

# SISTEM OPERASI

## S1 Teknologi Informasi



Official Line  
Informatics Lab



published by school of computing



## LEMBAR PENGESAHAN

Saya yang bertanda tangan di bawah ini:

Nama : Endro Ariyanto, S.T., M.T.

NIK : 04660014

Koordinator Mata Kuliah : Sistem Operasi

Prodi : S1 Teknologi Informasi

Menerangkan dengan sesungguhnya bahwa modul ini digunakan untuk pelaksanaan praktikum di Semester Genap Tahun Ajaran 2021/2022 di Laboratorium Informatika Fakultas Informatika Universitas Telkom.

Bandung, 12 Februari 2022

Mengesahkan,  
Koordinator Mata Kuliah Sistem Operasi

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

Mengetahui,  
Kaprodi S1 Teknologi Informasi

  
Hilal Huda Nuha, S. T., M. T., Ph. D.

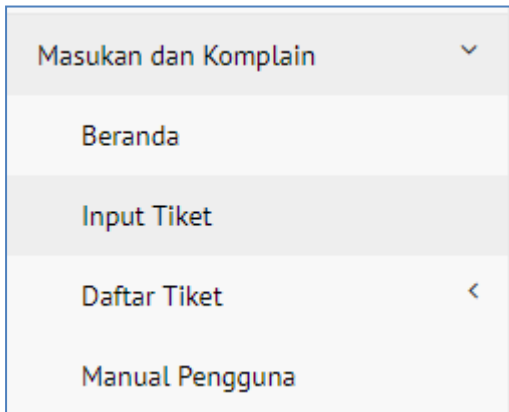
## Peraturan Praktikum

### Laboratorium Informatika 2021/2022

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 kedukaan (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 (Siti Amatullah Karimah): 082231501931, [karimahsiti@telkomuniversity.ac.id](mailto:karimahsiti@telkomuniversity.ac.id)  
Laboran IFLAB (Oku Dewi): 085294057905, [laboratorium.informatika@gmail.com](mailto:laboratorium.informatika@gmail.com)  
Fanspage IFLAB: [https://www.instagram.com/informaticslab\\_telu/](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 'Fakultas / Bagian' dropdown is set to 'BIDANG AKADEMIK (FIF)'. The 'Program Studi / Urusan' dropdown is set to 'URUSAN LABORATORIUM/BENGKEL/STUDIO (FIF)'. The 'Pelapor' field contains the name 'RIZQILLAH ZAHRA LESTARI'. The 'Layanan' dropdown is set to 'PRAKTIKUM'. The 'Kategori' dropdown is set to 'Pelaksanaan Praktikum'. The 'Sub Kategori' dropdown is set to 'Please Select...'. Below these fields are two radio buttons for 'Tipe Masukan', with 'Komplain' selected. At the bottom of the form is a rich text editor with a toolbar containing icons for bold, italic, underline, bulleted list, numbered list, link, unlink, image, video, table, 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 2021/2022 .....	ii
Tata cara Komplain Praktikum IFLab Melalui IGracias.....	iii
Daftar Isi.....	iv
Daftar Gambar .....	vii
Daftar Tabel .....	ix
Modul 1 Instalasi Xinu .....	1
1.1 Persiapan.....	1
1.2 Import dan Setting Development-System VM.....	1
1.3 Import dan Setting Backend VM.....	4
1.4 Hasil Akhir .....	6
Modul 2 Eksplorasi Xinu .....	8
2.1 Menjalankan Xinu .....	8
2.2 Eksplorasi Xinu .....	12
Modul 3 Membaca Source Code Xinu .....	14
3.1 Paradigma Embedded.....	14
3.2 Bahasa Pemrograman yang Digunakan .....	14
3.3 Organisasi Source Code Xinu .....	14
3.4 File-File Utama Xinu .....	15
3.5 Syscall.....	16
3.6 Tipe Data .....	16
3.7 Memahami Source Code.....	17
Modul 4 Proses.....	19
4.1 Proses.....	19
Modul 5 Sekuensial dan Konkuren Program .....	21
5.1 Pendahuluan .....	21
5.2 Sekuensial Program.....	21
5.3 Konkuren Program .....	23
Modul 6 Semaphore .....	25
6.1 Operasi Semaphore.....	25
6.1.1 Inisiasi.....	25
6.1.2 Signal .....	25
6.1.3 Wait.....	25
6.2 Pola Pada Semaphore .....	26
6.2.1 Signaling .....	26

6.2.2	Mutex.....	27
Modul 7	Syscall.....	29
7.1	Definisi dan Fungsi Syscall.....	29
7.2	Cara Kerja Syscall .....	29
7.3	Template Syscall Xinu.....	30
7.4	Syscall Resume .....	30
7.5	Membuat Syscall Baru pada Xinu .....	31
7.6	Cara Penggunaan Syscall.....	33
Modul 8	Shell.....	34
8.1	Shell.....	34
Modul 9	Memori.....	38
9.1	Memori .....	38
Modul 10	Linux dan Windows .....	41
10.1	Definisi Sistem Operasi .....	41
10.2	Sistem Operasi Windows .....	41
10.2.1	Sejarah Windows .....	41
10.3	Sistem Operasi Linux.....	46
10.3.1	Sejarah Linux.....	46
10.3.2	Pengenalan Ubuntu .....	46
10.3.3	Instalasi Ubuntu .....	47
Modul 11	Perintah Dasar Linux.....	50
11.1	Perintah-Perintah Dasar Linux .....	50
11.1.1	ls (List Dictionary).....	51
11.1.2	cd (Change Dictionary).....	51
11.1.3	cp (Copy) .....	51
11.1.4	rm (Remove) .....	52
11.1.5	mv (Move).....	52
11.1.6	mkdir (Make Dictionary) .....	53
11.1.7	pwd (Print Working Directory).....	53
11.1.8	man (Manual).....	53
11.1.9	(Pipeline) .....	53
11.1.10	Redirection.....	54
11.2	GCC.....	54
11.2.1	Instalasi GCC.....	54
11.2.2	Meng-Compile Program Menggunakan GCC .....	55
Modul 12	Scripting .....	56
12.1	Shell.....	56

12.1.1	Bash.....	56
12.1.2	Struktur Bash.....	56
12.2	Scripting Dasar .....	57
12.2.1	Variabel .....	57
12.2.2	Command Line Argument .....	57
12.2.3	Input.....	58
12.2.4	Aritmatik .....	58
12.2.5	If Statement .....	59
12.2.6	Loop.....	60
12.2.7	Function .....	61
Modul 13	Keamanan .....	62
13.1	Access Control (Permission) .....	62
13.1.1	Permission Dasar pada Linux .....	62
13.1.2	Cara Mengubah Permission pada Linux.....	63
13.2	Integrity (SHA-256).....	66
13.2.1	Dasar Integritas dan Hash .....	66
13.2.2	Memeriksa Integritas dengan Tool .....	67
13.3	Confidentiality (Enkripsi).....	68
13.3.1	Enkripsi <i>Disk/File System</i> (ENCFS) .....	68
13.4	GPG .....	70
13.4.1	Install GPG.....	70
13.4.2	Import Public Key Orang Lain.....	71
13.4.3	Membuat Public Key Kita Tersedia .....	71
13.4.4	Enkripsi dengan PGP .....	71
13.4.5	Dekripsi dengan PGP .....	72
Daftar Pustaka.....		73



## Daftar Gambar

Gambar 1-1 Development System Setting pada VirtualBox.....	1
Gambar 1-2 Development System Network Setting pada Adapter 1.....	2
Gambar 1-3 Development System Network Setting pada Adapter 2.....	2
Gambar 1-4 Development System Serial Ports Setting pada Port 1.....	3
Gambar 1-5 Development System Display Setting.....	3
Gambar 1-6 Backend Setting pada Virtual Box.....	4
Gambar 1-7 Backend Network Setting pada Adapter 1.....	4
Gambar 1-8 Backend Network Setting pada Adapter 2.....	5
Gambar 1-9 Backend Serial Ports Setting pada Port 1.....	5
Gambar 1-10 Hasil Akhir Development System.....	6
Gambar 1-11 Hasil Akhir Backend.....	6
Gambar 1-12 Arsitektur Xinu.....	7
Gambar 2-1 Login Xinu.....	8
Gambar 2-2 Pindah ke Directory Xinu.....	8
Gambar 2-3 Struktur Direktori Pada Xinu.....	9
Gambar 2-4 Direktori Home.....	9
Gambar 2-5 Direktori Xinu.....	10
Gambar 2-6 Hasil Running Backend Xinu.....	11
Gambar 2-7 Arsitektur Xinu dengan Dua VM.....	11
Gambar 2-8 Hasil Running Xinu.....	12
Gambar 3-1 Tipe Data Pada Xinu.....	17
Gambar 3-2 Tampilan Sourcetrail.....	17
Gambar 4-1 Struktur Procent.....	20
Gambar 5-1 Contoh Sekuensial Program pada Xinu.....	22
Gambar 5-2 Contoh Konkuren Program pada Xinu.....	23
Gambar 5-3 Deklarasi Syscall create().....	24
Gambar 7-1 Definisi dan Fungsi Syscall.....	29
Gambar 7-2 Disable Interrupts Syscall.....	29
Gambar 7-3 Template Syscall Xinu.....	30
Gambar 7-4 Kode Syscall resume().....	31
Gambar 7-5 Contoh Kode Syscall Baru Pada Xinu.....	32
Gambar 7-6 Kode File Prototypes.h.....	32
Gambar 7-7 Contoh Penggunaan Syscall Pada Perintah Uptime.....	33
Gambar 8-1 Shell.c.....	34
Gambar 8-2 Struct cmdent cmdtab[].....	35
Gambar 8-3 Penambahan extern shellcmd xsh_mycmd Pada shprototype.h.....	36
Gambar 8-4 xsh_mycmd.c.....	36
Gambar 8-5 Fungsi printMemUse.....	37
Gambar 9-1 Bagian Memory Image Pada Xinu.....	38
Gambar 9-2 Ilustrasi Memori.....	39
Gambar 9-3 Contoh List Memori Pada Xinu.....	39
Gambar 9-4 Ilustrasi Penggunaan Linked List Untuk Manajemen Memory.....	40
Gambar 10-1 Keluarga Windows.....	44
Gambar 10-2 Booting.....	45
Gambar 10-3 Windows Setup dan Aktivasi.....	45
Gambar 10-4 Tipe Instalasi dan Pemilihan Storage.....	45
Gambar 10-5 Proses Instalasi.....	46
Gambar 10-6 Pohon Keluarga Ubuntu.....	46



Gambar 10-7 Beberapa Linux Distro.....	47
Gambar 10-8 Pilihan Bahasa .....	48
Gambar 10-9 Pilihan Instalasi .....	48
Gambar 10-10 Tipe Instalasi dan Pemilihan Partisi .....	48
Gambar 10-11 Pilih Kota dan Pengisian Data .....	49
Gambar 10-12 Proses Instalasi.....	49
Gambar 10-13 Proses Instalasi Selesai.....	49
Gambar 12-1 Contoh Sesi Bash.....	56
Gambar 12-2 myscript.sh.....	56
Gambar 13-1 Daftar File.....	63
Gambar 13-2 Mode.....	63
Gambar 13-3 GtkHash Didukung Algoritma Checksum .....	67
Gambar 13-4 Menghasilkan SHA256 Checksum untuk UbuntuMATE iso .....	68
Gambar 13-5 Public Key.....	71



**Fakultas Informatika**  
**School of Computing**  
**Telkom University**



## Daftar Tabel

Tabel 6-1 Pola Signaling .....	26
Tabel 6-2 Pola Mutex .....	27
Tabel 10-1 Timeline Sistem Operasi Windows .....	41
Tabel 10-2 Spesifikasi Minimum dan Rekomendasi Windows 10 .....	44
Tabel 10-3 Spesifikasi Minimum dan Rekomendasi Ubuntu .....	47
Tabel 11-1 Penjelasan Struktur Perintah Linux.....	50
Tabel 11-2 Sebagian Daftar Opsi Perintah ls.....	51
Tabel 11-3 Sebagian Daftar Opsi Perintah cp .....	51
Tabel 11-4 Sebagian Daftar Opsi Perintah rm.....	52
Tabel 11-5 Sebagian Daftar Opsi Perintah mv .....	52
Tabel 11-6 Sebagian Daftar Opsi Perintah mkdir.....	53



# Modul 1 Instalasi Xinu

## Tujuan Praktikum

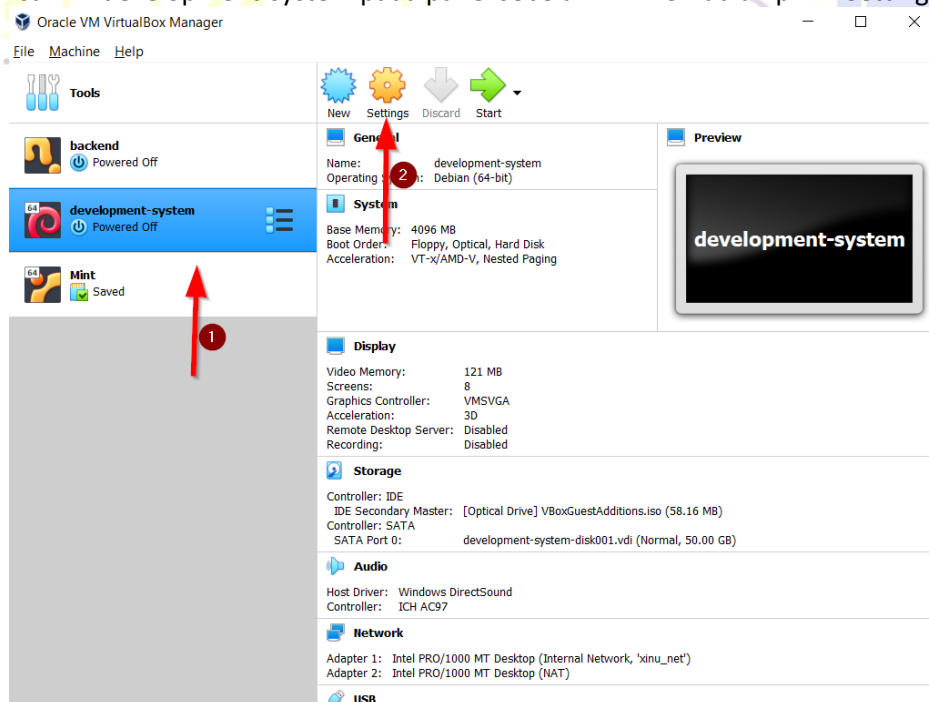
1. Praktikan dapat menginstal Xinu OS.
2. Praktikan memahami arsitektur sistem Xinu OS.

