

SISTEM OPERASI

S1 Teknologi Informasi



Official Line
Informatics Lab



published by school of computing



LEMBAR PENGESAHAN

Saya yang bertanda tangan di bawah ini:

Nama : Novian Anggis Suwastika, S.T., M.T.


NIK : 13850083

Koordinator Mata Kuliah : Sistem Operasi

Prodi : S1 Teknologi Informasi

Menerangkan dengan sesungguhnya bahwa modul ini digunakan untuk pelaksanaan praktikum di Semester Genap Tahun Ajaran 2020/2021 di Laboratorium Informatika Fakultas Informatika Universitas Telkom.

Bandung, 2 Februari 2021

 Mengesahkan,
Koordinator Mata Kuliah Sistem Operasi

 Mengetahui,
Kaprodi S1 Teknologi Informasi

Novian Anggis Suwastika, S.T., M.T.

Ir. Endro Ariyanto, S.T., M.T.

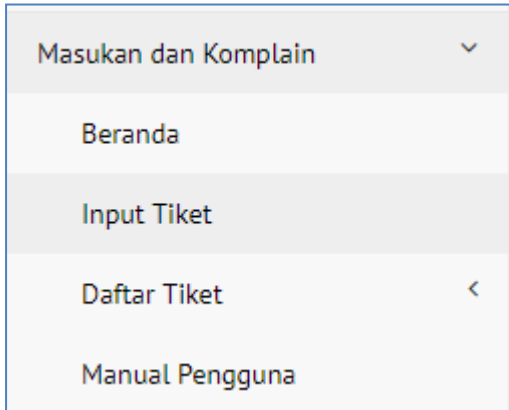
Peraturan Praktikum

Laboratorium Informatika 2020/2021

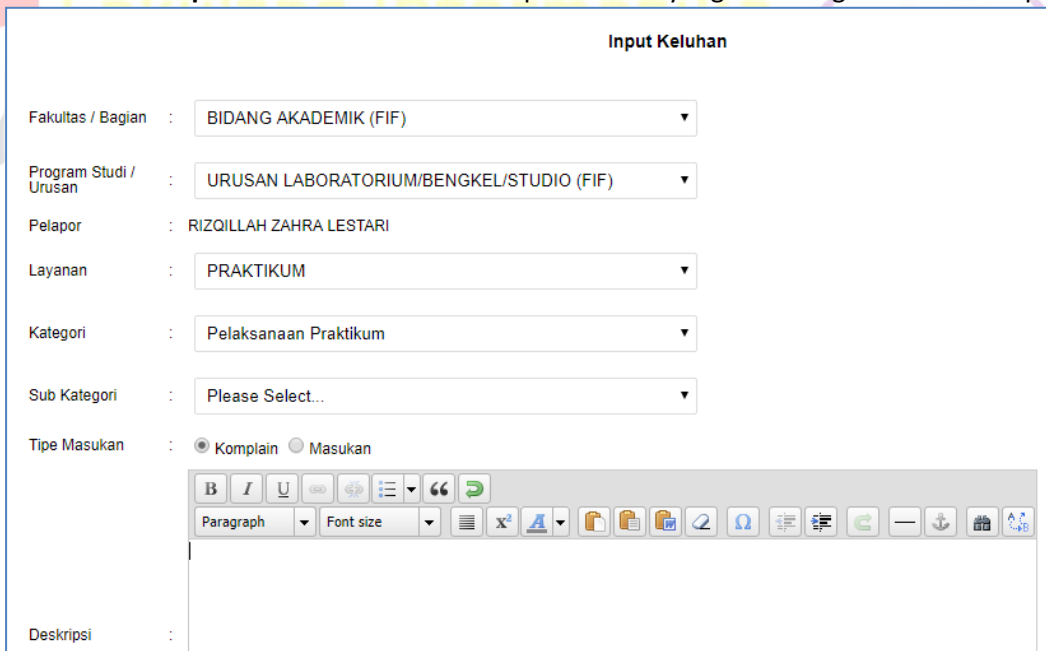
1. Praktikum diampu oleh dosen kelas dan dibantu oleh asisten laboratorium dan asisten praktikum.
2. Praktikum dilaksanakan di Gedung Kultubai Selatan & Kultubai Utara (IFLAB 1 s/d IFLAB 5, IFLAB 6 s/d IFLAB 7) sesuai jadwal yang ditentukan.
3. Praktikan wajib membawa modul praktikum, kartu praktikum, dan alat tulis.
4. Praktikan wajib mengisi daftar hadir *rooster* dan BAP praktikum dengan bolpoin bertinta hitam.
5. Durasi kegiatan praktikum S-1 = 2 jam (100 menit).
6. Jumlah pertemuan praktikum 14 kali di lab (dimulai dari minggu pertama perkuliahan).
7. Presensi praktikum termasuk presensi matakuliah.
8. Praktikan yang datang terlambat tidak mendapat tambahan waktu.
9. Saat praktikum berlangsung, asisten praktikum dan praktikan:
 - Wajib menggunakan seragam sesuai aturan institusi.
 - Wajib mematikan/ mengkondisikan semua alat komunikasi.
 - Dilarang membuka aplikasi yang tidak berhubungan dengan praktikum yang berlangsung.
 - Dilarang mengubah pengaturan *software* maupun *hardware* komputer tanpa ijin.
 - Dilarang membawa makanan maupun minuman di ruang praktikum.
 - Dilarang memberikan jawaban ke praktikan lain.
 - Dilarang menyebarkan soal praktikum.
 - Dilarang membuang sampah di ruangan praktikum.
 - Wajib meletakkan alas kaki dengan rapi pada tempat yang telah disediakan.
10. Setiap praktikan dapat mengikuti praktikum susulan maksimal dua modul untuk satu mata kuliah praktikum.
 - Praktikan yang dapat mengikuti praktikum susulan hanyalah praktikan yang memenuhi syarat sesuai ketentuan institusi, yaitu: sakit (dibuktikan dengan surat keterangan medis), tugas dari institusi (dibuktikan dengan surat dinas atau dispensasi dari institusi), atau mendapat musibah atau keduakaan (menunjukkan surat keterangan dari orangtua/wali mahasiswa.)
 - Persyaratan untuk praktikum susulan diserahkan sesegera mungkin kepada asisten laboratorium untuk keperluan administrasi.
 - Praktikan yang diijinkan menjadi peserta praktikum susulan ditetapkan oleh Asman Lab dan Bengkel Informatika dan tidak dapat diganggu gugat.
11. Pelanggaran terhadap peraturan praktikum akan ditindak secara tegas secara berjenjang di lingkup Kelas, Laboratorium, Fakultas, hingga Universitas.
12. Website IFLab : <http://informatics.labs.telkomuniversity.ac.id>,
Kaur IFLAB (Pak Aulia): +6285755219604, aulwardan@telkomuniversity.ac.id
Laboran IFLAB (Oku Dewi): 085294057905, laboratorium.informatika@gmail.com
Fanspage IFLAB: https://www.instagram.com/informaticslab_telu/

Tata cara Komplain Praktikum IFLab Melalui IGracias

1. Login IGracias
2. Pilih Menu **Masukan** dan **Komplain**, pilih **Input Tiket**



3. Pilih Fakultas/Bagian: **Bidang Akademik (FIF)**
4. Pilih Program Studi/Urusan: **Urusan Laboratorium/Bengkel/Studio (FIF)**
5. Pilih Layanan: **Praktikum**
6. Pilih Kategori: **Pelaksanaan Praktikum**, lalu pilih **Sub Kategori**.
7. Isi **Deskripsi** sesuai komplain yang ingin disampaikan.

A screenshot of a web form titled 'Input Keluhan'. The form contains several dropdown menus and radio buttons. The fields are: 'Fakultas / Bagian' (set to 'BIDANG AKADEMIK (FIF)'), 'Program Studi / Urusan' (set to 'URUSAN LABORATORIUM/BENGKEL/STUDIO (FIF)'), 'Pelapor' (text input with 'RIZQILLAH ZAHRA LESTARI'), 'Layanan' (set to 'PRAKTIKUM'), 'Kategori' (set to 'Pelaksanaan Praktikum'), 'Sub Kategori' (set to 'Please Select...'), and 'Tipe Masukan' (with radio buttons for 'Komplain' and 'Masukan', where 'Komplain' is selected). Below these fields is a rich text editor with a toolbar containing icons for bold, italic, underline, link, unlink, list, quote, and other formatting options. The text area of the editor is empty.

Lampirkan *file* jika perlu. Lalu klik Kirim.

Daftar Isi

LEMBAR PENGESAHAN	i
Peraturan Praktikum Laboratorium Informatika 2020/2021	ii
Tata cara Komplain Praktikum IFLab Melalui IGracias.....	iii
Daftar Isi.....	iv
Daftar Gambar	viii
Daftar Tabel	x
Modul 1 Pengenalan Sistem Operasi.....	1
1.1 Definisi Sistem Operasi	1
1.2 Sistem Operasi Windows	1
1.2.1 Sejarah Windows	1
1.3 Sistem Operasi Linux.....	6
1.3.1 Sejarah Linux	6
1.3.2 Pengenalan Ubuntu	6
1.3.3 Instalasi Ubuntu	7
Modul 2 Perintah Dasar Linux	10
2.1 Perintah–Perintah Dasar Linux	10
2.1.1 ls (<i>List Directory</i>).....	11
2.1.2 cd (<i>Change Directory</i>).....	11
2.1.3 cp (<i>Copy</i>)	11
2.1.4 rm (<i>Remove</i>).....	12
2.1.5 mv (<i>Move</i>).....	12
2.1.6 mkdir (<i>Make Directory</i>).....	12
2.1.7 pwd (<i>Print Working Directory</i>)	13
2.1.8 man (<i>Manual</i>).....	13
2.1.9 (<i>Pipeline</i>).....	13
2.1.10 Redirection	13
2.2 GCC.....	14
2.2.1 Instalasi GCC.....	14
2.2.2 Meng-compile Program Menggunakan GCC.....	15
Modul 3 Proses.....	16
3.1 Manajemen Proses pada Linux	16
3.1.1 Pembuatan Proses	16
3.1.2 PID dan PPID.....	16
3.1.3 Background Process dan Foreground Process	17
3.1.4 Kill Process	17

3.15	Mengirim Sinyal ke Proses	18
3.16	Mengubah Prioritas Proses	19
3.2	Syscall process.....	20
3.2.1	Syscall dengan Fork()	20
3.2.2	Syscall dengan Exec()	22
3.3	Vi IMproved (Vim).....	23
3.3.1	Insert Mode.....	24
3.3.2	Visual Mode	24
3.3.3	Command Mode	24
Modul 4	Thread dan Mutex.....	25
4.1	Thread	25
4.1.1	Manajemen Thread.....	25
4.1.2	Perilaku Random Thread.....	29
4.2	Mutex.....	29
4.2.1	Membuat Mutex	30
4.2.2	Implementasi Mutex dalam Program	30
Modul 5	Deadlock.....	33
5.1	Deadlock.....	33
5.2	Deadlock Avoidance: Algoritma Banker.....	34
Modul 6	Memori.....	37
6.1	Memori	37
6.2	Performansi Memori.....	39
6.2.1	Memory Management Techniques.....	39
6.2.2	Page Replacement	43
Modul 7	Penjadwalan.....	48
7.1	Manajemen Penjadwalan	48
7.1.1	Crontab	48
7.1.2	At.....	49
7.2	Algoritma dan Performansi Penjadwalan	50
7.2.1	Algoritma FCFS	50
7.2.2	Algoritma RR	51
7.2.3	Algoritma Priority.....	52
Modul 8	I/O dan File System	54
8.1	Manajemen I/O dan File System.....	54
8.1.1	File Hierarchy Standard Linux	54
8.1.2	Commands untuk Manajemen I/O dan File System	55
8.2	Performansi I/O dan <i>File System</i>	58

8.2.1	Program Penjadwalan Disk dengan FCFS	58
8.2.2	Program Penjadwalan Disk dengan SCAN.....	59
8.2.3	Program Penjadwalan Disk dengan C-SCAN	60
Modul 9	Keamanan	62
9.1	Access Control (Permission).....	62
9.1.1	Permission Dasar pada Linux	62
9.1.2	Cara Mengubah <i>Permission</i> pada Linux.....	63
9.2	Integrity (SHA-256).....	66
9.2.1	Dasar Integritas dan Hash	66
9.2.2	Memeriksa Integritas dengan Tool	67
9.3	Confidentiality (Enkripsi).....	68
9.3.1	Enkripsi <i>Disk/File System</i> (ENCFS)	68
9.4	GPG	70
9.4.1	Install GPG.....	70
9.4.2	Import Public Key Orang Lain.....	71
9.4.3	Membuat <i>Public Key</i> Kita Tersedia.....	71
9.4.4	Enkripsi dengan PGP	72
9.4.5	Dekripsi dengan PGP.....	72
Modul 10	Scripting	73
10.1	Shell.....	73
10.1.1	Bash.....	73
10.1.2	Struktur Bash.....	73
10.2	Scripting Dasar	74
10.2.1	Variabel	74
10.2.1	Command Line Argument	74
10.2.2	Input.....	75
10.2.3	Aritmatik	75
10.2.4	If statement.....	76
10.2.5	Loop.....	77
10.2.6	Function	78
Modul 11	Instalasi Xinu	79
11.1	Instalasi Xinu	79
11.1.1	Persiapan.....	79
11.1.2	Development-system	79
11.1.3	Backend.....	81
11.1.4	Hasil Akhir	83
Modul 12	Menjalankan Xinu	85

12.1	Menjalankan Xinu	85
Modul 13	Mengenal Xinu	88
13.1	Paradigma Embedded	88
13.2	Organisasi Source Code Xinu	88
13.3	Implementasi Xinu	89
13.4	Tipe Data	89
13.5	Memahami source code	90
Modul 14	Proses dan Eksekusi Program	92
14.1	Proses.....	92
14.2	Eksekusi Program	93
Daftar Pustaka.....		96



Fakultas Informatika
School of Computing
Telkom University



Daftar Gambar

Gambar 1-1 Keluarga Windows	4
Gambar 1-2 Booting.....	5
Gambar 1-3 Windows Setup dan Aktivasi 1.....	5
Gambar 1-4 Tipe Instalasi dan Pemilihan Storage	5
Gambar 1-5 Proses Instalasi.....	5
Gambar 1-6 Pohon Keluarga Linux	6
Gambar 1-7 Beberapa Linux Distro.....	7
Gambar 1-8 Pilihan Bahasa	7
Gambar 1-9 Pilihan Instalasi	8
Gambar 1-10 Tipe Instalasi dan Pemilihan Partisi	8
Gambar 1-11 Pilih Kota dan Pengisian Data	9
Gambar 1-12 Proses Instalasi.....	9
Gambar 1-13 Proses Instalasi Selesai.....	9
Gambar 3-1 Hubungan PID dengan PPID.....	16
Gambar 3-2 List Semua Sinyal Kill pada Linux.....	18
Gambar 3-3 Daftar Proses yang Berjalan pada Linux	19
Gambar 3-4 Ilustrasi saat fork() dieksekusi di parent dan child	20
Gambar 3-5 Ilustrasi fork() pada program 3_1.c	21
Gambar 3-6 Hasil proses 3_2.c	23
Gambar 4-1 Proses (kiri) dan Thread (kanan).....	25
Gambar 4-2 Thread membuat Thread yang lain (tidak ada hirarki dan dependensi antar Thread ketika dibuat).....	26
Gambar 4-3 Joining Thread.....	28
Gambar 4-4 Program Non-deterministik	29
Gambar 4-5 Ilustrasi Mutex	30
Gambar 4-6 Hasil Running Program Thread	32
Gambar 5-1 Ilustrasi Deadlock.....	33
Gambar 5-2 Circular Wait	33
Gambar 5-3 Input untuk Algoritma Banker	36
Gambar 5-4 Output untuk Algoritma Banker	36
Gambar 6-1 Free Memory dalam MB	37
Gambar 6-2 Total Memory dalam MB	37
Gambar 6-3 Free RAM	37
Gambar 6-4 Informasi Kernel dan Sistem.....	38
Gambar 6-5 Statistik Memori Virtual.....	38
Gambar 6-6 Input Algoritma Worst-Fit.....	41
Gambar 6-7 Output Algoritma Worst-Fit.....	41
Gambar 6-8 Input dan Output Algoritma Best-Fit	42
Gambar 6-9 Input dan Output Algoritma First-Fit	43
Gambar 6-10 Input dan Output Algoritma FIFO	44
Gambar 6-11 Input dan Output Algoritma LRU	46
Gambar 6-12 Input dan Output Algoritma LFU	47
Gambar 7-1 Input dan Output Algoritma FCFS.....	51
Gambar 7-2 Input dan Output Algoritma Round Robin.....	52
Gambar 7-3 Input dan Output Algoritma Priority.....	53
Gambar 8-1 Input dan Output Algoritma FCFS.....	59
Gambar 8-2 Input dan Output Algoritma SCAN.....	60
Gambar 8-3 Input dan Output Algoritma C-SCAN	61

Gambar 9-1 Daftar File.....	63
Gambar 9-2 Mode.....	63
Gambar 9-3 GtHash Didukung Algoritma Checksum	67
Gambar 9-4 Menghasilkan SHA256 Checksum untuk UbuntuMATE iso	68
Gambar 9-5 Public key	71
Gambar 10-1 Contoh Sesi Bash.....	73
Gambar 10-2 myscript.sh.....	73
Gambar 11-1 Development System Network Setting pada Adapter 1	79
Gambar 11-2 Development System Network Setting pada Adapter 2	80
Gambar 11-3 Development System Serial Setting pada Port 1	80
Gambar 11-4 Development System Display Setting	81
Gambar 11-5 Backend Network Setting pada Adapter 1.....	82
Gambar 11-6 Backend Network Setting pada Adapter 2.....	82
Gambar 11-7 Backend Serial Setting pada Port 1	83
Gambar 11-8 Hasil Akhir Development System.....	83
Gambar 11-9 Hasil Akhir Backend.....	84
Gambar 11-10 Arsitektur Xinu	84
Gambar 12-1 Login Xinu.....	85
Gambar 12-3 Pindah ke DDirectory Xinu	85
Gambar 12-4 Arsitektur Xine dengan Dua VM	86
Gambar 12-5 Hasil Running Xinu	87
Gambar 13-1 Tipe Data pada Xinu	90
Gambar 13-2 Tampilan Sourcetrail	90
Gambar 14-1 Struktur Procent.....	92
Gambar 14-2 Contoh Program Sekuensial pada Xinu.....	93
Gambar 14-3 Contoh Konkuren pada Xinu.....	94
Gambar 14-4 Deklarasi Syscall Create()	95
Gambar 14-5 Contoh Lain Konkurensi pada Xinu.....	95

Daftar Tabel

Tabel 1-1 Timeline Sistem Operasi Windows	1
Tabel 1-1-2 Spesifikasi Minimum dan Rekomendasi Windows 10	4
Tabel 1-3 Spesifikasi Minimum dan Rekomendasi Ubuntu	7
Tabel 2-1 Penjelasan Struktur Perintah Linux.....	10
Tabel 2-2 Sebagian Daftar Opsi Perintah ls.....	11
Tabel 2-3 Sebagian Daftar Opsi Perintah cp	11
Tabel 2-4 Sebagian Daftar Opsi Perintah rm.....	12
Tabel 2-5 Sebagian Daftar Opsi Perintah mv	12
Tabel 2-6 Sebagian Daftar Opsi Perintah mkdir	13
Tabel 3-1 Beberapa Daftar Sinyal untuk Perintah Kill	17
Tabel 8-1 Hirarki Standar pada Linux	54



Fakultas Informatika
School of Computing
Telkom University



informatics lab

Modul 1 Pengenalan Sistem Operasi

Tujuan Praktikum

1. Praktikan mengenal sistem operasi Windows dan Ubuntu.
2. Praktikan dapat melakukan instalasi Windows 10 dan Ubuntu 18.04.

1.1 Definisi Sistem Operasi

Sistem operasi adalah komponen perangkat lunak dari sebuah sistem komputer yang bertanggung jawab untuk mengatur dan mengkoordinasikan aktivitas-aktivitas dan pembagian *resource* komputer. Sistem operasi bertindak sebagai *host* dari program aplikasi yang berjalan di mesin. Sistem operasi menawarkan berbagai *service* bagi program aplikasi dan pengguna. Aplikasi mengakses *service* ini melalui *application programming interfaces* (APIs) atau *system calls*. Dengan menggunakan *interface* ini, aplikasi dapat meminta *service* dari sistem operasi, melewati parameter, dan menerima hasil dari suatu operasi.

1.2 Sistem Operasi Windows

1.2.1. Sejarah Windows

Microsoft Windows atau lebih dikenal dengan sebutan Windows adalah keluarga sistem operasi komputer pribadi yang dikembangkan oleh Microsoft yang menggunakan antarmuka dengan pengguna berbasis grafik (*graphical user interface*).

Sistem operasi Windows telah berevolusi dari MS-DOS, sebuah sistem operasi yang berbasis modus teks dan *command-line*. Windows versi pertama, Windows Graphic Environment 1.0 pertama kali diperkenalkan pada 10 November 1983, tetapi baru keluar pasar pada bulan November tahun 1985 yang dibuat untuk memenuhi kebutuhan komputer dengan tampilan bergambar. Windows 1.0 merupakan perangkat lunak 16-bit tambahan (bukan merupakan sistem operasi) yang berjalan di atas MS-DOS (dan beberapa varian dari MS-DOS), sehingga ia tidak akan dapat berjalan tanpa adanya sistem operasi DOS. Versi 2.x, versi 3.x juga sama. Beberapa versi terakhir dari Windows (dimulai dari versi 4.0 dan Windows NT 3.1) merupakan sistem operasi mandiri yang tidak lagi bergantung kepada sistem operasi MS-DOS. Microsoft Windows kemudian bisa berkembang dan dapat menguasai penggunaan sistem operasi hingga mencapai 90%. Versi *client* terbaru dari Windows adalah Windows 10, sedangkan versi *server* terbaru dari Windows adalah Windows Server 2008.

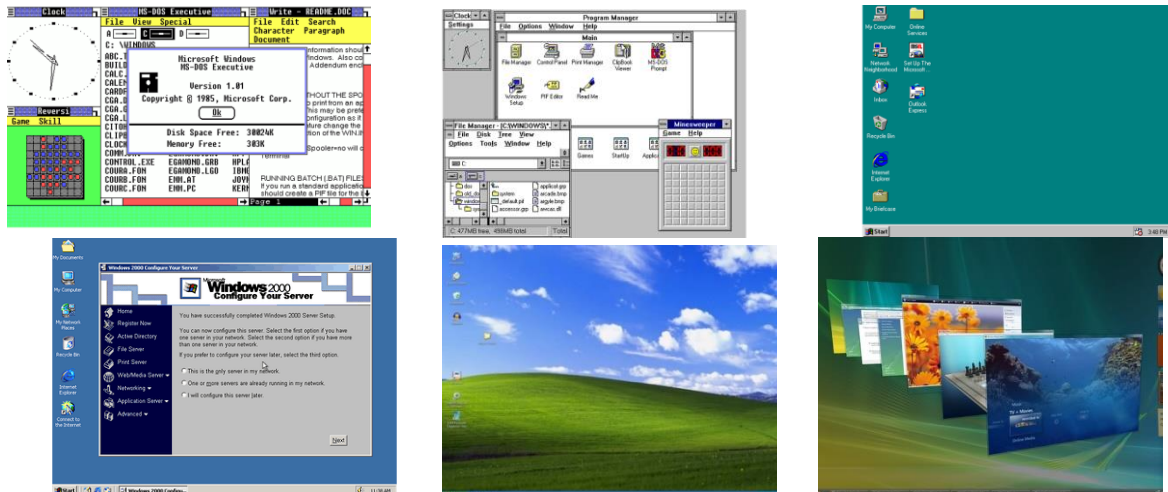
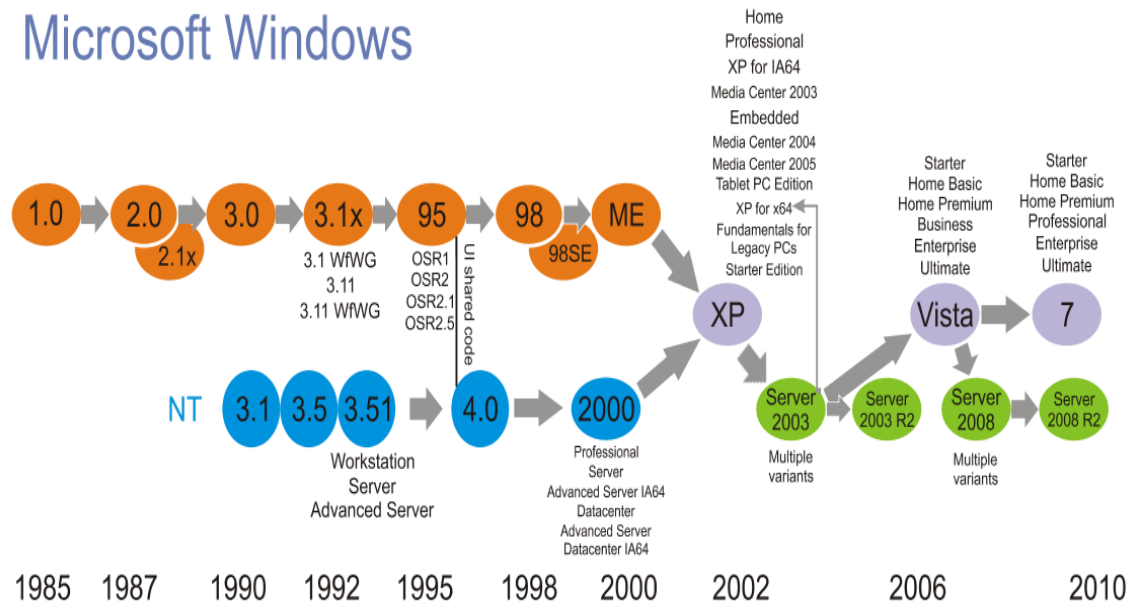
Tabel 1-1 Timeline Sistem Operasi Windows

Waktu rilis	Nama Produk	Versi	Dukungan	Kelebihan
November 1985	Windows 1.01	1.01	Tidak didukung lagi	memperluas kemampuan MS-DOS dengan tambahan antarmuka grafis
November 1987	Windows 2.03	2.03	Tidak didukung lagi	menggunakan <i>model</i> memori modus <i>real</i> , yang hanya mampu mengakses memori hingga 1 <i>megabit</i> saja
Maret 1989	Windows 2.11	2.11	Tidak didukung lagi	memiliki modus penampilan jendela secara <i>cascade</i> (bertumpuk)
Mei 1990	Windows 3.0	3.0	Tidak didukung lagi	diperkenalkan memori <i>virtual</i>
Maret 1992	Windows 3.1x	3.1	Tidak didukung lagi	kemampuan untuk menampilkan <i>font TrueType</i>

				Fonts dan perbaikan terhadap bug dan dukungan terhadap multimedia.
Oktober 1992	Windows For Workgroup 3.1	3.1	Tidak didukung lagi	mencakup driver jaringan komputer dan stack protokol yang lebih baik, dan juga mendukung jaringan secara <i>peer-to-peer</i>
Juli 1993	Windows NT 3.1	3.1	Tidak didukung lagi	Windows pertama yang dibuat dengan menggunakan kernel hibrida, setelah pada versi-versi sebelumnya hanya menggunakan kernel <i>monolithic</i> saja.
Desember 1993	Windows For Workgroups 3.11	3.11	Tidak didukung lagi	hanya berjalan di dalam modus 386 <i>Enhanced</i> , dan membutuhkan setidaknya mesin dengan prosesor Intel 80386SX.
Januari 1994	Windows 3.2 (Chinese Only)	3.2	Tidak didukung lagi	Mengembangkan subsistem WinFS yang merupakan implementasi dari <i>Object File System</i>
September 1994	Windows NT 3.5	NT 3.5	Tidak didukung lagi	Membuat <i>Application Programming Interface</i> (API) 32-bit yang baru untuk menggantikan Windows API 16-bit yang sudah lama
Mei 1995	Windows NT 3.51	NT 3.51	Tidak didukung lagi	menawarkan beberapa pilihan konektivitas jaringan yang luas dan juga tentunya sistem berkas NTFS yang efisien
Agustus 1995	Windows 95	4.0.950	Tidak didukung lagi	memiliki dukungan terhadap <i>multitasking</i> secara <i>pre-emptive</i> 32-bit
Juli 1996	Windows NT 4.0	NT 4.0.1381	Tidak didukung lagi	dikembangkan sebagai sebuah bagian dari usaha untuk memperkenalkan Windows NT kepada pasar <i>workstation</i>
Juni 1998	Windows 98	4.10.1998	Tidak didukung lagi	mencakup banyak driver perangkat keras baru dan dukungan sistem berkas FAT32
Mei 1999	Windows 98 SE	4.10.2222	Tidak didukung lagi	Memperkenalkan <i>Internet Connection Sharing</i>
Februari 2000	Windows 2000	NT 5.0.2195	Hingga 13 Juli 2010	<i>Device Manager</i> yang telah ditingkatkan (dengan menggunakan Microsoft <i>Management Console</i>)
September 2000	Windows Me	4.90.3000	Tidak didukung lagi	memperkenalkan Windows Movie Maker versi pertama dan juga memasukkan fitur <i>System Restore</i>

Oktober 2001	Windows XP	NT 5.1.2600	Untuk SP2 & SP3	menggantikan produk Windows 9x yang berbasis 16/32-bit yang sudah menua
Maret 2003	Windows XP 64-bit Edition	NT 5.2.3790	Tidak didukung lagi	untuk sistem-sistem rumahan dan <i>workstation</i> yang menggunakan prosesor 64-bit
April 2003	Windows Server 2003	NT 5.2.3790	Untuk SP1, R2, SP2	fitur-fitur manajemen untuk kantor-kantor cabang, dan integrasi identitas yang luas
April 2005	Windows XP Professional x64 Edition	NT 5.2.3790	Sekarang	dibuat berbasiskan basis kode Windows NT 5.2 (sama seperti Windows Server 2003)
Juli 2006	Windows Fundamentals for Legacy PCs	NT 5.1.2600	Sekarang	memberikan pilihan <i>upgrade</i> kepada para pelanggannya yang masih menggunakan Windows 95, Windows 98, Windows Me, dan Windows NT <i>Workstation</i>
November 2006	Windows Vista	NT 6.0.6000	Sekarang (SP1)	memperkenalkan sebuah modus pengguna yang terbatas, yang disebut sebagai <i>User Account Control</i> (UAC)
Juli 2007	Windows Home Server	NT 5.2.4500	Sekarang	memiliki fitur <i>Media Sharing</i> , <i>backup</i> terhadap <i>drive</i> lokal dan <i>drive</i> jarak jauh, dan duplikasi berkas
Februari 2008	Windows Server 2008	NT 6.0.6001	Sekarang	lebih modular secara signifikan, ketimbang pendahulunya, Windows Server 2003
Oktober 2009	Windows 7	NT 6.1.7600	Sekarang	memiliki keamanan dan fitur yang baru, diantaranya adalah: <i>Jump List</i>
Oktober 2012	Windows 8	NT 6.2.9200	Sekarang	Keamanan yang lebih baik, <i>startup</i> yang lebih cepat, masukan sentuhan.
Juli 2015	Windows 10		Sekarang	Adanya fitur asisten personal, fitur Windows Hello, fitur <i>virtual desktop</i> , memiliki <i>browser</i> bawaan, dan <i>upgrade</i> secara gratis

Microsoft Windows



Gambar 1-1 Keluarga Windows

Atas: Pohon Keluarga Windows

Bawah: Win 1, Win 3.1, Win 95, Windows 2000, XP dan Vista

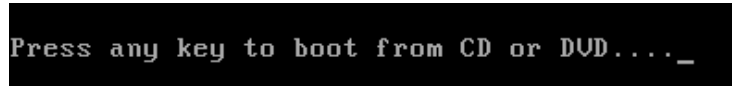
Sebelum melangkah ke proses instalasi, sebaiknya memperhatikan kebutuhan sistem operasi atau aplikasi. Berikut adalah kebutuhan minimal dan rekomendasi kebutuhan dari Windows 10.

