhana mengaktifkan program yang lebih memiliki sistem yang lebih rumit) yang memulai OS saat pengguna komputer menghidupkan sistem komputer.

6 Dikenal juga sebagai X, yakni *interface* jaringan berbasis grafis yang banyak dijumpai dalam distro Linux, yang menyediakan basis yang luas atas alat-alat grafis dan *window manager*.

7 Mekanisme yang digunakan untuk berinteraksi dengan OS komputer atau *software* dengan cara mengetikkan perintah-perintah untuk menjalankan tugas-tugas tertentu.

8 GNU’s Not Unix adalah suatu sistem operasi yang sepenuhnya terdiri atas perangkat-perangkat lunak bebas. GNU memiliki rancangan yang mirip dengan UNIX, namun tidak memiliki kode-kode UNIX.

9 Berkeley Software Distribution dibangun dan dikembangkan oleh *Computer System Research Group* (CSRG) di Universitas California di Berkeley. BSD memiliki lisensi tersendiri yang memungkinkan setiap orang bebas melakukan pengembangan dan menggunakan kode sumber BSD.