## 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. menggunakan aplikasi zip. Hasil ekstraksi adalah dua buah file yaitu: **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 virtual yang akan menjalankan Xinu. File berektensi .ova adalah file image virtual machine (vm). File-file .ova tersebut nanti akan diimport sehingga menjadi virtual machine. Pada praktikum ini akan lebih banyak menggunakan vm development-system daripada vm backend.

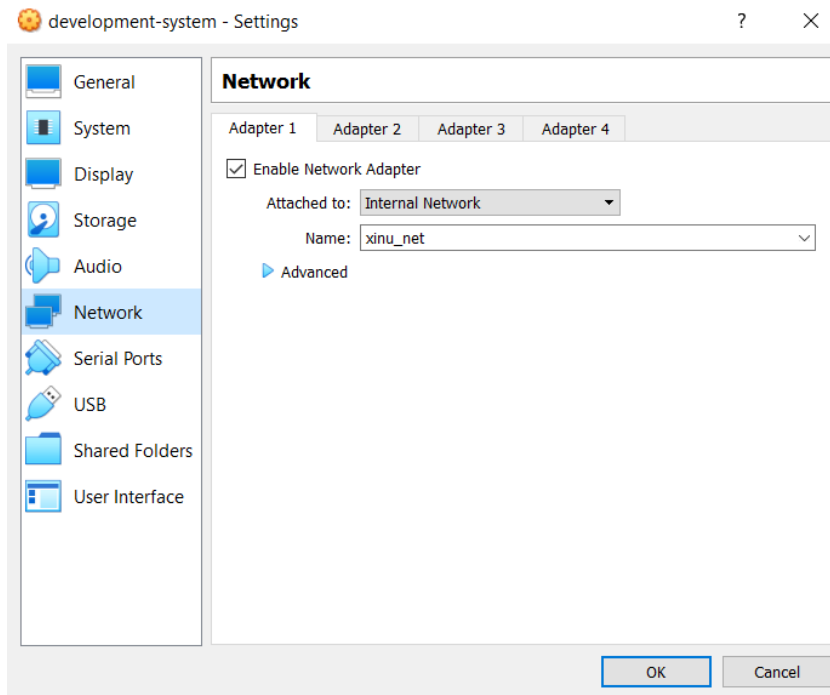
## 1.2 Import dan Setting Development-System VM

1. Jalankan VirtualBox.
2. File -> Import Appliance.
3. Pilih development-system.ova hasil dari tahap sebelumnya.
4. Next (tunggu hingga selesai). Lama waktu tunggu tergantung dari spesifikasi komputer.
5. Akan muncul vm development-system pada panel sebelah kiri. Kemudian pilih "Settings".

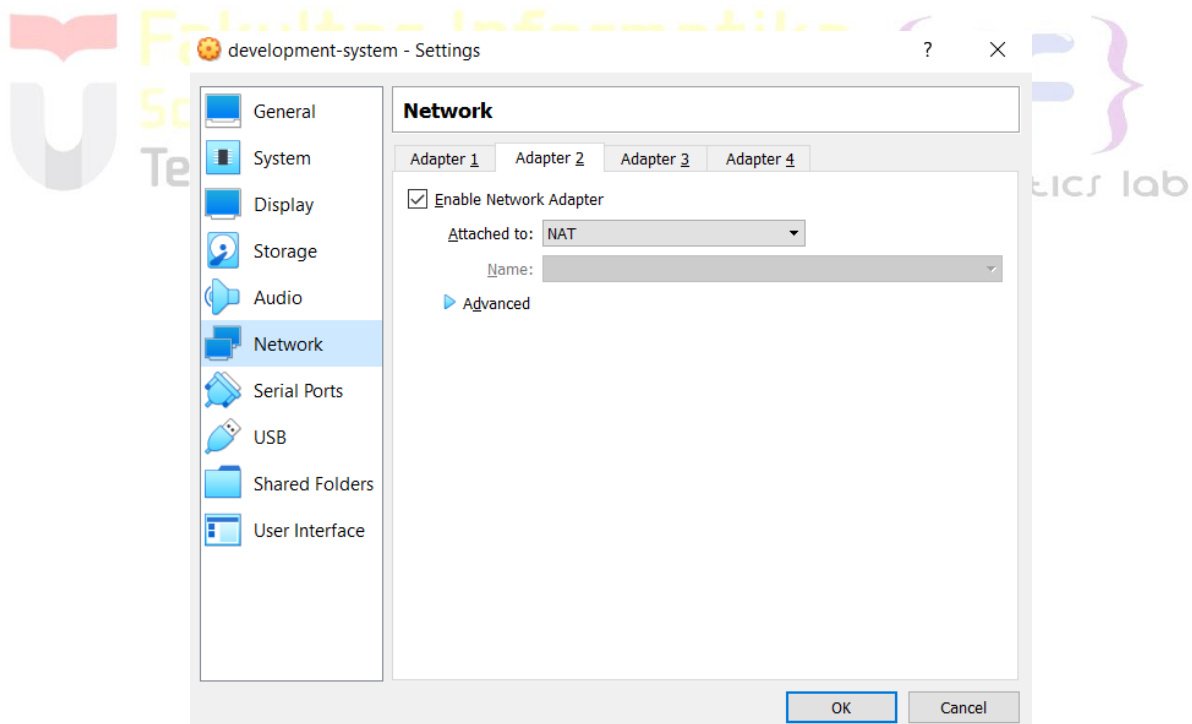


Gambar 1-1 Development System Setting pada VirtualBox

6. Pilih menu "Network" dan ubah setting menjadi seperti gambar di bawah ini:

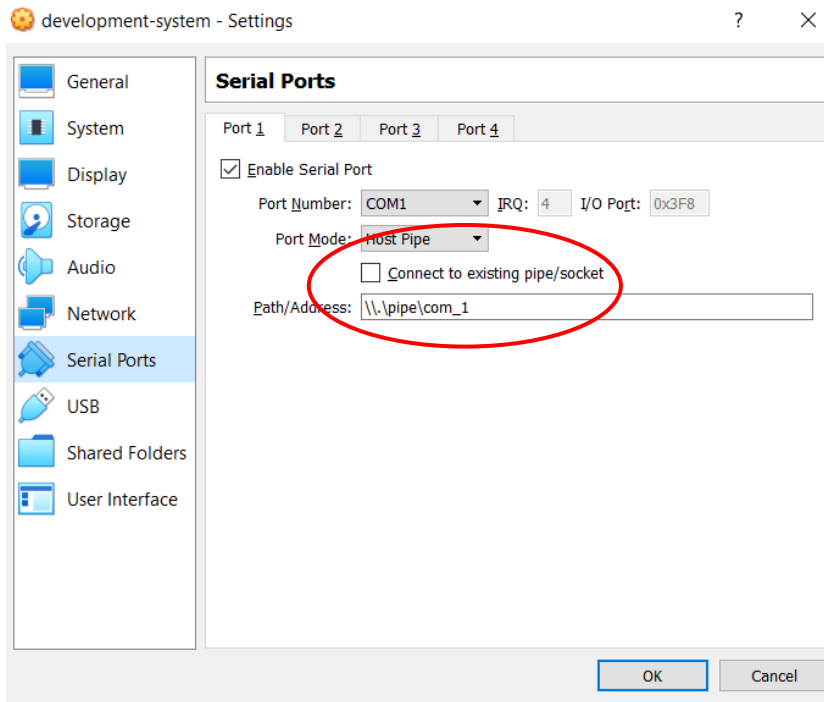


**Gambar 1-2 Development System Network Setting pada Adapter 1**



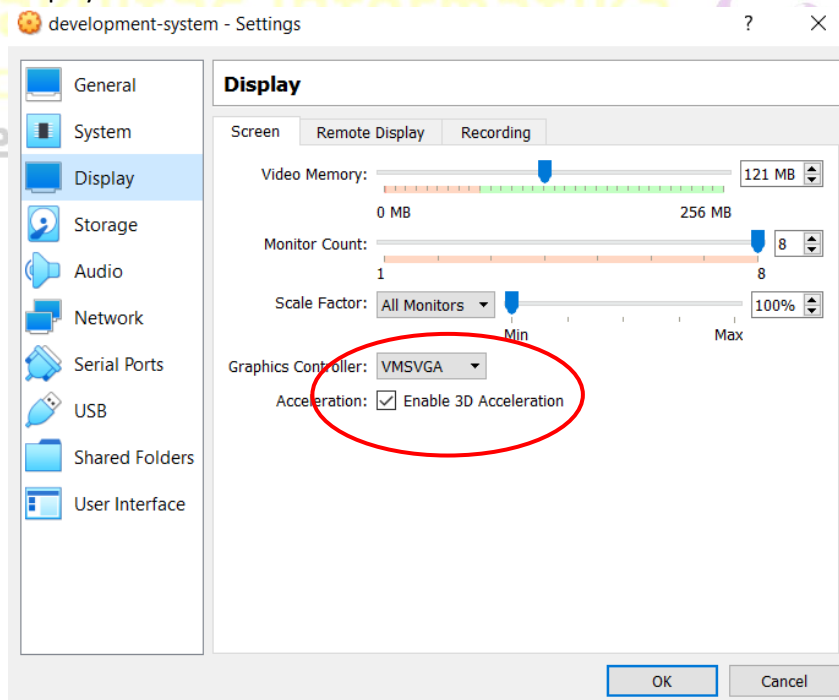
**Gambar 1-3 Development System Network Setting pada Adapter 2**

7. Pilih menu “Serial Ports” dan ubah setting menjadi seperti gambar di bawah ini:  
**Perhatikan “Connect to existing pipe/socket” tidak dipilih**



**Gambar 1-4 Development System Serial Ports Setting pada Port 1**

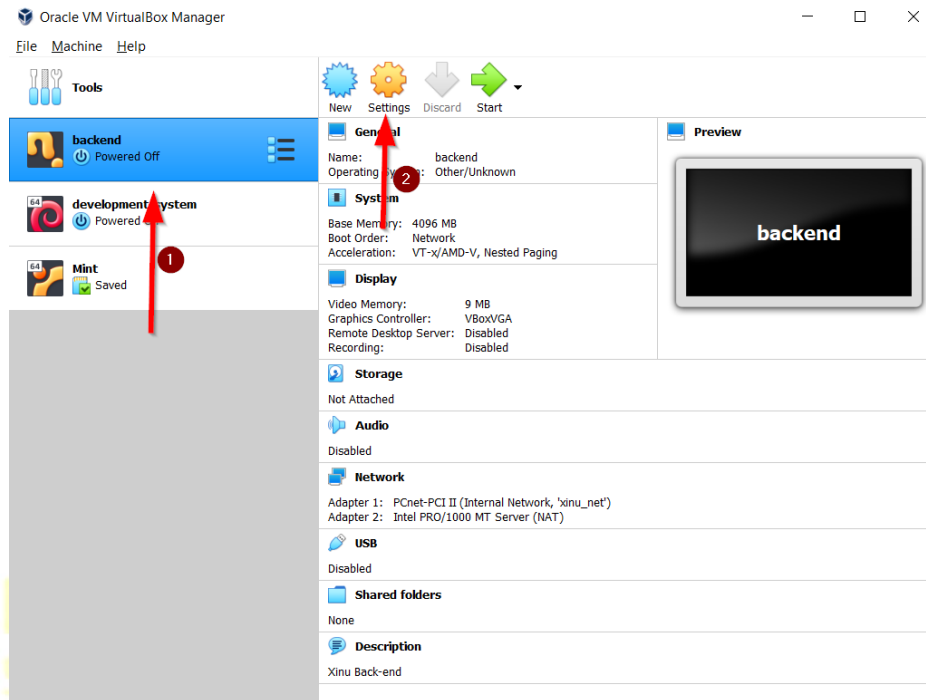
8. Ubah menu “Display”:



**Gambar 1-5 Development System Display Setting**

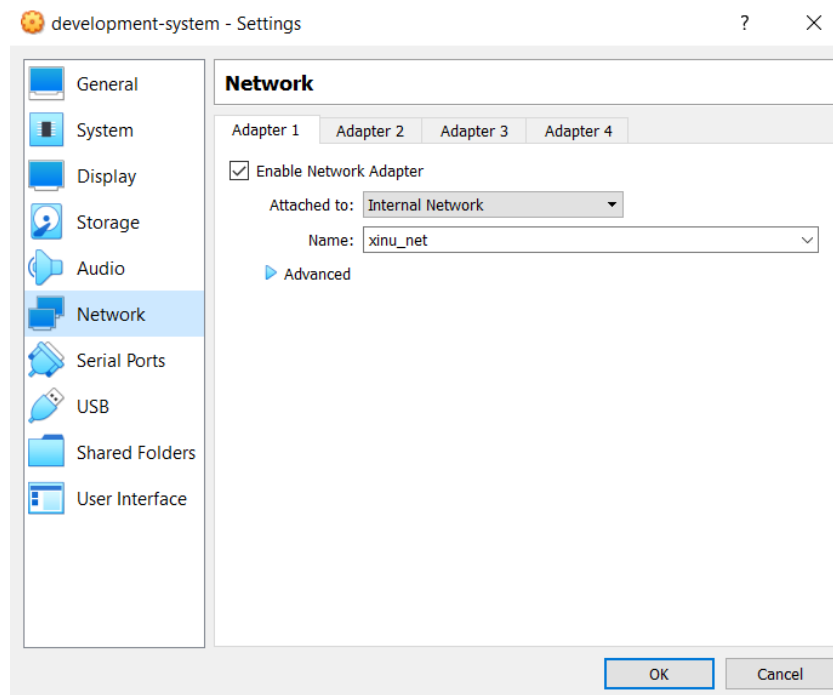
### 1.3 Import dan Setting Backend VM

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”.

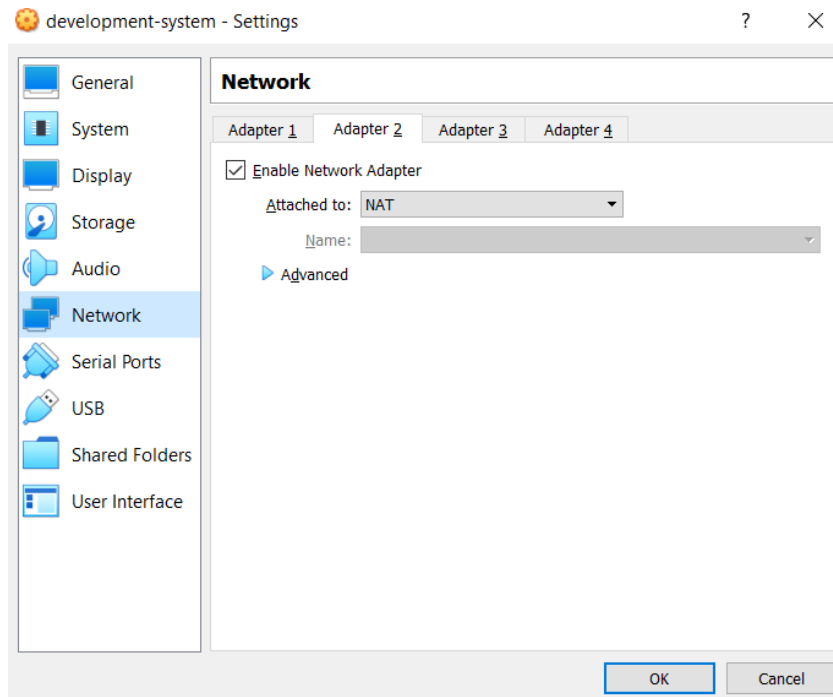


**Gambar 1-6 Backend Setting pada Virtual Box**

6. Pilih Network dan ubah setting menjadi seperti gambar di bawah ini:

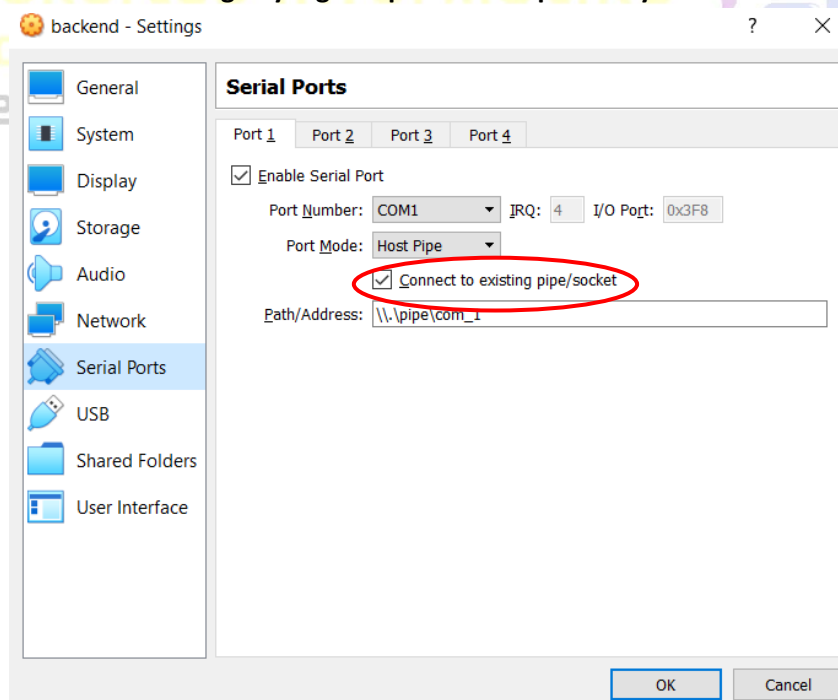


**Gambar 1-7 Backend Network Setting pada Adapter 1**



**Gambar 1-8 Backend Network Setting pada Adapter 2**

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

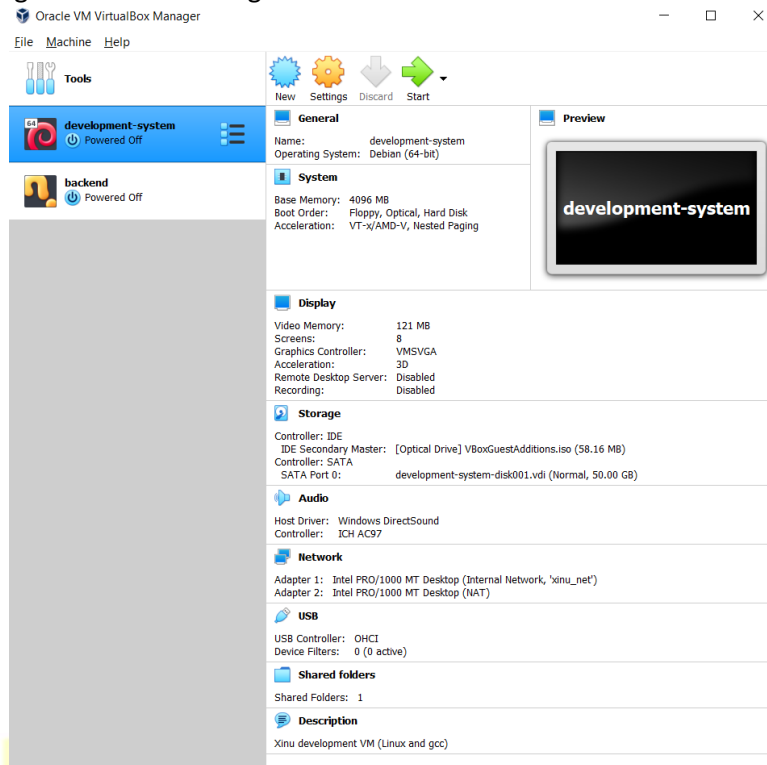


**Gambar 1-9 Backend Serial Ports Setting pada Port 1**

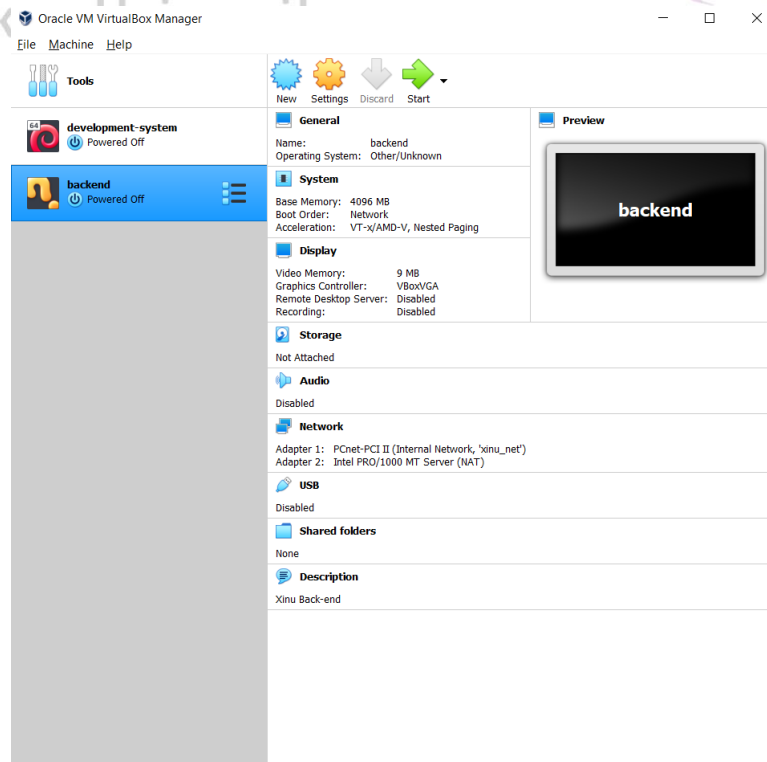


## 1.4 Hasil Akhir

1. Hasil akhir settings Xinu adalah sebagai berikut:



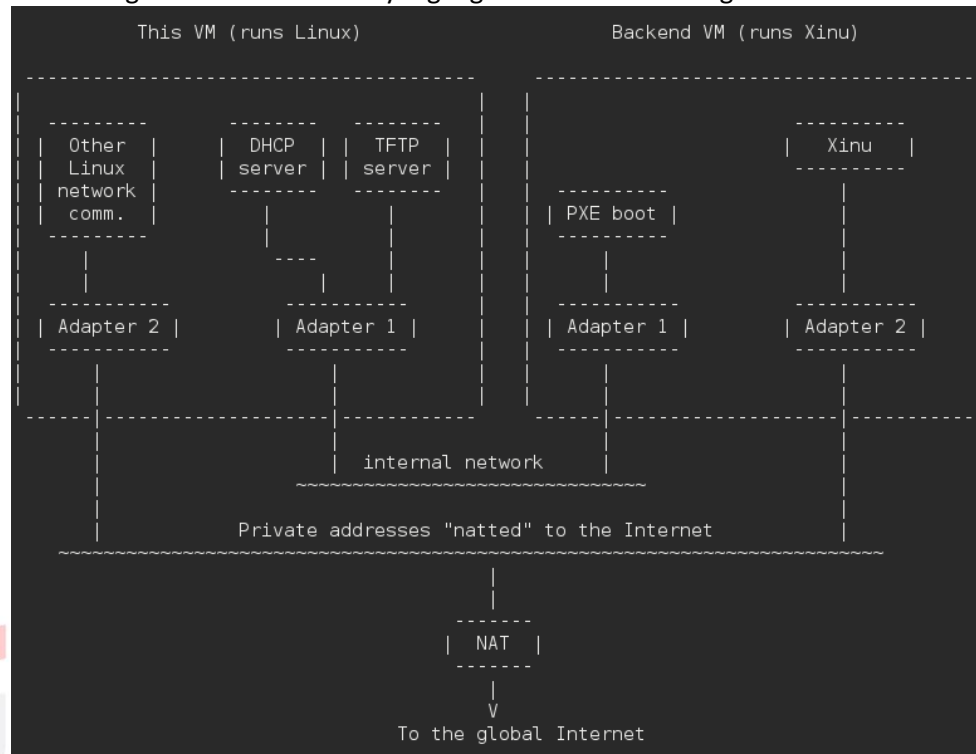
Gambar 1-10 Hasil Akhir Development System



Gambar 1-11 Hasil Akhir Backend

## 2. Arsitektur sistem

Mengapa terdapat 2 buah virtual machine? Xinu OS adalah OS kecil dan elegan yang dikhususkan untuk lingkungan embedded seperti smartphone atau mp3 player. Paradigma yang digunakan adalah programmer menggunakan tool umum (editor, compiler, linker, dll) untuk membuat Xinu image. Xinu image tersebut kemudian akan diupload pada target komputer, target komputer kemudian booting Xinu OS. Arsitektur yang digunakan adalah sebagai berikut ini:



**Gambar 1-12 Arsitektur Xinu**

### Catatan

- “This VM” adalah development-system vm yang disetting sesuai dengan langkah II. VM ini merupakan OS Debian Linux. Pada VM inilah kita akan melakukan pengembangan OS Xinu (edit source code, compile, dst).
- “Backend VM” adalah backend vm yang disetting sesuai dengan langkah III. VM ini merupakan target komputer yang akan menjalankan Xinu OS.