Tabel 1-1-2 Spesifikasi Minimum dan Rekomendasi Windows 10

Spesifikasi	Minimum	Rekomendasi
Kecepatan prosesor (GHz)	1 GB	1,5 GB atau lebih
RAM (GB)	Edisi IA-32: 1 GB Edisi x86-64: 2 GB	4 GB
Ruang kosong <i>harddisk</i> (GB)	Edisi IA-32: 16 GB Edisi x86-64: 20 GB	20 GB atau lebih
Resolusi tampilan	800 x 600 <i>pixels</i>	800 x 600 <i>pixels</i> atau lebih
Grafik	DirectX 9 WDDM 1.0	DirectX 9 atau lebih WDDM 1.3 atau lebih

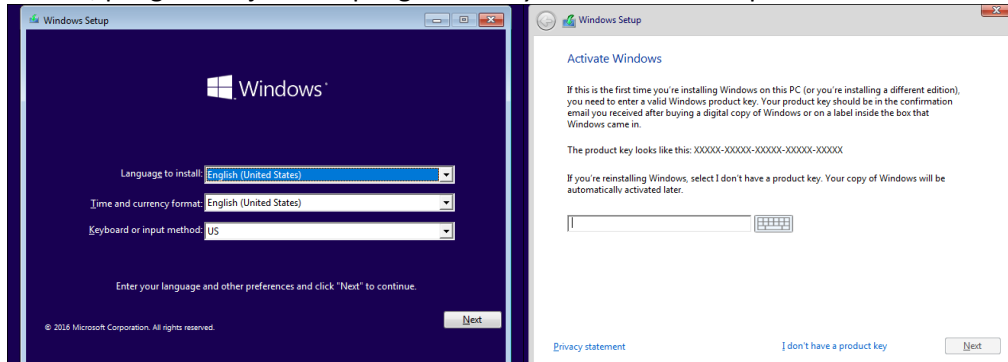
Berikut adalah langkah-langkah instalasi Windows 10:

1. Masukkan CD Windows 10 ke dalam komputer dan lakukan *restart*.
2. Jika muncul peringatan untuk memulai dari CD, tekan sembarang tombol. Jika peringatan terlewat (hanya muncul selama beberapa detik), *restart* komputer untuk mencoba lagi.



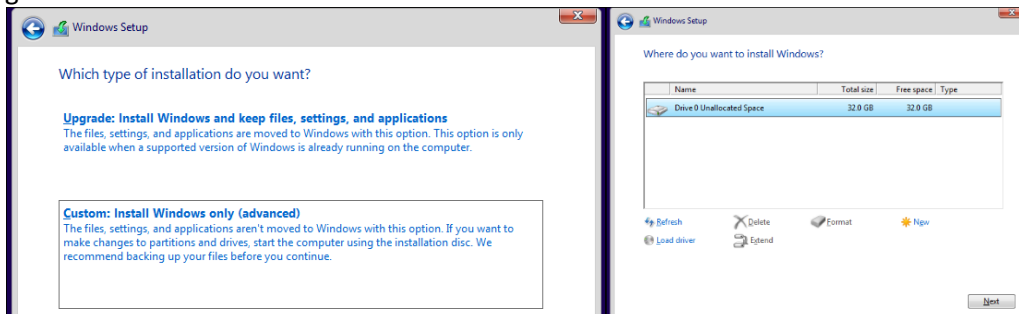
Gambar 1-2 Booting

3. Pilih Bahasa, pengaturan jam dan pengaturan *keyboard*. Kemudian pilih 'Next'



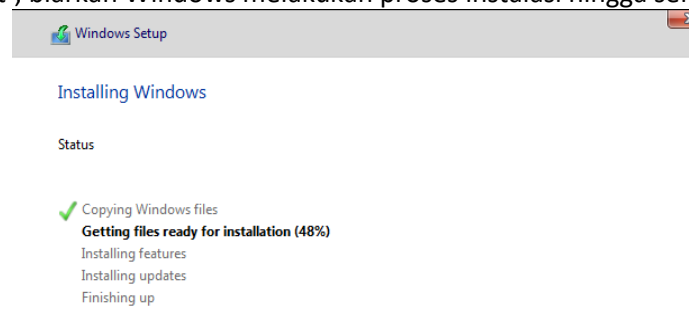
Gambar 1-3 Windows Setup dan Aktivasi 1

4. Ketika Anda telah mencapai layar instalasi, pilih 'Install Now'.
5. Pada layar 'Activate Windows', Anda akan diminta memasukkan kunci aktivasi atau melewatinya. Kemudian pilih 'Next'.
6. Kemudian pada layar selanjutnya, Anda akan diminta untuk memilih *type installation* sesuai keinginan Anda.



Gambar 1-4 Tipe Instalasi dan Pemilihan Storage

7. Pada layar selanjutnya, pilih *hard disk* yang ingin digunakan *install* Windows.
8. Setelah pilih 'Next', biarkan Windows melakukan proses instalasi hingga selesai.

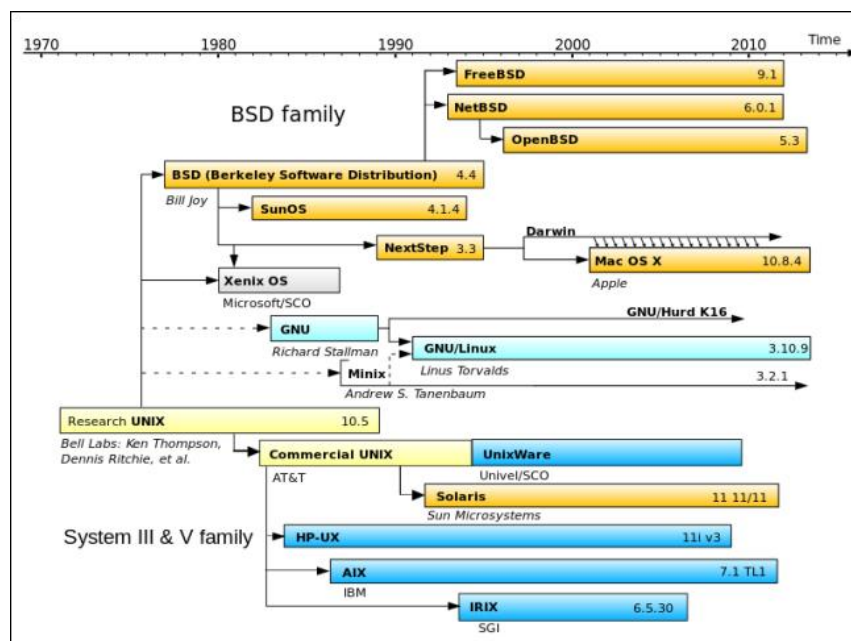


Gambar 1-5 Proses Instalasi

1.3 Sistem Operasi Linux

1.3.1 Sejarah Linux

Kernel Linux pada mulanya berasal dari proyek hobi mahasiswa Finlandia, bernama Linus Torvalds. Bermula dari ketidakpuasan Linus terhadap *kernel* Minix yang dikembangkan oleh seorang professor dari Belanda, Andrew S. Tanenbaum, yang dipakai di kampusnya, melahirkan keinginan untuk mendalami *kernel* tersebut sehingga lahirlah *kernel* Linux. Minix merupakan project Mini UNIX yang dipakai untuk kepentingan pendidikan dan non-komersial. Versi 0.01 yang merupakan versi pertama Linux dikeluarkan ke internet pada september 1991 sementara versi 0.02 nya yang sudah memiliki *bash* dan *gcc compiler* dirilis pada 5 oktober 1991.

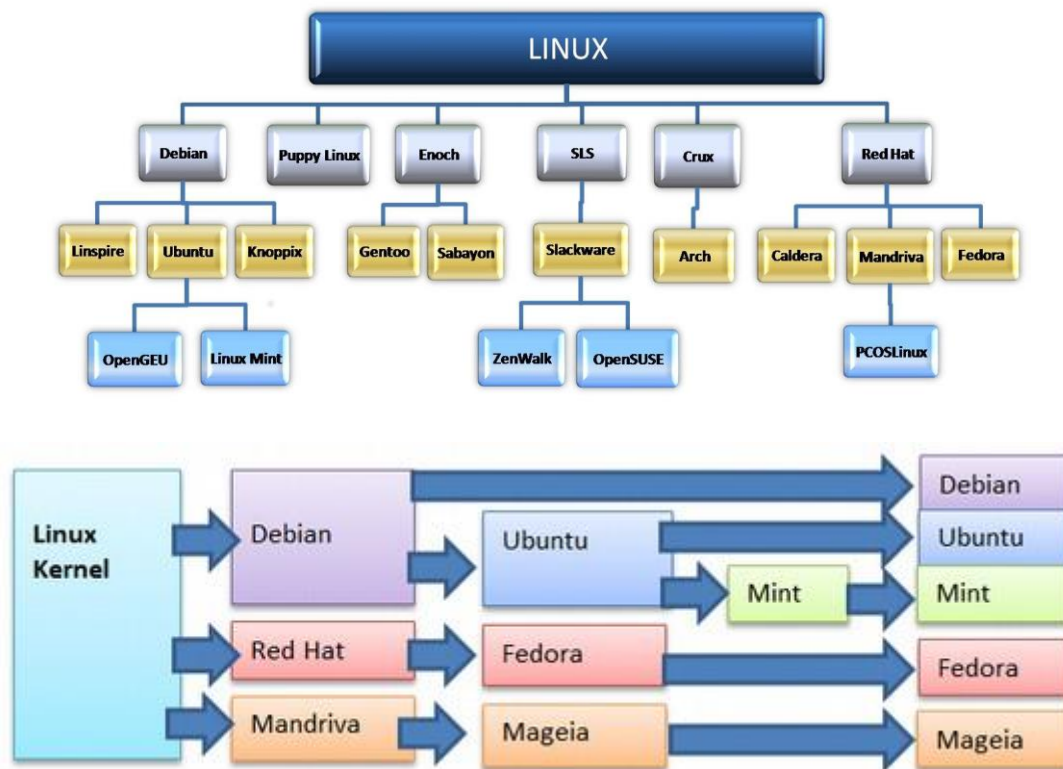


Gambar 1-6 Pohon Keluarga Linux

1.3.2 Pengenalan Ubuntu

Ubuntu adalah salah satu distro Linux yang tersedia secara gratis dengan dukungan komunitas yang profesional. Ubuntu berasal kata Afrika kuno yang berarti 'kemanusiaan untuk orang lain'. Hal ini sering digambarkan sebagai mengingatkan kita bahwa 'saya adalah saya karena kita semua'. Distribusi Ubuntu mewakili yang terbaik dari apa yang telah dibagikan oleh komunitas perangkat lunak dunia kepada dunia.

Linux sudah didirikan pada tahun 2004, tetapi itu terpecah menjadi edisi komunitas eksklusif dan tidak didukung, dan perangkat lunak bebas bukanlah bagian dari kehidupan sehari-hari bagi sebagian besar pengguna komputer. Saat itulah Mark Shuttleworth mengumpulkan tim kecil pengembang Debian yang bersama-sama mendirikan Canonical dan berangkat untuk membuat *desktop* Linux yang mudah digunakan yang disebut Ubuntu. Misi untuk Ubuntu bersifat sosial dan ekonomi. Pertama, Ubuntu memberikan perangkat lunak bebas di dunia, bebas kepada semua orang dengan persyaratan yang sama. Baik Anda seorang pelajar di India atau bank global, Anda dapat mengunduh dan menggunakan Ubuntu secara gratis. Kedua, Ubuntu bertujuan untuk memotong biaya layanan profesional - dukungan, manajemen, pemeliharaan, operasi - untuk orang-orang yang menggunakan Ubuntu dalam skala besar, melalui *portfolio* layanan yang disediakan oleh Canonical yang pada akhirnya mendanai perbaikan *platform*.



Gambar 1-7 Beberapa Linux Distro

1.3.3 Instalasi Ubuntu

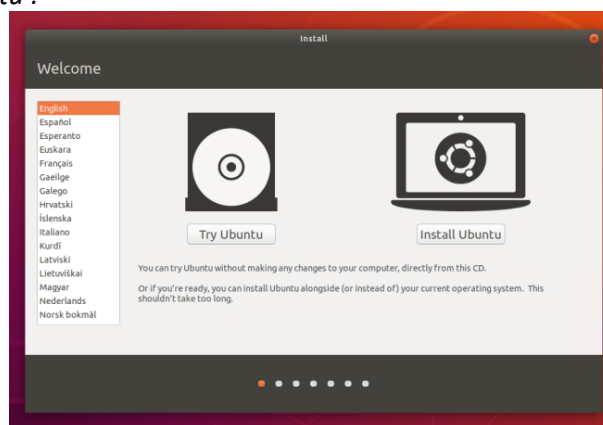
Sebelum melangkah ke proses instalasi, sebaiknya memperhatikan kebutuhan sistem operasi atau aplikasi. Berikut adalah kebutuhan minimal dan rekomendasi kebutuhan dari Ubuntu.

Tabel 1-3 Spesifikasi Minimum dan Rekomendasi Ubuntu

Spesifikasi	Minimum	Rekomendasi
Kecepatan prosesor (GHz)	2 GB	2 atau lebih
RAM (GB)	2 GB	2 GB
Ruang kosong <i>harddisk</i> (GB)	25 GB	25 GB atau lebih

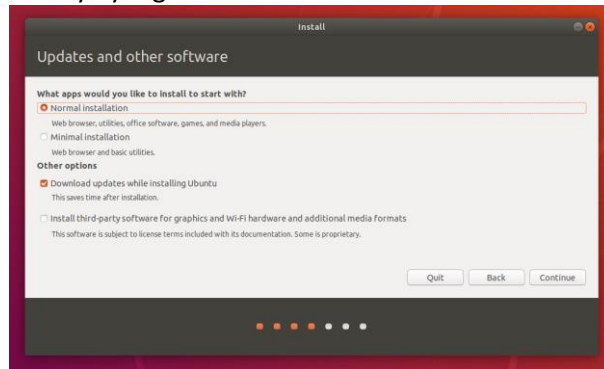
Berikut adalah langkah-langkah instalasi Ubuntu:

1. Masukkan DVD Ubuntu ke dalam DVD Drive Anda kemudian *restart* komputer Anda.
2. Setelah *restart*, akan muncul layar seperti dibawah ini. Pada layar ini, Anda dapat memilih bahasa yang ingin digunakan kemudian Anda dapat memilih jenis instalasi yang diperlukan. Pada tutorial ini, pilih '*Install Ubuntu*'.



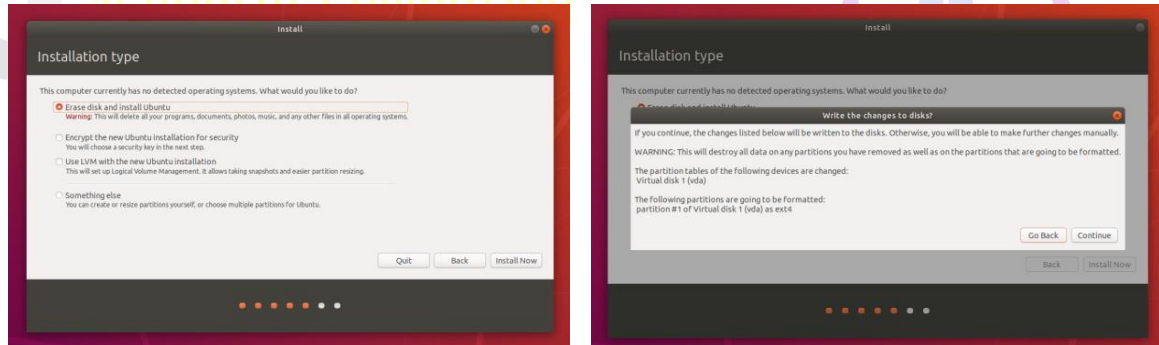
Gambar 1-8 Pilihan Bahasa

3. Jika Anda melakukan instalasi Ubuntu menggunakan *USB Drive*, lakukan hal yang sama seperti poin (2).
4. Pada layar selanjutnya, Anda akan diminta untuk memilih *keyboard layout*. Kemudian pilih 'Continue'.
5. Kemudian pada layar 'What apps would you like to install to start with', dua opsi tersebut adalah 'Instalasi normal' dan 'Instalasi minimal'. Yang pertama adalah setara dengan bundel bawaan lama dari utilitas, aplikasi, *game*, dan pemutar media - sebuah *launchpad* untuk setiap instalasi Linux. Yang kedua membutuhkan ruang penyimpanan yang jauh lebih sedikit dan memungkinkan Anda untuk menginstal hanya yang Anda butuhkan.



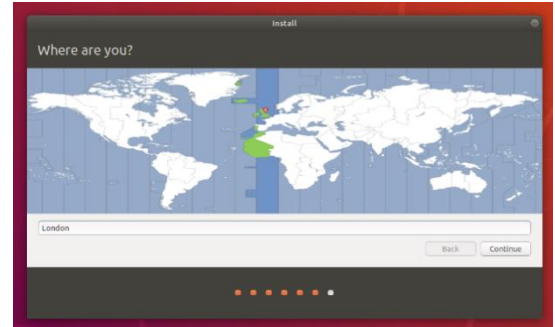
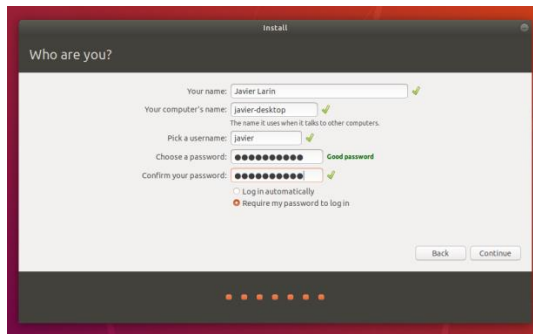
Gambar 1-9 Pilihan Instalasi

Di bawah pertanyaan tipe instalasi adalah dua kotak centang, satu untuk mengaktifkan pembaruan ketika menginstal dan yang lain untuk mengaktifkan perangkat lunak pihak ketiga. Alokasikan ruang *drive*. Gunakan kotak centang untuk memilih apakah Anda ingin menginstal Ubuntu di samping sistem operasi lain, hapus sistem operasi yang ada dan gantilah dengan Ubuntu, atau, jika Anda cukup mahir, pilih opsi 'Something else'.



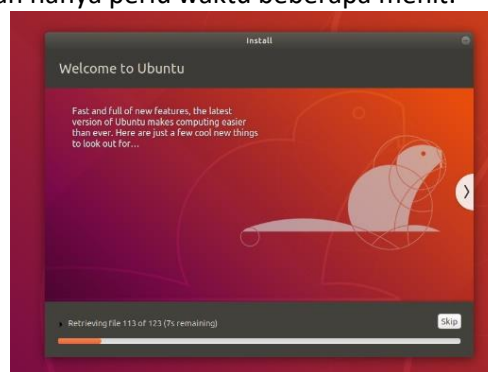
Gambar 1-10 Tipe Instalasi dan Pemilihan Partisi

6. Setelah mengonfigurasi penyimpanan, klik tombol 'Install Now'. Sebuah panel kecil akan muncul dengan gambaran tentang opsi penyimpanan yang Anda pilih, dengan kesempatan untuk kembali jika rinciannya salah. Klik 'Continue' untuk memperbaiki perubahan tersebut di tempat dan memulai proses instalasi.
7. Pada layar selanjutnya Anda akan diminta untuk memilih lokasi Anda. Jika Anda terhubung ke internet, lokasi Anda akan terdeteksi secara otomatis. Periksa lokasi Anda benar dan klik 'Forward' untuk melanjutkan. Jika Anda tidak yakin dengan zona waktu Anda, ketik nama kota atau kota setempat atau gunakan peta untuk memilih lokasi Anda.



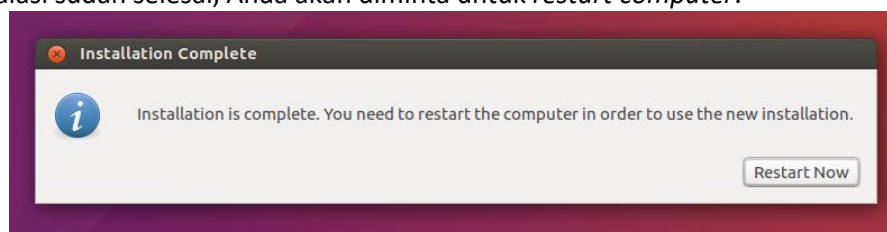
Gambar 1-11 Pilih Kota dan Pengisian Data

8. Kemudian isikan nama Anda, kata sandi, dan juga *username* Anda.
9. Penginstal akan selesai di latar belakang sementara jendela penginstalan mengajarkan Anda sedikit tentang betapa mengagumkannya Ubuntu. Tergantung pada kecepatan mesin dan koneksi jaringan Anda, penginstalan hanya perlu waktu beberapa menit.



Gambar 1-12 Proses Instalasi

10. Jika instalasi sudah selesai, Anda akan diminta untuk *restart computer*.



Gambar 1-13 Proses Instalasi Selesai

Modul 2 Perintah Dasar Linux

Tujuan Praktikum

1. Praktikan menguasai perintah-perintah dasar Linux.
2. Praktikan dapat meng-*compile* program dengan gcc.

2.1 Perintah–Perintah Dasar Linux

Pada umumnya *default* perintah dasar Linux dieksekusi pada *Bash* (*/bin/bash*). *Bash* mengeksekusi program dalam OS untuk setiap perintah yang dimasukkan dalam *console*. Berikut adalah beberapa petunjuk dalam menulis perintah:

- Tekan *enter*. Tidak ada yang terjadi hingga anda menekan *enter*.
- Gunakan *Tab* untuk *auto completion*. Biasakan gunakan *tab*.
- Ctrl+C untuk membatalkan perintah. Ctrl+C memberikan *attempt signal* ke program.
- Shift + *PageUP* untuk melihat hasil *text* sebelumnya.
- Panah atas/bawah untuk mengulang perintah.

Perintah–perintah di *console* Linux memiliki aturan–aturan penulisan, aturan struktur penulisan perintah Linux tersebut berlaku pada hampir semua perintah–perintah Linux. Penulisan perintah Linux bersifat *case sensitive* artinya adalah setiap penulisan huruf besar dan kecil sangat berpengaruh terhadap hasil yang diinginkan. Berikut struktur penulisan perintah Linux:

```
$ Perintah [-opsi] [argument_1] [argument_2]
```

Tabel 2-1 Penjelasan Struktur Perintah Linux

Nama bagian	Keterangan
\$	Merupakan tanda representasi <i>user mode</i> . Tanda representasi tersebut di bagi menjadi 2 yaitu \$ dan #, berikut adalah penjelasan dari tanda <i>user mode</i> tersebut: <ul style="list-style-type: none">• # (biasa disebut dengan <i>root</i>) adalah <i>user mode</i> yang memiliki hak akses tertinggi, <i>root</i> memiliki hak akses yang tak terbatas terhadap seluruh perintah Linux. Perintah – perintah yang dapat di jalan pada <i>mode root</i> terdapat di <i>folder /etc/sbin</i>.• \$ adalah tanda untuk <i>user mode</i> biasa yang hanya memiliki hak akses terbatas terhadap beberapa perintah – perintah Linux. Perintah – perintah yang dapat di jalan kan ketika <i>user</i> menggunakan <i>mode \$</i> disimpan di dalam <i>folder /etc/bin</i>.
Perintah	Perintah adalah program yang digunakan untuk berinteraksi dengan <i>system</i> operasi Linux.
[-opsi]	Opsi adalah fungsionalitas – fungsionalitas <i>default</i> yang dimiliki oleh perintah yang digunakan. Kita dapat mengkombinasikan beberapa opsi untuk mendapatkan hasil yang diinginkan. Fungsionalitas dari setiap perintah Linux dapat di baca dengan menggunakan perintah <div>\$ man perintah</div>
[argument_1]	<i>Argument_1</i> , biasanya adalah nama <i>file</i> , direktori atau alamat (<i>relative</i> atau <i>absolute</i>) dari suatu <i>file</i> ataupun <i>folder</i> .
[argument_2]	<i>Argument_2</i> , pada beberapa perintah Linux <i>argument_2</i> bersifat <i>optional</i> dengan kata lain tidak selalu harus ada, namun untuk beberapa perintah operasi <i>argument_2</i> biasanya berisi nama, direktori atau alamat (<i>relative</i> atau <i>absolute</i>) dari suatu <i>file</i> ataupun <i>folder</i>

	tujuan
--	--------

2.1.1 ls (*List Directory*)

\$ **ls** adalah perintah yang digunakan untuk melihat isi dari sebuah *directory* atau *file* di direktori aktif. Terdapat beberapa opsi fungsionalitas *default* yang dimiliki oleh perintah \$ **ls**, berikut adalah contoh penggunaan perintah \$ **ls**:

```
$ ls [opsi] [directory]
$ ls -al /home/praktikan
```

Berikut beberapa opsi yang dapat digunakan pada perintah \$ **ls** :

Tabel 2-2 Sebagian Daftar Opsi Perintah ls

Opsi	Keterangan
-l	menampilkan tampilan secara detail dari setiap <i>file</i> yang di tampilkan.
-a	menampilkan seluruh <i>file</i> yang terdapat di dalam <i>directory</i> termasuk <i>hidden file</i>
-s	menampilkan ukuran <i>file</i> (<i>in blocks, not bytes</i>)
-h	menampilkan kedalam bentuk " <i>human readable format</i> " (ie: 4K, 16M, 1G etc).

2.1.2 cd (*Change Directory*)

\$ **cd** adalah perintah yang digunakan untuk mengubah posisi dari posisi *directory* kerja aktif ke *directory* kerja yang akan kita gunakan. Contoh:

```
$ cd /usr/local
$ pwd /usr/local
```

Perintah diatas di gunakan untuk berpindah dari *directory* yang kita tempati sekarang menuju ke *directory* /usr/local

Selain itu perintah cd dapat digunakan untuk berpindah ke *parent directory* dari suatu *directory* aktif, yaitu dengan menambahkan (..) setelah perintah \$ **cd**.

Contoh:

```
$ pwd /usr/local
$ cd ..
$ pwd /usr
```

2.1.3 cp (*Copy*)

\$ **cp** adalah perintah untuk meng-*copy* satu atau banyak *file* dalam satu kali penulisan perintah atau duplikasi *file* atau *folder* dari sumber ke tujuan. Perintah yang di gunakan untuk meng-*copy file* atau *folder* adalah

```
$cp [opsi] /folder_asal/nama_file /folder_tujuan
```

Tabel 2-3 Sebagian Daftar Opsi Perintah cp

Opsi	Keterangan
-R	Digunakan untuk meng- <i>copy</i> suatu <i>directory</i> beserta dengan isinya
-v	Digunakan untuk melihat <i>file</i> yang di- <i>copy</i> -kan ke tujuan di <i>console</i>

Contoh:

```
$ cp -vR /home/student /data/backup
```


Keterangan: Perintah diatas akan meng-copy direktori *students* yang berada dibawah direktori */home* beserta seluruh isinya kedalam direktori */data/backup* dan menampilkan seluruh *file* yang di *copy* kan.

2.1.4 rm (Remove)

\$ **rm** adalah perintah yang digunakan untuk penghapusan (*delete*) satu atau banyak *directory* atau *file* melalui *console*. Untuk menghapus *file* digunakan perintah seperti dibawah ini:

```
$ rm [opsi] nama_file/folder
```

Beberapa opsi yang dapat digunakan dalam perintah **rm**:

Tabel 2-4 Sebagian Daftar Opsi Perintah rm

Opsi	Keterangan
-f	Memaksa <i>file</i> untuk di hapus, biasa di gunakan untuk menghapus <i>file system</i> .
-i	menampilkan peringatan sebelum proses penghapusan di lakukan
-r	Penghapusan secara berulang hingga akhir <i>directory</i> . HATI-HATI apabila menggunakan opsi ini!!!

Contoh:

```
$ rm -f UTS.java
```

Keterangan: Menghapus secara paksa *file* UTS.java

```
$ rm -rf /opt/test1
```

Keterangan: Menghapus seluruh *file* yang terdapat di dalam *folder test1*, dalam hal ini *test1* adalah *folder* yang di dalam nya terdapat beberapa *file*.

2.1.5 mv (Move)

\$ **mv** adalah perintah yang digunakan untuk melakukan perpindahan *file* atau direktori atau dapat juga digunakan untuk melakukan perubahan nama *file* atau direktori jika sumber dan tujuan yang diberikan terletak dalam satu struktur direktori yang sama. Ada beberapa opsi yang dimiliki perintah \$ **mv** ini, berikut adalah opsi yang dapat digunakan pada perintah \$ **mv**

Tabel 2-5 Sebagian Daftar Opsi Perintah mv

Opsi	Keterangan
-R	Digunakan untuk meng-copy suatu <i>directory</i> beserta dengan isinya
-v	Digunakan untuk melihat <i>file</i> yang di-copy-kan ke tujuan

Contoh:

```
$ mv contoh1.html /home/student/contoh2.html
```

Keterangan: Perintah di atas digunakan untuk mengganti nama *file contoh1.html* menjadi *contoh2.html* dan memindahkan *file* tersebut kedalam *folder student* yang berada di dalam *folder home*.

2.1.6 mkdir (Make Directory)

\$ **mkdir** adalah perintah yang digunakan untuk membuat satu atau beberapa *directory* melalui konsol. Sama hal nya dengan perintah Linux lainnya pada perintah \$ **mkdir** terdapat beberapa opsi yang dapat digunakan, berikut adalah daftar opsi yang dapat digunakan:

Tabel 2-6 Sebagian Daftar Opsi Perintah *mkdir*

Opsi	Keterangan
-p	Opsi untuk membuat <i>directory</i> secara bertingkat dengan hanya menggunakan satu buah <i>command</i> .
-m	Opsi untuk memberikan <i>permissions</i> dari <i>directory</i> yang kita buat, dengan menggunakan bilangan oktal (akan dijelaskan pada sub bab <i>file permissions</i>).

mkdir memiliki *syntax* penulisan:

```
$ mkdir [opsi] dir_name
```

dir_name adalah nama dari sebuah *directory* yang akan dibuat oleh *user*.

```
$ mkdir -p /tmp/{a/{b,c},opt/test1/test2,var/coba}
```

2.1.7 *pwd* (*Print Working Directory*)

\$ *pwd* adalah perintah yang digunakan untuk melihat tempat *directory* aktif atau alamat *directory* yang sedang kita tempati saat ini. Contoh:

```
$ pwd /home/praktikan
```

Hasil dari perintah di atas ditunjukan setelah perintah tersebut di jalankan, dan saat ini *directory* kerja aktif nya sedang berada di **/home/praktikan**.

2.1.8 *man* (*Manual*)

\$ *man* adalah perintah yang digunakan untuk melihat manual dari suatu perintah Contoh:

```
$ man ls
```

Perintah di atas untuk mengetahui fungsi dari perintah *ls* dan opsi-opsi yang ada dalam perintah *ls*.

2.1.9 *|* (*Pipeline*)

Pipeline adalah perintah di Linux yang memungkinkan Anda menggunakan dua atau lebih perintah sedemikian *output* dari satu perintah berfungsi sebagai masukan ke yang berikutnya. Singkatnya, *output* dari setiap proses langsung sebagai masukan ke yang berikutnya seperti pipa. Simbol '*|*' menandakan pipa. Pipa membantu Anda menggabungkan dua atau lebih perintah pada waktu yang sama dan menjalankan mereka berturut-turut. Contoh:

```
$ cat README.txt | grep rules
```

Terdapat 2 perintah yaitu "*cat*" dan "*grep*". Perintah "*cat*" berfungsi untuk menampilkan isi *file* ke konsol. Perintah "*grep*" berguna untuk mem-filter, *grep rules* berarti akan menampilkan baris dengan kata "*rules*" saja. Pipa (*|*) berguna untuk menggabungkan 2 perintah tersebut. *Output* dari perintah "*cat*" digunakan sebagai *input* dari perintah "*grep*".

2.1.10 *Redirection*

Fasilitas *redirection* memungkinkan kita untuk dapat menyimpan *output* dari sebuah proses untuk disimpan ke *file* lain (*Output Redirection*) atau sebaliknya menggunakan isi dari *file* sebagai *input* dalam suatu proses (*Input redirection*). Komponen-komponen dari *redirection* adalah *<*, *>*, *<<*, *>>*.

>: menyimpan hasil ke *file* (*overwrite*)

Contoh:

```
$ ls -al > result.txt
```

>>: menyimpan hasil ke *file* (*append*)

Contoh:

```
$ cat README >> result.txt
$ cat INSTALL.txt >> result.txt
```

<: *file* sebagai *input* proses

<<: *file* sebagai *input* proses

2.2 GCC

GCC dikembangkan oleh Richard Stallman, pendiri dari proyek GNU. Richard Stallman mendirikan proyek GNU pada tahun 1984 untuk menciptakan sistem operasi Unix-like lengkap sebagai perangkat lunak bebas (*open*), untuk mempromosikan kebebasan (*free*) dan kerjasama antara komputer pengguna dan pemrogram. GCC ("GNU C Compiler" atau "GNU Compiler Collection") telah berkembang seiring dengan waktu untuk mendukung banyak bahasa C (gcc), C++ (g ++), Java (gcj), Fortran (gfortran), Ada (Agas), Go (gccgo), OpenMP, Cilk Plus, dan OpenAcc.

GCC adalah komponen kunci dari apa yang disebut "GNU Toolchain", untuk mengembangkan aplikasi dan menulis sistem operasi. GCC *portable* dan berjalan di banyak *platform*. GCC (dan GNU Toolchain) saat ini tersedia pada semua Unix/Linux. GCC juga diporting ke Windows (oleh Cygwin, MinGW dan MinGW-W64). GCC adalah juga sebuah *cross-compiler* sehingga dapat memproduksi *executable* pada *platform* yang berbeda. Situs utama untuk GCC adalah <http://gcc.gnu.org/>. Versi saat ini adalah 7.3 GCC, dirilis pada tahun 25-01-2018.

2.2.1 Instalasi GCC

Instalasi pada Linux

Buka Terminal, dan masukkan "*gcc--version*". Jika gcc tidak diinstal, sistem akan meminta Anda untuk menginstal gcc.

```
$ gcc --version
```

Cara *install* gcc pada Linux:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install build-essential
```

atau

```
$ sudo apt-get gcc
```

Instalasi pada Windows

Untuk Windows, Anda juga dapat meng-*install* Cygwin GCC, MinGW GCC atau MinGW-W64 GCC.

- Cygwin GCC: Cygwin adalah lingkungan berbasis Unix dan antarmuka baris perintah untuk Microsoft Windows. Cygwin sangat besar dan mencakup sebagian besar Unix alat dan utilitas. Ini juga termasuk umum digunakan *Bash shell*.
- MinGW: MinGW (minimalis GNU for Windows) adalah pelabuhan *GNU Compiler Collection* (GCC) dan GNU Binutils untuk digunakan dalam Windows. Ini juga termasuk MSYS (*Minimal System*), yang pada dasarnya *Bourne shell* (*bash*).

- MinGW-W64: sebuah *fork* dari MinGW yang mendukung 32-bit dan 64-bit Windows. MinGW-W64 (sebuah *fork* dari MinGW, tersedia di <http://mingw-w64.org/doku.php>) mendukung sasaran 32-bit dan 64-bit Windows asli. Anda dapat menginstal "MinGW-W64" di bawah "Cygwin" dengan memilih paket ini (di bawah "devel" kategori):
- mingw64-x86_64-gcc-core: 64-bit C *compiler* untuk target asli 64-bit Windows. *Executable* adalah "x86_64-w64-mingw32-gcc".
- mingw64-x86_64-gcc-g ++: *compiler* C++ 64-bit untuk target asli 64-bit Windows. *Executable* adalah "x86_64-w64-mingw32-g++".
- mingw64-i686-gcc-core: 64-bit C *compiler* untuk target asli 32-bit Windows. *Executable* adalah "i686-w64-mingw32-gcc".
- mingw64-i686-gcc-g ++: *compiler* C++ 64-bit untuk target asli 32-bit Windows. *Executable* adalah "i686-w64-mingw32-g++".

2.2.2 Meng-compile Program Menggunakan GCC

Cara meng-*compile source code* menjadi program adalah sebagai berikut:

Pada *folder* dengan *source code* jalankan perintah ini:

```
$ gcc source_code.c -o program_hasil
```

atau

```
$ gcc source_code.c
```

Apabila tidak diberikan opsi hasil maka gcc akan menghasilkan program dengan nama: **a.out**

Cara menjalankan program yang telah *dcompile*:

```
$ chmod +x ./program_hasil
$ ./program_hasil
```

Modul 3 Proses

Tujuan Praktikum

1. Praktikan dapat melakukan manajemen proses.
2. Praktikan dapat menggunakan *syscall* pada program.

3.1 Manajemen Proses pada Linux

3.1.1 Pembuatan Proses

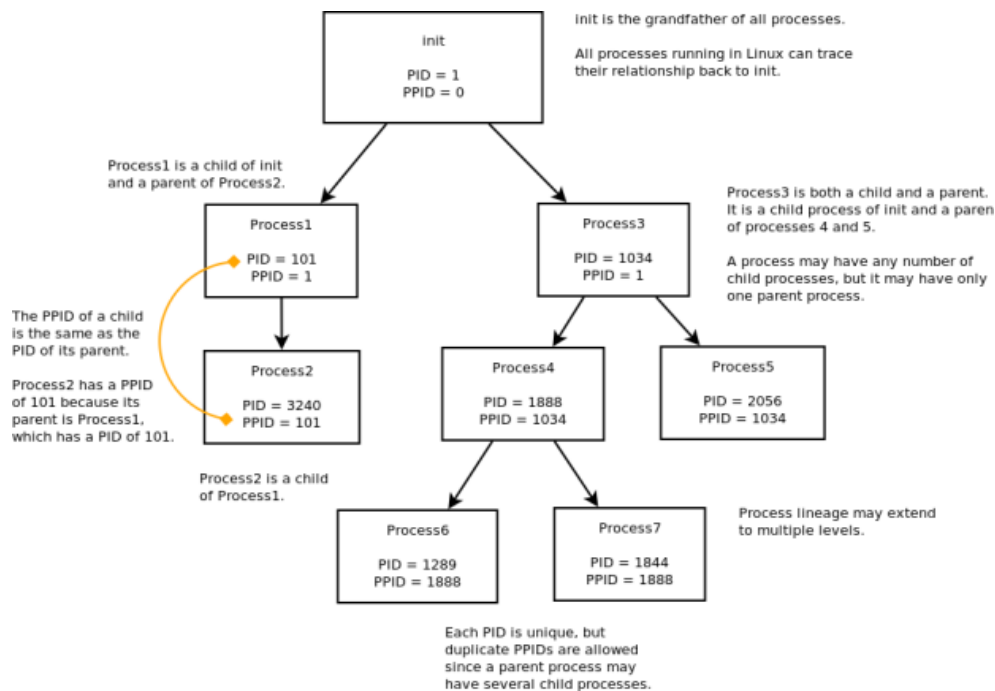
Proses baru biasanya dibuat ketika proses yang ada membuat salinan persis dari dirinya sendiri dalam memori. Proses anak atau proses salinan akan memiliki lingkungan yang sama dengan induknya (proses asal), tetapi hanya nomor ID proses berbeda. Ada dua cara konvensional yang digunakan untuk membuat proses baru di Linux:

- a. Menggunakan fungsi **system()**. Metode ini relatif sederhana, namun tidak efisien dan memiliki risiko keamanan.
- b. Menggunakan fungsi **fork()** dan **exec()**. Metode ini adalah sedikit lebih rumit namun menawarkan fleksibilitas yang lebih besar, kecepatan, dan juga keamanan.

3.1.2 PID dan PPID

Pengeksekusi suatu aplikasi disimpan pada *disk* disebut *program*, kemudian *program* di-load ke memori dan menjalankan suatu proses. Ketika proses tersebut berjalan, maka *system* akan secara otomatis memberikan nomor yang bisa disebut *process ID* atau **PID** sebagai identitas suatu proses. PID yang diberikan oleh *system* selalu unik sehingga tidak akan terjadi adanya ambiguitas antar proses yang berjalan.

Selain nomor PID yang unik, setiap proses juga memiliki PPID (*Parents Process ID*). PPID ini mengidentifikasi setiap proses yang memulainya atau proses sebelumnya. Setiap PPID tunggal dapat melahirkan banyak PID, dengan masing-masing PID yang unik namun dengan PPID yang sama.



Gambar 3-1 Hubungan PID dengan PPID

3.1.3 Background Process dan Foreground Process

Pada dasarnya ada dua jenis proses di Linux:

- Proses *foreground* (juga disebut sebagai proses interaktif) adalah proses yang diinisialisasi dan dikendalikan melalui sesi terminal. Dengan kata lain, harus ada pengguna terhubung ke sistem untuk memulai proses tersebut; mereka belum dimulai secara otomatis sebagai bagian dari sistem fungsi/layanan.
- Proses *background* (juga disebut sebagai non-interaktif/otomatis proses) adalah proses yang tidak terhubung ke terminal; mereka tidak mengharapkan *input* pengguna.

3.1.4 Kill Process

Sering kali proses yang sedang berjalan perlu dihentikan untuk keperluan tertentu. Sistem operasi Linux dilengkapi dengan perintah **Kill** untuk mengakhiri proses.

Perintah **Kill** mengirimkan sinyal yang telah ditentukan sebelumnya. Untuk melakukan proses **Kill** dapat dilakukan dalam beberapa cara, baik secara langsung maupun menggunakan *script*. Menggunakan perintah **Kill** dari */usr/bin* menyediakan beberapa fitur tambahan untuk memberhentikan proses dengan proses yang menggunakan **pskill**. Sintaks perintah *kill* secara umum adalah:

```
# kill [sinyal] PID
```

Untuk menghentikan proses berdasarkan nama sinyal dapat menggunakan sintaks berikut:

Tabel 3-1 Beberapa Daftar Sinyal untuk Perintah Kill

Nama Sinyal	Nomor Sinyal	Keterangan
SIGHUP	1	<i>Hangup</i>
SIGKILL	9	<i>Kill Signal</i>
SIGTERM	15	<i>Terminate</i>

Untuk membunuh suatu proses, kita perlu mengetahui ID proses. Proses adalah *instance* dari sebuah program. Setiap kali program dimulai, secara otomatis PID dengan unik akan diberikan untuk proses itu. Setiap proses di Linux, memiliki PID. Proses pertama yang dimulai ketika sistem Linux *boot* disebut *init* proses (PID bernilai '1'). *Init* adalah proses utama dan tidak akan bisa dimatikan dengan cara ini, yang menjamin bahwa proses master tidak mendapat dibunuh secara tidak sengaja. *Init* bisa dimatikan dengan *shutdown*.

Sebelum mengeksekusi perintah *kill*, beberapa poin yang penting untuk dicatat:

- Pengguna dapat membunuh semua prosesnya.
- Pengguna tidak dapat membunuh proses pengguna lain.
- Pengguna tidak dapat membunuh proses sistem menggunakan.
- Root user* dapat membunuh proses sistem tingkat dan proses pengguna.

Berikut contoh:

PID	TTY	TIME	CMD
1	?	00:00:01	init
2	?	00:00:00	kthreadd
3	?	00:00:00	migration/0
4	?	00:00:00	ksoftirqd/0
5	?	00:00:00	migration/0
6	?	00:00:00	watchdog/0
7	?	00:00:01	events/0
8	?	00:00:00	cgroup
9	?	00:00:00	khelper
3139	?	00:00:01	mysql

Sebagai contoh, akan dilakukan **Kill** pada proses *mysql*. Untuk mendapatkan **PID** dari proses *mysql*, tuliskan sintaks berikut.

```
# pgrep mysql
```

Pgrep adalah *command* yang digunakan untuk mencari PID dari proses yang sedang aktif berdasarkan aplikasi yang dicari. Kemudian akan muncul *output* seperti ini:

```
3139
```

Untuk mengeksekusi perintah *Kill Signal* pada PID 3139, gunakan sintaks berikut.

```
# kill -9 3139
```

Perintah diatas akan memberhentikan proses dengan PID 3139, dimana PID adalah nilai numerik dari suatu proses.

Cara lain yang dapat digunakan untuk fungsi yang sama dapat menggunakan sintaks berikut.

```
# kill -SIGTERM 3139
```

Bagaimana dengan menghentikan suatu proses yang menggunakan nama proses?
Untuk melakukan ini, Anda harus mengetahui nama proses dengan benar, sebelum melakukan eksekusi proses *Kill* dan menuliskan nama proses. Untuk melakukan ini, gunakan *command pkill*.
Contoh:

```
# pkill mysql
```

Untuk melakukan *kill process* lebih dari satu proses dalam satu waktu dapat dilakukan dengan menuliskan sintaks:

```
# kill PID1 PID2 PID3
atau
# kill -9 PID1 PID2 PID3
atau
# kill -SIGKILL PID1 PID2 PID3
```

Untuk menghentikan dengan proses yang memiliki terlalu banyak dan sejumlah anak proses, Linux memiliki perintah *killall*. Contoh:

```
# killall mysql
```

3.1.5 Mengirim Sinyal ke Proses

Cara mendasar mengendalikan proses di Linux adalah dengan mengirimkan sinyal kepada mereka. Ada beberapa sinyal yang dapat dikirim ke proses, untuk melihat semua sinyal, jalankan sintaks berikut:

```
$ kill -l
```

```
[root@tecmint ~]# 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
21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF   28) SIGWINCH   29) SIGIO        30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5 60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
[root@tecmint ~]#
```

Gambar 3-2 List Semua Sinyal Kill pada Linux

Untuk mengirim sinyal ke proses, gunakan perintah *kill*, *pkill* atau *pgrep* yang disebutkan sebelumnya. Program hanya dapat merespon sinyal jika mereka diprogram untuk mengenali sinyal. Kebanyakan sinyal digunakan untuk keperluan internal proses atau untuk *programmer* ketika mereka menulis kode. Berikut ini adalah sinyal yang berguna untuk pengguna sistem:

- SIGHUP 1 - dikirim ke proses ketika terminal pengendali ditutup.
- SIGINT 2 - dikirim ke proses oleh terminal pengendali ketika pengguna menyela proses dengan menekan [Ctrl+C].
- SIGQUIT 3 - dikirim ke proses jika pengguna mengirimkan sinyal berhenti [Ctrl+D].
- SIGKILL 9 - sinyal ini segera berakhir (membunuh) proses dan proses tidak akan melakukan operasi pembersihan.
- SIGTERM 15 - ini penghentian program sinyal (membunuh akan mengirim ini secara *default*).
- SIGTSTP 20 - dikirim ke proses oleh terminal pengendali untuk memintanya untuk menghentikan (terminal berhenti); diprakarsai oleh pengguna menekan [Ctrl+Z].

Berikut ini adalah perintah **kill** contoh untuk membunuh aplikasi Firefox menggunakan PID:

```
$ pidof firefox
$ kill 9 2687
atau
$ kill -KILL 2687
atau
$ kill -SIGKILL 2687
```

3.1.6 Mengubah Prioritas Proses

Pada sistem Linux, semua proses aktif memiliki prioritas dan nilai tertentu. Proses dengan prioritas yang lebih tinggi biasanya akan mendapatkan lebih banyak waktu CPU daripada proses prioritas yang lebih rendah. Namun, sistem pengguna dengan akses *root* dapat mempengaruhi ini dengan perintah **nice** dan **renice**.

```
$ top
```

Dari *output* perintah di atas, pada kolom NI (kolom ke-4) menunjukkan nilai dari proses. Berikut *output* yang akan muncul:

```
top - 08:53:06 up 16 min, 1 user, load average: 0.20, 0.28, 0.35
Tasks: 238 total, 1 running, 237 sleeping, 0 stopped, 0 zombie
%Cpu(s): 5.4 us, 1.0 sy, 0.0 ni, 93.0 id, 0.6 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 3742792 total, 1144416 free, 1236544 used, 1361832 buff/cache
KiB Swap: 5631996 total, 5631996 free, 0 used, 2249948 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2469	aaronki+	20	0	1757760	168424	56580	S	11.6	4.5	0:37.61	cinnamon
3691	aaronki+	20	0	479484	35208	26268	S	6.3	0.9	0:00.42	gnome-scre+
1946	root	20	0	463000	96932	85920	S	4.3	2.6	0:21.98	Xorg
170	root	20	0	0	0	0	S	1.3	0.0	0:00.69	kworker/u1+
6	root	20	0	0	0	0	S	1.0	0.0	0:00.85	kworker/u1+
921	root	20	0	449740	19428	14048	S	0.7	0.5	0:01.05	NetworkMan+
1743	shinken	20	0	1557824	31040	6504	S	0.7	0.8	0:06.95	shinken-sc+
1817	shinken	20	0	1631460	32172	7856	S	0.7	0.9	0:04.32	shinken-re+
7	root	20	0	0	0	0	S	0.3	0.0	0:01.17	rcu sched
1865	shinken	20	0	1632116	32604	7284	S	0.3	0.9	0:05.92	shinken-br+
1908	shinken	20	0	1557024	30232	6556	S	0.3	0.8	0:03.24	shinken-re+
1953	root	20	0	1633896	34552	5712	S	0.3	0.9	0:04.19	shinken-ar+
2082	shinken	20	0	1631232	29112	4728	S	0.3	0.8	0:00.20	shinken-po+
3684	aaronki+	20	0	41908	3808	3104	R	0.3	0.1	0:00.04	top
1	root	20	0	119696	5924	4040	S	0.0	0.2	0:01.45	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd/0

Gambar 3-3 Daftar Proses yang Berjalan pada Linux

Gunakan perintah **nice** untuk menetapkan nilai untuk proses. Perlu diingat bahwa *user* normal dapat atribut nilai bagus dari 0 sampai 20 proses mereka sendiri. Hanya *user root* dapat menggunakan nilai-nilai yang negatif.

Untuk **renice** prioritas proses, gunakan perintah **renice** sebagai berikut:

```
$ renice +8 2687
$ renice +8 2103
```

3.2 Syscall process

3.2.1 Syscall dengan Fork()

Tujuan **fork()** adalah untuk menciptakan proses baru, yang menjadi proses anak dari pemanggil. Tidak perlu ada argumen dan tidak perlu mengembalikan ID proses. Setelah proses anak baru dibuat, kedua proses akan menjalankan instruksi berikutnya setelah panggilan sistem **fork()**. Karena itu, kita harus membedakan orang tua dari anak. Ini dapat dilakukan dengan menguji nilai yang dikembalikan oleh **fork()**:

- Jika **fork()** mengembalikan nilai negatif, penciptaan proses anak tidak berhasil.
- **fork()** mengembalikan nilai nol ke proses anak yang baru dibuat.
- **fork()** mengembalikan nilai positif, ID proses dari proses anak, ke induk. ID proses yang dikembalikan adalah jenis **pid_t** yang ditentukan dalam **sys/types.h**. Biasanya, ID proses adalah bilangan integer. Selain itu, proses dapat menggunakan fungsi **getpid()** untuk mengambil ID proses dan **getppid()** untuk mendapatkan *parent id*.

Berikut adalah contoh program 3_1.c untuk memperjelas tiga poin di atas.

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <stdlib.h>

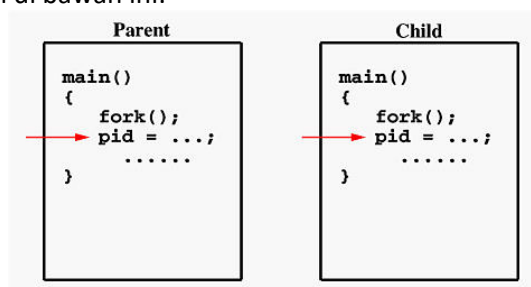
#define MAX_COUNT 200
#define BUF_SIZE 100

void main(void)
{
    pid_t pid;
    pid_t ppid;
    int i;
    char buf[BUF_SIZE];

    fork();
    pid = getpid();
    ppid = getppid();
    for (i = 1; i <= MAX_COUNT; i++) {
        sprintf(buf, "This line is from pid %d with ppid %d, value = %d\n", pid,
        ppid, i);
        write(1, buf, strlen(buf));
    }
    exit(1);
}
```

Jika prosedur **fork()** berhasil dijalankan, maka Unix akan:

- Membuat dua ruang alamat yang identik, satu untuk orang tua dan yang lainnya untuk anak.
- Kedua proses akan memulai eksekusi mereka pada pernyataan berikutnya setelah panggilan **fork()**. Dalam hal ini, kedua proses akan memulai eksekusinya pada pernyataan penugasan seperti yang ditunjukkan di bawah ini:

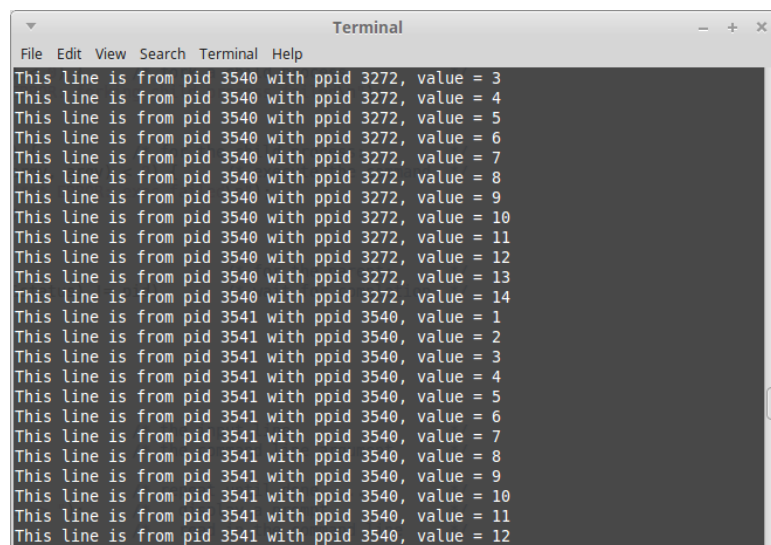


Gambar 3-4 Ilustrasi saat `fork()` dieksekusi di parent dan child

Kedua proses memulai eksekusinya tepat setelah *system call* **fork()**. Karena kedua proses memiliki ruang alamat yang identik tetapi terpisah, variabel-variabel tersebut diinisialisasi sebelum panggilan **fork()** memiliki nilai yang sama di kedua ruang alamat. Karena setiap proses memiliki ruang alamat sendiri, modifikasi apa pun akan independen dari yang lain. Dengan kata lain, jika induk mengubah nilai variabelnya, modifikasi hanya akan mempengaruhi variabel di ruang alamat proses induk. Ruang alamat lain yang dibuat oleh **fork()** panggilan tidak akan terpengaruh meskipun mereka memiliki nama variabel yang identik.

Apa alasan menggunakan **write** daripada **printf**? Ini karena **printf()** adalah "buffered" yang berarti **printf()** akan mengelompokkan *output* dari suatu proses secara bersamaan. Sementara *buffering output* untuk proses induk, anak juga dapat menggunakan **printf** untuk mencetak beberapa informasi, yang juga akan di-buffer. Akibatnya, karena *output* tidak akan dikirim ke layar segera, Anda mungkin tidak mendapatkan urutan hasil yang diharapkan.

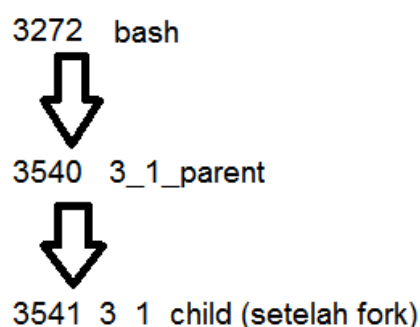
Jika Anda menjalankan program ini, Anda mungkin melihat hal-hal berikut di layar:



```

Terminal
File Edit View Search Terminal Help
This line is from pid 3540 with ppid 3272, value = 3
This line is from pid 3540 with ppid 3272, value = 4
This line is from pid 3540 with ppid 3272, value = 5
This line is from pid 3540 with ppid 3272, value = 6
This line is from pid 3540 with ppid 3272, value = 7
This line is from pid 3540 with ppid 3272, value = 8
This line is from pid 3540 with ppid 3272, value = 9
This line is from pid 3540 with ppid 3272, value = 10
This line is from pid 3540 with ppid 3272, value = 11
This line is from pid 3540 with ppid 3272, value = 12
This line is from pid 3540 with ppid 3272, value = 13
This line is from pid 3540 with ppid 3272, value = 14
This line is from pid 3541 with ppid 3540, value = 1
This line is from pid 3541 with ppid 3540, value = 2
This line is from pid 3541 with ppid 3540, value = 3
This line is from pid 3541 with ppid 3540, value = 4
This line is from pid 3541 with ppid 3540, value = 5
This line is from pid 3541 with ppid 3540, value = 6
This line is from pid 3541 with ppid 3540, value = 7
This line is from pid 3541 with ppid 3540, value = 8
This line is from pid 3541 with ppid 3540, value = 9
This line is from pid 3541 with ppid 3540, value = 10
This line is from pid 3541 with ppid 3540, value = 11
This line is from pid 3541 with ppid 3540, value = 12

```



Gambar 3-5 Ilustrasi *fork()* pada program 3_1.c

Proses 3_1 dijalankan pada *bash* sehingga *parent* dari proses tersebut adalah *bash* (pid = 3272). PID untuk proses 3_1 itu sendiri adalah 3540. Proses 3_1 akan menampilkan angka dari 1 s.d 200. Setelah **fork()** dieksekusi, program utama/*parent* (PID = 3540) akan membuat proses baru (*child* dengan PID 3541) yang mempunyai fungsi yang sama dengan *parent*.

Karena fakta bahwa proses ini dijalankan secara bersamaan, jalur *output* mereka dicampurkan dengan cara yang agak tidak dapat diprediksi. Selain itu, urutan garis-garis ini ditentukan oleh penjadwal CPU. Oleh karena itu, jika Anda menjalankan program ini lagi, Anda mungkin mendapatkan hasil yang sama sekali berbeda.

3.2.2 Syscall dengan Exec()

Proses anak yang dibuat tidak harus menjalankan program yang sama seperti yang dilakukan oleh proses induk. Panggilan jenis sistem **exec** memungkinkan proses untuk menjalankan *file* program apa pun, yang mencakup biner yang dapat dieksekusi atau skrip *shell*. Kita hanya membahas satu panggilan sistem seperti **execvp()**. Panggilan sistem **execvp()** membutuhkan dua argumen:

- Argumen pertama adalah *string* karakter yang berisi nama *file* yang akan dieksekusi.
- Argumen kedua adalah *pointer* ke *array string* karakter. Lebih tepatnya, tipenya adalah `char **`, yang persis sama dengan *array* `argv` yang digunakan dalam program utama:

`int main (int argc, char ** argv)`

Perhatikan bahwa argumen ini harus diakhiri dengan *null*. Ketika **execvp ()** dijalankan, *file* program yang diberikan oleh argumen pertama akan dimuat ke ruang alamat pemanggil dan *over-write* program di sana. Kemudian, argumen kedua akan diberikan kepada program dan memulai eksekusi. Akibatnya, setelah *file* program yang ditentukan memulai eksekusinya, program asli di ruang alamat pemanggil hilang dan digantikan oleh program baru. **execvp()** mengembalikan nilai negatif jika eksekusi gagal (misalnya, *file* permintaan tidak ada).

Berikut adalah contoh program dalam *file* 3_2.c:

```
#include <stdio.h>
#include <sys/types.h>

void parse(char *line, char **argv)
{
    while (*line != '\0') {          /* if not the end of line ..... */
        while (*line == ' ' || *line == '\t' || *line == '\n')
            *line++ = '\0';          /* replace white spaces with 0 */
        *argv++ = line;              /* save the argument position */
        while (*line != '\0' && *line != ' ' &&
               *line != '\t' && *line != '\n')
            line++;                  /* skip the argument until ... */
    }
    *argv = '\0';                   /* mark the end of argument list */
}

void execute(char **argv)
{
    pid_t pid;
    int status;

    if ((pid = fork()) < 0) {         /* fork a child process */
        printf("*** ERROR: forking child process failed\n");
        exit(1);
    }
    else if (pid == 0) {              /* for the child process: */
        if (execvp(*argv, argv) < 0) /* execute the command */
            printf("*** ERROR: exec failed\n");
        exit(1);
    }
    else {                            /* for the parent: */
        while (wait(&status) != pid) /* wait for completion */
            ;
    }
}

void main(void)
{
    char line[1024];                 /* the input line */
    char *argv[64];                  /* the command line argument */

    while (1) {                      /* repeat until done .... */
```

```

printf("Shell -> ");      /* display a prompt      */
gets(line);              /* read in the command line */
printf("\n");
parse(line, argv);        /* parse the line          */
if (strcmp(argv[0], "exit") == 0) /* is it an "exit"?      */
    exit(0);              /* exit if it is          */
execute(argv);            /* otherwise, execute the command */
}

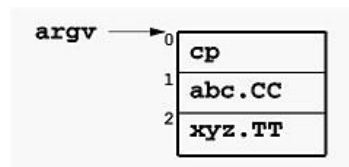
```

Fungsi **parse()** mengambil jalur *input* dan mengembalikan *array zero-terminated* dari *pointer char*, yang masing-masing menunjuk ke *string* karakter yang diakhiri nol. Fungsi ini mengulang hingga nol biner ditemukan, yang berarti akhir dari jalur **garis** masukan tercapai. Jika karakter **baris** saat ini bukan nol biner, **parse()** melewati semua *white-space* dan menggantikannya dengan nol biner sehingga *string* diakhiri secara efektif. Setelah **parse()** menemukan *non-white space*, alamat lokasi tersebut disimpan ke posisi **argv** saat ini dan indeks dimajukan. Kemudian, **parse()** melompati semua karakter *non-whitespace*. Proses ini berulang hingga akhir **garis string** tercapai dan pada saat itu **argv** diakhiri dengan nol.

Misalnya, jika baris *input* adalah *string* sebagai berikut:

"cp abc.CC xyz.TT"

Fungsi **parse ()** akan mengembalikan *array argv []* dengan konten berikut:



Fungsi **execute ()** mengambil *array argv []*, memperlakukannya sebagai argumen baris perintah dengan nama program dalam **argv [0]**, melakukan *forks* pada proses *child*, dan mengeksekusi program yang ditunjukkan dalam proses *child* itu. Saat proses anak mengeksekusi perintah, *parent* menjalankan **wait()**, menunggu penyelesaian proses anak. Dalam kasus khusus ini, *parent* mengetahui ID proses *child* dan oleh karena itu dapat menunggu *child* tertentu untuk menyelesaikannya.

Program utamanya sangat sederhana. Ini mencetak *command prompt*, membaca dalam garis, mem-**parse** menggunakan fungsi **parse()**, dan menentukan apakah nama itu "keluar". Jika "keluar", gunakan **exit()** untuk mengakhiri eksekusi program ini, sebaliknya, menggunakan fungsi **execute()** untuk menjalankan perintah.