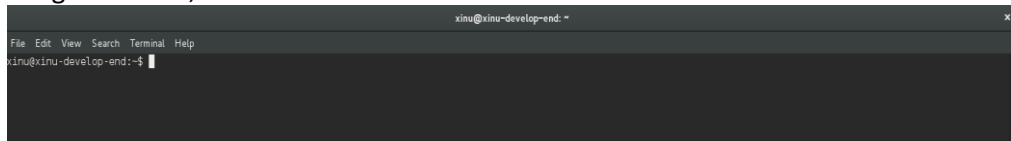
## Modul 2 Eksplorasi Xinu

### Tujuan Praktikum

1. Praktikan dapat menjalankan Xinu.
2. Praktikan dapat menggunakan shell dan perintah-perintah Xinu.

### 2.1 Menjalankan Xinu

1. Jalankan VirtualBox kemudian “Start” development-system vm.
2. Login pada development-sysyem vm menggunakan username: **xinu** password:**xinurocks**
3. Setelah login berhasil, akan muncul terminal berikut ini:

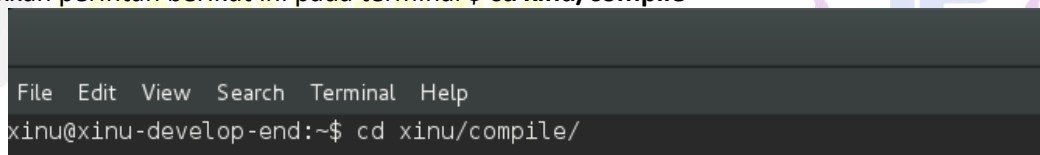


**Gambar 2-1 Login Xinu**

Pada terminal tersebut kita bisa memberikan perintah-perintah kepada komputer. Sebagai contoh perintah “halt”. Perintah tersebut digunakan untuk shutdown komputer. Jika kita mengetikkan perintah tersebut pada terminal maka komputer akan shutdown.

Terdapat banyak perintah pada Linux. Selama praktikum ini kita akan mempelajari beberapa perintah-perintah penting.

4. Ketikkan perintah berikut ini pada terminal **\$ cd xinu/compile**



**Gambar 2-2 Pindah ke Directory Xinu**

Untuk menjalankan perintah tersebut tekan ENTER. **Selalu tekan ENTER setelah mengetikkan perintah.**

Apa arti perintah “cd xinu/compile/”? Perintah tersebut berarti kita melakukan perpindahan direktori menuju direktori xinu/compile. Istilah direktori sama dengan istilah folder pada Windows.

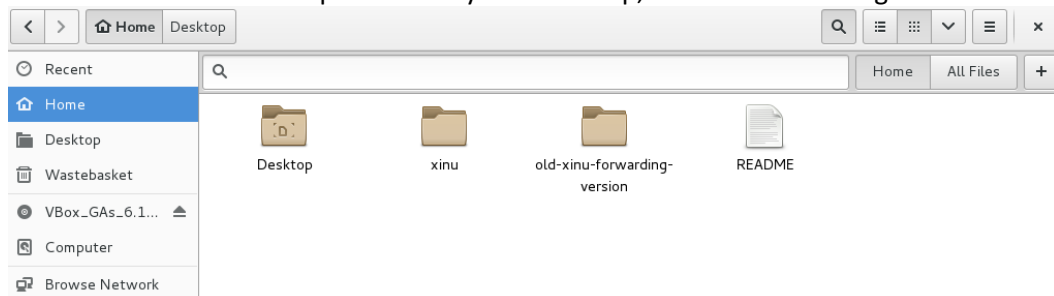
Direktori awal pada saat terminal berjalan umumnya berada pada direktori home (~). Biasanya direktori home adalah nama user (pada kasus kita berarti username xinu).



**Gambar 2-3 Struktur Direktori Pada Xinu**

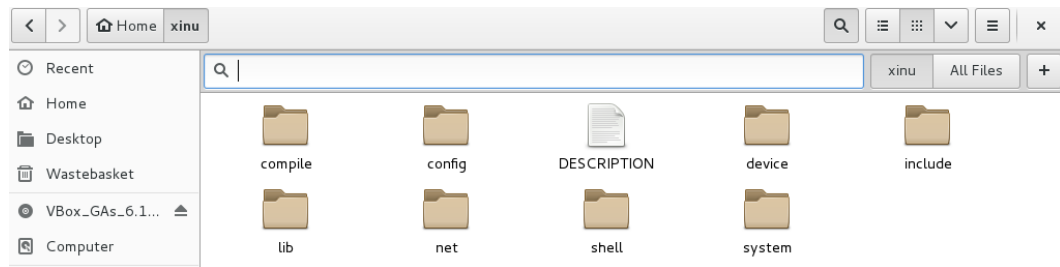
Perhatikan gambar di atas.

- Direktori home kita adalah xinu (direktori paling atas)
- Pada direktori home terdapat 3 folder yaitu: Desktop, old-xinu-forwarding-version dan xinu



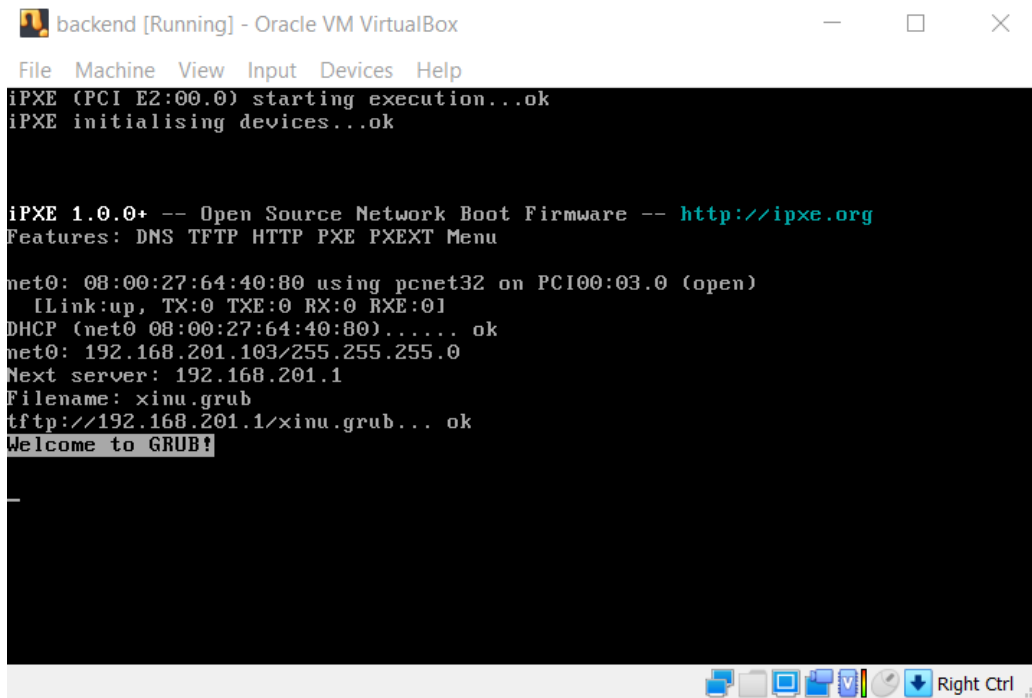
**Gambar 2-4 Direktori Home**

- Jika kita berpindah dari direktori home ke direktori xinu, kita menggunakan perintah **\$ cd xinu**
- Pada direktori xinu terdapat beberapa folder dan file. Folder yang ada dalam direktori xinu adalah compile, config, device, include, lib, net, shell dan system



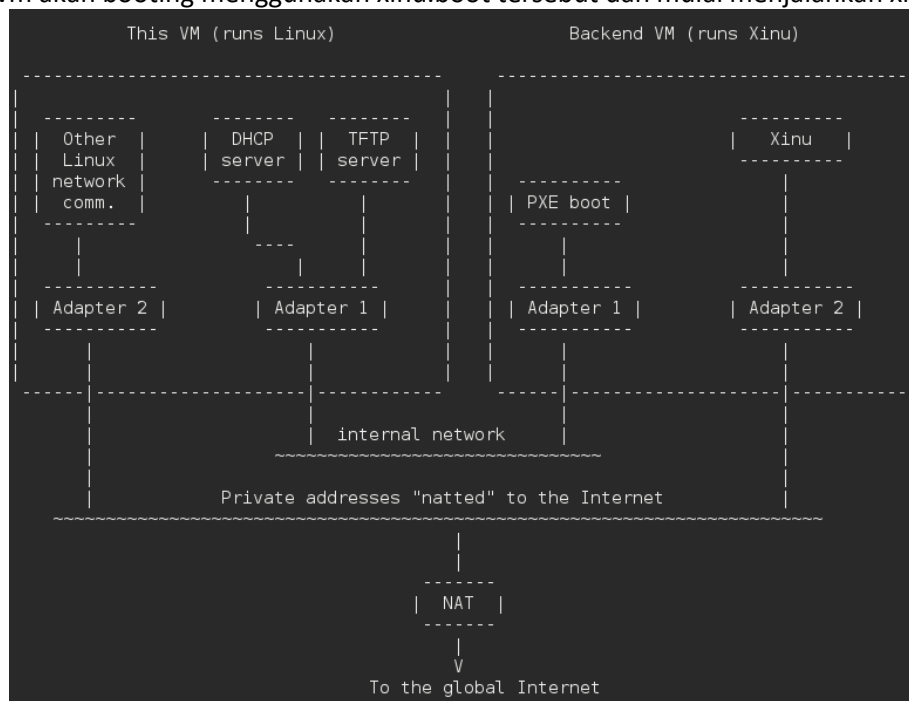
**Gambar 2-5 Direktori Xinu**

- Setelah mengetikkan perintah **\$ cd xinu** berarti sekarang kita berada pada direktori xinu. Jika ingin berpindah ke direktori compile maka kita menggunakan perintah **\$ cd compile**
  - Bagaimana jika ada folder lagi yang ingin kita tuju? Kita bisa memberikan perintah lagi **\$cd nama\_direktori\_tujuan\_kita** begitu seterusnya hingga sampai ke direktori tujuan yang diinginkan
  - Akan membutuhkan banyak usaha untuk menulis jika folder kita terlalu dalam. Solusinya adalah menggunakan path
  - Dari pada menulis **\$cd xinu kemudian \$cd compile** akan lebih mudah jika kita menulisnya sebagai berikut **\$ cd xinu/compile**
5. Ketikkan perintah berikut ini pada terminal **\$ make clean**  
Perintah tersebut digunakan untuk menghapus image xinu os yang ada (untuk memastikan bahwa xinu os yang akan kita compile merupakan xinu os yang terbaru).
  6. Ketikkan perintah berikut ini pada terminal **\$ make**  
Perintah tersebut untuk mengcompile xinu os (proses compile cukup lama tergantung dari spesifikasi komputer yang digunakan).
  7. Dengan mengcompile source code xinu, kita akan mendapatkan image xinu bernama **xinu.elf** yang berada dalam folder xinu/compile/. Image xinu os tersebut (**xinu.elf**) kemudian juga dicopy sebagai **xinu.boot** pada folder server TFTP (/srv/tftp/).
  8. Xinu pada backend vm akan selalu berkomunikasi dengan development-system vm melalui serial port virtual. Pada saat kita mengetikkan perintah pada terminal development-system, perintah tersebut dikirimkan ke backend vm untuk dijalankan oleh xinu.
  9. Untuk berkomunikasi menggunakan serial port digunakan aplikasi bernama minicom. Minicom adalah software untuk berinteraksi dengan serial port pada Linux. Masih berada pada terminal development-system vm jalankan perintah berikut ini **\$ sudo minicom**  
Jika diminta password gunakan password: **xinuroids**  
Sudo diperlukan karena minicom mengakses langsung hardware sehingga butuh root akses
  10. **Sekarang berpindah ke Backend VM.** Jalankan virtualbox kemudian **“Start” virtual machine backend.** Tunggu hingga muncul pesan **“Welcome to GRUB”**. Jika pesan tersebut muncul berarti Xinu telah berhasil dijalankan pada backend vm.



**Gambar 2-6 Hasil Running Backend Xinu**

11. **Cara Xinu Booting.** Virtual machine yang berada pada backend vm tidak berisi apapun. Backend vm hanya bisa melakukan boot melalui jaringan (PXE). OS bisa melakukan booting melalui jaringan menggunakan protokol PXE. Backend vm yang terhubung melalui ethernet virtual ke development-system vm akan memperoleh informasi berupa IP address dan lokasi file image yang bisa didownload dari protokol PXE. DHCP server akan memberikan IP address dan TFTP server akan menyediakan file image xinu.boot. Setelah mendapatkan informasi tersebut (IP address dan lokasi image), backend vm akan mendownload file xinu.boot. Setelah selesai download image, backend vm akan booting menggunakan xinu.boot tersebut dan mulai menjalankan xinu os.