```

Shell -> ps

  PID TTY          TIME CMD
 3272 pts/0    00:00:00 bash
 3570 pts/0    00:00:00 dua
 3571 pts/0    00:00:00 ps
Shell -> exit

```

Gambar 3-6 Hasil proses 3_2.c

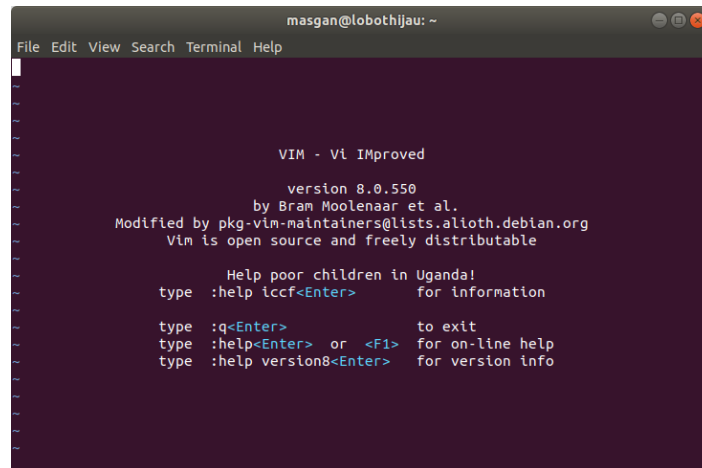
3.3 Vi IMproved (Vim)

Vim adalah program komputer yang dipakai untuk menulis teks (*text editor*). Vim merupakan singkatan dari Vi IMproved yang dirilis pertama kali tahun 1991. Vim merupakan tiruan *editor* yang sudah ada bernama vi dengan menambah beragam fitur baru (perhatikan lagi kepanjangan dari vim). Vi sendiri sudah ada sejak tahun 1972.

Untuk sistem Linux atau Unix (Keluarga BSD, Solaris, dkk.), pemasangan vim bisa dilakukan dengan menggunakan *package manager* masing-masing sistem. Untuk memasang vim dengan sistem operasi Linux keluarga Debian dapat dilakukan dengan memasukkan perintah dalam terminal sebagai berikut :

```
sudo apt install -y vim
```

Dan untuk mengecek apakah vim sudah terinstall atau belum dapat dilakukan dengan cara memasukkan perintah *vim* dalam terminal.



Gambar 3-7 hasil perintah *vim* pada terminal

3.3.1 Insert Mode

Insert Mode memungkinkan kita untuk menyisipkan teks ke dalam dokumen. Gunakan *shortcut* “i” untuk menyisipkan teks sesuai letak kursor atau “o” untuk menyisipkan teks di awal baris.

3.3.2 Visual Mode

Visual Mode memungkinkan kita untuk memilih teks menggunakan *keyboard* layaknya seperti menggunakan *mouse*. Berguna untuk meng-copy beberapa baris teks. *Shortcut* yang digunakan adalah “V”.

3.3.3 Command Mode

Command Mode memungkinkan kita untuk memberikan perintah atau *command* pada vim. Untuk masuk ke dalam *command mode* dapat dilakukan dengan menekan “esc”. *Command* pada vim diawali dengan simbol “:”.

Beberapa contoh *command* yang ada adalah sebagai berikut:

- *save* (:w)
- *save dan exit* (:wq)
- *exit* (:q)
- *force* (“!”), contoh :w! :q!)
- *copy* (y)
- *copy a line* (yy)
- *paste* (p)
- *cut* (d)
- *cut a line* (dd)

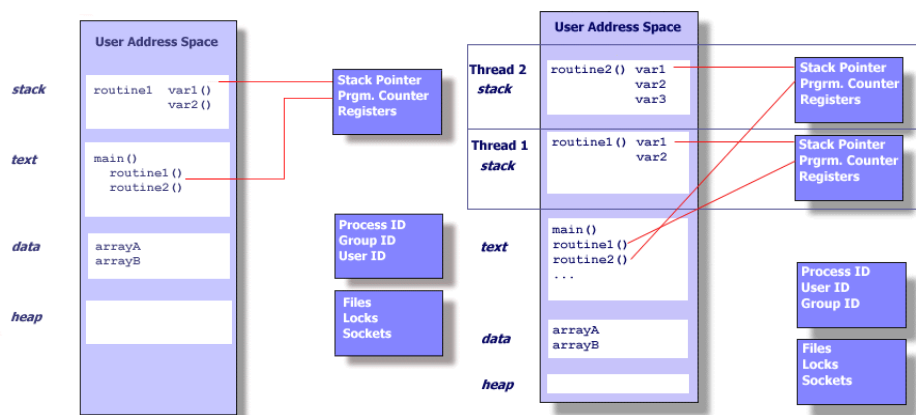
Modul 4 Thread dan Mutex

Tujuan Praktikum

1. Praktikan dapat membuat program dengan *thread*.
2. Praktikan menguasai teknik sinkronisasi dalam *thread* (*mutex*).

4.1 Thread

Thread merupakan urutan instruksi independen yang dapat dijadwalkan untuk dijalankan pada CPU. *Thread* yang terdiri dari *Thread_id*, program, counter, register set, dan stack. Sebuah *Thread* akan berbagi code section (text), data section, dan resource (misal: file) dengan *Thread* lain yang dimiliki oleh proses yang sama (perhatikan gambar di bawah).



Gambar 4-1 Proses (kiri) dan Thread (kanan)

Thread juga sering disebut *lightweight process*. Sebuah proses tradisional atau *heavyweight process* hanya mempunyai *Thread* tunggal. Perbedaan antara proses dengan *Thread* tunggal adalah proses dengan *Thread* yang banyak dapat mengerjakan lebih dari satu tugas pada satu satuan waktu.

Banyak sistem operasi *modern* telah memiliki metode yang memungkinkan sebuah proses untuk memiliki eksekusi *multi-Threads* sehingga memungkinkan proses untuk menjalankan lebih dari satu tugas pada satu waktu. Contohnya sebuah *web browser* mempunyai *Thread* untuk menampilkan gambar atau tulisan sedangkan *Thread* yang lain berfungsi sebagai penerima data dari *network*.

4.1.1 Manajemen Thread

Membuat Thread

Untuk membuat *Thread* dapat menggunakan prosedur sebagai berikut:

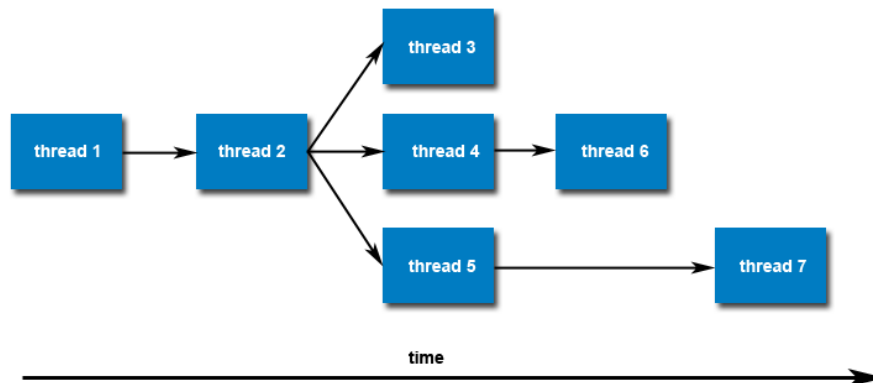
```
pthread_create (Thread, attr, start_routine, arg)
```

pthread_create menciptakan *Thread* baru dan membuatnya dapat dieksekusi. Rutin ini dapat dipanggil beberapa kali dari mana saja dalam kode. Setelah diciptakan, *Thread* adalah *peers* dan bisa membuat *Thread* yang lain.

Argumen **pthread_create** :

- *Thread*: Identifikasi yang unik untuk *Thread* baru dikembalikan oleh *subroutine*.
- *attr*: Objek atribut yang dapat digunakan untuk mengatur atribut *Thread*. Anda dapat menentukan objek atribut *Thread*, atau *NULL* untuk nilai *default*.

- *start_routine*: Sebuah C routine yang akan mengeksekusi Thread ketika sudah dibuat.
- *arg*: Satu argumen yang dapat dilewatkan ke *start_routine*. Ini harus dilalui oleh referensi sebagai penunjuk jenis *void*. *NULL* dapat digunakan jika tidak ada argumen yang dilewatkan.



Gambar 4-2 Thread membuat Thread yang lain (tidak ada hirarki dan dependensi antar Thread ketika dibuat)

Berikut adalah contoh program untuk membuat dan terminasi Thread:

```

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 5

void *PrintHello(void *Threadid)
{
    long tid;
    tid = (long)Threadid;
    printf("Hello World! It's me, Thread #%ld!\n", tid);
    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    pthread_t Threads[NUM_THREADS];
    int rc;
    long t;
    for(t=0;t<NUM_THREADS;t++){
        printf("In main: creating Thread %ld\n", t);
        rc = pthread_create(&Threads[t], NULL, PrintHello, (void *)t);
        if (rc){
            printf("ERROR: return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }

    /* Last thing that main() should do */
    pthread_exit(NULL);
}

```

Untuk meng-*compile* program di atas gunakan perintah sebagai berikut:

```
gcc -pthread -o 3_1 3_1.c
```

Catatan: *library pthread* harus di-*install* terlebih dahulu dengan cara:

```
sudo apt-get install libpthread-stubs0-dev
```

Passing argument

Pthread memungkinkan *programmer* untuk memberikan *argument* pada saat *Thread* dimulai. Semua *argument* harus menggunakan "pass by reference" dan di-cast ke (void *).

Berikut adalah contoh *passing argument* (3_2.c):

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 8

char *messages[NUM_THREADS];

struct Thread_data
{
    int Thread_id;
    int sum;
    char *message;
};

struct Thread_data Thread_data_array[NUM_THREADS];

void *PrintHello(void *Threadarg)
{
    int taskid, sum;
    char *hello_msg;
    struct Thread_data *my_data;

    sleep(1);
    my_data = (struct Thread_data *) Threadarg;
    taskid = my_data->Thread_id;
    sum = my_data->sum;
    hello_msg = my_data->message;
    printf("Thread %d: %s Sum=%d\n", taskid, hello_msg, sum);
    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    pthread_t Threads[NUM_THREADS];
    int *taskids[NUM_THREADS];
    int rc, t, sum;

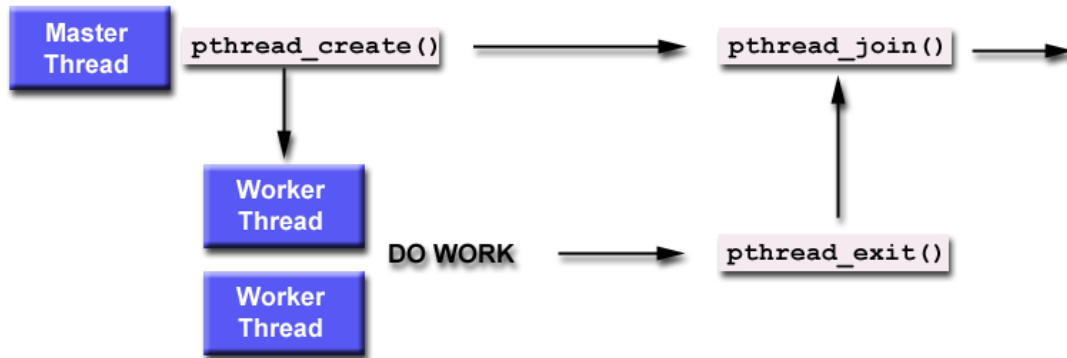
    sum=0;
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvyye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    for(t=0;t<NUM_THREADS;t++) {
        sum = sum + t;
        Thread_data_array[t].Thread_id = t;
        Thread_data_array[t].sum = sum;
        Thread_data_array[t].message = messages[t];
        printf("Creating Thread %d\n", t);
        rc = pthread_create(&Threads[t], NULL, PrintHello, (void *)
            &Thread_data_array[t]);
        if (rc) {
            printf("ERROR: return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
}
```

```
pthread_exit(NULL);
}
```

Join

Joining adalah salah satu cara untuk melakukan sinkronisasi. Perhatikan gambar berikut:



Gambar 4-3 Joining Thread

Subrutin **pthread_join()** akan mem-block “the calling Thread” (dalam hal ini master Thread). Worker Thread akan berjalan. Ketika worker Thread telah selesai (yaitu dengan memanggil **pthread_exit()**) maka subrutin **pthread_join()** akan memperoleh **Thread_id** dari worker Thread tersebut. Jika Subrutin **pthread_join()** memperoleh **Thread_id** yang sesuai maka **pthread_join()** akan unblock Thread master. Contoh: Thread master membuat 2 Thread untuk menggambar dan menghitung. Thread master akan menampilkan hasil gambar dan hitung. Selama Thread menghitung belum selesai maka Thread master akan block. Setelah Thread hitung selesai maka Thread master akan menampilkan hasil hitungan.

Berikut adalah contoh Thread_join():

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 4

void *BusyWork(void *t)
{
    int i;
    long tid;
    double result=0.0;
    tid = (long)t;
    printf("Thread %ld starting...\n",tid);
    for (i=0; i<1000000; i++)
    {
        result = result + sin(i) * tan(i);
    }
    printf("Thread %ld done. Result = %e\n",tid, result);
    pthread_exit((void*) t);
}

int main (int argc, char *argv[])
{
    pthread_t Thread[NUM_THREADS];
    pthread_attr_t attr;
    int rc;
    long t;
    void *status;

    /* Initialize and set Thread detached attribute */
    pthread_attr_init(&attr);
```

```

pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

for(t=0; t<NUM_THREADS; t++) {
    printf("Main: creating Thread %ld\n", t);
    rc = pthread_create(&Thread[t], &attr, BusyWork, (void *)t);
    if (rc) {
        printf("ERROR: return code from pthread_create() is %d\n", rc);
        exit(-1);
    }
}

/* Free attribute and wait for the other Threads */
pthread_attr_destroy(&attr);
for(t=0; t<NUM_THREADS; t++) {
    rc = pthread_join(Thread[t], &status);
    if (rc) {
        printf("ERROR: return code from pthread_join() is %d\n", rc);
        exit(-1);
    }
    printf("Main: completed join with Thread %ld having a status of %ld\n", t, (long)status);
}
printf("Main: program completed. Exiting.\n");
pthread_exit(NULL);
}

```

4.1.2 Perilaku Random Thread

Perilaku *random Thread* yaitu perilaku dari *Thread* yang tidak bisa ditebak karena perilaku *Thread* tergantung dari penjadwalan oleh cpu. Program yang berjalan pada saat yang bersamaan akan berperilaku non-deterministik (artinya: tidak mungkin untuk mengetahui hasil akhir jika hanya dengan melihat program saja). Berikut adalah contoh sederhana dari sebuah program non-deterministik:



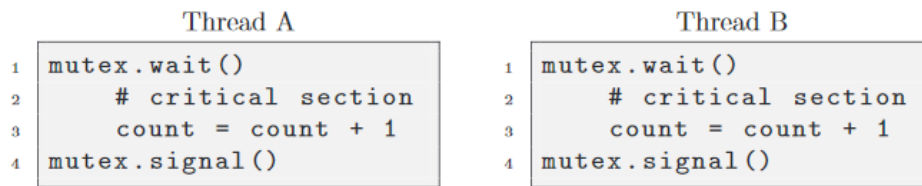
Gambar 4-4 Program Non-deterministik

Karena dua *Thread* berjalan secara bersamaan, urutan eksekusi tergantung pada *scheduler*. Ketika menjalankan program ini, *output* mungkin "yes no" atau "no yes". Non-deterministik adalah salah satu hal yang membuat program tidak berjalan sesuai spesifikasi. Sebuah program mungkin berjalan benar 1000 kali berturut-turut, dan kemudian pada eksekusi ke 1001 menghasilkan *output* yang salah, karena keputusan tertentu *scheduler*. Jenis *bug*/kesalahan seperti ini hampir mustahil untuk menemukan dengan pengujian; mereka hanya dapat dihindari dengan berhati-hati dalam implementasi dan perencanaan sinkronisasi yang benar dalam pemrograman.

Sinkronisasi pada *Thread* harus dilakukan oleh *programmer* karena sifat non-deterministik dari *Thread*. Sinkronisasi *Thread* dapat dilakukan dengan menggunakan *semaphore*, *mutex* ataupun *conditional* variabel.

4.2 Mutex

Mutual Exclusion (Mutex) adalah syarat atau kondisi yang harus dipenuhi untuk mencegah terjadinya pengaksesan *critical section* oleh lebih dari satu proses dalam waktu yang bersamaan. Penerapan *mutex* dalam *Thread* yaitu agar tidak terjadi data *overwritten* sehingga data tetap konsisten. *Critical Section* adalah *resource* yang dalam satu waktu hanya boleh diakses oleh satu proses saja. Ilustrasi *mutex*:



Gambar 4-5 Ilustrasi Mutex

Instruksi `count=count+1` adalah *critical section*. Harus dipastikan bahwa instruksi ini hanya bisa dieksekusi oleh satu *Thread* saja (jika 2 *Thread* dapat mengakses instruksi ini secara bersamaan maka hasil akhirnya akan salah).

4.2.1 Membuat Mutex

Variabel *mutex* harus dideklarasikan dengan tipe **`pthread_mutex_t`**, dan harus diinisialisasi sebelum digunakan. Ada dua cara untuk menginisialisasi variabel *mutex*:

1. Secara statis, ketika dideklarasikan diawal program. Sebagai contoh :
`pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;`
2. Secara dinamis, dengan rutinitas **`pthread_mutex_init()`**. Metode ini memungkinkan pengaturan atribut objek *mutex*, *attr*.

Mutex awalnya tidak terkunci. Objek *attr* digunakan untuk membangun properti untuk objek *mutex*, dan harus bertipe **`pthread_mutexattr_t`** jika digunakan (dapat ditetapkan sebagai *NULL* untuk menerima default). Standar *Pthreads* mendefinisikan tiga atribut opsional *mutex*:

- Protokol: Menentukan protokol yang digunakan untuk mencegah inversi prioritas untuk *mutex*.
- *Prioceiling*: Menentukan batas prioritas suatu *mutex*.
- Proses-bersama: Menentukan proses dalam berbagi dari sebuah *mutex*.

Perhatikan bahwa tidak semua implementasi dapat menyediakan tiga atribut *mutex* opsional. **`pthread_mutexattr_init()`** dan **`pthread_mutexattr_destroy()`** secara terus-menerus digunakan untuk membuat dan menghancurkan objek atribut *mutex* masing-masing. Sedangkan **`pthread_mutex_destroy()`** harus digunakan untuk membebaskan objek *mutex* yang tidak lagi diperlukan. Selain itu, terdapat beberapa 3 jenis tipe data (rutin) untuk *mutex* yang dapat digunakan untuk mengunci dan membuka *mutex*:

1. **`Pthread_mutex_lock()`** digunakan oleh *Thread* untuk mendapatkan kunci pada variabel *mutex* yang ditentukan. Jika *mutex* sudah dikunci oleh *Thread* lain, panggilan ini akan memblokir *Thread* panggilan sampai *mutex* dibuka kuncinya.
2. **`Pthread_mutex_trylock()`** akan berusaha mengunci *mutex*. Namun, jika *mutex* sudah terkunci, rutinitas akan segera kembali dengan kode kesalahan "*busy*". Rutin ini mungkin berguna dalam mencegah kondisi *deadlock*, seperti dalam situasi prioritas-inversi.
3. **`Pthread_mutex_unlock()`** akan membuka kunci *mutex* jika dipanggil oleh pemilik *Thread*. Memanggil rutin ini diperlukan setelah *Thread* selesai menggunakan data terlindungi jika untaian lain untuk memperoleh *mutex* untuk pekerjaan mereka dengan data yang dilindungi. Kesalahan akan dikembalikan: Jika *mutex* sudah dibuka kuncinya atau jika *mutex* dimiliki oleh *Thread* lain.

4.2.2 Implementasi Mutex dalam Program

Berikut adalah implementasi penggunaan *variable mutex* dalam program *Thread* dalam menjalankan *dotproduct*. Data utama tersedia untuk semua *Thread* melalui struktur yang dapat diakses secara global. Setiap *Thread* bekerja pada bagian data yang berbeda. Bagian inti *Thread* akan menunggu

semua *Thread* selesai melakukan pekerjaannya dan kemudian akan mencetak hasil akhir berupa penjumlahan total.

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

typedef struct
{
    double      *a;
    double      *b;
    double      sum;
    int         veclen;
} DOTDATA;

#define NUMTHRDS 4
#define VECLEN 100
DOTDATA dotstr;
pthread_t callThd[NUMTHRDS];
pthread_mutex_t mutexsum;

/*
Fungsi dotprod diaktifkan ketika Thread dibuat.
Semua masukan untuk rutin ini diperoleh dari struktur
dari tipe DOTDATA dan semua output dari fungsi ini ditulis ke dalam
struktur ini. Manfaat dari pendekatan ini jelas untuk
program multi-Thread: ketika sebuah Thread yang dibuat dapat melewati satu
argumen ke fungsi yang diaktifkan - biasanya argumen ini
adalah nomor Thread. Semua informasi lain yang dibutuhkan oleh
fungsi diakses dari struktur yang dapat diakses secara global.
*/

void *dotprod(void *arg)
{
    int i, start, end, len ;
    long offset;
    double mysum, *x, *y;
    offset = (long)arg;

    len = dotstr.veclen;
    start = offset*len;
    end   = start + len;
    x = dotstr.a;
    y = dotstr.b;

    /* Melakukan dotproduct dan menyimpan hasil ke variable yang telah ditentukan*/

    mysum = 0;
    for (i=start; i<end ; i++)
    {
        mysum += (x[i] * y[i]);
    }

    /* Mengunci mutex sebelum memperbarui nilai yang dibagikan dalam struktur,
kemudian akan dibuka setelah meng-update nilai */

    pthread_mutex_lock (&mutexsum);
    dotstr.sum += mysum;
    pthread_mutex_unlock (&mutexsum);

    pthread_exit((void*) 0);
}

/* Program utama membuat Thread yang melakukan semua pekerjaan dan kemudian cetak
hasil setelah selesai. Sebelum membuat Thread, data input dibuat. Karena semua
Thread memperbarui struktur bersama kita membutuhkan mutex. Thread utama harus
menunggu semua Thread untuk diselesaikan, menunggu setiap Thread.
```

```

*/

int main (int argc, char *argv[])
{
    long i;
    double *a, *b;
    void *status;
    pthread_attr_t attr;

    a = (double*) malloc (NUMTHRDS*VECLEN*sizeof(double));
    b = (double*) malloc (NUMTHRDS*VECLEN*sizeof(double));

    for (i=0; i<VECLEN*NUMTHRDS; i++)
    {
        a[i]=1.0;
        b[i]=a[i];
    }

    dotstr.vecLEN = VECLen;
    dotstr.a = a;
    dotstr.b = b;
    dotstr.sum=0;

    pthread_mutex_init(&mutexsum, NULL);

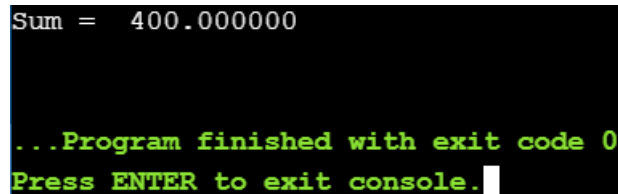
    /* Membuat Thread untuk dotproduct */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    for(i=0; i<NUMTHRDS; i++)
    {
        pthread_create(&callThd[i], &attr, dotprod, (void *)i);
    }
    /* Setiap Thread bekerja pada kumpulan data yang berbeda.
    Offset ditentukan oleh 'i'.
    Ukuran data untuk setiap Thread ditunjukkan oleh VECLen */
    pthread_attr_destroy(&attr);

    /* Menunggu Threads lainnya */
    for(i=0; i<NUMTHRDS; i++)
    {
        pthread_join(callThd[i], &status);
    }

    /* Setelah digabungkan, cetak hasil akhir dan destroy mutex */
    printf ("Sum = %f \n", dotstr.sum);
    free (a);
    free (b);
    pthread_mutex_destroy(&mutexsum);
    pthread_exit(NULL);
}
}

```



```

Sum = 400.000000

...Program finished with exit code 0
Press ENTER to exit console.

```

Gambar 4-6 Hasil Running Program Thread

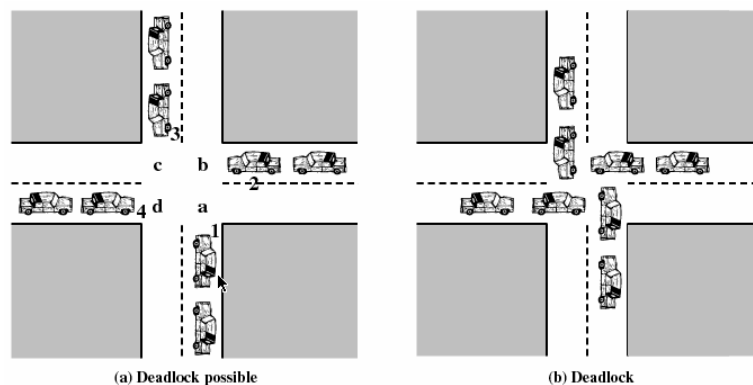
Modul 5 Deadlock

Tujuan Praktikum

1. Praktikan dapat membuat program dan analisis *deadlock*.

5.1 Deadlock

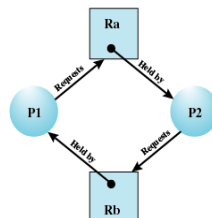
Deadlock adalah kondisi dimana sejumlah proses ter-*block* secara permanen akibat saling memperebutkan *resource* atau saling menunggu pesan dari proses lain. Contoh sederhana *deadlock* pada kehidupan sehari-hari yaitu kemacetan.



Gambar 5-1 Ilustrasi Deadlock

Ada beberapa kondisi yang memungkinkan terjadinya *deadlock*:

1. *Mutual Exclusion*; kondisi saat sebuah *resource* hanya boleh digunakan oleh sebuah proses dalam satu waktu.
2. *Hold-and-Wait*; kondisi dimana sebuah proses boleh terus-menerus menggunakan *resource* sambil menunggu *resource* yang lain.
3. *No-preemption*; kondisi saat sebuah *resource* yang sedang digunakan oleh suatu proses tidak boleh direbut/disela.
4. *Circular wait*; merupakan rangkaian beberapa proses dan *resource* yang membentuk cincin dan setiap proses sedang menggunakan **minimal** satu *resource* yang juga sedang dibutuhkan oleh proses didekatnya.



Gambar 5-2 Circular Wait

5.2 Deadlock Avoidance: Algoritma Banker

Dibawah ini adalah contoh program C untuk mensimulasikan *algoritma banker* untuk *deadlock avoidance*. Dalam lingkungan yang *multiprogramming*, beberapa proses mungkin bersaing untuk sejumlah *resource* yang terbatas. Proses meminta *resource*; jika *resource* tidak tersedia pada waktu saat yang tepat, proses memasuki keadaan menunggu. Terkadang, proses yang menunggu tidak pernah lagi mampu mengubah *state*, karena *resource* yang diminta telah dipegang oleh proses yang sudah menunggu lainnya. Situasi ini disebut *avoidance*. *Deadlock Avoidance* adalah salah satu teknik untuk menangani kasus kebuntuan. Pendekatan ini mensyaratkan bahwa sistem operasi diberikan informasi tambahan sebelumnya mengenai *resource* yang proses akan meminta dan menggunakan selama hidup. Dengan pengetahuan tambahan ini, sistem operasi dapat memutuskan untuk setiap permintaan apakah proses harus menunggu. Untuk menentukan apakah permintaan saat ini bisa dieksekusi atau harus ditunda, sistem harus mempertimbangkan *resource* yang tersedia saat ini, *resource* saat ini dialokasikan untuk setiap proses, dan permintaan yang akan datang dan rilis masing-masing proses. Algoritma *Banker* adalah algoritma untuk *deadlock avoidance* yang dapat diterapkan ke sistem dengan beberapa contoh setiap jenis *resource*.

```
#include <stdio.h>

struct file
{
    int all[10];
    int max[10];
    int need[10];
    int flag;
};

void main() {
    struct file f[10];
    int fl;
    int i, j, k, p, b, n, r, g, cnt=0, id, newr;
    int avail[10], seq[10];
    printf("Enter number of processes -- ");
    scanf("%d", &n);
    printf("Enter number of resources -- ");
    scanf("%d", &r);
    for(i=0; i<n; i++) {
        printf("Enter details for P%d", i);
        printf("\nEnter allocation\t -- \t");
        for(j=0; j<r; j++) {
            scanf("%d", &f[i].all[j]);
            printf("Enter Max\t\t -- \t");
            for(j=0; j<r; j++)
                scanf("%d", &f[i].max[j]);
            f[i].flag=0;
        }
        printf("\nEnter Available Resources\t -- \t");
        for(i=0; i<r; i++)
            scanf("%d", &avail[i]);

        printf("Enter New Request Details -- ");
        printf("\nEnter pid \t -- \t");
        scanf("%d", &id);
        printf("Enter Request for Resources \t -- \t");

        for(i=0; i<r; i++) {
            scanf("%d", &newr);
            f[id].all[i] += newr;
            avail[i]=avail[i] - newr;
        }

        for(i=0; i<n; i++) {
```

```

        for(j=0;j<r;j++) {
            f[i].need[j]=f[i].max[j]-f[i].all[j];
            if(f[i].need[j]<0) f[i].need[j]=0;
        }
    }
    cnt=0;
    fl=0;
    while(cnt!=n) {
        g=0;
        for(j=0;j<n;j++) {
            if(f[j].flag==0) {
                b=0;
                for(p=0;p<r;p++)
                {
                    if(avail[p]>=f[j].need[p])
                        b=b+1;
                    else
                        b=b-1;
                }
                if(b==r)
                {
                    printf("\nP%d is visited",j);
                    seq[fl++]=j;
                    f[j].flag=1;
                    for(k=0;k<r;k++)
                        avail[k]=avail[k]+f[j].all[k];
                    cnt=cnt+1;
                    printf("(");
                    for(k=0;k<r;k++)
                        printf("%3d",avail[k]);
                    printf(")");
                    g=1;
                }
            }
        }
        if(g==0)
        {
            printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED");
            printf("\n SYSTEM IS IN UNSAFE STATE");
            goto y;
        }
    }
    printf("\nSYSTEM IS IN SAFE STATE");
    printf("\nThe Safe Sequence is -- (");
    for(i=0;i<fl;i++)
        printf("P%d ",seq[i]); printf(")");
    y: printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");
    for(i=0;i<n;i++)
    {
        printf("P%d\t",i);
        for(j=0;j<r;j++)
            printf("%6d",f[i].all[j]);
        for(j=0;j<r;j++)
            printf("%6d",f[i].max[j]);

        for(j=0;j<r;j++)
            printf("%6d",f[i].need[j]);
        printf("\n");
    }
}

```

INPUT					
Enter number of processes	--	5			
Enter number of resources	--	3			
Enter details for P0					
Enter allocation	--	0	1	0	
Enter Max	--		7	5	3
Enter details for P1					
Enter allocation	--	2	0	0	
Enter Max	--	3	2	2	
Enter details for P2					
Enter allocation	--	3	0	2	
Enter Max	--	9	0	2	
Enter details for P3					
Enter allocation	--	2	1	1	
Enter Max	--	2	2	2	
Enter details for P4					
Enter allocation	--	0	0	2	
Enter Max	--	4	3	3	
Enter Available Resources	--	3	3	2	
Enter New Request Details --					
Enter pid	--	1			
Enter Request for Resources	--	1	0	2	

Gambar 5-3 Input untuk Algoritma Banker

OUTPUT				
P1 is visited(5 3 2)				
P3 is visited(7 4 3)				
P4 is visited(7 4 5)				
P0 is visited(7 5 5)				
P2 is visited(10 5 7)				
SYSTEM IS IN SAFE STATE				
The Safe Sequence is -- (P1 P3 P4 P0 P2)				
Process	Allocation			Need
P0	0	1	0	7 4 3
P1	3	0	2	0 2 0
P2	3	0	2	6 0 0
P3	2	1	1	0 1 1
P4	0	0	2	4 3 1

Gambar 5-4 Output untuk Algoritma Banker

Modul 6 Memori

Tujuan Praktikum

1. Praktikan dapat melakukan manajemen memori.
2. Praktikan dapat mengukur performansi memori.

6.1 Memori

Linux dilengkapi dengan beberapa perintah yang berbeda untuk memeriksa penggunaan memori. Perintah *free* menampilkan jumlah total memori fisik, digunakan dalam sistem, yang tersisa, *buffer* yang digunakan oleh *kernel* dan *cache memory*. Selain informasi memori, *free* juga menampilkan informasi mengenai *swap file*.

1. Free

Untuk menampilkan ukuran memori bebas dalam MB (*megabyte*) dapat menggunakan perintah:

```
$ free -m
```

Contoh output:

	total	used	free	shared	buffers	cached
Mem:	750	625	125	0	35	335
-/+ buffers/cache:		254	496			
Swap:	956	0	956			

Gambar 6-1 Free Memory dalam MB

Menampilkan garis yang berisi total memori dalam MB:

```
$ free -t -m
```

Contoh output:

	total	used	free	shared	buffers	cached
Mem:	750	625	125	0	35	335
-/+ buffers/cache:		253	496			
Swap:	956	0	956			
Total:	1707	625	1082			

Gambar 6-2 Total Memory dalam MB

Untuk menampilkan *free* RAM yang ada:

```
$ free -m
```

Contoh output:

	total	used	free	shared	buffers	cached
Mem:	11909	11749	160	0	349	8573
-/+ buffers/cache:		2825	9083			
Swap:	6143	193	5950			

Gambar 6-3 Free RAM

Pada *output* di atas, *server* tersebut telah menggunakan 2825 MB RAM, dan 9083 MB tersedia untuk pengguna dan program lain.

2. Cat/proc/meminfo

Cara berikutnya untuk memeriksa penggunaan memori adalah untuk membaca *file* /proc/meminfo. Perlu diketahui bahwa sistem *file* /proc tidak berisi *file* nyata/*file* asli. Mereka adalah *virtual file* yang berisi dinamis informasi tentang *kernel* dan sistem.

\$ cat /proc/meminfo	
MemTotal:	8167848 kB
MemFree:	1409696 kB
Buffers:	961452 kB
Cached:	2347236 kB
SwapCached:	0 kB
Active:	3124752 kB
Inactive:	2781308 kB
Active(anon):	2603376 kB
Inactive(anon):	309056 kB
Active(file):	521376 kB
Inactive(file):	2472252 kB
Unevictable:	5864 kB
Mlocked:	5880 kB
SwapTotal:	1998844 kB
SwapFree:	1998844 kB
Dirty:	7180 kB
Writeback:	0 kB
AnonPages:	2603272 kB
Mapped:	788380 kB
Shmem:	311596 kB
Slab:	200468 kB
SReclaimable:	151760 kB
SUnreclaim:	48708 kB
KernelStack:	6488 kB
PageTables:	78592 kB
NFS_Unstable:	0 kB
Bounce:	0 kB
WritebackTmp:	0 kB
CommitLimit:	6082768 kB
Committed_AS:	9397536 kB
VmallocTotal:	34359738367 kB
VmallocUsed:	420204 kB
VmallocChunk:	34359311104 kB
HardwareCorrupted:	0 kB
AnonHugePages:	0 kB
HugePages_Total:	0
HugePages_Free:	0
HugePages_Rsvd:	0
HugePages_Surp:	0
Hugepagesize:	2048 kB

Gambar 6-4 Informasi Kernel dan Sistem

3. Vmstat

Perintah *vmstat* melaporkan informasi tentang proses, memori, *paging*, blok IO, perangkat, dan aktivitas *cpu*. Cara lain adalah dengan menggunakan perintah *vmstat* (*Virtual Memory Stats*) dengan *-s* beralih. Ini akan memberikan daftar memori dalam rincian dengan baris pertama menjadi jumlah memori pada *server*. Berikut adalah contoh dengan memori ditampilkan dalam *kilobyte*:

vmstat -s
132039544 total memory
1218692 used memory
181732 active memory
----output trimmed----

Gambar 6-5 Statistik Memori Virtual

4. Swappiness

Parameter *swappiness* mengendalikan kecenderungan *kernel* untuk memindahkan proses memori fisik dan ke *swap disk*. Karena *disks* jauh lebih lambat daripada *RAM disk*, dapat membuat *response time* lebih lambat untuk sistem dan aplikasi jika proses-proses yang dieksekusi yang terlalu agresif dipindahkan dari memori ke *harddisk* dan sebaliknya. *Swappiness* dapat memiliki nilai diantara 0 dan 100.

- *swappiness*=0 : Menonaktifkan *swappiness* (tidak ada yang dipindahkan dari memori ke *harddisk*)
- *swappiness* = 100 : Memberitahu *kernel* yang agresif *swap* proses memori fisik dan memindahkan mereka untuk *swap cache*.

Pengaturan *default* untuk *swappiness* di Ubuntu adalah 60. Mengurangi nilai *default* dari *swappiness* mungkin akan meningkatkan kinerja keseluruhan untuk instalasi *desktop* Ubuntu biasa. Nilai *swappiness* yang direkomendasikan adalah 10, tetapi jangan ragu untuk mencoba.

Linux *kernel* menyediakan pengaturan yang mengontrol seberapa sering *swap file* digunakan yang disebut *swappiness*. *Swappiness* nol berarti bahwa *disk* akan dihindari kecuali benar-benar diperlukan (Anda kehabisan memori), sementara *swappiness* 100 berarti bahwa program akan bertukar ke *disk* hampir seketika.

Berikut sintaks yang digunakan untuk memeriksa sistem sendiri *swappiness* nilai dengan menjalankan:

```
~$ cat /proc/sys/vm/swappiness
```

Sebagai contoh, RAM yang tersedia ada sebesar 4GB dan akan di *turn-down* menjadi 10 dari 15. *Swap file* akan kemudian hanya digunakan ketika penggunaan RAM saya sekitar 80 atau 90 persen. Untuk mengubah nilai *swappiness* sistem, buka `/etc/sysctl.conf` sebagai *root*. Kemudian, mengubah atau menambahkan baris ini ke *file*:

```
vm.swappiness = 10
```

Reboot agar perubahan diterapkan. Anda juga dapat mengubah nilai sementara sistem Anda masih berjalan dengan:

```
sysctl vm.swappiness=10
```

Anda juga dapat menghapus *swap* Anda dengan menjalankan `swapoff -a` dan kemudian `swapon -a` sebagai *root* bukan *reboot* untuk mencapai efek yang sama.

Untuk menghitung *swap* Anda Formula:

```
free -m (total) / 100 = A
```

```
A * 10
```

```
root@onezero:/home/one# free -m
```

	total	used	free	shared	buffers	cached
Mem:	3950	2262	1687	0	407	952
-/+ buffers/cache:		903	3047			
Swap:	1953	0	1953			

Jadi total adalah $3950 / 100 = 39.5 * 10 = 395$. Jadi apa artinya adalah bahwa ketika 10% (395 MB) RAM yang tersisa maka akan mulai menggunakan *swap*.

6.2 Performansi Memori

6.2.1 Memory Management Techniques

Salah satu metode yang paling sederhana untuk alokasi memori adalah membagi memori menjadi beberapa partisi berukuran tetap. Setiap partisi berisi satu proses. Dalam metode multi-partisi ini, ketika sebuah partisi bebas, proses yang dipilih dari antrian masukan dan dimuat ke partisi tersebut. Ketika proses diakhiri, partisi menjadi tersedia untuk proses lain.

Sistem operasi membuat sebuah *tabel* yang menunjukkan bagian mana dari memori tersedia dan yang digunakan. Akhirnya, ketika proses tiba dan membutuhkan memori, bagian memori yang cukup besar untuk proses ini disediakan oleh sistem operasi. Terdapat beberapa strategi yang digunakan sistem operasi untuk mengalokasikan memori. Strategi *Best-Fit* memilih blok yang terdekat dalam

ukuran dengan permintaan. *First-fit* memilih blok tersedia pertama yang cukup besar. *Worst-fit* memilih blok tersedia terbesar.

6.2.1.1 Algoritma Worst Fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main(){
    int frag[max],b[max],f[max],i,j,nb,nf,temp;
    static int bf[max],ff[max];

    printf("\n\tMemory Management Scheme - First Fit");
    printf("\n\tEnter the number of blocks:");
    scanf("%d",&nb);
    printf("Enter the number of files:");
    scanf("%d",&nf);
    printf("\n\tEnter the size of the blocks:-\n");
    for(i=1;i<=nb;i++)
    {
        printf("Block %d:",i);
        scanf("%d",&b[i]);
    }
    printf("Enter the size of the files :-\n");
    for(i=1;i<=nf;i++)
    {
        printf("File %d:",i);
        scanf("%d",&f[i]);
    }
    for(i=1;i<=nf;i++)
    {
        for(j=1;j<=nb;j++)
        {
            if(bf[j]!=1)
            {
                temp=b[j]-f[i];
                if(temp>=0)
                {
                    ff[i]=j;
                    break;
                }
            }
        }
        frag[i]=temp;
        bf[ff[i]]=1;
    }
    printf("\n\tFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for(i=1;i<=nf;i++)

    printf("\n\t%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

INPUT

Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4

Gambar 6-6 Input Algoritma Worst-Fit

OUTPUT				
File No	File Size	Block No	Block Size	Fragment
1	1	1	5	4
2	4	3	7	3

Gambar 6-7 Output Algoritma Worst-Fit

6.2.1.2 Algoritma Best Fit

```
#include<stdio.h>
#include<conio.h>
#define max 25

void main()
{
    int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
    static int bf[max],ff[max];

    printf("\nEnter the number of blocks:");
    scanf("%d",&nb);

    printf("Enter the number of files:");
    scanf("%d",&nf);

    printf("\nEnter the size of the blocks:-\n");
    for(i=1;i<=nb;i++)
        printf("Block %d:",i);scanf("%d",&b[i]);

    printf("Enter the size of the files :-\n");
    for(i=1;i<=nf;i++)
    {
        printf("File %d:",i);
        scanf("%d",&f[i]);
    }
    for(i=1;i<=nf;i++)
    {
        for(j=1;j<=nb;j++)
        {
            if(bf[j]!=1)
            {
                temp=b[j]-f[i];
                if(temp>=0)
                {
                    if(lowest>temp)
                    {
                        ff[i]=j;
                        lowest=temp;
                    }
                }
            }
        }
    }
}
```

```

    }
    frag[i]=lowest;
    bf[ff[i]]=1;
    lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}

```

INPUT				
Enter the number of blocks: 3				
Enter the number of files: 2				
Enter the size of the blocks:-				
Block 1: 5				
Block 2: 2				
Block 3: 7				
Enter the size of the files:-				
File 1: 1				
File 2: 4				
OUTPUT				
File No	File Size	Block No	Block Size	Fragment
1	1	2	2	1
2	4	1	5	1

Gambar 6-8 Input dan Output Algoritma Best-Fit

6.2.1.3 Algoritma First Fit

```

#include<stdio.h>
#include<conio.h>
#define max 25

void main() {
    int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
    static int bf[max],ff[max];

    printf("\n\tMemory Management Scheme - Worst Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d",&nb);

    printf("Enter the number of files:");
    scanf("%d",&nf);

    printf("\nEnter the size of the blocks:-\n");
    for(i=1;i<=nb;i++) {
        printf("Block %d:",i);
        scanf("%d",&b[i]);
    }
    printf("Enter the size of the files :-\n");
    for(i=1;i<=nf;i++) {
        printf("File %d:",i);
        scanf("%d",&f[i]);
    }
    for(i=1;i<=nf;i++) {
        for(j=1;j<=nb;j++) {
            if(bf[j]!=1) //if bf[j] is not allocated
            {
                temp=b[j]-f[i];

```

```

        if(temp>=0)
            if(highest<temp)
            {
                ff[i]=j;
                highest=temp;
            }
        }
        frag[i]=highest;
        bf[ff[i]]=1;
        highest=0;
    }
    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for(i=1;i<=nf;i++)

    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}

```

INPUT
Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

Gambar 6-9 Input dan Output Algoritma First-Fit

6.2.2 Page Replacement

Terdapat beberapa algoritma *page replacement*. Setiap sistem operasi mungkin memiliki skema penggantian sendiri-sendiri. Algoritma penggantian FIFO (*First In First Out*): *page* yang harus diganti adalah *page* yang pertama kali masuk. Algoritma *Least Recently Used* (LRU): ketika *page* harus diganti, LRU memilih *page* yang belum digunakan untuk periode terpanjang waktu. *Least frequently used* (LFU) halaman-penggantian algoritma memerlukan bahwa *page* dengan jumlah terkecil yang diganti.

6.2.2.1 FIFO (First In First Out) Page Replacement Algorithm

```

#include<stdio.h>
#include<conio.h>
void main() {
    int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

    printf("\n Enter the length of reference string -- ");
    scanf("%d",&n);

    printf("\n Enter the reference string -- ");
    for(i=0;i<n;i++)
        scanf("%d",&rs[i]);

    printf("\n Enter no. of frames -- ");

```

```

scanf("%d",&f);
for(i=0;i<f;i++)
    m[i]=-1;

printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
    for(k=0;k<f;k++) {
        if(m[k]==rs[i])
            break;
    }
    if(k==f) {
        m[count++]=rs[i];
        pf++;
    }
    for(j=0;j<f;j++)
        printf("\t%d",m[j]);
    if(k==f)
        printf("\tPF No. %d",pf);
    printf("\n");
    if(count==f)
        count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
}

```

INPUT

Enter the length of reference string – 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter no. of frames -- 3

OUTPUT

The Page Replacement Process is –

7	-1	-1	PF No. 1
7	0	-1	PF No. 2
7	0	1	PF No. 3
2	0	1	PF No. 4
2	0	1	
2	3	1	PF No. 5
2	3	0	PF No. 6
4	3	0	PF No. 7
4	2	0	PF No. 8
4	2	3	PF No. 9
0	2	3	PF No. 10
0	2	3	
0	2	3	
0	1	3	PF No. 11
0	1	2	PF No. 12
0	1	2	
0	1	2	
7	1	2	PF No. 13
7	0	2	PF No. 14
7	0	1	PF No. 15

The number of Page Faults using FIFO are 15

Gambar 6-10 Input dan Output Algoritma FIFO

6.2.2.2 LRU (Least Recently Used) Page Replacement Algorithm

```
#include<stdio.h>
#include<conio.h>
void main() {
    int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;

    printf("Enter the length of reference string -- ");
    scanf("%d",&n);
    printf("Enter the reference string -- ");
    for(i=0;i<n;i++) {
        scanf("%d",&rs[i]);
        flag[i]=0;
    }
    printf("Enter the number of frames -- ");
    scanf("%d",&f);
    for(i=0;i<f;i++) {
        count[i]=0;
        m[i]=-1;
    }
    printf("\nThe Page Replacement process is -- \n");
    for(i=0;i<n;i++) {
        for(j=0;j<f;j++) {
            if(m[j]==rs[i]) {
                flag[i]=1;
                count[j]=next;
                next++;
            }
        }
        if(flag[i]==0) {
            if(i<f) {
                m[i]=rs[i];
                count[i]=next;
                next++;
            }
            else {
                min=0;
                for(j=1;j<f;j++)
                    if(count[min] > count[j])
                        min=j;
                m[min]=rs[i];
                count[min]=next;
                next++;
            }
            pf++;
        }
        for(j=0;j<f;j++)
            printf("%d\t", m[j]);
        if(flag[i]==0)
            printf("PF No. -- %d" , pf);
        printf("\n");
    }
    printf("\nThe number of page faults using LRU are %d",pf);
}
```

INPUT
Enter the length of reference string -- 20
Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the number of frames -- 3

OUTPUT
The Page Replacement process is --

7	-1	-1	PF No. -- 1
7	0	-1	PF No. -- 2
7	0	1	PF No. -- 3
2	0	1	PF No. -- 4
2	0	1	
2	0	3	PF No. -- 5
2	0	3	
4	0	3	PF No. -- 6
4	0	2	PF No. -- 7
4	3	2	PF No. -- 8
0	3	2	PF No. -- 9
0	3	2	
0	3	2	
1	3	2	PF No. -- 10
1	3	2	
1	0	2	PF No. -- 11
1	0	2	
1	0	7	PF No. -- 12
1	0	7	

1 0 7
The number of page faults using LRU are 12

Gambar 6-11 Input dan Output Algoritma LRU

6.2.2.3 LFU (Least Frequently Used) Page Replacement Algorithm

```
#include<stdio.h>
#include<conio.h>
void main() {
    int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;

    printf("\nEnter number of page references -- ");
    scanf("%d",&m);

    printf("\nEnter the reference string -- ");
    for(i=0;i<m;i++)
        scanf("%d",&rs[i]);
    printf("\nEnter the available no. of frames -- ");
    scanf("%d",&f);
    for(i=0;i<f;i++) {
        cntr[i]=0;
        a[i]=-1;
    }
    printf("\nThe Page Replacement Process is - \n");
    for(i=0;i<m;i++) {
        for(j=0;j<f;j++)
            if(rs[i]==a[j]) {
                cntr[j]++;
                break;
            }
        if(j==f) {
            min = 0;
            for(k=1;k<f;k++)
                if(cntr[k]<cntr[min])
```



```

                                min=k;
                                a[min]=rs[i];
                                cntr[min]=1;
                                pf++;
                                }
                                printf("\n");
                                for(j=0;j<f;j++)
                                    printf("\t%d",a[j]);
                                if(j==f)
                                    printf("\tPF No. %d",pf);
                                }
                                printf("\n\n Total number of page faults -- %d",pf);
                                }

```

INPUT

Enter number of page references -- 10

Enter the reference string --

1 2 3 4 5 2 5 2 5 1 4 3

Enter the available no. of frames -- 3

OUTPUT

The Page Replacement Process is --

1	-1	-1	PF No. 1
1	2	-1	PF No. 2
1	2	3	PF No. 3
4	2	3	PF No. 4
5	2	3	PF No. 5
5	2	3	
5	2	3	
5	2	1	PF No. 6
5	2	4	PF No. 7
5	2	3	PF No. 8

Total number of page faults -- 8

Gambar 6-12 Input dan Output Algoritma LFU

Modul 7 Penjadwalan

Tujuan Praktikum

1. Praktikan dapat melakukan manajemen penjadwalan.
2. Praktikan dapat membuat program penjadwalan sederhana.
3. Praktikan dapat mengukur performansi penjadwalan.

7.1 Manajemen Penjadwalan

7.1.1 Crontab

Crontab adalah salah satu aplikasi untuk penjadwalan pada Linux. *Crontab* memungkinkan *user* melakukan eksekusi aplikasi atau *script* program sesuai dengan waktu yang telah ditentukan. *Crontab* menggunakan daemon Cron, konfigurasi ini terdapat pada masing-masing *home* direktori *user*, disimpan di `/var/spool/cron/crontab`. File *crontab* ada di direktori `/etc/crontab`.

Penjadwalan/kerja otomatis yang dilakukan oleh sistem adalah hal yang umum dilakukan, dikarenakan Linux atau UNIX adalah berupa sistem operasi *multitasking* dan *multiuser*. Cron dijalankan menggunakan perintah *crontab*. *Crontab* akan menyimpan baris demi baris perintah pada direktori `/var/spool/cron/crontab`. Untuk menjalankan *crontab*, harus dipastikan bahwa sistem Linux telah menjalankan daemon bernama *crond* pada saat *booting* berlangsung. Pada dasarnya tak ada perintah bernama *cron*, tetapi hanya menggunakan utilitas *crontab* dan daemon *crond*.

Contoh penggunaan *crontab*:

```
crontab -e
```

Untuk *edit file crontab*, atau membuatnya jika belum ada.

```
crontab -l
```

Menampilkan isi dari *file crontab*

```
crontab -r
```

Menghapus *file crontab*

```
crontab -v
```

Menampilkan kapan terakhir kalinya Anda mengedit *file crontab* tersebut.

Untuk membuat *schedule* silahkan lakukan perintah dibawah ini. *crontab -e*, tekan tombol 'i' untuk melakukan proses *insert scheduler* format nya seperti ini:

* * * * * perintah_untuk_dilaksanakan, setelah selesai tekan esc lalu: wq untuk menyimpannya.

keterangan:

bintang ke 1: Untuk Menit 0 – 59

bintang ke 2: Untuk Jam 0-23

bintang ke 3: Untuk Hari 1-31

bintang ke 4: Untuk Bulan 1-12

bintang ke 5: Untuk Hari (senin-minggu), Senin =1

Contoh:

```
30 18 * * * rm /home/ram/tmp/*
```

artinya "Setiap jam 18.30 rm (*remove*) semua *file* di path `/home/ram/tmp/`"

Contoh lainnya:

30 1 2 1,8,12 * artinya "Setiap jam 01.30 setiap tanggal 2 Januari, 2 Agustus, dan 2 Desember".

5,10 0 12 * 2 artinya "Setiap jam 00.5 dan 00.10 setiap hari Selasa di tanggal 12 setiap bulannya".

0,10,20,30,40,50 * * * * * artinya setiap 10 menit sekali, dan seterusnya.

7.1.2 At

Perintah penjadwalan *at* adalah perintah untuk menjalankan pada waktu khususnya yang biasanya memiliki perizinan untuk berjalan. Perintah *at* bisa menjalankan apapun dari pengingat pesat sederhana hingga *script* yang kompleks. Anda mulai dengan menjalankan perintah *at* pada baris perintah, memberikannya waktu yang dijadwalkan sebagai pilihan. Kemudian menempatkan anda pada permintaan khusus, yang mana bisa diketik pada perintah (atau serangkaian perintah) untuk dijalankan pada waktu penjadwalan. Pada saat anda berhasil, tekan Ctrl + D pada garis baru, dan perintah anda akan ditempatkan pada antrian.

At dan *batch* membaca perintah - perintah dari masukan standar atau *file* yang ditentukan yang akan dieksekusi nantinya menggunakan *sh*.

- *at*: mengeksekusi perintah - perintah pada waktu yang ditentukan.
- *atq*: daftar pekerjaan tertunda pengguna, kecuali pengguna adalah *superuser*; yang mana setiap pekerjaan sudah terdata. Format untuk baris keluaran (satu untuk setiap pekerjaan) adalah : *job, number, date, hour, year, queue*, dan *username*.
- *atrm*: menghapus pekerjaan - pekerjaan, mengidentifikasi berdasarkan jumlah pekerjaan
- *batch*: mengeksekusi perintah pada saat sistem mengeluarkan tingkatan perizinan; dengan kata lain, ketika rata - rata yang dikeluarkan dibawah 1,5 atau nilai yang ditentukan dalam *atd*

At *command* tidak hanya menulis perintah dan menekan *enter*. Terdapat hal yang lebih dari itu, contohnya kita ingin mempunyai *command* *play/usr/share/sounds/KDE_Startup.wav* dieksekusi pada 12:49. Gambarnya akan seperti ini di terminal.

```
$ at 12:49 -m
warning: commands will be executed using (in order) a)$SHELL b)login shell
c)/bin/sh
at>play/usr/share/sounds/KDE_Startup.wav
at><EOT>
job 7 at 2006-05-01 12:49
```

- Di baris pertama dapat dilihat *command* "*at 12:49 -m*", maksudnya dieksekusi pada 12:49 dan "-m" berarti kabarkan saya disaat pekerjaan selesai.
- Di baris kedua, baris ini akan mencetak beberapa info tentang *at command* dan lompat ke baris 3.
- Di baris ketiga, kalian akan melihat "*at>*", di "*at>*" kalian mengetik perintah yang ingin kalian eksekusi dan tekan *enter* lagi.
- Di baris keempat, kalian melihat "*at>*" lagi, pada kali ini tekan Ctrl+D dan akan mencetak <EOT>.
- Kelima ini akan secara otomatis mencetak jumlah pekerjaan dan waktu perintah diberikan akan dieksekusi.

Contoh lainnya, kita mempunyai sebuah *script* *"/usr/local/bin/backup-script"* yang ingin dieksekusi pada 12:32

```
#at 12:32 -m -f/usr/local/bin/backup-script
warning: commands will be executed using (in order) a)$SHELL b)login shell
c)/bin/sh
job 8 at 2006-05-01 12:32
job 7 at 2006-05-01 12:49
```

- Di *line* pertama, dapat dilihat "*at 12:32 -m*" seperti pada contoh pertama, hanya setelahnya muncul "-f" yang berarti "*from file*" dan "*usr/local/bin/backup-script*" yang mana adalah jalur ke *file/script* yang ingin dieksekusi

- Pada *line* kedua dan ketiga dapat dilihat sama dengan baris kedua dan kelima pada contoh pertama
- Ingat jumlah pekerjaan? Itu dapat berguna jika ingin memberhentikan pekerjaan sebelum berjalan. Pertama, kita gunakan keduanya "at -l" or "atq" di daftar pekerjaan yang tertunda:

```
$ at -l
7      2006-05-01 12:49 a bruno
8      2006-05-01 12:32 a root
```

Lalu kita lihat angkat 7 adalah pekerjaan pertama, sekarang kita hilangkan pekerjaan itu dengan:

```
$ atrm 7
```

7.2 Algoritma dan Performansi Penjadwalan

7.2.1 Algoritma FCFS

Pada algoritma penjadwalan FCFS, setiap proses akan dieksekusi berdasarkan waktu kedatangannya.

```
#include<stdio.h>
#include<conio.h>
main()
{
    int bt[20], wt[20], tat[20], i, n;
    float wtavg, tatavg;
    printf("\nEnter the number of processes -- ");
    scanf("%d", &n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter Burst Time for Process %d -- ", i);
        scanf("%d", &bt[i]);
    }
    wt[0] = wtavg = 0;
    tat[0] = tatavg = bt[0];
    for(i=1;i<n;i++)
    {
        wt[i] = wt[i-1] +bt[i-1];
        tat[i] = tat[i-1] +bt[i];
        wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];
    }
    printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
    for(i=0;i<n;i++)
        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time -- %f", wtavg/n);
    printf("\nAverage Turnaround Time -- %f", tatavg/n);
}
```

Input

```
Enter the number of processes -- 3
Enter Burst Time for Process 0 -- 24
Enter Burst Time for Process 1 -- 3
Enter Burst Time for Process 2 -- 3
```

Output

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P0	24	0	24
P1	3	24	27
P2	3	27	30

Average Waiting Time-- 17.000000
Average Turnaround Time -- 27.000000

Gambar 7-1 Input dan Output Algoritma FCFS

7.2.2 Algoritma RR

Pada algoritma penjadwalan *Round-Robin*, memungkinkan setiap proses untuk mendapatkan kesempatan yang sama karena proses bergantian dieksekusi dalam *time slice* yang sama.

```
#include<stdio.h>
main()
{
    int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
    float awt=0,att=0,temp=0;
    printf("Enter the no of processes -- ");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("\nEnter Burst Time for process %d -- ", i+1);
        scanf("%d",&bu[i]);
        ct[i]=bu[i];
    }

    printf("\nEnter the size of time slice -- ");
    scanf("%d",&t);
    max=bu[0];
    for(i=1;i<n;i++)
        if(max<bu[i])
            max=bu[i];
    for(j=0;j<(max/t)+1;j++)
        for(i=0;i<n;i++)
            if(bu[i]!=0)
                if(bu[i]<=t)
                {
                    tat[i]=temp+bu[i];
                    temp=temp+bu[i];
                    bu[i]=0;
                }
                else
                {
                    bu[i]=bu[i]-t;
                    temp=temp+t;
                }

    for(i=0;i<n;i++)
    {
        wa[i]=tat[i]-ct[i];
        att+=tat[i];
    }
}
```

```

        awt+=wa[i];
    }
    printf("\nThe Average Turnaround time is -- %f",att/n);
    printf("\nThe Average Waiting time is -- %f ",awt/n);
    printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
    for(i=0;i<n;i++)
        printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
}

```

Input

Enter the no of processes – 3
 Enter Burst Time for process 1 – 24
 Enter Burst Time for process 2 -- 3
 Enter Burst Time for process 3 -- 3

 Enter the size of time slice – 3

Output

The Average Turnaround time is – 15.666667
 The Average Waiting time is -- 5.666667

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
1	24	6	30
2	3	4	7
3	3	7	10

Gambar 7-2 Input dan Output Algoritma Round Robin

7.2.3 Algoritma Priority

Pada algoritma penjadwalan *priority*, sistem operasi mengatur semua pekerjaan agar sesuai dengan prioritas mereka. Jika terdapat 2 pekerjaan dalam antrian dengan prioritas yang sama, pendekatan FCFS bisa dilakukan.

```

#include<stdio.h>
main()
{
    int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
    float wtavg, tatavg;
    printf("Enter the number of processes --- ");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        p[i] = i;
        printf("Enter the Burst Time & Priority of Process %d --- ",i);
        scanf("%d %d",&bt[i], &pri[i]);
    }
    for(i=0;i<n;i++)
        for(k=i+1;k<n;k++)
            if(pri[i] > pri[k])
            {
                temp=p[i];
                p[i]=p[k];
                p[k]=temp;

                temp=bt[i];
                bt[i]=bt[k];
                bt[k]=temp;
            }
}

```


Modul 8 I/O dan File System

Tujuan Praktikum

1. Praktikan dapat melakukan manajemen I/O dan *file system*.
2. Praktikan dapat mengukur performansi I/O dan *file system*.

8.1 Manajemen I/O dan File System

Dalam sistem komputer, manajemen I/O sangatlah diperlukan. Dikarenakan I/O merupakan sarana *user* untuk bisa berkomunikasi dengan komputer. Contoh dari perangkat I/O adalah seperti *keyboard, mouse, audio controller, video controller, disk drives, networking port*, dll. Hal ini sangat diperlukan agar *user* dapat langsung menggunakan perangkat tersebut tanpa harus bersusah payah. Oleh karena itu, dalam setiap sistem operasi selalu terdapat *I/O manager*.

Fungsi manajemen I/O:

1. Mengirim perintah ke perangkat I/O agar dapat menyediakan layanan.
2. Menangani interupsi perangkat I/O.
3. Menangani kesalahan perangkat I/O.
4. Menyediakan antarmuka ke pemakai.

8.1.1 File Hierarchy Standard Linux

File System Hierarchy Standard (FHS) adalah pedoman untuk direktori standar dalam membuat sebuah distribusi di Linux. FHS dibuat pertama kali di tahun 1993, dengan nama FSSTND berfungsi untuk menyatukan struktur *file* serta direktori Linux. Pada tahun 1994, Linux merilis FHS untuk pertama kalinya. Tujuan FHS adalah untuk interoperabilitas aplikasi, program administrasi sistem, *script*, dan menyatukan dokumentasi sistem.

Tabel 8-1 Hirarki Standar pada Linux

Nama	Keterangan
/	Merupakan direktori akar (<i>root</i>) dari seluruh hirarki
/bin/	Merupakan direktori berisi program - program yang esensial, yang harus tersedia bahkan pada modus <i>single user</i>
/boot/	Merupakan direktori tempat menyimpan <i>file</i> - <i>file</i> yang dibutuhkan oleh <i>boot loader</i>
/dev/	Sebagai kontainer penyimpan <i>device file</i>
/etc/	Sebagai penyimpan konfigurasi sistem
/home/	Direktori untuk menyimpan data, konfigurasi, dan <i>file</i> - <i>file</i> pribadi pengguna sistem
/lib/	Tempat <i>file</i> pustaka yang dibutuhkan oleh program - program yang terdapat pada direktori /bin/ dan /sbin/
/media/	Direktori yang berisi <i>mountpoint</i> dari <i>removable media</i>
/mnt/	<i>Mountpoint</i> sementara

/opt/	Tempat penyimpanan paket aplikasi opsional
/proc/	<i>Filesystem</i> maya yang mendokumentasikan status <i>kernel</i> dan proses sebagai <i>file</i> teks
/root/	Serupa dengan <i>/home/</i> , tetapi khusus untuk <i>user 'root'</i>
/sbin/	Program esensial yang hanya boleh dijalankan oleh <i>superuser (user root)</i>
/tmp/	Direktori penyimpanan sementara
/usr/	Hirarki kedua untuk menyimpan program dan data aplikasi multi pengguna (berisi direktori - direktori seperti pada <i>/</i> , misal <i>/usr/bin/</i> , <i>/usr/sbin/</i> , <i>/usr/lib/</i> , <i>/usr/var/</i> , dan lain - lain)
/var/	Tempat penyimpanan <i>file</i> - <i>file</i> variabel, seperti <i>file log</i> , antrian <i>printer</i> , <i>file email</i> sementara, dan lain - lain

8.1.2. Commands untuk Manajemen I/O dan File System

a. df

Dengan menggunakan perintah **df**, kita dapat menampilkan info *disk* yang tersedia di dalam sistem *file* milik kita. Untuk melihat jumlah *space disk* yang tersedia di dalam semua sistem *file* yang terpasang di komputer Linux anda, ketik **df** tanpa pilihan:

```
$ df
Filesystem    1k-blocks      Used    Available    Use%    Mounted on
/dev/hda3      30645460  2958356    26130408     11%    /
/dev/hda2        46668      8340       35919      19%    /boot
/dev/fd0         1412        13        1327        1%    /mnt/floppy
```

Keluaran memperlihatkan ruang yang tersedia di dalam partisi *harddisk* terpasang di dalam partisi *root* (*/dev/hda1*), partisi */boot* (*/dev/hda2*), dan *floppy disk* yang terpasang pada direktori */mnt/floppy* (*/dev/fd0*). Ruang *disk* ditunjukkan dalam 1K blok. Untuk memproduksi keluaran yang lebih bentuknya manusiawi, gunakan **-h** sebagai berikut.

```
$ df -h
Filesystem      Size    Used Avail Use% Mounted on
/dev/hda3       29G     2.9G   24G   11% /
/dev/hda2       46M     8.2M   25M   19% /boot
/dev/fd0        1.4M     13k   1.2M    1% /mnt/floppy
```

Dengan **df -h**, keluaran muncul lebih bersahabat dengan daftar megabyte atau gigabyte. Pilihan lainnya dengan **df** adalah:

- Hanya mencetak sistem *file* dari pilihan tertentu (ketik **-t**)
- Pengecualian sistem *file* dari pilihan tertentu (ketik **-x**)
- Memasukkan sistem *file* yang tidak mempunyai ruang, seperti */proc* dan */dev/pts* (ketik **-a**)
- Mendaftarkan hanya yang tersedia dan menggunakan inode (ketik **-i**)
- Memunculkan ruang *disk* di ukuran blok tertentu (ketik **--block-size=#**)

b. du

Untuk mengetahui seberapa besar ruang yang terkonsumsi oleh direktori tertentu (dan subdirektornya), Anda bisa menggunakan perintah **du**. Dengan tanpa pilihan, **du** mendata semua direktori dibawah direktori yang terbaru, bersamaan dengan ruang yang terkonsumsi oleh setiap direktori. Pada akhirnya, **du** menampilkan total ruang *disk* yang terpakai dalam struktur direktori tersebut. Perintah **du** adalah cara yang tepat untuk melihat seberapa besar ruang yang terpakai oleh *user* tertentu (misal: `du /home/user1`) atau partisi *file* sistem tertentu (misal: `du /var`). Defaultnya, ruang *disk* akan ditampilkan dalam ukuran 1K blok. Untuk membuat keluaran yang lebih bersahabat (dalam *kilobytes*, *megabytes*, dan *gigabytes*), gunakan pilihan **-h** seperti berikut:

```
/home/jake
/home/jake/httpd/stuff
/home/jake/httpd
/home/jake/uucp/data
/home/jake/uucp
/home/jake
```

Keluaran akan memperlihatkan ruang *disk* yang terpakai dari setiap direktori di bawah direktori *home* pengguna bernama jake (`/home/jake`). Ruang *disk* terkonsumsi akan terlihat dalam *kilobytes* (k) dan *megabytes* (M). Jumlah ruang yang terkonsumsi oleh `/home/jake` akan terlihat di baris terakhir.

c. fdisk

Untuk melihat partisi apa yang saat ini diatur pada *hard disk* Anda, gunakan perintah **fdisk** sebagai berikut:

```
# fdisk -l
Disk /dev/hda: 40.0 GB, 40020664320
255 heads, 63 sectors/track, 4825 cylinders
Units = cylinders of 16065 * 512 bytes = 8225280 bytes

   Device Boot      Start         End      Blocks    Id  System
/dev/hda1 *          1           13         104      b   Win95 FAT32
/dev/hda2            84           89        48195     83   Linux
/dev/hda3           90          522    3478072+     83   Linux
/dev/hda4          523          554     257040      5   Extended
/dev/hda5          523          554     257008+     82   Linux swap
```

Keluaran ini memperlihatkan partisi *disk* untuk komputer telah ter-install Linux dan Microsoft Windows. Anda dapat melihat partisi Linux di `/dev/hda3` yang memiliki ruang paling kosong untuk data. Terdapat partisi Windows (`/dev/hda1`) dan partisi *swap* Linux (`/dev/dha5`). Terdapat juga partisi *boot* kecil (46MB) di `/dev/hda2`. Dalam kasus ini, partisi *root* untuk Linux memiliki 3.3GB dari ruang *disk* dan tinggal di `/dev/hda3`.

d. Mount

Untuk melihat partisi yang sebenarnya digunakan untuk sistem Linux, anda dapat menggunakan perintah **mount** (tanpa pilihan). Perintah **mount** dapat memperlihatkan anda yang mana partisi *disk* yang tersedia sebenarnya terpasang dan di mana terpasangnya.

```
# mount
/dev/hda3 on / type ext3 (rw)
/dev/hda2 on /boot type ext3 (rw)
/dev/hda1 on /mnt/win type vfat (rw)
none on /proc type proc (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/cdrom on /mnt/cdrom type iso9660 (ro,nosuid,nodev)
```

Catatan: Anda mungkin memperhatikan `/proc`, `/dev/pts`, dan entri lainnya tidak berhubungan dengan partisi yang diperlihatkan sebagai *file* sistem. Mereka mewakili jenis sistem *file* yang berbeda (`proc`, dan `devpts` masing - masing).

e. **Umount**

Ketika anda selesai menggunakan sistem *file* sementara, atau anda ingin melepas permanen sebuah sistem *file* sementara, anda dapat menggunakan perintah **umount**. Perintah ini memisahkan sistem *file* dari titik *mount*nya dalam sistem *file*. Untuk menggunakan **umount**, anda dapat memberikan nama direktori atau nama perangkat. Contohnya:

```
# umount /mnt/floppy
```

Ini melepaskan dari perangkat (mungkin `/dev/fd0`) dari titik *mount* `/mnt/floppy`. Anda juga bisa melakukan ini menggunakan bentuk:

```
# umount /dev/fd0
```

Secara umum, lebih baik menggunakan nama direktori dikarenakan **umount** akan gagal jika perangkat dipasang dilebih dari satu lokasi.

f. **mkfs**

Digunakan untuk membuat sebuah sistem *file*. Ini adalah contoh untuk menggunakan **mkfs** untuk membuat sebuah sistem *file* di *floppy disk*:

```
# mkfs -t ext3 /dev/fd0
mke2fs 1.34, (25-Jul-2003)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
184 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
184 inodes per group

Writing inode tables: done

Filesystem too small for a journal
```

```
Writing superblocks and filesystem accounting information: done
```

```
The filesystem will be automatically checked every 32 mounts or  
180 days, whichever comes first. Use tune2fs -c or -i to override.
```

Anda dapat melihat statistik yang dihasilkan dengan pemformatan yang dilakukan oleh perintah **mkfs**. Banyaknya *inode* atau blok yang dibuat adalah keluaran. Serta, jumlah dari blok per grup dan potongan - potongan per grup jugalah keluaran. Anda dapat memasang sistem *file* ini (*mount /mnt/floppy*), mengubahnya ke direktori terbaru (*cd /mnt/floppy*), dan membuat *file - file* di tempat yang diminta.