**Gambar 2-7 Arsitektur Xinu dengan Dua VM**

12. Kembali ke terminal pada development-system vm. Jika tidak ada masalah maka akan muncul gambar berikut ini pada terminal development-system vm:

```
[0x00000000 to 0x00FFFFFF]
25649 bytes of Xinu code.
[0x00000000 to 0x00006430]
18011 bytes of data.
[0x00006431 to 0x0000AAB8]
611696 bytes of heap space below 640K.
15728640 bytes of heap space above 1M.
[0x00100000 to 0x00FFFFFF]

-----
XINU
-----

Welcome to Xinu!

xsh $ _
```

*Gambar 2-8 Hasil Running Xinu*

13. Kita bisa memberikan perintah kepada Xinu yang berada pada backend vm melalui terminal tersebut pada development-system vm.

#### Ringkasan Perintah

##### Development-system

```
$ cd xinu/compile
$ make clean
$ make
$ sudo minicom
```

##### Backend

Jalankan backend setelah menjalankan perintah \$ sudo minicom

## 2.2 Eksplorasi Xinu

- Perhatikan terminal Xinu tersebut!
  - Prompt sekarang berubah menjadi **xsh\$**
  - Prompt tersebut menandakan kita sedang berinteraksi dengan Xinu yang ada pada backend vm.
- Memberikan perintah pada Xinu sama dengan memberikan perintah pada Linux. Ketik perintah yang ingin dijalankan dan diakhiri dengan **ENTER**.



3. Perintah-perintah yang ada pada Xinu berbeda dengan perintah pada Linux. Terdapat beberapa perintah yang mirip tetapi cara kerjanya berbeda.
4. Ketik perintah **xsh\$ help**  
Perintah ini menampilkan semua perintah yang ada pada Xinu dan memberikan informasi mengenai perintah yang akan dijalankan. Silahkan eksplorasi semua perintah Xinu yang ada.



## Modul 3 Membaca Source Code Xinu

### Tujuan Praktikum

1. Praktikan memahami struktur direktori source code Xinu.
2. Praktikan melakukan navigasi source code menggunakan tool Source Trail

### 3.1 Paradigma Embedded

Xinu ditujukan untuk sistem embedded, oleh karena itu Xinu mengikuti paradigma cross-development. Pada paradigma cross-development, developer (programmer) akan menggunakan komputer standar pada umumnya (yaitu: PC atau laptop) dan memakai sistem operasi biasa seperti Linux atau Windows. Pada komputer tersebut, developer akan membuat kode, mengedit kode, cross-compile dan cross-link Xinu. Output dari cross-development adalah file image. File tersebut berisi os yang lengkap dan bisa dibooting. Ketika image tersebut telah berhasil dibuat, developer akan mengupload image ke sistem target (embedded system, microcontroller, dll) biasanya melalui jaringan, melalui USB atau kabel serial. Terakhir, developer akan menjalankan komputer sistem target dan Xinu akan berjalan pada target sistem embedded tersebut.

### 3.2 Bahasa Pemrograman yang Digunakan

Sama seperti sistem operasi-sistem operasi hebat lainnya (Unix, Linux, MacOS, dll), bahasa pemrograman yang digunakan adalah bahasa C. Bahasa C dipilih karena bersifat low level yaitu dapat langsung memanipulasi hardware.

### 3.3 Organisasi Source Code Xinu

Berikut adalah struktur direktori yang digunakan dalam mendevlop 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)

Tidak seperti bahasa scripting (contoh: python) yang tidak perlu dicompile dan bisa langsung dijalankan, bahasa c adalah bahasa yang mengharuskan source code untuk dicompile. Hasil proses compile adalah file binari executable (jika pada Windows .exe). Untuk mengcompile source code yang banyak dan kompleks diperlukan tool yang disebut "Make". Tanpa tool "Make" kita harus satu per satu memberikan perintah kepada compiler untuk mengcompile dan hal tersebut akan sangat merepotkan. Make akan mempermudah proses kompilasi. Kita cukup membuat script-script yang akan kita gunakan dan diletakkan pada direktori ./bin. Hasil compile akan ditaruh pada direktori ./binaries.

**./config** Source for the configuration program and Makefile  
**/conf.h** Configuration include file (copied to ../include)  
**/conf.c** Configuration declarations (copied to ../system)

Pada folder ini, berisi program dan **konfigurasi** untuk device driver pada Xinu. Device driver adalah program yang mengendalikan device tertentu pada komputer. Ketika OS memanggil suatu fungsi pada driver maka driver akan memberikan perintah kepada device. Device mengeksekusi perintah tersebut dan memberikan data/informasi ke driver. Driver kemudian akan meneruskan ke OS. Deskripsi lengkap bisa dibaca pada file DESCRIPTION.

**./device** Source code for device drivers, organized into one subdirectory for each device type

<b>/tty</b>	Source code for the tty driver
<b>/rfs</b>	Source code for the remote file access system (both the master device and remote file pseudo-devices)
<b>/eth</b>	Source code for the Ethernet driver
<b>/rds</b>	Source code for the remote disk driver
<b>/...</b>	Directories for other device drivers

Dua folder ini `./config` dan `./device` saling berhubungan. Jika pada `./config` hanya berisi konfigurasi (setting), maka `./device` berisi source code device driver tersebut yaitu cara kerja driver tersebut. Pada `./eth` akan ditemukan cara device driver bekerja, pada `./tty` akan ditemukan cara kerja terminal, dst.

**./include** All include files

`./include` berisi include file (header file) yang digunakan oleh Xinu. Tidak semua include file harus digunakan. Cukup diinclude file yang dibutuhkan pada saat mendevlop program. Sebagai contoh: `semaphore.h` file ini wajib diinclude Ketika kita akan membangun program yang menggunakan semaphore. File `proses.h` juga harus diinclude jika berhubungan dengan proses. Pada C, untuk melakukan include menggunakan kata kunci `#include <xinu.h>`

**./shell** Source code for the Xinu shell and shell commands

Folder ini berisi source code shell dan source code perintah-perintah pada shell. Setiap perintah berkorespondensi dengan satu source code. Sebagai contoh perintah `uptime`. Source code perintah `uptime` adalah `xsh_uptime.c`. Dengan mengubah `xsh_uptime.c` kita bisa mengubah perilaku perintah yang diberikan.

**./system** Source code for Xinu kernel functions

Folder ini merupakan kernel pada Xinu. Manajemen resource, penjadwalan dan semua fungsi sistem operasi ada pada source code ini.

**./lib** Source code for library functions

**./net** Source code for network protocol software

Dua Folder ini berisi library function yang disediakan oleh Xinu. Sebagai contoh, untuk melakukan print maka kita bisa memanggil fungsi `printf.c`. Khusus untuk fungsi-fungsi yang berhubungan dengan jaringan (ping, dns, dll) dikelompokkan pada folder `.net`

### 3.4 File-File Utama Xinu

Berikut adalah beberapa file penting dan akan sering digunakan pada kernel Xinu:

<b>Configuration</b>	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> .
<b>conf.h</b>	Generated by <i>config</i> , it contains declarations and constants including defined names of I/O devices, such as <i>CONSOLE</i> .
<b>conf.c</b>	Generated by <i>config</i> , it contains initialization for the device switch table.
<b>kernel.h</b>	General symbolic constants and type declarations used throughout the kernel.
<b>prototypes.h</b>	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.

### 3.5 Syscall

Secara umum system call pada Xinu akan berkorespondensi dengan nama file yang sama. contoh: syscall untuk resume berada pada file resume.c pada folder ./system.

### 3.6 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.

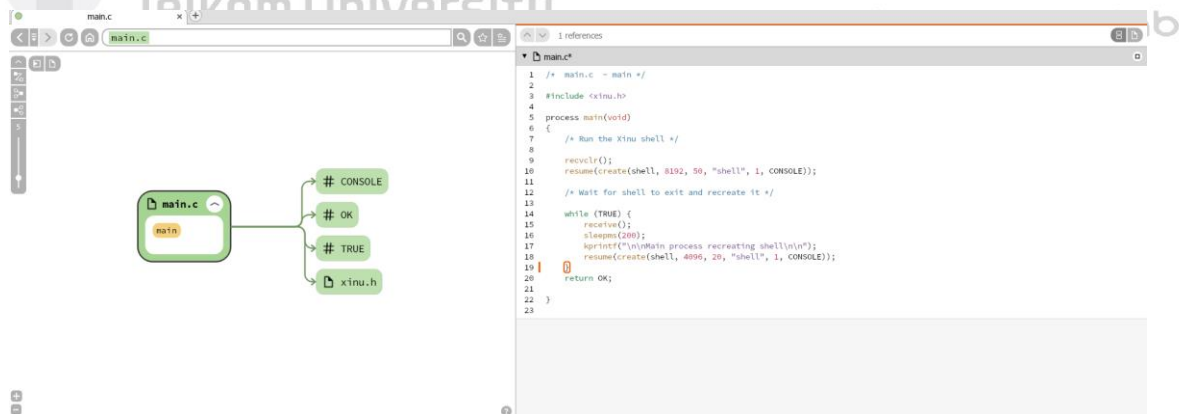
Ukuran\_data: menunjukkan besarnya data yang digunakan. Pada C, ukuran data tergantung pada arsitektur komputer. Tipe *long* berukuran 32 bit pada sebuah komputer dan berukuran 64 bit pada komputer yang lain. Untuk menggaransi ukuran data, Xinu mendefinisikan ukuran secara spesifik. Berikut adalah ukuran 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 3-1 Tipe Data Pada Xinu**

### 3.7 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 3-2 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
- 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



## Modul 4 Proses

### Tujuan Praktikum

1. Praktikan memahami PCB pada Xinu.
2. Praktikan mampu memanfaatkan informasi PCB pada program yang dibuat.

### 4.1 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* tersebut. Entri pada *process table* akan dibuat pada saat proses diciptakan dan entri pada *process table* akan dihapus pada saat proses diterminasi.
2. Beberapa source code utama yang menangani proses adalah:
  - a. `./include/proses.h` yang berisi konfigurasi setiap proses pada Xinu.
  - b. `./system/create.c` untuk membuat proses.
  - c. `./system/kill.c` untuk terminasi proses.
  - d. `./system/resume.c` untuk resume proses.
3. 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 fungsi apapun). Berikut adalah ilustrasi array `proctab[]`:

0	1	2	3	4	5	6
Struct procent{...A.. }	Struct procent{...B.. }	Struct procent{...C.. }	Struct procent{ }	Struct procent{ }	Struct procent{ }	Struct procent{ }

Pada ilustrasi di atas, array `proctab[]` berindex ke 0 akan berisi struct `procent{...A...}`. Isi dari struct `procent` dapat dilihat pada `./include/process.h`

4. 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 4-1 Struktur Procent**

Jadi untuk mengelola suatu proses Xinu cukup mengubah struktur data **struct procent** pada proses yang diinginkan tersebut. Sebagai contoh, jika ingin mengubah nama proses maka kernel cukup mengupdate variabel char prname.

5. Xinu menggunakan implicit data structure yaitu dengan tidak secara eksplisit membuat ID sebagai entri pada struct procent (tidak ada variabel ID pada **struct procent**). Alih-alih, **process ID merupakan indeks pada array proctab[]**. Atau dengan kata lain sebuah proses direferensi hanya oleh indeks pada array proctab[]. Contoh: jika Xinu mengakses proctab[3] artinya mengakses proses dengan pid = 3 dan kemudian Xinu dapat memanipulasi misalkan nilai prioritas proses tersebut melalui struct procent yang ada.

Contoh kode:

```

struct procent*prptr;          /* pointer to process          */

prptr = &proctab[3];          /*mengakses array indeks ke-3 (ID = 3) */

printf("%s", prptr->prname);  /* print nama proses dengan ID = 3 */

```

## Modul 5      Sekuensial dan Konkuren Program

### Tujuan Praktikum

1. Praktikan memahami perbedaan konsep sekuensial dan konkuren pada Xinu
2. Praktikan mampu membuat program konkuren pada Xinu

### 5.1 Pendahuluan

1. Secara intuisi, ketika program berjalan, programmer membayangkan komputer akan melakukan eksekusi kode baris per baris, statement per statement. **Pada setiap waktu komputer hanya mengeksekusi sebuah statement.** Cara pandang tersebut disebut sebagai sekuensial.
2. Sistem operasi memperluas pandangan mengenai komputasi dengan konsep *concurrent processing*. *Concurrent processing* berarti banyak komputasi dapat berlangsung “pada saat yang bersamaan”. Catatan: pada komputer dengan banyak core atau cpu, komputer benar-benar dapat melakukan banyak hal secara bersamaan (*parallelism*). Pada komputer dengan 1 CPU maka OS akan menciptakan ilusi konkurensi.
3. Untuk menciptakan ilusi “berjalan pada saat yang bersamaan”, komputer menggunakan teknik yang disebut multiprogramming/multitasking yaitu sistem operasi memilih proses yang ada dari kumpulan proses yang tersedia, mengeksekusi proses tersebut, memilih proses lain dan kemudian mengeksekusinya. Komputer akan mengeksekusi satu proses dalam beberapa millisecond kemudian berpindah ke proses lainnya. Ketika dilihat oleh manusia, tampak bahwa semua proses berjalan bersamaan karena cepatnya prosesor dan lambatya indera manusia.
4. Terdapat 2 kategori multitasking yaitu: timesharing dan realtime. Timesharing memberikan prioritas yang sama untuk setiap komputasi, membolehkan komputasi berjalan dan berakhir sesuai dengan durasi yang telah ditentukan saja. Realtime memberikan prioritas yang berbeda untuk setiap komputasi karena terdapat performansi waktu yang ingin dicapai. Realtime akan memberikan prioritas dan menjadwalkan proses secara terencana.