8.2 Performansi I/O dan File System

Salah satu fungsi dari sistem operasi adalah menggunakan *hardware* secara efisien. Untuk *disk drive*, memenuhi tanggung jawab ini memerlukan waktu akses yang cepat serta *bandwidth disk* yang besar. Baik waktu akses atau *bandwidth* dapat ditingkatkan dengan mengelola urutan di mana I/O meminta untuk dilayani yang mana disebut penjadwalan *disk*.

Bentuk paling sederhana dari penjadwalan *disk* adalah tentu saja algoritma FCFS. Algoritma ini secara intrinsik adil, namun secara umum tidak membuktikan menjadi pelayanan yang tercepat. Pada algoritma SCAN, *disk* dimulai dari ujung dan bergerak menuju ujung yang lain, melayani permintaan saat mencapai setiap silinder, hingga mencapai ujung *disk* yang lain. Di bagian akhir, arah gerakannya dibalik dan terus berlanjut. Bagian kepala terus - menerus memindai bolak - balik di *disk*. C-SCAN adalah variasi dari SCAN didesain untuk menyediakan waktu tunggu yang lebih seragam. Di mana bagian kepala mencapai bagian akhir lainnya, namun, sesegera kembali ke awal dari *disk* tanpa melayani permintaan apapun pada perjalanan kembali.

8.2.1 Program Penjadwalan Disk dengan FCFS

```
#include<stdio.h>
main()
{
    int t[20], n, I, j, tohm[20], tot=0;
    float avhm;
    printf("enter the no.of tracks");
    scanf("%d",&n);
    printf("enter the tracks to be traversed");
    for(i=2;i<n+2;i++)
        scanf("%d",&t*i+);
    for(i=1;i<n+1;i++)
    {
        tohm[i]=t[i+1]-t[i];
        if(tohm[i]<0)
            tohm[i]=tohm[i]*(-1);
    }
    for(i=1;i<n+1;i++)
        tot+=tohm[i];
    avhm=(float)tot/n;
    printf("Tracks traversed\tDifference between tracks\n");
    for(i=1;i<n+1;i++)
        printf("%d\t\t\t%d\n",t*i+,tohm*i+);
    printf("\nAverage header movements:%f",avhm);
}
```

INPUT

Enter no.of tracks:9
Enter track position:55 58 60 70 18 90 150 160 184

OUTPUT

Tracks traversed	Difference between tracks
55	45
58	3
60	2
70	10
18	52
90	72
150	60
160	10
184	24

Average header movements:30.888889

Gambar 8-1 Input dan Output Algoritma FCFS

8.2.2 Program Penjadwalan Disk dengan SCAN

```
#include<stdio.h>
main()
{
    int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;

    printf("enter the no of tracks to be traversed");
    scanf("%d",&n);
    printf("enter the position of head");
    scanf("%d",&h);
    t[0]=0;t[1]=h;
    printf("enter the tracks");
    for(i=2;i<n+2;i++)
        scanf("%d",&t[i]);
    for(i=0;i<n+2;i++)
    {
        for(j=0;j<(n+2)-i-1;j++)
        {
            if(t[j]>t[j+1])
            {
                temp=t[j];
                t[j]=t[j+1];
                t[j+1]=temp;
            }
        }
    }

    for(i=0;i<n+2;i++)
        if(t[i]==h)
            j=i;k=i;

    p=0;
    while(t[j]!=0)
    {
        atr[p]=t[j];
        j--;
        p++;
    }
    atr[p]=t[j];
```

```

    for(p=k+1;p<n+2;p++,k++)
        atr[p]=t[k+1];
    for(j=0;j<n+1;j++)
    {
        if(atr[j]>atr[j+1])
            d[j]=atr[j]-atr[j+1];
        else
            d[j]=atr[j+1]-atr[j];
        sum+=d[j];
    }
    printf("\nAverage header movements:%f", (float)sum/n);
}

```

INPUT

Enter no.of tracks:9
Enter track position:55 58 60 70 18 90 150 160 184

OUTPUT

Tracks traversed	Difference between tracks
150	50
160	10
184	24
90	94
70	20
60	10
58	2
55	3
18	37

Average header movements: 27.77

Gambar 8-2 Input dan Output Algoritma SCAN

8.2.3 Program Penjadwalan Disk dengan C-SCAN

```

#include<stdio.h>
main()
{
    int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
    printf("enter the no of tracks to be traversed");
    scanf("%d",&n);
    printf("enter the position of head");
    scanf("%d",&h);
    t[0]=0;t[1]=h;
    printf("enter total tracks");
    scanf("%d",&tot);
    t[2]=tot-1;
    printf("enter the tracks");
    for(i=3;i<=n+2;i++)
        scanf("%d",&t[i]);
    for(i=0;i<=n+2;i++)
        for(j=0;j<=(n+2)-i-1;j++)
            if(t[j]>t[j+1])
            {
                temp=t[j];
                t[j]=t[j+1];
                t[j+1]=temp;
            }
}

```

```
for(i=0;i<=n+2;i++)
    if(t[i]==h)
        j=i;break;
p=0;
while(t[j]!=tot-1)
{
    atr[p]=t[j];
    j++;
    p++;
}
atr[p]=t[j];
p++;
i=0;
while(p!=(n+3) && t[i]!=t[h])
{
    atr[p]=t[i];
    i++;
    p++;
}
for(j=0;j<n+2;j++)
{
    if(atr[j]>atr[j+1])
        d[j]=atr[j]-atr[j+1];
    else
        d[j]=atr[j+1]-atr[j];
    sum+=d[j];
}
printf("total header movements%d",sum);
printf("avg is %f",(float)sum/n);
}
```

INPUT

Enter the track position : 55 58 60 70 18 90 150 160 184
Enter starting position : 100

OUTPUT

Tracks traversed	Difference Between tracks
150	50
160	10
184	24
18	240

55	37
58	3
60	2
70	10
90	20
Average seek time : 35.7777779	

Gambar 8-3 Input dan Output Algoritma C-SCAN

Modul 9 Keamanan

Tujuan Praktikum

1. Praktikan dapat mengimplementasikan *access control* pada OS.
2. Praktikan dapat mengimplementasikan aspek integritas pada OS.
3. Praktikan dapat mengimplementasikan aspek konfidensialitas pada OS.

9.1 Access Control (Permission)

9.1.1 Permission Dasar pada Linux

Linux adalah OS multi-pengguna yang didasarkan pada konsep Unix kepemilikan *file* dan *permissions* untuk memberikan keamanan pada tingkat sistem *file*.

1. Tentang User

Pada Linux, ada dua jenis pengguna: pengguna sistem dan pengguna biasa. Secara tradisional, pengguna sistem digunakan untuk menjalankan proses non-interaktif atau *background* pada sistem, sementara pengguna biasa digunakan untuk masuk dan menjalankan proses secara interaktif. Cara mudah untuk melihat semua pengguna pada sistem adalah dengan melihat isi *file* `/etc/passwd`. Setiap baris dalam *file* ini berisi informasi tentang satu pengguna, dimulai dengan nama pengguna. Tampilkan isi *file* `passwd` dengan perintah ini: `cat /etc/passwd`

2. Superuser

Selain dua jenis pengguna, ada *superuser*, atau pengguna *root*, yang memiliki kemampuan untuk mengganti kepemilikan *file* dan pembatasan izin. Dalam praktiknya, hal ini berarti bahwa *superuser* memiliki hak untuk mengakses apa pun di *server*-nya sendiri. Pengguna ini digunakan untuk membuat perubahan seluruh sistem, dan harus dijaga keamanannya.

Juga dimungkinkan untuk mengkonfigurasi akun pengguna lain dengan kemampuan untuk melakukan "hak *superuser*". Bahkan, membuat pengguna normal yang memiliki hak *sudo* untuk tugas administrasi sistem dianggap sebagai praktik terbaik.

3. Tentang Grup

Grup adalah kumpulan dari nol atau lebih pengguna. Seorang pengguna termasuk grup *default*, dan juga dapat menjadi anggota dari grup lain di *server*. Cara mudah untuk melihat semua grup dan anggotanya adalah dengan melihat *file* `/etc/group` pada *server*.
`cat /etc/group`

4. Ownership dan Permission

Pada Linux, setiap *file* dimiliki oleh satu pengguna dan satu grup, dan memiliki izin aksesnya sendiri. Mari kita lihat cara melihat kepemilikan dan izin *file*.

Cara paling umum untuk melihat izin *file* adalah menggunakan `ls` dengan opsi daftar panjang, misalnya `ls -l myfile`. Jika Anda ingin melihat izin dari semua *file* di direktori Anda saat ini, jalankan perintah tanpa argumen, seperti ini:

```
ls -l
```

Petunjuk: Jika Anda berada di direktori home kosong, dan Anda belum membuat file untuk dilihat, Anda dapat mengikuti dengan daftar isi direktori `/etc` dengan menjalankan perintah ini:

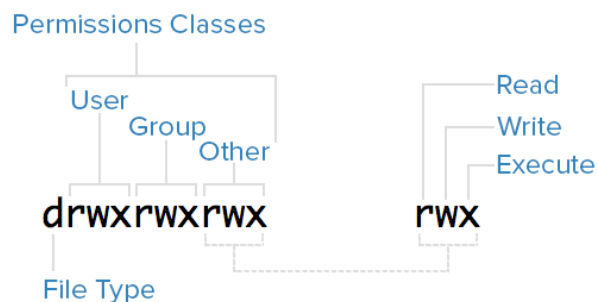
```
ls -l /etc
```

Berikut ini contoh *screenshot* dari tampilan *output*, dengan label setiap kolom *output*:

Mode		Owner	Group	File Size	Last Modified	Filename
drwxrwxrwx	2	sammy	sammy	4096	Nov 10 12:15	everyone_directory
drwxrwx---	2	root	developers	4096	Nov 10 12:15	group_directory
-rw-rw----	1	sammy	sammy	15	Nov 10 17:07	group_modifiable
drwx-----	2	sammy	sammy	4096	Nov 10 12:15	private_directory
-rw-----	1	sammy	sammy	269	Nov 10 16:57	private_file
-rwxr-xr-x	1	sammy	sammy	46357	Nov 10 17:07	public_executable
-rw-rw-rw-	1	sammy	sammy	2697	Nov 10 17:06	public_file
drwxr-xr-x	2	sammy	sammy	4096	Nov 10 16:49	publicly_accessible_directory
-rw-r--r--	1	sammy	sammy	7718	Nov 10 16:58	publicly_readable_file
drwx-----	2	root	root	4096	Nov 10 17:05	root_private_directory

Gambar 9-1 Daftar File

5. Mode



Gambar 9-2 Mode

- **Jenis File:** ada dua jenis *file* dasar: normal dan khusus. Jenis *file* ditunjukkan oleh karakter pertama dari *mode file*. *File* normal ditunjukkan dengan tanda (-). *File* khusus dapat diidentifikasi oleh *file* yang memiliki karakter selain (-). Sebagai contoh, sebuah direktori, yang merupakan jenis *file* khusus yang paling umum, diidentifikasi oleh karakter "d" yang muncul di bidang jenis *file* (seperti gambar diatas).
- **Permission Classes:** kolom *Mode* menunjukkan jenis *file*, diikuti oleh tiga triad. Terdiri dari *User* (Pemilik *file*), *Group* (anggota grup), dan *Other*.
- **Symbolic Permission:** Terdiri dari tiga triad. Membaca ditunjukkan oleh 'r' di posisi pertama. Tulis ditunjukkan oleh 'w' pada posisi kedua dan *execute* yang ditunjukkan oleh 'x' di posisi ketiga.

6. Contoh Mode dan Permission:

- `-rw-----`: *File* yang hanya dapat diakses oleh pemiliknya
- `-rwxr-xr-x`: *File* yang dapat dieksekusi oleh setiap pengguna pada sistem. *File* " *world-Executable*"
- `-rw-rw-rw-`: *File* yang terbuka untuk modifikasi oleh setiap pengguna pada sistem. *File* " *world-Executable*"
- `drwxr-xr-x`: Direktori yang dapat dibaca dan diakses setiap pengguna di sistem
- `drwxrwx---`: Direktori yang dapat dimodifikasi (termasuk kontennya) oleh pemilik dan grupnya
- `drwxr-x---`: Direktori yang dapat diakses oleh kelompoknya

9.1.2 Cara Mengubah *Permission* pada Linux

Setiap kategori perizinan (pemilik, pemilik grup, dan lainnya) dapat diberikan izin yang memungkinkan atau membatasi kemampuan mereka untuk membaca, menulis, atau mengeksekusi *file*. Untuk *file* biasa, izin membaca diperlukan untuk membaca isi *file*, izin menulis diperlukan untuk

memodifikasinya, dan izin mengeksekusi diperlukan untuk menjalankan *file* sebagai skrip atau aplikasi. Untuk direktori, izin membaca diperlukan untuk **ls** (daftar) isi direktori, izin menulis diperlukan untuk mengubah isi direktori, dan menjalankan izin memungkinkan pengguna untuk mengubah direktori ke direktori. Linux mewakili jenis perizinan menggunakan dua notasi simbolis terpisah: alfabetik dan oktal.

Notasi Alfabet

Notasi alfabetik mudah dimengerti dan digunakan oleh beberapa program umum untuk mewakili izin. Setiap izin diwakili oleh satu huruf:

- r = izin membaca
- w = izin menulis
- x = izin mengeksekusi

Penting untuk diingat bahwa izin menggunakan notasi abjad selalu ditentukan dalam urutan ini. Jika hak istimewa tertentu diberikan, itu diwakili oleh huruf yang sesuai. Jika akses dibatasi, itu diwakili oleh tanda hubung (-). Izin diberikan untuk pemilik *file* terlebih dahulu, diikuti oleh pemilik grup, dan akhirnya untuk pengguna lain. Hal ini memberi kita tiga kelompok dari tiga nilai. Perintah **ls** menggunakan notasi abjad ketika dipanggil dengan opsi format panjangnya:

```
ls-l
drwxr-xr-x 3 root root 4096 Apr 26 2012 acpi
-rw-r - r-- 1 root root 2981 Apr 26 2012 adduser.conf
drwxr-xr-x 2 root root 4096 Jul 5 20:53 alternatif
-rw-r - r-- 1 root root 395 Jun 20 2010 anacrontab
drwxr-xr-x 3 root root 4096 Apr 26 2012 apm
drwxr-xr-x 3 root root 4096 Apr 26 2012 apparmor
drwxr-xr-x 5 root root 4096 Jul 5 20:52 apparmor.d
drwxr-xr-x 6 root root 4096 26 Apr 2012 apt
```

Kolom pertama dalam *output* dari perintah ini menunjukkan izin dari *file*. Sepuluh karakter mewakili izin ini. Karakter pertama sebenarnya bukan nilai perizinan dan tetapi menandakan jenis *file* (- untuk *file* biasa, d untuk direktori, dll). Sembilan karakter berikutnya mewakili izin yang kita *diskusikan* di atas. Tiga grup yang mewakili: pemilik, pemilik grup, dan izin lainnya, masing-masing dengan nilai yang menunjukkan membaca, menulis, dan mengeksekusi izin. Dalam contoh di atas, pemilik direktori "acpi" (yaitu: *user root*) dapat membaca, menulis, dan mengeksekusi *file*. Pemilik grup dan pengguna lain dapat membaca dan mengeksekusi *file*. *File "anacrontab"* memungkinkan pemilik *file* untuk membaca dan memodifikasi, tetapi anggota grup dan pengguna lain hanya memiliki izin untuk membaca.

Notasi Oktal

Cara yang lebih ringkas, tetapi sedikit kurang intuitif untuk merepresentasikan perizinan adalah dengan notasi oktal. Dengan menggunakan metode ini, setiap kategori perizinan (pemilik, pemilik grup, dan lainnya) diwakili oleh angka antara 0 dan 7.

Setiap jenis izin dengan nilai numerik:

- 4= izin membaca
- 2= izin menulis
- 1= izin mengeksekusi

Ini akan menjadi angka antara 0 dan 7 (0 mewakili tidak ada izin dan 7 mewakili izin baca, tulis, dan eksekusi penuh) untuk setiap kategori. Misalnya, jika pemilik *file* memiliki izin membaca dan menulis, ini akan direpresentasikan sebagai 6 di kolom pemilik *file*. Jika pemilik grup hanya membutuhkan izin

baca, maka 4 dapat digunakan untuk mewakili izin mereka. Serupa dengan notasi abjad, notasi oktal dapat menyertakan karakter utama opsional yang menentukan jenis *file*. Ini diikuti oleh izin pemilik, izin pemilik grup, dan izin lainnya. Program penting yang bermanfaat dari menggunakan notasi oktal adalah perintah **chmod**.

Menggunakan Perintah Chmod

Cara paling populer untuk mengubah izin *file* adalah dengan menggunakan notasi oktal dengan perintah **chmod**:

```
cd
touch testfile
```

Kemudian lihat izin yang diberikan ke *file* ini setelah pembuatan:

```
ls -l testfile
Output: -rw-rw-r-- 1 demouser demouser 0 Jul 10 17:23 testfile
```

Jika diinterpretasikan, dapat dilihat bahwa pemilik *file* dan pemilik grup *file* memiliki hak baca dan tulis, dan pengguna lain memiliki kemampuan membaca. Jika kita mengubah itu menjadi notasi oktal, pemilik dan pemilik grup akan memiliki nilai izin 6 (4 untuk dibaca, ditambah 2 untuk menulis) dan kategori lainnya akan memiliki 4 (untuk dibaca). Izin penuh akan diwakili oleh triplet 664. Selanjutnya *file* ini berisi skrip *bash* diibaratkan *file* yang ingin kami eksekusi, sebagai pemilik. Dan tidak ingin orang lain memodifikasi *file*, termasuk pemilik grup, dan tidak ingin orang lain tidak berada di grup untuk dapat membaca *file* sama sekali. Sehingga dapat diubah izinnya menggunakan **chmod**:

```
chmod 740 testfile
ls -l testfile
Output: -rwxr----- 1 demouser demouser 0 Jul 10 17:23 testfile
```

Pengaturan Izin Default dengan Umask

Perintah **umask** mendefinisikan izin *default* untuk *file* yang baru dibuat berdasarkan pada "dasar" perizinan yang ditetapkan untuk *file* dan direktori. *File* memiliki seperangkat hak akses dasar 666, atau akses baca dan tulis lengkap untuk semua pengguna. Jalankan izin tidak ditetapkan secara *default* karena sebagian besar *file* tidak dibuat untuk dieksekusi (menetapkan izin yang dapat dieksekusi juga membuka beberapa masalah keamanan). Direktori memiliki izin dasar mengatur 777, atau membaca, menulis, dan mengeksekusi izin untuk semua pengguna. **Umask** beroperasi dengan menerapkan "topeng" yang subtraktif ke izin dasar yang ditunjukkan di atas. Jika ingin pemilik dan anggota grup pemilik dapat menulis ke direktori yang baru dibuat, tetapi tidak dengan pengguna lain, dapat menetapkan izin ke 775. Sehingga ini membutuhkan tiga digit angka yang akan menyatakan perbedaan antara izin dasar dan izin yang diinginkan. Angka itu adalah 002.

```
777
- 775
-----
002
```

Jumlah yang dihasilkan ini adalah nilai **umask** yang ingin diterapkan. Ini adalah nilai **umask default** untuk banyak sistem, seperti yang kita lihat ketika membuat *file* dengan perintah **touch** sebelumnya. Mari kita coba lagi:

```
touch test2
ls -l test2
Output: -rw-rw-r-- 1 demouser demouser 0 Jul 10 18:30 test2
```

Jika ingin memberikan izin secara penuh untuk setiap *file* dan direktori yang dibuatnya:

```
umask 077
touch restricted
ls -l restricted
Output: -rw----- 1 demouser demouser 0 Jul 10 18:33 restricted
```

9.2 Integrity (SHA-256)

9.2.1 Dasar Integritas dan Hash

Terdapat beberapa jenis *Secure Hash* algoritma SHA yaitu: SHA1, SHA256 dan SHA512. Secara teknis SHA256 dan SHA512 menggunakan algoritma yang sama, tetapi mempunyai *block* data berbeda (SHA256 menggunakan blok 32 bit dan SHA512 64-bit blok).

Contoh penggunaan sha mereka seperti ini:

```
sha1sum Fedora-19-i386-netinst.iso
```

Output akan terlihat seperti ini:

```
b24e9b7bd49168839fd056bbd0ac8f2aec6b68b9  Fedora-19-i386-netinst.iso
```

Sha256 *hash* yang dihasilkan dengan cara yang sama:

```
sha256 Fedora-19-i386-netinst.iso
```

Dan *output* mirip, kecuali catatan bahwa *hash* lebih panjang:

```
2b16f5826a810bb8c17aa2c2e0f70c4895ee4b89f7d59bb8bf39b07600d6357c  Fedora-19-i386-netinst.iso
```

Dan demikian juga untuk SHA512:

```
sha512sum Fedora-19-i386-netinst.iso
```

Hasil *hash*:

```
9eb35d03cc289aa5d5a29cfc9080c3152a3da1b91a2b12d352b16a3d817a7479b9d1be3c7ec
f011abf6a01f3122c66892f96a2c213756df786117412d8df99b3
Fedora-19-i386-netinst.iso
```

Perintah `sha` dapat digunakan untuk memeriksa integritas *file*. Seseorang akan mem-*publish file* beserta *hash*-nya, jika antara *hash* yang dilakukan *user* dan *hash* yang diberikan tidak sesuai maka telah terjadi perubahan pada *file* tersebut.

Sebagai contoh, pada saat men-*download file* Fedora-19-i386-netinst.iso terdapat juga *file* Fedora-19-i386-CHECKSUM yang berisi *hash* untuk *file* Fedora-19-i386-netinst.iso. Berikut adalah isi *file* Fedora-19-i386-CHECKSUM.

```
2b16f5826a810bb8c17aa2c2e0f70c4895ee4b89f7d59bb8bf39b07600d6357c  Fedora-19-i386-netinst.iso
```

Untuk memeriksa *file* Fedora-19-i386-netinst.iso digunakan parameter `-c` seperti ini:

```
sha256sum -c Fedora-19-i386-CHECKSUM
```

Jika *hash* cocok, maka *output* akan terlihat seperti ini:

```
Fedora-19-i386-netinst.iso: OK
```

Artinya tidak ada perubahan dalam *file*.

Jika ada kesalahan dalam *file* yang didownload, *output* akan menjadi:

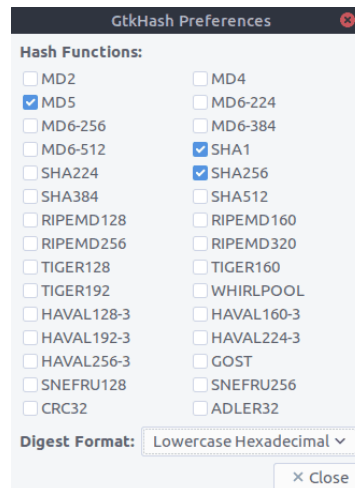
```
Fedora-19-i386-netinst.iso: FAILED
sha256sum: WARNING: 1 computed checksum did NOT match
```

9.2.2 Memeriksa Integritas dengan Tool

Hash adalah seperti sidik jari *digital file*. Ada berbagai algoritma untuk menghasilkan *hash*. Algoritma *hash* yang paling populer adalah:

- Aman Algoritma *Hash* dan varian (SHA-1, SHA-2 dll) dan
- MD5 Algoritma

GtkHash adalah alat yang bagus untuk menghasilkan dan memverifikasi *checksum*



Gambar 9-3 GtkHash Didukung Algoritma Checksum

Untuk menginstal *GtkHash* pada sistem Ubuntu Anda, jalankan perintah berikut:

```
sudo apt install gtkhash
```

Untuk memilih algoritma menggunakan:

- Pergi ke **Edit** > opsi menu **preferensi** .
- Pilih yang Anda ingin gunakan.
- Tekan tombol **tutup** .
- Secara *default*-MD5, SHA-1 dan SHA256 dipilih.

Menggunakan GtkHash

- Pilih *file* yang Anda ingin memeriksa
- Mendapatkan nilai *hash* dari situs dan meletakkannya di kotak **cek**.
- Klik tombol **Hash**.
- Ini akan menghasilkan nilai *checksum* dengan algoritma yang Anda pilih.
- Jika salah satu dari mereka cocok dengan **memeriksa** kotak, ia akan menampilkan tanda centang kecil di samping itu.

Setiap distribusi Linux dilengkapi dengan *tool* untuk berbagai algoritma *hashing*. Anda dapat membuat dan memverifikasi *hash*. Alat – alat *command-line checksum* yang berikut:

- Md5 *checksum* alat disebut: **md5sum**
- SHA-1 *checksum* alat disebut: **sha1sum**
- Alat *checksum* SHA-256 disebut: **sha256sum**

Ada beberapa lebih tersedia, misalnya: *sha224sum*, *sha384sum* dll. Semua dari mereka menggunakan format perintah yang serupa. Mari kita lihat contoh penggunaan *sha256sum*. Kami akan menggunakan *file* gambar "*ubuntu-mate-16,10-desktop-amd64.iso*" sama seperti yang biasa kami sebelum.

Menghasilkan dan memverifikasi *Checksum* SHA256 dengan *sha256sum*:

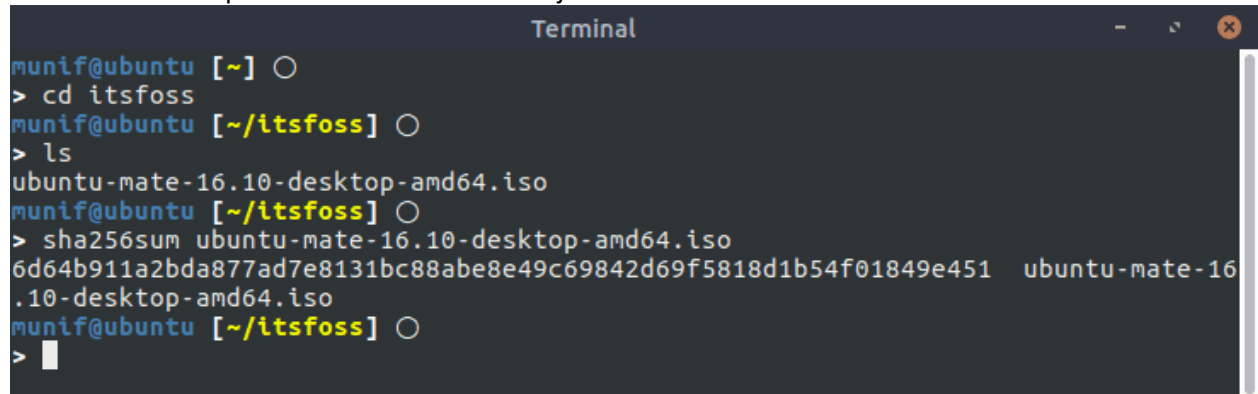
Pertama pergi ke direktori di mana gambar *.iso* disimpan:

```
cd ~/itsfoss
```

Sekarang, untuk menghasilkan SHA256 *checksum*, masukkan perintah berikut:

```
sha256sum ubuntu-mate-16.10-desktop-amd64.iso
```

Anda akan mendapatkan SHA256 hash dalam jendela terminal.



```
Terminal
munif@ubuntu [~] ○
> cd itsfoss
munif@ubuntu [~/itsfoss] ○
> ls
ubuntu-mate-16.10-desktop-amd64.iso
munif@ubuntu [~/itsfoss] ○
> sha256sum ubuntu-mate-16.10-desktop-amd64.iso
6d64b911a2bda877ad7e8131bc88abe8e49c69842d69f5818d1b54f01849e451  ubuntu-mate-16
.10-desktop-amd64.iso
munif@ubuntu [~/itsfoss] ○
>
```

Gambar 9-4 Menghasilkan SHA256 Checksum untuk UbuntuMATE iso

Jika *hash* yang dihasilkan sesuai dengan yang diberikan pada halaman *download* UbuntuMATE, yang akan berarti data tidak berubah ketika Anda men-*download file* atau menempatkan sebaliknya, *file download* Anda tidak rusak.

9.3 Confidentiality (Enkripsi)

Enkripsi adalah penyandian, pengkodean atau pengacakan sebuah teks atau *file* menjadi bentuk lain menggunakan algoritma tertentu agar tidak dapat dimengerti oleh siapapun secara langsung. Untuk membaca *file* yang di enkripsi maka perlu melakukan proses dekripsi *file* yaitu mengubah atau mengembalikan pesan atau *file* yang dienkripsi menjadi bentuk semula sehingga dapat dibaca dan diketahui format atau bentuknya.

9.3.1 Enkripsi *Disk/File System* (ENCFS)

EncFS menyediakan *filesystem* terenkripsi di ruang pengguna (*user space*). EncFS berjalan tanpa izin khusus apapun. Encfs adalah perangkat lunak *open source*, berlisensi di bawah GPL. Ikuti contoh di bawah.

membuat direktori induk untuk enkripsi direktori

```
lec:/home/bob>mkdir Encrypt
```

menghapus grup dan izin lainnya

```
lec:/home/bob>chmod 700 Encrypt
```

membuat terenkripsi (.crypt) dan didekripsi direktori (crypt)

```
lec:/home/bob>mkdir Encrypt/.crypt  
lec:/home/bob>mkdir Encrypt/crypt
```

membaca dan mengeksekusi pada mengenkripsi mendekripsi direktori

```
lec:/home/bob>chmod 755 Encrypt/.crypt Encrypt/crypt
```

Periksa hasil

```
lec:/home/bob>ls -al Encrypt  
total 36  
drwx----- 4 bob bob 4096 Feb 23 13:18 .  
drwxr-xr-x 45 bob bin 24576 Feb 23 13:17 ..  
drwxr-xr-x 2 bob bob 4096 Feb 23 13:18 crypt  
drwxr-xr-x 2 bob bob 4096 Feb 23 13:18 .crypt
```

Buat struktur dienkrpsi

```
lec:/home/bob/Encrypt>encfs ~/Encrypt/.crypt ~/Encrypt/crypt  
Creating new encrypted volume.  
Please choose from one of the following options:  
enter "x" for expert configuration mode,  
enter "p" for pre-configured paranoia mode,  
anything else, or an empty line will select standard mode.  
?>  
  
Standard configuration selected.  
  
Configuration finished. The filesystem to be created has  
the following properties:  
Filesystem cipher: "ssl/blowfish", version 2:1:1  
Filename encoding: "nameio/block", version 3:0:1  
Key Size: 160 bits  
Block Size: 512 bytes  
Each file contains 8 byte header with unique IV data.  
Filenames encoded using IV chaining mode.  
  
Now you will need to enter a password for your filesystem.  
You will need to remember this password, as there is absolutely  
no recovery mechanism. However, the password can be changed  
later using encfsctl.  
  
New Encfs Password:  
Verify Encfs Password:
```

(.crypt adalah direktori dienkrpsi; crypt adalah versi dekripsi dari .crypt)

Sekarang pindahkan *file* ke dalam direktori dekripsi:

```
lec:/home/bob/Encrypt>mv ~/CS750.xls crypt  
lec:/home/bob/Encrypt>mv ~/secret.stuff crypt
```

Berikut adalah *file* kami pindah (didekripsi)

```
lec:/home/bob/Encrypt>ls -al crypt  
total 36  
drwx----- 2 bob bob 4096 Feb 23 09:13 .  
drwxr-xr-x 5 bob wheel 4096 Feb 23 09:12 ..  
-rw----- 1 bob bob 1589 Jan 9 08:25 CS750.xls  
-rw----- 1 bob bob 1325 Jan 14 13:42 secret.stuff
```


Versi dienkripsi

```
lec:/home/bob/Encrypt>ls -al .crypt
total 40
drwx----- 2 bob bob 4096 Feb 23 09:13 .
drwxr-xr-x 5 bob wheel 4096 Feb 23 09:12 ..
-rw----- 1 bob bob 1597 Jan 9 08:25 dQxWUeTso7NiojItcTHbmdy2
-rw----- 1 bob bob 1333 Jan 14 13:42 u5gpyk3WhD8DHhylPl-ntd9X
-rw----- 1 bob bob 224 Feb 23 09:12 .encfs5
```

Ketika selesai, *umount* dekripsi direktori:

```
lec:/home/bob/Encrypt>fusermount -u ~/Encrypt/crypt
```

Dicatat bahwa direktori **crypt** sekarang menunjukkan kosong

```
lec:/home/bob/Encrypt>ls -al crypt
total 8
drwx----- 2 bob bob 4096 Feb 23 09:13 .
drwxr-xr-x 5 bob wheel 4096 Feb 23 09:12 ..
```

Perhatikan bahwa *file* yang dienkripsi masih menunjukkan di **.crypt**

```
lec:/home/bob/Encrypt>ls -al .crypt
total 40
drwx----- 2 bob bob 4096 Feb 23 09:13 .
drwxr-xr-x 5 bob wheel 4096 Feb 23 09:12 ..
-rw----- 1 bob bob 1597 Jan 9 08:25 dQxWUeTso7NiojItcTHbmdy2
-rw----- 1 bob bob 1333 Jan 14 13:42 u5gpyk3WhD8DHhylPl-ntd9X
-rw----- 1 bob bob 224 Feb 23 09:12 .encfs5
```

Untuk kembali *me-mount* direktori dekripsi:

```
lec:/home/bob/Encrypt>
```

```
encfs /home/bob/Encrypt/.crypt /home/bob/Encrypt/crypt
EncFS Password:
```

Melihat *file* dalam direktori dekripsi **crypt**:

```
lec:/home/bob/Encrypt>ls -al crypt
total 36
drwx----- 2 bob bob 4096 Feb 23 09:13 .
drwxr-xr-x 5 bob wheel 4096 Feb 23 09:12 ..
-rw----- 1 bob bob 1589 Jan 9 08:25 CS750.xls
-rw----- 1 bob bob 1325 Jan 14 13:42 secret.stuff
```

9.4 GPG

9.4.1 Install GPG

GPG (GNU *Privacy Guard*) adalah adalah sebuah implementasi kriptografi *public key*. Hal ini memungkinkan untuk transmisi aman informasi antara pihak dan dapat digunakan untuk memverifikasi bahwa asal-usul pesan asli. Cara *install* GPG dapat dilakukan dengan sintaks berikut:

```
sudo apt-get install gnupg
```

Untuk mulai menggunakan GPG untuk mengenkripsi komunikasi Anda, Anda perlu membuat sepasang kunci (kunci *public* dan kunci *private*). Kunci *public* akan Anda bagikan kepada teman-teman Anda atau ditaruh di *server*. Sedangkan kunci *private* harus Anda simpan di komputer Anda dan tidak boleh keluar dari komputer tersebut atau bisa diakses oleh orang lain. Anda dapat melakukan ini dengan mengeluarkan perintah berikut:

```
gpg --gen-key
```

Ini akan membawa Anda melalui beberapa pertanyaan yang akan mengkonfigurasi kunci:

- *Please select what kind of key you want: **(1) RSA and RSA (default)***
- *What keysize do you want? **4096***
- *Key is valid for? **1y** (expires after 1 year. If you are just testing, you may want to create a short-lived key the first time by using a number like "3" instead.)*
- *Is this correct? **y***
- *Real name: **your real name here***
- *Email address: **your_email@address.com***
- *Comment: **Optional comment that will be visible in your signature***
- *Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? **O***
- *Enter passphrase: Enter **a secure passphrase here (upper & lower case, digits, symbols)***

Pilihlah *passphrase* yang panjang. Semakin panjang semakin bagus.

9.4.2 Import Public Key Orang Lain

GPG tidak akan berguna jika tidak dapat menerima *public key* dari orang yang kita kehendaki. Cara untuk meng-*import public key* dari orang lain yaitu dengan:

```
gpg --import nama_public_key_file
```

Kita mendapatkan *file* tersebut secara langsung dari teman kita atau men-*download file public key* dari *server* yang tersedia. *Server* ini menyimpan *public key* – *public key* dari orang – orang di seluruh dunia. Salah satu *server* ini adalah: <https://pgp.mit.edu/>

9.4.3 Membuat Public Key Kita Tersedia

Gunakan perintah ini untuk mem-*publish public key* kita:

```
gpg -output ~/mypgp.key --armor --export your_email@address.com
```

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.11 (GNU/Linux)

mQINBFJPCuABEACiog/sInjg002SqgmG1T8n9FroSTdN74uGsRMHHAOuAmGLsTse
90xeLQpN+r75Ko39RVE88dRcW710fPY0+fjSXBKhpN+raRMUKJp4AX9BJd00YA/4
EpD+8cDK4DuLlLdn1x0q41VUsznXrnMpQedRmAL9f9bL6pbLTJhaKeorTokTvdn6
5VT3pb2o+jr6NETaUxd99ZG/osPar9tNthVLIIzG1nDabcTFbMB+w7w0JuhXyTLQ
JBU9xmavTM71PfV6Pkh4j1pfWImXc1D8dS+jcvKeXInBfm2XZsfOCesk12YnK3Nc
u1Xe1lxzSt7Cegum4S/YuxmYoh462oGZ7FA4Cr2lvAPVpO9zmgQ8JITXiYg2wB3
. . .
```

Gambar 9-5 Public key

Anda dapat mengirimkan *file* ini ke orang lain melalui media pengiriman yang sesuai dan Anda kehendaki.

9.4.4 Enkripsi dengan PGP

Anda dapat mengenkripsi pesan menggunakan "--encrypt" untuk GPG. Sintaks dasar adalah:

```
gpg --encrypt --sign --armor -r person@email.com name_of_file
```

- encrypt : Ini mengenkripsi pesan yang menggunakan kunci publik penerima.
- sign : beri *signature* dengan kunci pribadi Anda sendiri untuk menjamin bahwa pesan itu berasal dari Anda.
- armor : *output* pesan dalam format teks daripada mentah *byte*. Nama *file* akan sama sebagai nama *file input*, tetapi dengan sebuah ekstensi .asc .
- r : *user* penerima pesan. Penerima kedua dengan alamat *email* Anda sendiri jika Anda ingin untuk dapat membaca pesan terenkripsi tersebut. Hal ini karena pesan akan dienkripsi dengan kunci publik penerima, dan hanya akan dapat didekripsi dengan kunci *private* penerima. Jadi jika pesan hanya dienkripsi dengan kunci publik dari pihak lain, Anda tidak akan dapat melihat pesan lagi, kecuali jika Anda entah bagaimana memperoleh kunci pribadi pihak lain tersebut.

9.4.5 Dekripsi dengan PGP

Untuk dekripsi gunakan perintah:

```
gpg file_name.asc
```



School of Computing
Telkom University



informatics lab

Modul 10 Scripting

Tujuan Praktikum

1. Praktikan dapat melakukan otomasi dalam OS.
2. Praktikan dapat melakukan administrasi OS.

10.1 Shell

10.1.1 Bash

Bash (Bourne-Again Shell) adalah unix *shell* (*command line interpreter/command line user interface*) yang ditulis oleh Brian Fox sebagai pengganti dari Bourne *shell*. Bash adalah *command processor* yang biasanya berjalan *mode text* dimana *user* mengetikkan perintah di layar untuk kemudian dieksekusi oleh komputer. Bash dapat juga membaca dan mengeksekusi perintah dari *file* yang disebut *bash script*.

```
kar@karswain: /usr/portage/app-shells/bash $ grep -i /dev/sda /etc/fstab | cut --fields=3
/dev/sda1      /boot
/dev/sda2      none
/dev/sda3      /
kar@karswain: /usr/portage/app-shells/bash $ date
Sat Aug  8 02:42:24 MSD 2009
kar@karswain: /usr/portage/app-shells/bash $ lsmod
Module                  Size  Used by
rmdis_wlan              23424  0
rmdis_host             8696   1 rmdis_wlan
cdc_ether               5672   1 rmdis_host
usbnet                 18688   3 rmdis_wlan,rmdis_host,cdc_ether
parport_pc             38424   0
fglrx                 2388128  20
parport               39648   1 parport_pc
ltdc_udt               12272   0
i2c_i801               9388   0
kar@karswain: /usr/portage/app-shells/bash $
```

Gambar 10-1 Contoh Sesi Bash

Bash script adalah *plain text* yang berisi urutan perintah-perintah. Dari pada menulis satu per satu perintah ke dalam terminal, dengan adanya *script*, perintah akan dieksekusi sesuai dengan urutan yang ada dalam *script*. Pada umumnya *bash script* diberi ekstensi *.sh* (walaupun bukan sebuah keharusan).

Menjalankan *bash script* sangat mudah yaitu hanya dengan mengeksekusi *script* tersebut. Jangan lupa untuk mengubah *permission* dari *script* jika tidak bisa dijalankan (dengan memberikan perintah `chmod +x myscript.sh` atau `chmod 755 myscript.sh`).

```
$ ./myscript.sh
Hello World!
```

10.1.2 Struktur Bash

Berikut adalah contoh sederhana dari *bash script*:

```
myscript.sh
1. #!/bin/bash
2. # A sample Bash script
3.
4. echo Hello World!
```

Gambar 10-2 myscript.sh

Mari kita analisis baris per baris:

- Baris 1: disebut sebagai **shebang**. Karakter **#!** Dikenal sebagai **shebang** diikuti dengan *path* ke interpreter (dalam hal ini adalah *bash*). Letak program *bash* ada di */bin/bash*. Format sangat penting. **Shebang** harus berada pada baris pertama (jika diletakkan pada baris ke dua tidak akan berjalan). Tidak boleh ada spasi antara **#** dan **!** dan path ke interpreter.
- Baris 2: contoh komentar, apapun perintah setelah **#** tidak akan dieksekusi.
- Baris 3: *spacing* agar mempermudah pembacaan.
- Baris 4: perintah yang akan dieksekusi. Pada contoh ini perintah *echo* (menampilkan pesan ke terminal) akan dieksekusi.

10.2 Scripting Dasar

10.2.1 Variabel

Variabel adalah tempat menyimpan informasi sementara. Terdapat 2 hal yang dapat dilakukan pada variabel yaitu *setting* dan *getting* variabel.

Setting Variabel

Pola dasar setting variabel adalah `nama_variabel=value`. Perhatikan bahwa tidak ada spasi setelah tanda `=`.

- *Single quote* akan memperlakukan setiap karakter secara literal (sesuai dengan yang ditulis).
- *Double quote* memungkinkan adanya substitusi variabel

```
user@bash: myvar='Hello World'
user@bash: echo $myvar
Hello World
user@bash: newvar="More $myvar"
user@bash: echo $newvar
More Hello World
user@bash: newvar1='More $myvar'
user@bash: echo $newvar1
More $myvar
```

Getting Variabel

Untuk membaca variabel gunakan tanda **\$** pada variabel yang ingin diakses.

```
#!/bin/bash
myvar=hello
var1=world
echo $myvar $var1
```

Command Substitution

Command substitution memungkinkan kita untuk menulis suatu perintah dan menyimpan perintah tersebut ke dalam variabel. Untuk melakukan hal tersebut, perintah harus berada dalam **()** dan diawali dengan tanda **\$**.

```
user@bash: myvar=$(ls /etc)
user@bash: echo Terdapat file $myvar di folder /etc
```

10.2.1 Command Line Argument

Command line argument adalah argumen yang diberikan pada perintah/*script*.

Contoh: `myscript.sh /home/data.zip /tmp`

Pada contoh tersebut terdapat 2 *command line argument* yaitu: */home/data.zip* dan */tmp*. */home/data.zip* disimpan dalam variabel **\$1** sedangkan */tmp* disimpan dalam variabel **\$2**.

```
user@bash: myscript.sh /home/data.zip /tmp
user@bash: echo $1
/home/data.zip
user@bash: echo $2
/tmp
user@bash: cp $1 $2
```

variabel spesial lainnya:

- **\$0** = *nama script*
- **\$1-\$9** = argumen 1 s.d 9
- **\$\$** = *pid script*
- **\$USER** = *nama user yang menjalankan script*
- **\$RANDOM** = *angka random*

10.2.2 Input

Untuk meminta *input* dari *user* dapat digunakan perintah **read**.

```
#!/bin/bash
echo Hello, Who are you?
read varname
echo Nice to meet you $varname

echo What cars do you like?
read car1 car2 car3

echo your first car: $car1
echo your second car: $car2
echo your third car: $car3
```

10.2.3 Aritmatik

Bash mempunyai fungsi *built-in* untuk melakukan aritmatik. Terdapat beberapa teknik untuk melakukan aritmatik yaitu: **let** dan **expr**.

Let

Format dasarnya perintah adalah **let** <ekpresi aritmatik>.

Bagian pertama biasanya adalah variabel dimana hasil akan disimpan. Berikut adalah contoh menggunakan aritmatik:

```
#!/bin/bash
let a=20+22
echo $a
let "a = 5 + 4"
echo $a
let a++
echo $a
let "a = $1 + 10"
echo $a # 10 + argument pertama
```

Hasil, jika dieksekusi contoh_let.sh 13

```
$
42
9
10
23
```

Operator

+, -, *, / = tambah, kurang, kali, bagi

Var++ = *increment*

Var-- = *decrement*

% = modulo

Expr

Expr dapat digunakan substitusi dan tidak perlu menggunakan *quote*. Perlu diperhatikan: harus ada spasi antar *item* dalam ekspresi.

```
#!/bin/bash
expr 5 + 4
expr "5 + 4"
expr 5+4 #jika tidak ada spasi maka tidak dievaluasi, hanya diprint
expr 5 \* $1
a=$(expr 10-3)
echo $a
```

Hasil, jika dieksekusi contoh_expr.sh 6

```
$
9
9
5+4
30
7
```

10.2.4 If statement

Format *if statement* adalah sebagai berikut:

```
if [some test]
then
    <perintah>
else
    <perintah lain>
fi
```

Contoh:

```
#!/bin/bash
if [$1 -ge 17]
then
    echo Boleh melakukan pemilu
else
    echo Tunggu pemilu selanjutnya
fi
```

If statement dapat *dinested* jika diperlukan.

Test

Terdapat beberapa *test* yang dapat dilakukan dalam *if []* yaitu:

String1 = *String2* *string* 1 sama dengan *string* 2

String1 != *String2* *string* 1 tidak sama dengan *string* 2

Integer1 -eq integer 2 integer 1 sama dengan integer 2

Integer1 -gt integer 2	integer 1 lebih besar dari integer 2
Integer1 -lt integer 2	integer 1 lebih kecil dari integer 2
-e file	file ada
&&	logika <i>and</i>
	logika <i>or</i>

Case

```
case <variabel> in
<pola 1>)
    perintah
    ;;
<pola 2>)
    perintah
    ;;
esac
```

Contoh:

```
#!/bin/bash
case $1 in
start)
    echo Mulai
    ;;
stop)
    echo Berhenti
    ;;
esac
```

10.2.5 Loop

Terdapat 3 dasar iterasi yaitu *for*, *while* dan *until*.

For

```
for var in <list>
do
    perintah
done
```

Contoh:

```
#!/bin/bash
for value in {1..10}
do
    echo $value
done

for value in {10..0..2}
do
    echo $value
done
```

While

```
while [some test]
do
    perintah
done
```


Contoh:

```
#!/bin/bash
counter=1
while [$counter -le 10]
do
    echo $counter
    ((counter++))
done
```

Until

```
until [some test]
do
    perintah
done
```

Contoh:

```
#!/bin/bash
counter=1
until [$counter -gt 10]
do
    echo $counter
    ((counter++))
done
```

Untuk mengendalikan iterasi dapat digunakan perintah **continue** dan **break**. Perintah ini mirip dengan perintah pada bahasa C.

10.2.6 Function

Fungsi digunakan untuk *code reuse*. Jika terdapat perintah-perintah yang sering digunakan maka dapat dibuat sebuah fungsi sehingga kita hanya akan memanggil fungsi tersebut dan bukan menulis ulang *code* dari awal.

```
nama_fungsi() {
    Perintah-perintah
}
```

Contoh:

```
#!/bin/bash

tampilkan_sesuatu() {
    echo Hello $1
}

tampilkan_sesuatu Mars
tampilkan_sesuatu Jupiter
```

Hasil:

```
$/myfunction.sh
Hello Mars
Hello Jupiter
```

Modul 11 Instalasi Xinu

Tujuan Praktikum

1. Praktikan dapat menginstal Xinu

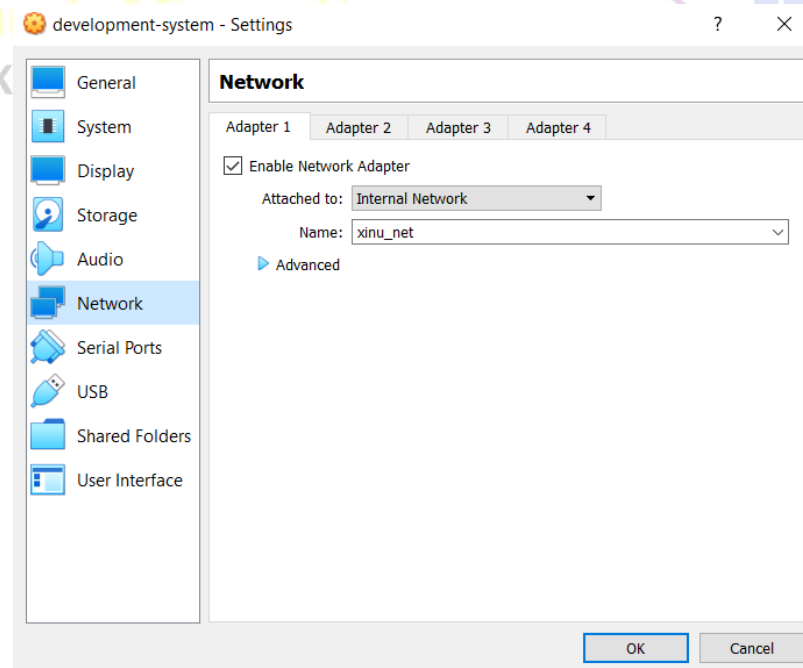
11.1 Instalasi Xinu

11.1.1 Persiapan

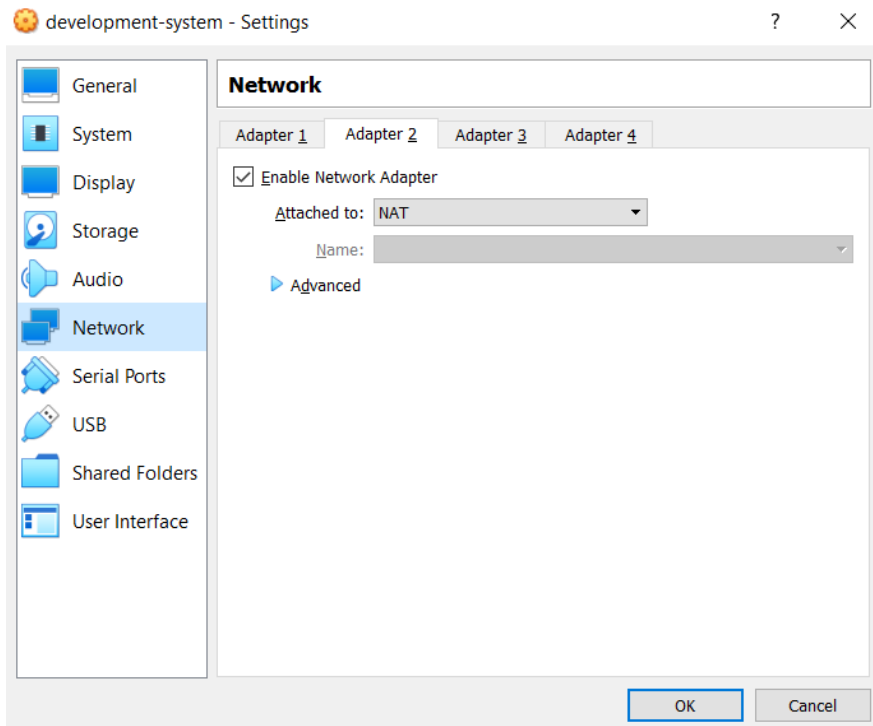
1. Download kemudian install VirtualBox terbaru (<https://www.virtualbox.org/wiki/Downloads>)
2. Download Xinu (<ftp://ftp.cs.purdue.edu/pub/comer/private/Xinu/xinu-vbox-appliances.tar.gz>)
3. Ekstrak xinu-vbox-appliances.tar.gz. Hasil ekstraksi adalah 2 file (**development-system.ova** dan **backend.ova**). File development-system.ova merupakan image Linux Debian yang berisi source code Xinu, compiler untuk mengcompile Xinu, DHCP server dan TFTP server. File backend.ova merupakan komputer yang akan menjalankan Xinu. Pada praktikum ini akan lebih banyak menggunakan development-system daripada backend.

11.1.2 Development-system

1. Jalankan VirtualBox
2. File -> Import Appliance
3. Pilih **development-system.ova** hasil dari tahap sebelumnya
4. Next (tunggu hingga selesai)
5. Akan muncul vm development-system pada panel sebelah kiri. Pilih "Settings".
6. Pilih Network dan ubah setting menjadi seperti gambar di bawah ini:



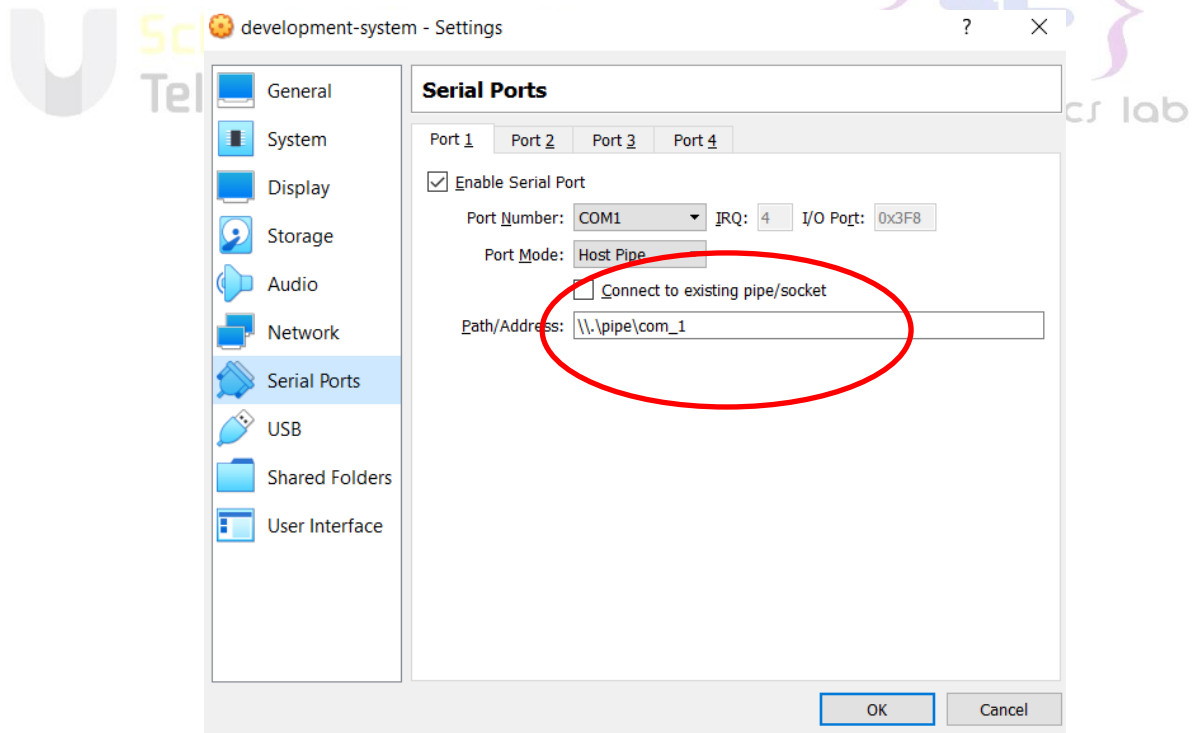
Gambar 11-1 Development System Network Setting pada Adapter 1



Gambar 11-2 Development System Network Setting pada Adapter 2

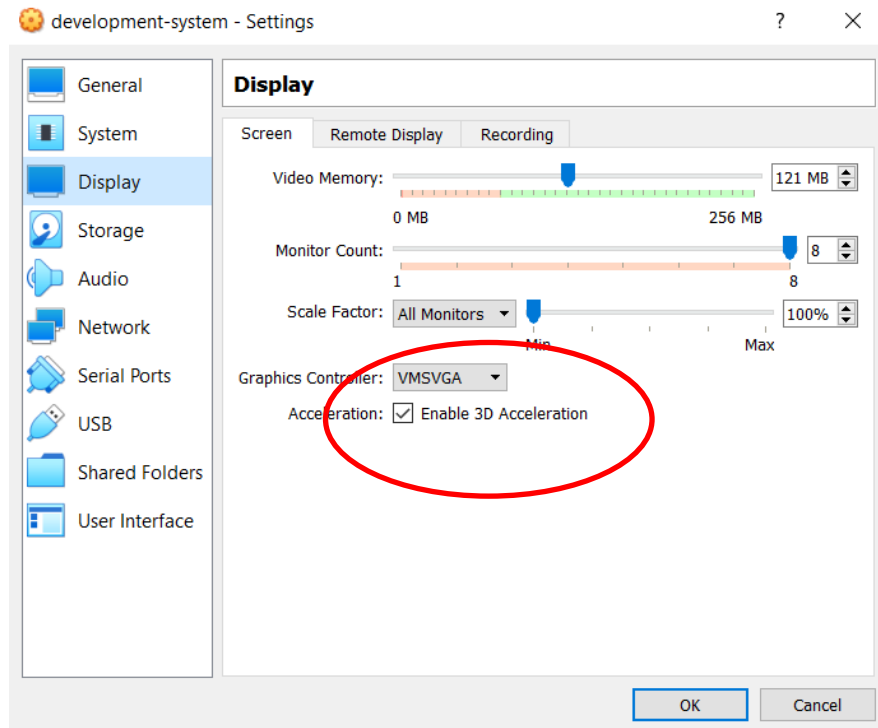
7. Pilih Serial Ports dan ubah setting menjadi seperti gambar di bawah ini:

Perhatikan “Connect to existing pipe/socket” tidak dipilih



Gambar 11-3 Development System Serial Setting pada Port 1

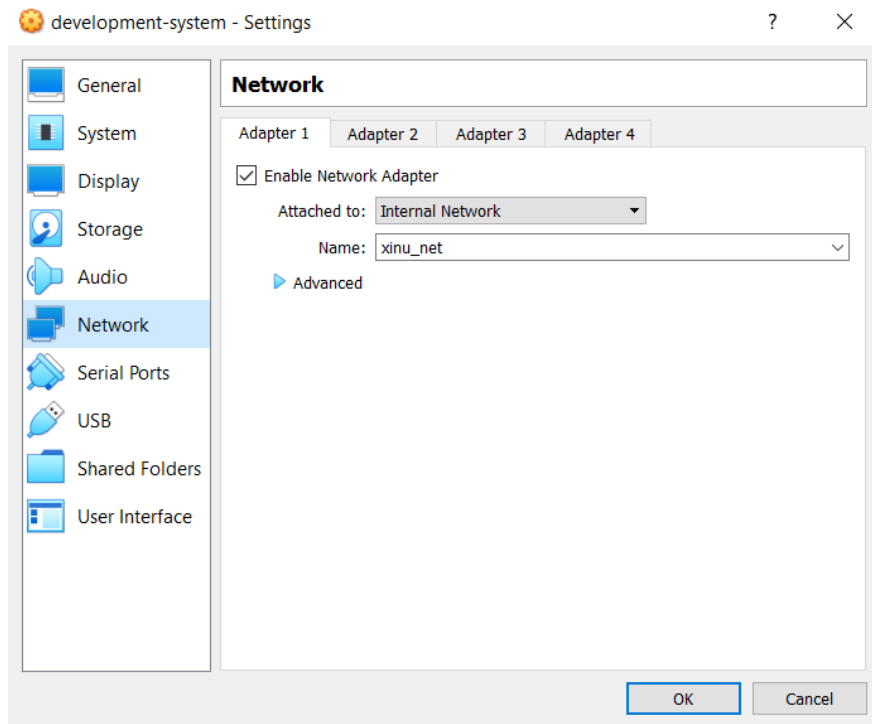
8. Ubah Setting Display



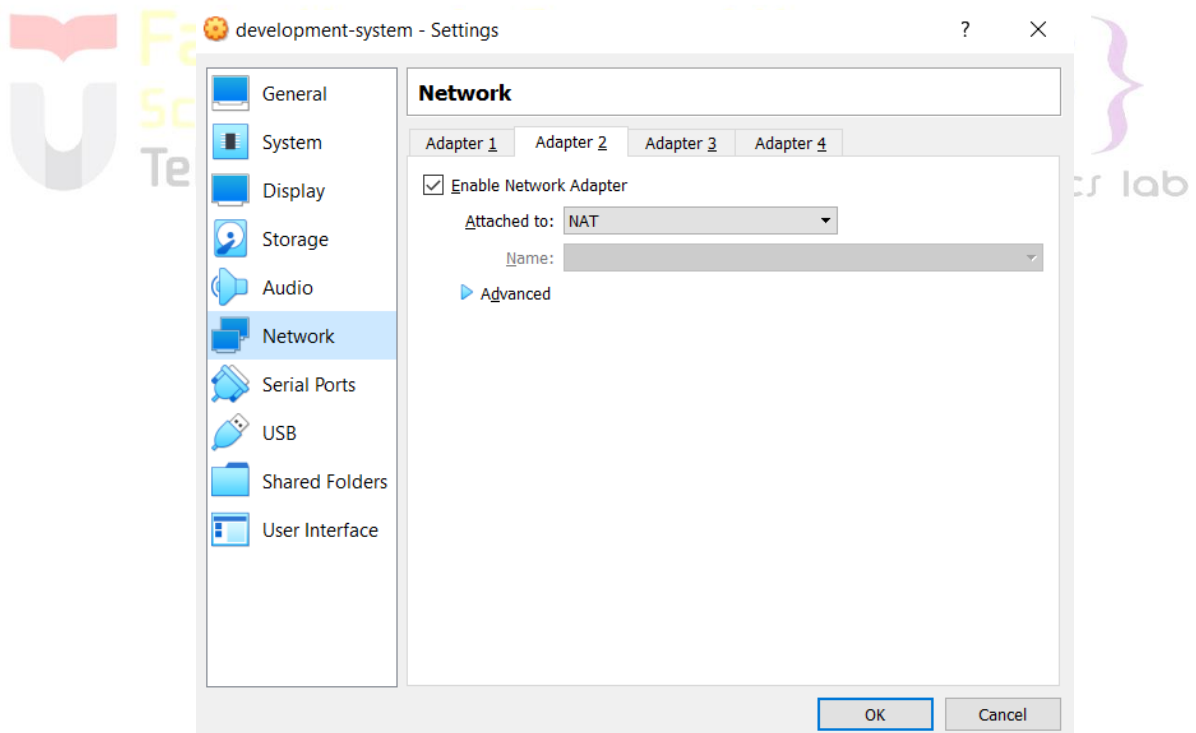
Gambar 11-4 Development System Display Setting

11.1.3 Backend

1. Jalankan VirtualBox
2. File -> Import Appliance
3. Pilih **backend.ova** hasil dari tahap sebelumnya
4. Next (tunggu hingga selesai)
5. Akan muncul vm backend pada panel sebelah kiri. Pilih "Settings".
6. Pilih Network dan ubah setting menjadi seperti gambar di bawah ini:

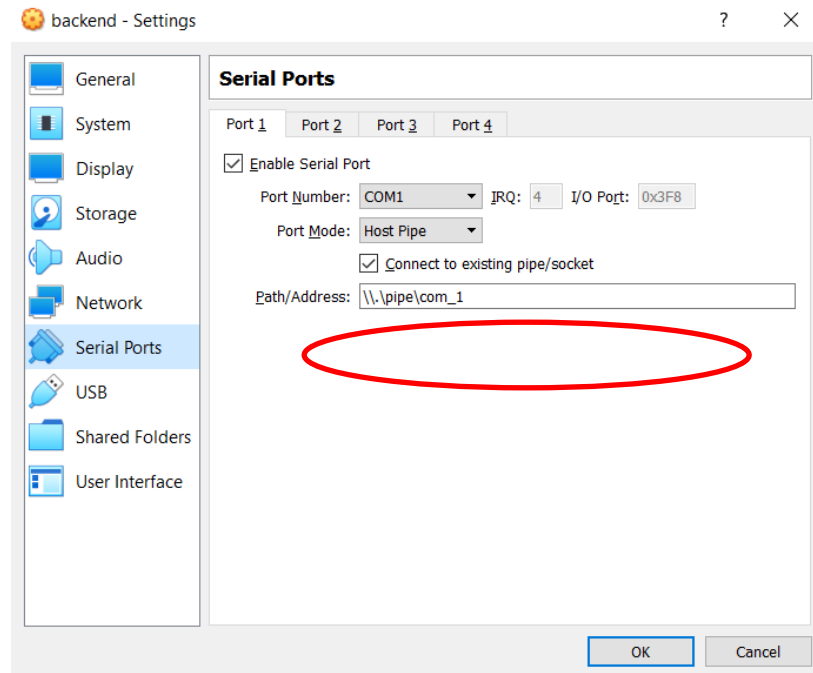


Gambar 11-5 Backend Network Setting pada Adapter 1



Gambar 11-6 Backend Network Setting pada Adapter 2

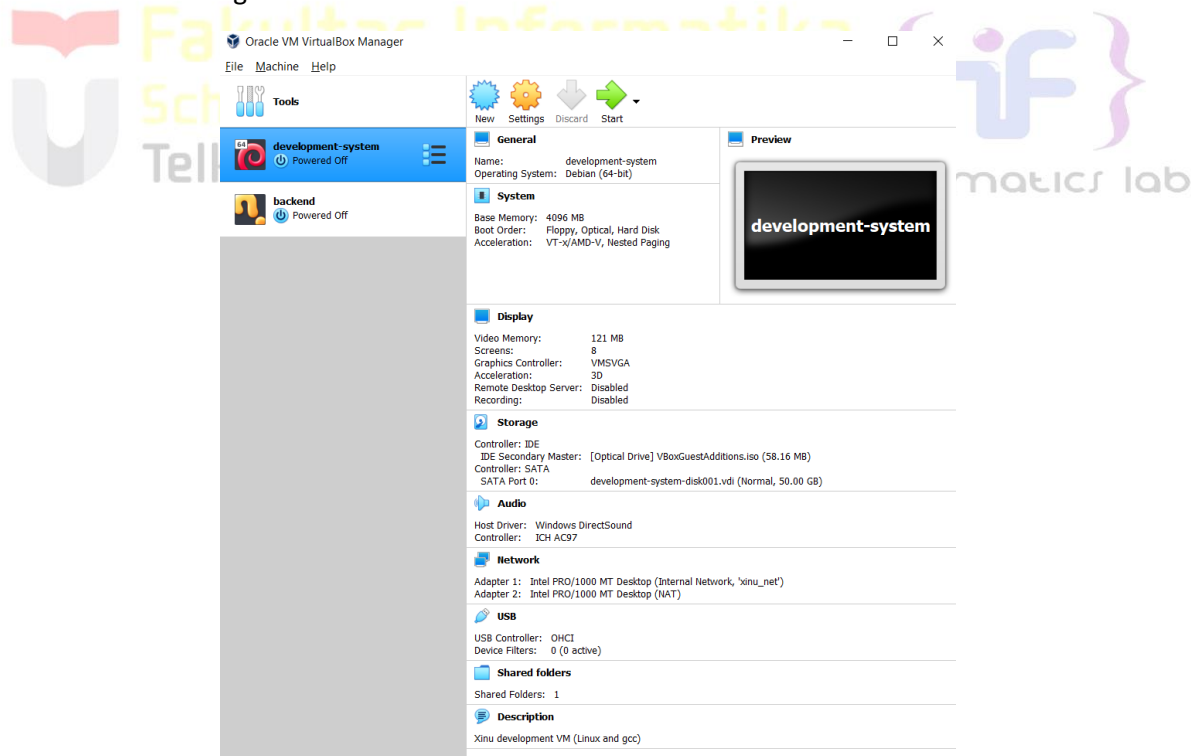
7. Pilih Serial Ports dan ubah setting menjadi seperti gambar di bawah ini:
Path/Address harus sama dengan yang ada pada development-system



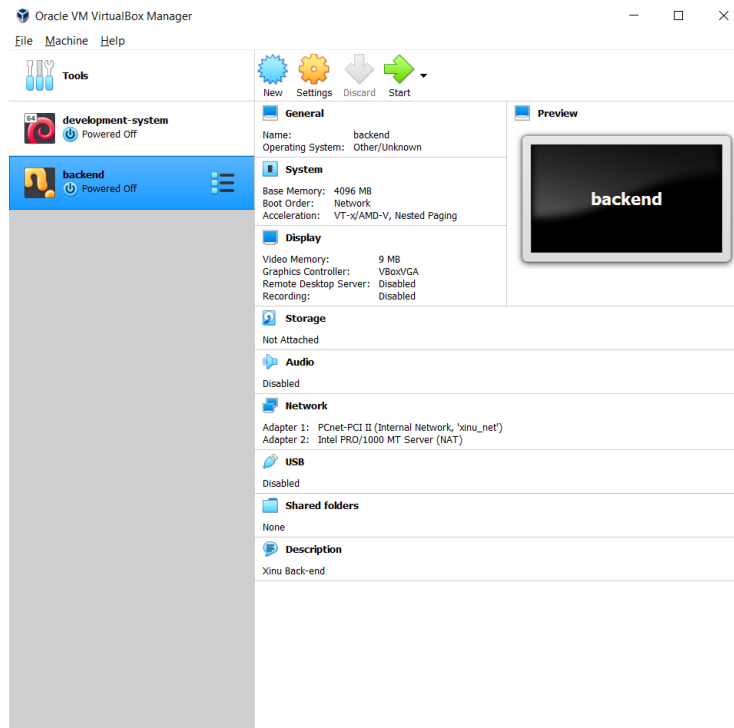
Gambar 11-7 Backend Serial Setting pada Port 1

11.1.4 Hasil Akhir

1. Hasil akhir settings



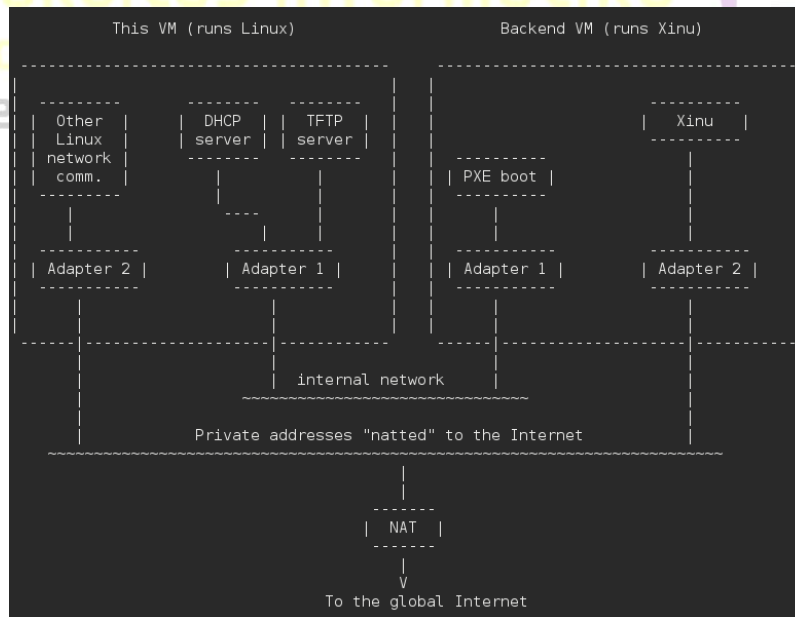
Gambar 11-8 Hasil Akhir Development System



Gambar 11-9 Hasil Akhir Backend

2. Arsitektur Jaringan

Berikut adalah arsitektur keseluruhan yang dibangun menggunakan 2 VM di atas:



Gambar 11-10 Arsitektur Xinu

Catatan:

- This VM adalah VM development-system (langkah II)
- Backend VM adalah VM backend (langkah III)

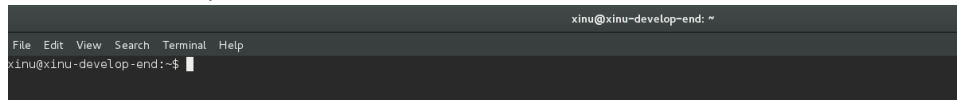
Modul 12 Menjalankan Xinu

Tujuan Praktikum

1. Praktikan dapat menjalankan Xinu

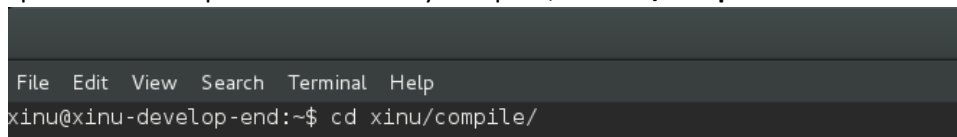
12.1 Menjalankan Xinu

1. Buka VirtualBox kemudian “Start” virtual machine development-system
2. Login menggunakan username: **xinu** password: **xinu rocks**
3. Akan muncul terminal seperti berikut ini:



Gambar 12-1 Login Xinu

4. Jalankan perintah untuk pindah ke directory compile \$ **cd xinu/compile**



Gambar 12-2 Pindah ke Directory Xinu

Perintah tersebut berarti melakukan perpindahan direktori ke direktori xinu/compile

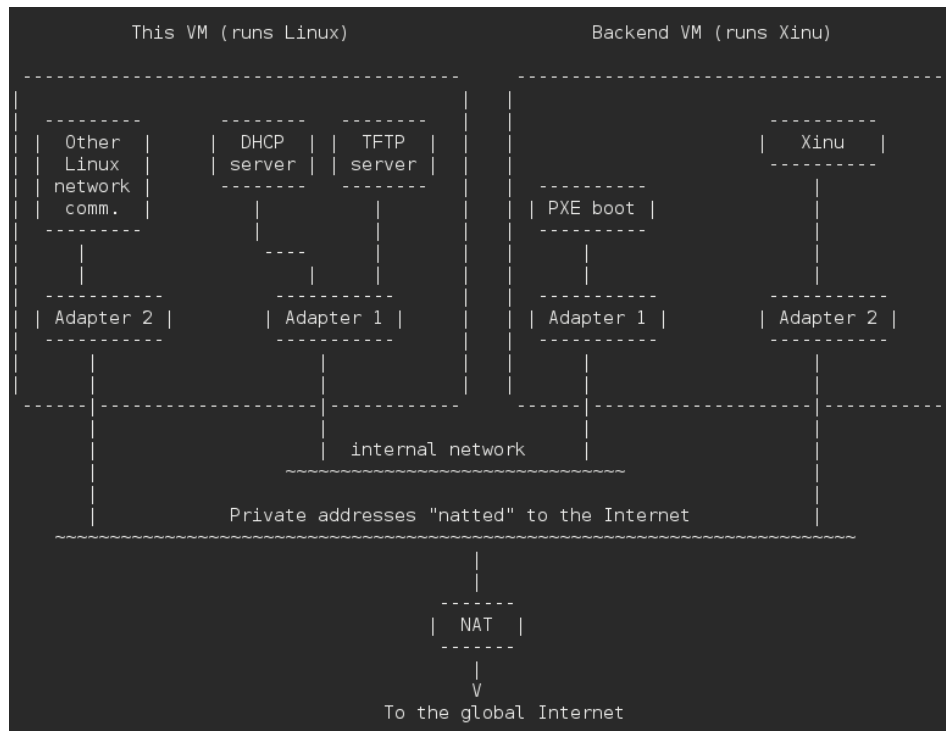
5. Jalankan perintah untuk menghapus image xinu os yang ada (untuk memastikan bahwa xinu os yang akan kita compile merupakan xinu os yang akan kita compile)