### 5.2 Sekuensial Program

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         sleepms(1000);
21     }
22 }
23
24 void sndB(void)
25 {
26
27     while(1){
28         printf("B\n");
29         sleepms(1000);
30     }
31 }
32
33

```

**Gambar 5-1 Contoh Sekuensial Program pada Xinu**

Pada kode tersebut terdapat **sebuah proses** yaitu proses main (baris 7) dan **dua buah fungsi** yaitu sndA() dan sndB(). Ketika program dieksekusi, proses main akan memanggil 2 buah fungsi yaitu sndA() dan sndB(). Jika sudah selesai memanggil dua fungsi tersebut maka proses main akan terminasi. Mohon pahami perbedaan konsep proses dan fungsi!

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() akan terus menerus berjalan karena ada while(1). Jadi output fungsi sndA(): print "A", sleep selama 1 detik, print "A", sleep 1 detik, dst. Fungsi sndB() sama dengan sndA() hanya menampilkan huruf B secara terus menerus.

**Fungsi akan berjalan secara sekuensial. Sedangkan proses akan berjalan secara konkuren.** Konsekuensi dari kode di atas adalah: fungsi sndB() tidak akan pernah dieksekusi karena fungsi sndA()

tidak pernah terminasi (selalu berada pada loop while (1){}). Tidak akan muncul huruf “B”, hanya selalu muncul huruf “A”.

### 5.3 Konkuren Program

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
24 void sndB(void)
25 {
26     while(1) {
27         printf("B\n");
28         sleepms(1000);
29     }
30 }
31
32
33 }
```

**Gambar 5-2 Contoh Konkuren Program pada Xinu**

Perbedaan utama kode ini dengan kode sebelumnya adalah **terdapat 3 proses dalam kode ini yaitu proses main, proses A dan process B**. Pada kode sebelumnya hanya terdapat 1 proses yaitu proses main saja. Untuk membuat proses baru dapat dicapai menggunakan baris ke 10 dan 11. Perhatikan 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 proses bernama “process B” dan menggunakan fungsi sndB().

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 5-3 Deklarasi Syscall create()**

create(nama\_fungsi\_dipanggil, ukuran\_stack, prioritas, nama\_proses, banyaknya\_argument)

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() sehingga ditulis 0

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);

resume(pid\_proses\_yang\_akan\_diresume)

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



## Modul 6 Semaphore

### Tujuan Praktikum

1. Praktikan mampu mengimplementasikan semaphore.
2. Praktikan mampu mengimplementasikan mutex.

### 6.1 Operasi Semaphore

Terdapat 3 operasi yang bisa dilakukan pada semaphore yaitu: inisiasi, signal dan wait.

#### 6.1.1 Inisiasi

Pada Xinu, inisiasi semaphore dilakukan sebagai berikut:

```
nama_semaphore = semcreate(nilai_inisiasi_semaphore)
```

Contoh kode:

```
sid32 s1,s2;           // variabel s1 dan s2 dengan tipe sid32
s1 = semcreate(3);      // inisiasi s1 dengan nilai 3
s2 = semcreate(1);      // inisiasi s2 dengan nilai 1
```

```
resume(create(sndA, 1024, 20, "process A", 1, s1, s2)); //proses A menggunakan 2 semaphore
```

```
resume(create(sndB, 1024, 20, "process B", 1, s1)); // proses B hanya butuh 1 semaphore
```

#### 6.1.2 Signal

Signal berarti increment nilai semaphore. Pada Xinu, semaphore signal menggunakan sintak sebagai berikut:

```
signal(nama_semaphore)
```

Contoh kode:

```
void sndB(sid32 s1)
```

```
{
... kode sebelumnya. // nilai s1 = 3
```

```
signal(s1); //increment nilai s1; jika nilai s1 sebelum dieksekusi adalah 3 maka setelah signal dieksekusi
maka nilai s1 menjadi 4
```

```
...kode sesudahnya // sekarang nilai s1 = 4
}
```

#### 6.1.3 Wait

Wait berarti decrement nilai semaphore. Jika bernilai negatif maka proses yang memanggil wait akan block (perintah setelah wait akan dieksekusi setelah proses diunblock). Pada Xinu, semaphore wait menggunakan sintak sebagai berikut:

```
wait(nama_semaphore)
```

Contoh kode:

```
void sndA(sid32 s1, sid32 s2)
{
... kode sebelumnya. // misalkan nilai s1 = 3; s2 = 0

wait(s1); // nilai s1 sekarang menjadi 2
print ("A"); // perintah ini akan dieksekusi
wait(s2); // nilai s2 akan menjadi -1 karena negatif maka proses A akan diblock
print("B"); // perintah ini akan dieksekusi setelah proses A diunblock oleh proses lain

...kode sesudahnya //
}
```

## 6.2 Pola Pada Semaphore

Pola-pola yang sering digunakan dalam semaphore adalah **signaling dan mutex**.

### 6.2.1 Signaling

Pola signaling digunakan sebagai cara untuk **memastikan** bahwa suatu proses awal (P1) telah selesai dan baru proses selanjutnya (P2) bisa dijalankan.

Berikut adalah pseudo codenya:

**Inisiasi semaphore:**

s1 = 1 (P1 harus berjalan dahulu sehingga diinisiasi dengan 1).

s2 = 0 (P2 berjalan setelah P1 sehingga harus menunggu terlebih dahulu dan diinisiasi dengan nilai 0).

**Pola semaphore:**

**Tabel 6-1 Pola Signaling**

P1	P2
wait(s1)	wait(s2)
//P1 melakukan sesuatu	//P2 melakukan sesuatu
signal(s2)	signal(s1)

Perhatikan letak signal(s1), signal(s2), wait(s1) dan wait(s2)! Letaknya harus mengikuti pola di atas untuk memastikan signaling terjadi.

Contoh kode:

```
sndA(sid32 s1, sid32 s2){

wait(s1);
print ("A");
signal(s2);

}
```



```
sndB(sid32 s1, sid32 s2){
```

```
wait(s2);
print ("B");
signal(s1);
```

```
}
```

Memastikan "A" diprint sebelum "B".

### 6.2.2 Mutex

Pola mutex digunakan untuk memastikan bahwa suatu variable (resource) hanya dapat diakses oleh satu proses saja.

#### Inisiasi mutex:

sem\_1 = 1 (mutex selalu diinisiasi dengan 1. Jika ingin suatu resource bisa diakses secara bersamaan oleh n proses maka mutex diinisiasi dengan n).

Contoh kode:

```
sid32 mutex;
m1 = semcreate(1); // mutex selalu diinisiasi dengan 1
```

#### Pola mutex:

Tabel 6-2 Pola Mutex

P1	P2
wait(m1)	wait(m1)
//Critical section, hanya boleh 1 proses akses	//Critical section, hanya boleh 1 proses akses
signal(m1)	signal(m1)

Jika terdapat P3, pola tetap sama yaitu wait(m1); critical section; signal(m1)

P3
wait(m1)
//Critical section, hanya boleh 1 proses akses
signal(m1)

Contoh kode:

```
sndA(sid32 m1)
{
wait(m1);
saldo = saldo + setor; //critical section
signal(m1);
```

```
}  
  
sndB(sid32 m1)  
{  
  wait(m1);  
  saldo = saldo + setor; //critical section  
  signal(m1);  
}
```



## Modul 7 Syscall

### Tujuan Praktikum

1. Praktikan mampu memahami syscall pada Xinu.
2. Praktikan mampu mengimplementasikan syscall pada Xinu

### 7.1 Definisi dan Fungsi Syscall

Syscall didefinisikan sebagai interface layanan yang diberikan oleh sistem operasi. Proses akan menggunakan syscall jika ingin mendapatkan layanan dari os. Selain itu, syscall juga memberikan lapisan perlindungan keamanan dan information hiding. Proses tidak perlu mengetahui detail internal syscall. Proses cukup memanggil syscall yang diinginkan, kernel os akan mengeksekusinya dan proses menerima hasilnya (berhasil atau gagal syscall tersebut). Syscall berbeda dengan fungsi yang dibuat oleh developer misal fungsi printf().

*System calls, which define operating system services for applications, protect the system from illegal use and hide information about the underlying implementation.*

**Gambar 7-1 Definisi dan Fungsi Syscall**

### 7.2 Cara Kerja Syscall

- a. Disable semua interupsi pada saat syscall dipanggil

Hal ini dilakukan untuk memastikan bahwa perubahan struktur data global yang dilakukan pada saat syscall konsisten untuk seluruh os. Syscall harus menjamin bahwa tidak ada proses lain yang mengubah struktur data os pada saat yang bersamaan atau dapat menghasilkan struktur data yang tidak konsisten.

Selain untuk memastikan struktur data konsisten secara global, disable interupsi juga mencegah terjadinya context switching. Jangan sampai terjadi context switching ke proses lain pada saat syscall sedang dijalankan.

*System calls must disable interrupts to prevent other processes from changing global data structures; using a disable / restore paradigm increases generality.*

**Gambar 7-2 Disable Interrupts Syscall**

- b. Memeriksa seluruh argument yang diberikan oleh proses(caller).

Syscall tidak boleh membuat asumsi apapun mengenai proses yang memanggil. Syscall akan memeriksa semua argument dan ijin yang ada. Jika terdapat argument yang tidak sesuai atau tidak mempunyai ijin maka syscall akan menolak layanan yang diminta oleh proses. Contoh: syscall sleep() hanya menerima argument integer positif. Jika argument adalah string atau bilangan negatif maka akan ditolak.

- c. Mengesekusi layanan yang diminta oleh process (caller). Inilah tugas utama pada syscall. Contoh: jika syscall adalah freemem() maka fungsi utama adalah membebaskan memori block pada alamat tertentu.
- d. Restore interupsi sehingga proses lain bisa menggunakan syscall/context switch.
- e. Memberikan laporan hasil (sukses/gagal) kepada proses (caller).

### 7.3 Template Syscall Xinu

Untuk menjamin semua syscall berjalan dengan baik maka semua syscall Xinu menggunakan template berikut ini:

```
syscall function_name ( args ) {
    intmask mask;           /* Saved interrupt mask          */
    mask = disable( );      /* Disable interrupts at start of function */
    if ( args are incorrect ) {
        restore(mask);      /* Restore interrupts before error return */
        return SYSERR;
    }
    ... other processing ...
    if ( an error occurs ) {
        restore(mask);      /* Restore interrupts before error return */
        return SYSERR;
    }
    ... more processing ...
    restore(mask);          /* Restore interrupts before normal return*/
    return appropriate value ;
}
```

**Gambar 7-3 Template Syscall Xinu**

### 7.4 Syscall Resume

Berikut adalah contoh kode syscall resume. Syscall resume() akan mengubah status proses dari state suspend ke state ready dan akan mengembalikan (return value) prioritas proses ready tersebut.

```

5  /*-----
6  * resume - Unsuspend a process, making it ready
7  *-----
8  */
9  pri16 resume(
10     pid32 pid /* ID of process to unsuspend */
11 )
12 {
13     intmask mask; /* Saved interrupt mask */
14     struct proctab *prptr; /* Ptr to process's table entry */
15     pri16 prio; /* Priority to return */
16
17     mask = disable(); 1
18     if (isbadpid(pid)) {
19         restore(mask); 2
20         return (pri16) SYSERR;
21     }
22     prptr = &proctab[pid];
23     if (prptr->prstate != PR_SUSP) { 3
24         restore(mask);
25         return (pri16) SYSERR;
26     }
27     prio = prptr->prprio; /* Record priority to return */
28     ready(pid); 4
29     restore(mask); 5
30     return prio; 6
31 }

```

Gambar 7-4 Kode Syscall resume()

Menggunakan template yang diberikan berikut adalah penjelasan detail cara kerja syscall resume: Setelah mendeklarasikan variabel-variabel yang akan digunakan (baris 13-15),

[1] syscall disable interupsi (baris 17)

[2] syscall memeriksa validitas argument yang diberikan . PID tidak boleh negatif, terdapat batas maksimum PID , dst. Jika PID tidak memenuhi syarat maka restore interupsi dan berikan hasil SYSERR.

[3] dan [4] merupakan pekerjaan utama syscall. Pertama pastikan bahwa proses dalam status suspend (baris 23) jika proses tidak dalam state suspend maka berikan hasil SYSERR. Jika dalam state suspend maka ubah menjadi state ready (baris 28). Fungsi ready() adalah fungsi biasa bukan syscall.

[5] dan [6] Enable interupsi dan berikan hasil syscall yaitu prioritas proses (prio).

## 7.5 Membuat Syscall Baru pada Xinu

Cara membuat Syscall baru pada Xinu:

- Buat file baru pada folder ./system
- Edit file tersebut dan pastikan mengikuti template code syscall Xinu.
- Edit ./include/prototypes.h dengan menambahkan nama fungsi yang dibuat.
- Compile Xinu dengan tambahan syscall baru dan bisa digunakan.

Contoh:

Kita akan membuat dummy syscall bernama chname(). Syscall ini tidak melakukan apapun hanya menampilkan kalimat "New Syscall is easy". Template diambil dari chprio.c

- a. \$ touch ./system/chname.c
- b. Berikut isi dari chname.c

```
/* chname.c - chname */
#include <xinu.h>

/*-----
 * chname - Dummy syscall
 *-----
 */
pri16 chname(
    pid32    pid,          /* ID of process to change */
    pri16    newprio       /* New priority */
)
{
    intmask mask;          /* Saved interrupt mask */
    struct procent *prptr;  /* Ptr to process's table entry */
    pri16    oldprio;       /* Priority to return */

    mask = disable();
    if (isbadpid(pid)) {
        restore(mask);
        return (pri16) SYSERR;
    }
    prptr = &proctab[pid];
    oldprio = prptr->prprio;
    prptr->prprio = newprio;
    kprintf("\n\nNew Syscall is easy\n\n");
    restore(mask);
    return oldprio;
}
```

**Gambar 7-5 Contoh Kode Syscall Baru Pada Xinu**

Pada dasarnya sama dengan chprio.c, hanya ditambahkan kprintf() sebelum restore(mask).

- c. Edit ./include/prototypes.h

```
/* in file asctime.c */
extern status asctime(uint32, char *);

/* in file bufinit.c */
extern status bufinit(void);

/* in file chprio.c */
extern pri16 chprio(pid32, pri16);

/* in file chname.c */
extern pri16 chname(pid32, pri16); 1

/* in file clkupdate.S */
extern uint32 clkcount(void);

/* in file clkhandler.c */
extern interrupt clkhandler(void);
```

**Gambar 7-6 Kode File Prototypes.h**

Kata kunci extern berarti fungsi tersebut (syscall) dapat diakses secara global oleh semua fungsi lainnya.

pri16 adalah return value syscall/fungsi.

chname(pid32, pri16) adalah nama fungsi (syscall) beserta parameternya.

- d. Compile
  - \$ make clean
  - \$ make

## 7.6 Cara Penggunaan Syscall

Setelah syscall dibuat maka syscall sudah bisa digunakan. Cara menggunakan syscall cukup dengan langsung memanggil syscall tersebut.

Contoh: Kita akan edit xsh\_uptime.c dan menambahkan syscall chname() sehingga setiap kali perintah uptime dieksekusi syscall chname() juga akan dieksekusi.

- a. Buka ./shell/xsh\_uptime.c
- b. Edit hingga isinya xsh\_uptime.c seperti ini

```
printf("Xinu has been up ");
if (days > 0) {
    printf(" %d day(s) ", days);
}

if (hrs > 0) {
    printf(" %d hour(s) ", hrs);
}

if (mins > 0) {
    printf(" %d minute(s) ", mins);
}

if (secs > 0) {
    printf(" %d second(s) ", secs);
}
printf("\n");
chname(1,20);
return 0;
}
```

**Gambar 7-7 Contoh Penggunaan Syscall Pada Perintah Uptime**

- c. Compile

## Modul 8 Shell

### Tujuan Praktikum

1. Praktikan mampu memahami shell pada Xinu.
2. Praktikan mampu mengimplementasikan perintah baru pada Shell.

### 8.1 Shell

1. Pada modul sebelumnya telah dijelaskan mengenai syscall (layanan yang disediakan oleh sistem operasi). Akan tetapi, user normal tidak akan pernah/jarang mengakses syscall secara langsung. Biasanya user mengakses aplikasi dan aplikasi tersebut yang akan mengakses syscall.
2. Shell adalah **program** yang memproses perintah yang diberikan oleh user pada terminal. Shell akan looping secara terus menerus membaca baris input yang diberikan. Setelah sebuah baris input dibaca ditandai dengan adanya ENTER, shell harus mengekstrak nama perintah, argumen dan hal-hal lainnya. Jika proses ekstraksi berhasil maka perintah akan dieksekusi sesuai dengan argumen yang diberikan.

Contoh perintah pada shell:

ls -al

nama\_perintah = "ls"

argument 1 = "-al"

3. Cara kerja shell dapat dipelajari pada source code ./shell/shell.c.

```
/* Print shell banner and startup message */
fprintf(dev, "\n\n%s%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",
    SHELL_BAN0, SHELL_BAN1, SHELL_BAN2, SHELL_BAN3, SHELL_BAN4,
    SHELL_BAN5, SHELL_BAN6, SHELL_BAN7, SHELL_BAN8, SHELL_BAN9);

fprintf(dev, "%s\n\n", SHELL_STRMSG);

/* Continually prompt the user, read input, and execute command */
while (TRUE) {
    /* Display prompt */
    fprintf(dev, SHELL_PROMPT);

    /* Read a command */
    len = read(dev, buf, sizeof(buf));

    /* Exit gracefully on end-of-file */
    if (len == EOF) {
        break;
    }

    /* If line contains only NEWLINE, go to next line */
    if (len <= 1) {
        continue;
    }

    buf[len] = SH_NEWLINE; /* terminate line */

    /* Parse input line and divide into tokens */
```

**Gambar 8-1 Shell.c**



4. Perintah-perintah Linux (shell command) berada pada file system/harddisk (contoh: perintah ls berada pada direktori /bin/ls). Pada Xinu, shell command berada pada image (yaitu image yang dibuat pada kompilasi Xinu). Jadi untuk menambahkan atau mengurangi shell command (perintah pada shell), Xinu harus dicompile ulang seluruhnya.

5. Berikut adalah cara menambahkan perintah baru “mycmd” pada Xinu

a. Edit shell.c yang berada pada direktori xinu/shell/

**Tambahkan {“mycmd”, FALSE, xsh\_mycmd}, pada struct cmdent cmdtab[]**

```

/*****
/* Table of Xinu shell commands and the function associated with each */
*****/

const struct cmdent cmdtab[] = {
    {"argecho",    TRUE,  xsh_argecho},
    {"arp",        FALSE, xsh_arp},
    {"cat",        FALSE, xsh_cat},
    {"clear",      TRUE,  xsh_clear},
    {"date",       FALSE, xsh_date},
    {"devdump",    FALSE, xsh_devdump},
    {"echo",       FALSE, xsh_echo},
    {"exit",       TRUE,  xsh_exit},
    {"help",       FALSE, xsh_help},
    {"kill",       TRUE,  xsh_kill},
    {"memdump",    FALSE, xsh_memdump},
    {"memstat",    FALSE, xsh_memstat},
    {"netinfo",    FALSE, xsh_netinfo},
    {"ping",       FALSE, xsh_ping},
    {"ps",         FALSE, xsh_ps},
    {"sleep",      FALSE, xsh_sleep},
    {"udp",        FALSE, xsh_udpdump},
    {"udpecho",    FALSE, xsh_udpecho},
    {"udpserver",  FALSE, xsh_udpserver},
    {"uptime",    FALSE, xsh_uptime},
    {"mycmd",      FALSE, xsh_mycmd},
    {"?",         FALSE, xsh_help}
};

```

**Gambar 8-2 Struct cmdent cmdtab[]**

b. Edit “shprototype.h” yang berada pada folder xinu/include/  
 Tambahkan extern shellcmd xsh\_mycmd (int32, char \*[]);

```

/* in file xsh_sleep.c */
extern shellcmd xsh_sleep      (int32, char *[]);

/* in file xsh_udpdump.c */
extern shellcmd xsh_udpdump    (int32, char *[]);

/* in file xsh_udpecho.c */
extern shellcmd xsh_udpecho     (int32, char *[]);

/* in file xsh_udpserver.c */
extern shellcmd xsh_udpserver   (int32, char *[]);

/* in file xsh_uptime.c */
extern shellcmd xsh_uptime      (int32, char *[]);

/* in file xsh_help.c */
extern shellcmd xsh_help        (int32, char *[]);

extern shellcmd xsh_mycmd       (int32, char *[]);

```

**Gambar 8-3 Penambahan extern shellcmd xsh\_mycmd Pada shprototype.h**

- c. Buat file baru bernama xsh\_mycmd.c pada folder xinu/shell

```

/* xsh_mycmd.c - xsh_mycmd */

#include <xinu.h>
#include <stdio.h>
#include <string.h>
// deklarasi nama fungsi
static void printMemUse(void);

shellcmd xsh_mycmd(int nargs, char *args[]) {

    /* For argument '--help', emit help about the 'ping' command */
    // mycmd --help
    // jika perintah adalah "mycmd --help" tampilkan help
    if (nargs == 2 && strcmp(args[1], "--help", 7) == 0) {
        printf("Use: %s argument1 argument2\n\n", args[0]);
        printf("Description:\n");
        printf("\tMy own command with 2 argument\n");
        printf("Options:\n");
        printf("\t--help\t display this help and exit\n");
        return 0;
    }

    /* Check for valid number of arguments */
    // banyaknya argument harus 2; args[0] = nama_perintah args[1]= argumen 1 args[2] = argumen 2
    if (nargs != 3) {
        fprintf(stderr, "%s: jumlah argumen tidak cocok\n", args[0]);
        fprintf(stderr, "Try '%s --help' for more information\n",
                args[0]);
        return 1;
    }

    printMemUse();
    printf("Nama perintah adalah: %s \n", args[0]);
    printf("Argumen pertama adalah: %s \n", args[1]); // 123
    printf("Argumen kedua adalah: %s \n", args[2]); // xxx
    return 0;
}

/*-----
 * printMemUse - Print statistics about memory use
 *-----
*/

```

**Gambar 8-4 xsh\_mycmd.c**

```

/*-----
 * printMemUse - Print statistics about memory use
 *-----
 */
void printMemUse(void){
    printf("My memory usage is xxxx\n");
}

```

**Gambar 8-5 Fungsi printMemUse**

d. Compile ulang Xinu

Pada development-system lakukan hal-hal berikut ini:

- cd /home/xinu/xinu/compile
- make clean
- make
- sudo minicom

e. Test hasil perintah baru pada shell

- xsh \$ mycmd
- xsh \$ mycmd 111
- xsh \$ mycmd 111 xxx



**Fakultas Informatika**  
**School of Computing**  
**Telkom University**



informatics lab

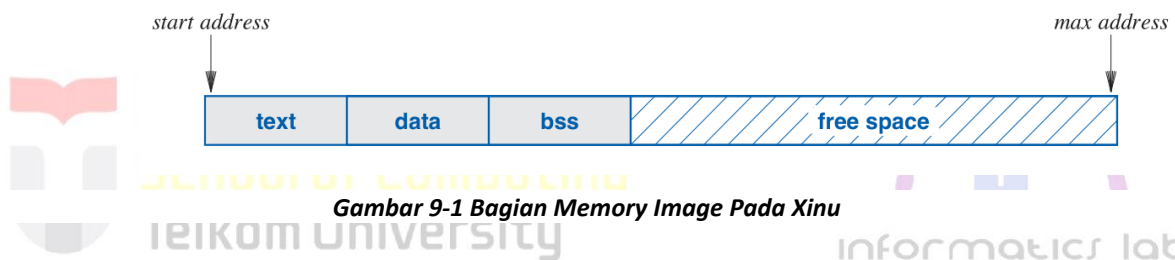
## Modul 9 Memori

### Tujuan Praktikum

1. Praktikan mampu memahami struktur memori Xinu.

### 9.1 Memori

1. Pada Xinu, memory image dibagi menjadi menjadi 4 bagian:
  - a. Text segment: berisi kode yang telah dicompile dalam bahasa mesin (instruksi mesin) untuk semua fungsi. Hasil kompilasi fungsi `printf()`, perintah-perintah shell berada pada segmen ini. Text segmen berada pada bagian alamat memory paling bawah.
  - b. Data segment: berisi semua variable global yang telah diinisiasi nilainya, berada setelah text segment.
  - c. Bss segment (block started by symbol segment): berisi global variable yang tidak diinisiasi, berada setelah data segment.
  - d. Free space: alamat setelah bss segment



Gambar 9-1 Bagian Memory Image Pada Xinu

2. Image di atas adalah ilustrasi image Xinu pada saat belum berjalan (static). Apa yang terjadi jika Xinu berjalan/booting? Pada saat booting maka akan dibuat proses-proses baru. Proses-proses baru tersebut akan menempati free space yang ada. Setiap kali ada proses baru akan dialokasikan free space yang ada dan ketika proses terminasi maka alokasi proses tersebut pada free space akan dihapus. Adanya alokasi dan dealokasi pada free space sangat dinamis tergantung dari kekomplekan kode.
3. Bagian **free space** pada memory dapat digunakan untuk alokasi dinamis memory. Terdapat 2 jenis dinamis memori:
  - a. Stack: pada saat suatu fungsi dipanggil maka akan dibuatkan stack untuk fungsi tersebut. Pada saat fungsi telah selesai maka stack akan dihapus.
  - b. Heap: digunakan ketika fungsi meminta/membuat/membutuhkan alokasi memori baru (pada c misalkan menggunakan perintah `malloc()`). Contoh: pada saat membaca file, konten file disimpan dahulu pada memori. Untuk menyimpan konten membutuhkan memori sehingga perlu dialokasikan memori heap menggunakan `malloc()`.