```
$ make clean
```

6. Jalankan perintah untuk mengcompile xinu os (proses compile cukup lama tergantung dari spesifikasi komputer yang digunakan)

```
$ make
```

7. Dengan mengcopile kita mendapatkan image xinu os bernama **xinu.elf** yang berada dalam folder xinu/compile/. Image xinu os tersebut (**xinu.elf**) kemudian dicopy **xinu.boot** pada folder server TFTP (/srv/tftp/). Virtual machine yang berada pada vm backend tidak berisi apapun. VM backend hanya bisa melakukan boot melalui jaringan (PXE). VM backend yang terhubung melalui ethernet ke VM development-system akan memperoleh IP dari DHCP server beserta lokasi boot yang bisa didownload (yaitu xinu.boot pada TFTP). Setelah xinu.boot didownload oleh vm backend, vm backend akan booting menggunakan xinu.boot tersebut dan mulai menjalankan xinu os.



Gambar 12-3 Arsitektur Xine dengan Dua VM

8. Setelah Xinu berjalan, Xinu pada vm backend akan berkomunikasi dengan vm development-system menggunakan serial port. Pada saat kita nanti mengetikkan perintah pada terminal vm development-system, kita telah terhubung ke xinu.
9. Untuk berkomunikasi menggunakan serial port digunakan aplikasi bernama minicom. Minicom adalah software untuk berinteraksi dengan serial port pada Linux. Perintah untuk menjalankan minicom adalah.

```
$ sudo minicom
```

Jika diminta password gunakan: **xinuroids**

Sudo diperlukan karena minicom mengakses langsung hardware sehingga butuh root akses.

10. **Berpindah ke VM Backend.** Jalankan virtualbox kemudian **"Start" virtual machine backend.** Tunggu hingga muncul pesan "Welcome to GRUB". Jika pesan tersebut muncul berarti Xinu berhasil dijalankan pada vm backend.
11. **Kembali lagi ke terminal pada vm development-system.** Jika tidak ada masalah maka akan muncul gambar berikut ini pada vm development-system:

```
[0x00000000 to 0x00FFFFFF]
25649 bytes of Xinu code.
[0x00000000 to 0x00006430]
18011 bytes of data.
[0x00006431 to 0x0000AA8B]
611696 bytes of heap space below 640K.
15728640 bytes of heap space above 1M.
[0x00100000 to 0x00FFFFFF]

-----
XINU
-----

Welcome to Xinu!

xsh $ _
```

Gambar 12-4 Hasil Running Xinu

Selamat Xinu sudah berjalan!



Modul 13 Mengenal Xinu

Tujuan Praktikum

1. Praktikan dapat menggunakan tool untuk explorasi source code
2. Praktikan dapat memahami source code Xinu

13.1 Paradigma Embedded

Xinu ditujukan untuk sistem embedded oleh karena itu Xinu mengikuti paradigma cross-development. Developer akan menggunakan komputer standar pada umumnya (PC atau laptop) dan menggunakan sistem operasi biasa seperti Linux atau Windows. Pada computer tersebut, developer akan membuat, mengedit, cross-compile dan cross-link Xinu software. Output dari cross-development adalah memory image. Ketika image tersebut telah berhasil dibuat, developer akan mendownload image ke sistem target (embedded system, microcontroller, dll) biasanya melalui jaringan. Terakhir, developer akan mulai menjalan image tersebut dan Xinu akan berjalan pada target sistem embedded.

13.2 Organisasi Source Code Xinu

Berikut adalah struktur direktori yang digunakan dalam mendvelop Xinu:

./compile	The Makefile used to compile and link a Xinu image
/bin	Executable scripts invoked during compilation
/binaries	Compiled binaries for Xinu functions (.o files)
./config	Source for the configuration program and Makefile
/conf.h	Configuration include file (copied to ../include)
/conf.c	Configuration declarations (copied to ../system)
./device	Source code for device drivers, organized into one subdirectory for each device type
/tty	Source code for the tty driver
/rfs	Source code for the remote file access system (both the master device and remote file pseudo-devices)
/eth	Source code for the Ethernet driver
/rds	Source code for the remote disk driver
/...	Directories for other device drivers
./include	All include files
./shell	Source code for the Xinu shell and shell commands
./system	Source code for Xinu kernel functions
./lib	Source code for library functions
./net	Source code for network protocol software

13.3 Implementasi Xinu

Secara umum, setiap file berkorespondensi pada system call (contoh: resume.c merupakan syscall untuk resume). Mayoritas fungsi-fungsi utama pada sistem Xinu dapat ditemukan pada file tersendiri. Berikut adalah beberapa source code yang berisi fungsi-fungsi utama Xinu:

<i>Configuration</i>	A text file containing device information and constants that describe the system and the hardware. The <i>config</i> program takes file <i>Configuration</i> as input and produces <i>conf.c</i> and <i>conf.h</i> .
<i>conf.h</i>	Generated by <i>config</i> , it contains declarations and constants including defined names of I/O devices, such as <i>CONSOLE</i> .
<i>conf.c</i>	Generated by <i>config</i> , it contains initialization for the device switch table.
<i>kernel.h</i>	General symbolic constants and type declarations used throughout the kernel.
<i>prototypes.h</i>	Prototype declarations for all system functions.
<i>xinu.h</i>	A master include file that includes all header files in the correct order. Most Xinu functions only need to include <i>xinu.h</i> .
<i>process.h</i>	Process table entry structure declaration; state constants.
<i>semaphore.h</i>	Semaphore table entry structure declaration; semaphore constants.
<i>tty.h</i>	Tty device control block, buffers, and other tty constants.
<i>bufpool.h</i>	Buffer pool constants and format.
<i>memory.h</i>	Constants and structures used by the low-level memory manager.
<i>ports.h</i>	Definitions by the high-level inter-process communication mechanism.
<i>sleep.h</i>	Definitions for real-time delay functions.
<i>queue.h</i>	Declarations and constants for the general-purpose process queue manipulation functions.
<i>resched.c</i>	The Xinu scheduler that selects the next process to run from the eligible set; <i>resched</i> calls the context switch.
<i>ctxsw.S</i>	The context switch that changes from one executing process to another; it consists of a small piece of assembly code.
<i>initialize.c</i>	The system initialization function, <i>sysinit</i> , and other initialization code as well as code for the null process (process 0).
<i>userret.c</i>	The function to which a user process returns if the process exits. <i>Userret</i> must never return. It must kill the process that executed it because the stack does not contain a legal frame or return address.
<i>platinit.c</i>	Platform-specific initialization.

13.4 Tipe Data

Pada Xinu untuk menunjukkan ukuran dan kegunaan suatu data digunakan konvensi: nama_kegunaan ukuran_data.

Nama_kegunaan: menunjukkan kegunaan data tersebut. Sebagai contoh *sid* adalah untuk semaphore id. Data tersebut (*sid*) harus digunakan ketika menggunakan semaphore dan tidak digunakan untuk keperluan lainnya. Contoh lain: *qid* adalah untuk queue id.

Ukuran_data: menunjukkan besarnya data yang digunakan. Pada C, ukuran data tergantung pada arsitektur komputer. Tipe *long* berukuran 32 bit pada sebuah komputer dan berukuran 64 bit pada

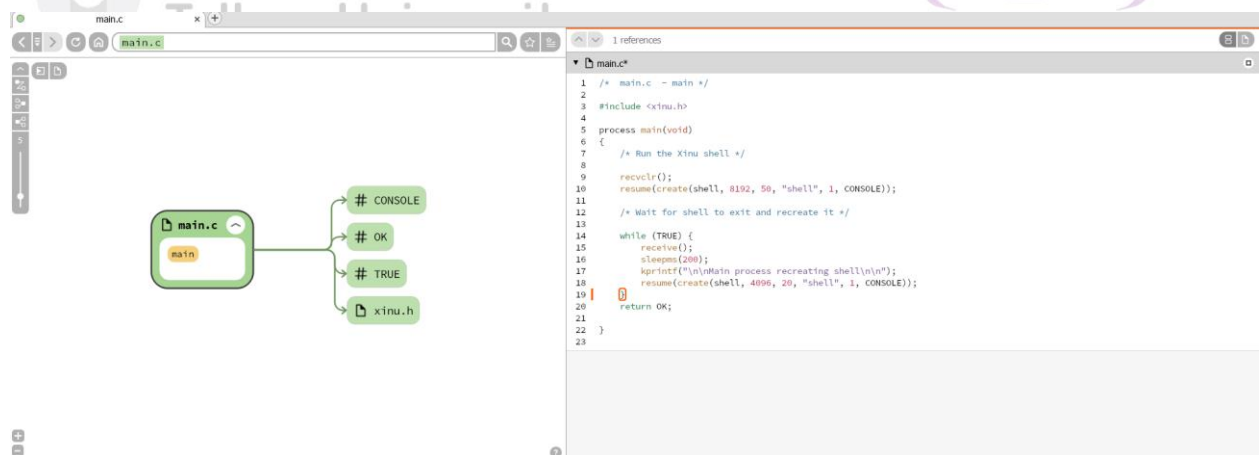
komputer yang lain. Untuk menggaransi ukuran data, Xinu mendefinisikan ukuran secara spesifik. Berikut adalah ukuran yang digunakan dalam Xinu:

Type	Meaning
byte	unsigned 8-bit value
bool8	8-bit value used as a Boolean
int16	signed 16-bit integer
uint16	unsigned 16-bit integer
int32	signed 32-bit integer
uint32	unsigned 32-bit integer

Gambar 13-1 Tipe Data pada Xinu

13.5 Memahami source code

Salah satu tool untuk memahami source code adalah SourceTrail (<https://www.sourcetrail.com/>). SourceTrail adalah software untuk mengeksplorasi source code. Programmer yang mumpuni lebih banyak membaca kode daripada menulis kode. SourceTrail sangat membantu dalam memahami kode karena dapat mencari fungsi dengan cepat (tinggal klik saja), mempunyai anotasi dan menggunakan GUI yang intuitive.



Gambar 13-2 Tampilan Sourcetrail

Instalasi SourceTrail

- Download SourceTrail (<https://www.sourcetrail.com/>). SourceTrail adalah software untuk mengeksplorasi source code. Programmer yang bagus lebih banyak membaca kode daripada menulis kode.
- Jalankan SourceTrail
- Project-> New Project
- Isi nama project xinu dan pilih lokasi project di manapun

- e) Add Source Groups pilih C pilih Empty Source Group
- f) File & Directories to Index: masukkan semua folder Xinu (hasil dari download source code xinu pada modul sebelumnya).
- g) Include Paths: .../xinu/include (path dari download source code xinu)
- h) Create
- i) Silahkan eksplorasi source code Xinu



Fakultas Informatika
School of Computing
Telkom University



Modul 14 Proses dan Eksekusi Program

Tujuan Praktikum

1. Praktikan memahami proses dan eksekusi pada Xinu
2. Praktikan dapat memodifikasi Xinu dengan mengubah source code

14.1 Proses

1. Sistem operasi menyimpan semua informasi mengenai proses pada struktur data yang disebut sebagai process table. Sebuah proses direpresentasikan sebagai sebuah entri dalam process table. Entri pada process table harus dibuat pada saat proses diciptakan (./system/create.c) dan entri pada process table akan dihapus pada saat proses diterminasi (./system/kill.c).
2. Pada Xinu, implementasi process table menggunakan global array bernama proctab[]. Deklarasi proctab[] dapat dilihat pada ./include/process.h. Pada source code tersebut menggunakan keyword extern sehingga variable array proctab[] bersifat global (yaitu dapat diakses oleh apapun).

0	1	2	3	4	5	6

3. Isi dari array proctab[] adalah process control block (PCB) suatu proses. Pada Xinu, isi dari setiap array proctab berupa struct procent (./include/process.h). Dengan kata lain struct procent adalah PCB. Berikut adalah struct procent:

```
41  /* Definition of the process table (multiple of 32 bits) */
42
43  struct procent {          /* Entry in the process table */
44      uint16 prstate;      /* Process state: PR_CURR, etc. */
45      pri16 prprio;        /* Process priority */
46      char *prstkptr;      /* Saved stack pointer */
47      char *prstkbase;     /* Base of run time stack */
48      uint32 prstklen;     /* Stack length in bytes */
49      char prname[PNMLEN]; /* Process name */
50      sid32 prsem;         /* Semaphore on which process waits */
51      pid32 prparent;      /* ID of the creating process */
52      umsg32 prmsg;        /* Message sent to this process */
53      bool8 prhasmsg;      /* Nonzero iff msg is valid */
54      int16 prdesc[NDESC]; /* Device descriptors for process */
55  };
```

Gambar 14-1 Struktur Procent

4. Xinu menggunakan implicit data structure yaitu dengan menghilangkan/mengabaikan process ID. Process ID merupakan indek pada array proctab[]. Atau dengan kata lain sebuah proses direferensi hanya oleh process ID nya yang merupakan indek pada array proctab[]. Misalkan: jika mengakses proctab[3] artinya mengakses proses dengan pid = 3 dan kemudian dapat memanipulasi status proses tersebut melalui struct procent yang ada.

14.2 Eksekusi Program

1. Program konvensional disebut sebagai sequential karena programmer membayangkan komputer melakukan eksekusi kode baris per baris, statement per statement; Pada setiap waktu komputer hanya mengeksekusi sebuah statement.
2. Sistem operasi memperluas pandangan mengenai komputasi dengan konsep concurrent processing. Concurrent processing berarti banyak komputasi dapat berlangsung “pada saat yang bersamaan”. Catatan: komputer dengan banyak core/cpu, komputer benar-benar dapat melakukan banyak hal secara bersamaan (parallelism). Pada komputer dengan 1 CPU maka OS akan menciptakan ilusi konkuren.
3. Untuk menciptakan ilusi “pada saat yang bersamaan”, komputer menggunakan teknik yang disebut multiprocessing/multitasking yaitu sistem operasi berganti dan memilih proses yang ada dari kumpulan proses yang tersedia. Komputer akan mengeksekusi satu program dalam beberapa millisecond kemudian berpindah ke program lainnya. Ketika dilihat oleh manusia, tampak bahwa semua program berjalan bersamaan karena cepatnya prosesor dan lambatnya indra manusia.
4. Terdapat 2 kategori multitasking yaitu: timesharing dan realtime. Timesharing memberikan prioritas yang sama untuk setiap komputasi, membolehkan komputasi berjalan dan berakhir kapan saja. Realtime memberikan prioritas yang berbeda untuk setiap komputasi karena terdapat performansi waktu yang ingin dicapai. Realtime akan memberikan prioritas, menjadwalkan proses secara terencana.
5. Pada sekuensial program, prosesor akan mengeksekusi instruksi satu per satu. Perhatikan kode main.c berikut ini:



```
1  /* main.c - main */
2
3  #include <xinu.h>
4  void sndA(void);
5  void sndB(void);
6
7  process main(void)
8  {
9
10     sndA();
11     sndB();
12     return OK;
13 }
14
15 void sndA(void)
16 {
17
18     while(1) {
19         printf("A\n");
20         sleeps(1000);
21     }
22 }
23
24 void sndB(void)
25 {
26
27     while(1) {
28         printf("B\n");
29         sleeps(1000);
30     }
31 }
32
33 }
```

Gambar 14-2 Contoh Program Sekuensial pada Xinu

Pada kode tersebut terdapat sebuah proses yaitu main (baris 7) dan 2 fungsi sndA() dan sndB(). Proses main kemudian memanggil 2 buah fungsi yaitu sndA() dan sndB() jika sudah selesai maka proses main akan terminasi. Fungsi sndA() akan menampilkan huruf A (baris 19) kemudian sleep selama 1 detik (baris 20) menggunakan system call sleepms() yang disediakan oleh Xinu. Fungsi sndA() secara terus menerus berjalan. Fungsi sndB() sama dengan sndA() hanya menampilkan huruf B secara terus menerus.

Fungsi akan berjalan secara sekuensial. Konsekuensi kode tersebut adalah: fungsi sndB() tidak akan pernah dieksekusi karena fungsi sndA() tidak pernah terminasi (selalu berada pada loop while (1){}

6. Berikut adalah contoh konkuren pada sistem operasi Xinu

```

1  /* main.c - main */
2
3  #include <xinu.h>
4  void sndA(void);
5  void sndB(void);
6
7
8  process main(void)
9  {
10     resume(create(sndA, 1024, 20, "process A", 0));
11     resume(create(sndB, 1024, 20, "process B", 0));
12     return OK;
13 }
14
15 void sndA(void)
16 {
17     while(1){
18         printf("A\n");
19         sleepms(1000);
20     }
21 }
22
23 void sndB(void)
24 {
25     while(1){
26         printf("B\n");
27         sleepms(1000);
28     }
29 }
30
31
32
33

```

Gambar 14-3 Contoh Konkuren pada Xinu

Perbedaan utama kode ini dengan kode sebelumnya adalah **terdapat 3 proses dalam kode ini yaitu proses main, sndA dan sndB**. Pada kode sebelumnya hanya terdapat 1 proses yaitu main saja. Hal ini dapat dicapai menggunakan baris ke 10 dan 11. Pada baris 10, create() adalah syscall pada Xinu. Syscall create() akan membuat proses baru bernama "process A". Output dari syscall create () adalah PID proses yang baru saja dibuat. Pada Xinu, proses yang baru saja dibuat berada dalam state suspend. Oleh karena itu, digunakan syscall resume() untuk menjalankan proses tersebut. Baris ke 11, sama dengan baris 10 hanya saja fungsi sndB() yang digunakan. Berikut adalah deklarasi syscall create():

```

11 pid32  create(
12     void      *funcaddr, /* Address of the function */
13     uint32     ssize,     /* Stack size in bytes */
14     pri16      priority,  /* Process priority > 0 */
15     char       *name,     /* Name (for debugging) */
16     uint32     nargs,     /* Number of args that follow */
17     ...
18 )

```

Gambar 14-4 Deklarasi Syscall Create()

create(sndA, 1024, 20, "process A, 0) berarti:

- + fungsi yang akan digunakan dalam proses tersebut adalah sndA()
- + ukuran stack adalah 1024
- + proses tersebut mempunyai prioritas 20
- + nama proses tersebut adalah "process A"
- + tidak ada argument yang digunakan pada fungsi sndA().

Misalkan kita mempunyai fungsi sum(int a, int b) dan akan membuat proses tersebut maka create(sum, 1024, 20, "penjumlahan", 2, int a, int b);

Setelah syscall resume() dieksekusi proses akan aktif dan dijadwalkan pada CPU. Hasil dari kode di atas adalah output A B muncul bergantian.

7. Contoh lain mengenai konkurensi dapat dilihat pada kode berikut ini:

```

1  /* main.c - main */
2
3  #include <xinu.h>
4  void sndChar(char);
5
6  process main(void)
7  {
8      resume(create(sndChar, 1024, 20, "Send A", 1, 'A'));
9      resume(create(sndChar, 1024, 20, "Send B", 1, 'B'));
10     return OK;
11 }
12
13 void sndChar(char c)
14 {
15
16     while(1){
17         printf("%c \n", c);
18         sleeps(1000);
19     }
20 }
21

```

Gambar 14-5 Contoh Lain Konkurensi pada Xinu

Pada kode ini akan ada 2 proses yaitu proses bernama "Send A" dan "send B" (ada total 3 proses jika proses main juga dihitung). Penciptaan 2 proses tersebut dilakukan pada baris ke 8 dan 9. Perhatikan bahwa 2 proses tersebut menggunakan fungsi/kode yang sama yaitu sndChar(). Walaupun 2 proses tersebut mengeksekusi kode yang sama, 2 proses tadi tidak mempengaruhi satu dengan lainnya. Setiap proses mempunyai argument, alokasi memory, variable local, stack, dst sendiri-sendiri. Hal penting yang perlu diingat adalah: sistem operasi harus mengalokasikan memory untuk setiap proses, bahkan jika proses berbagi kode yang sama dengan proses lain. Sebagai konsekuensinya, ukuran memory akan membatasi banyaknya proses yang dapat berjalan.

Daftar Pustaka

1. 15 Examples To Master Linux Command Line History. The Geek Stuff. [Online] Ramesh Natarajan, August 11, 2008. [Cited: Mei 20, 2018.] <https://www.thegeekstuff.com/2008/08/15-examples-to-master-linux-command-line-history>.
2. 30 Useful 'ps Command' Examples for Linux Process Monitoring. TecMint.com. [Online] September 11, 2017. [Cited: Mei 21, 2018.] <https://www.tecmint.com/ps-command-examples-for-linux-process-monitoring/>.
3. A Guide to the Linux "Top" Command. Boolean World. [Online] April 30, 2018. [Cited: Mei 20, 2018.] <https://www.booleanworld.com/guide-linux-top-command/>.
4. Sejarah Dan Perkembangan Linux. Linux.or.id. [Online] Linux, Maret 12, 2016. [Cited: Mei 20, 2018.] <https://www.linux.or.id/sejarah-dan-perkembangan-linux.html>.
5. POSIX Threads Programming. [Online] U.S. Department of Energy by Lawrence Livermore National Laboratory, August 03, 2017. [Cited: July 10, 2018.] <https://computing.llnl.gov/tutorials/pthreads/>.
6. POSIX Thread Programming or Pthreads. CS 50 Software Design and Implementation. [Online] Trustees of Dartmouth College. [Cited: July 11, 2018.] <http://www.cs.dartmouth.edu/~campbell/cs50/threads.html>.
7. Hijau, Lobot. ngoding-di-terminal-dengan-vim-bagian-1-perkenalan-5a8984c1ec057. Codepolitan. [Online] Februari 19, 2018. <https://www.codepolitan.com/ngoding-di-terminal-dengan-vim-bagian-1-perkenalan-5a8984c1ec057>.
8. bad_crow. vim-basics. howtoforge. [Online] <https://www.howtoforge.com/vim-basics>.
9. openvim. [Online] <https://www.openvim.com/>.
10. Xinu. [Online] <https://xinu.cs.purdue.edu/>.
11. Sourcetrail. [Online] <https://www.sourcetrail.com/>.



Kontak Kami :



@fiflab



Praktikum IF LAB



informaticslab@telkomuniversity.ac.id



informatics.labs.telkomuniversity.ac.id