Berikut adalah perbedaan lain antara stack dan heap. Stack dialokasikan dari alamat maksimum memori, sedangkan Heap dialokasikan dari alamat terendah dari free space. Stack terstruktur yaitu LIFO sedangkan heap bebas dialokasikan dan dealokasi kapanpun.

#### 4. Ilustrasi stack dan heap



**Gambar 9-2 Ilustrasi Memori**

Gambar di atas merupakan ilustrasi memori setelah 3 buah proses dibuat dan salah satu proses membuat alokasi memori. Pada saat proses 1 dibuat maka proses 1 berada pada stack 1. Ketika proses 2 dibuat maka proses 2 berada pada stack 2. Begitu juga untuk proses 3 sehingga proses 3 berada pada stack 3. Perhatikan bahwa stack dialokasikan dari alamat memori paling tinggi menuju alamat yang rendah.

Misalkan proses 3 membutuhkan memori heap. Xinu akan mengalokasikan heap mulai dari alamat free memory yang paling rendah. Begitu seterusnya hingga free space tidak mampu menampung lagi stack atau heap.

#### 5. Implementasi manajemen memori pada Xinu

Pada Xinu terdapat empat fungsi utama untuk manajemen memori:

- getstk() - alokasi stack pada saat proses dibuat (create)
- freestk() – release stack pada saat proses terminasi
- getmem() – alokasi heap sesuai permintaan
- freemem() – release heap sesuai permintaan

6. Alokasi dan dealokasi stack hanya terjadi pada saat proses dibuat dan diterminasi. Hal ini menjamin bahwa dealokasi stack pasti terjadi pada saat proses terminasi. Akan tetapi, ada permasalahan pada dealokasi heap. Pada Heap, saat proses yang meminta alokasi heap terminasi maka heap tidak secara otomatis dihapus. Proses tersebut harus secara manual dan eksplisit menghapus heap yang dibuat proses tersebut. Masalah lain pada heap adalah permintaan akan heap bisa melebihi free space yang ada. Heap juga akan mengalami fragmentasi.

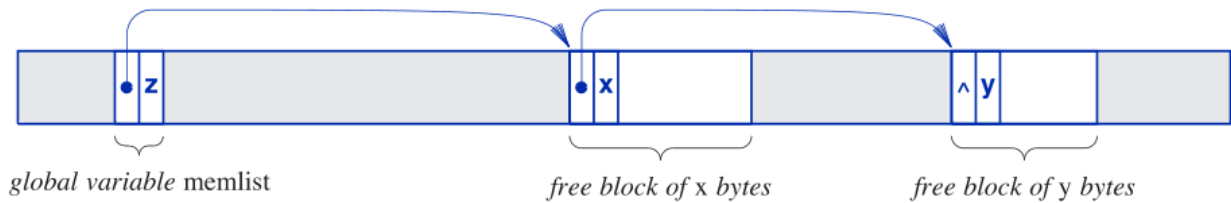
7. Untuk mengatasi permasalahan tersebut maka Xinu harus menyimpan informasi mengenai semua free block memory. Xinu menggunakan list yang mencatat awal memori dan ukuran memori yang kosong.

Block	Address	Length
1	0x84F800	4096
2	0x850F70	8192
3	0x8A03F0	8192
4	0x8C01D0	4096

**Gambar 9-3 Contoh List Memori Pada Xinu**

Jika ada proses yang membutuhkan memori maka Xinu akan mencari pada list tersebut, menentukan awal alamat yang bisa digunakan, mengalokasikan block sesuai permintaan dan mengupdate list tersebut.

8. Xinu menggunakan linked list untuk manajemen memory. Terdapat global variabel Bernama *memlist* yang berisi pointer ke free block kosong pertama. Kemudian *memlist* akan menunjuk ke alamat free block kosong selanjutnya. Pada free block ini akan menunjukkan ukuran free block dan free block selanjutnya.



**Gambar 9-4 Ilustrasi Penggunaan Linked List Untuk Manajemen Memory**



**Fakultas Informatika**  
School of Computing  
Telkom University



informatics lab

## Modul 10 Linux dan Windows

### Tujuan Praktikum

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

### 10.1 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.

### 10.2 Sistem Operasi Windows

#### 10.2.1 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 10-1 Timeline Sistem Operasi Windows**

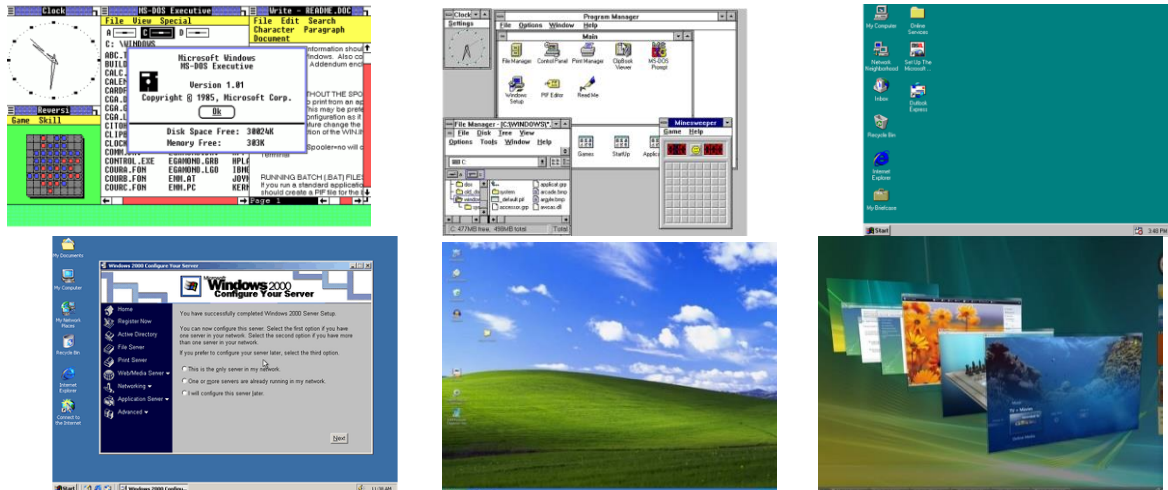
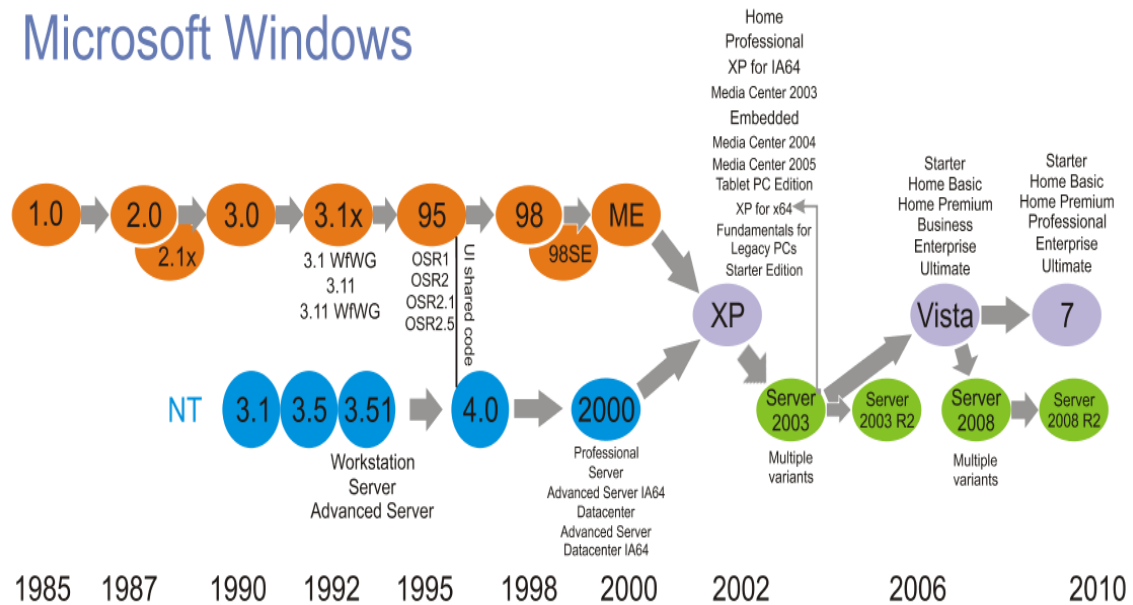
Waktu rilis	Nama Produk	Versi	Dukungan	Kelebihan
November 1985	Windows 1.01	1.01	Tidak didukung lagi	memperluas kemampuan MS-DOS dengan tambahan antarmuka grafis
November 1987	Windows 2.03	2.03	Tidak didukung lagi	menggunakan <i>model</i> memori modus <i>real</i> , yang hanya mampu mengakses memori hingga 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 Fonts</i> dan perbaikan terhadap <i>bug</i> dan dukungan terhadap multimedia.
Oktober 1992	Windows For Workgroup 3.1	3.1	Tidak didukung lagi	mencakup <i>driver</i> jaringan komputer dan <i>stack</i> 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 <i>kernel</i> hibrida, setelah pada versi-versi sebelumnya hanya menggunakan <i>kernel 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 (API)</i> 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 <i>driver</i> 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 <i>Server</i> 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 <i>Home Server</i>	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 <i>Server</i> 2008	NT 6.0.6001	Sekarang	lebih modular secara signifikan, ketimbang pendahulunya, Windows <i>Server</i> 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 10-1 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.

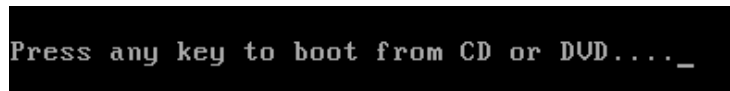
**Tabel 10-2 Spesifikasi Minimum dan Rekomendasi Windows 10**

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
--------	-----------------------	---

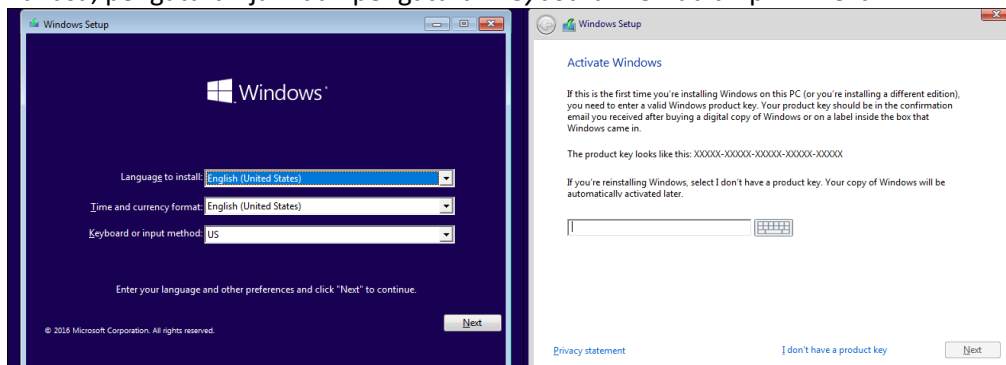
Berikut adalah langkah-langkah instalasi Windows 10:

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.



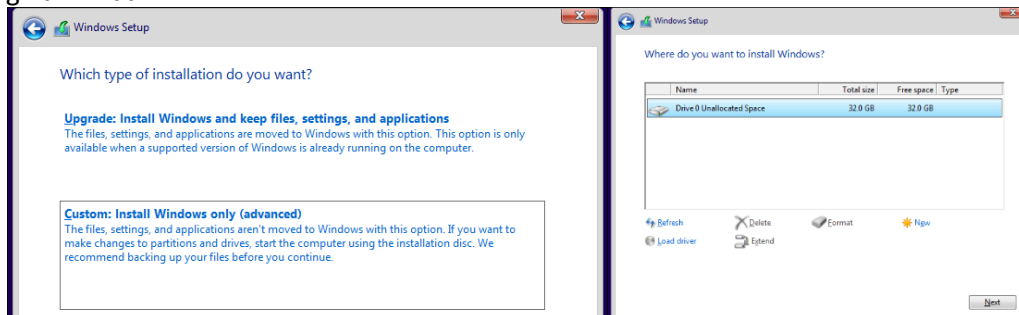
**Gambar 10-2 Booting**

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



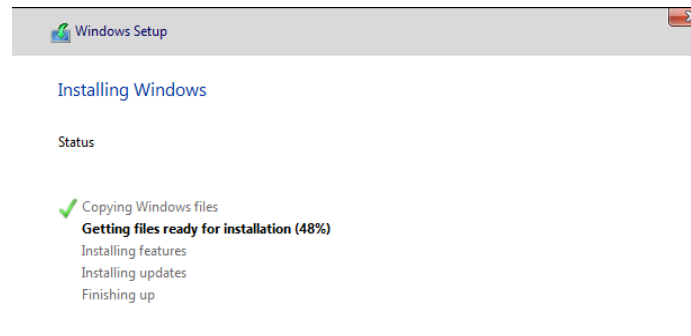
**Gambar 10-3 Windows Setup dan Aktivasi**

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 10-4 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.

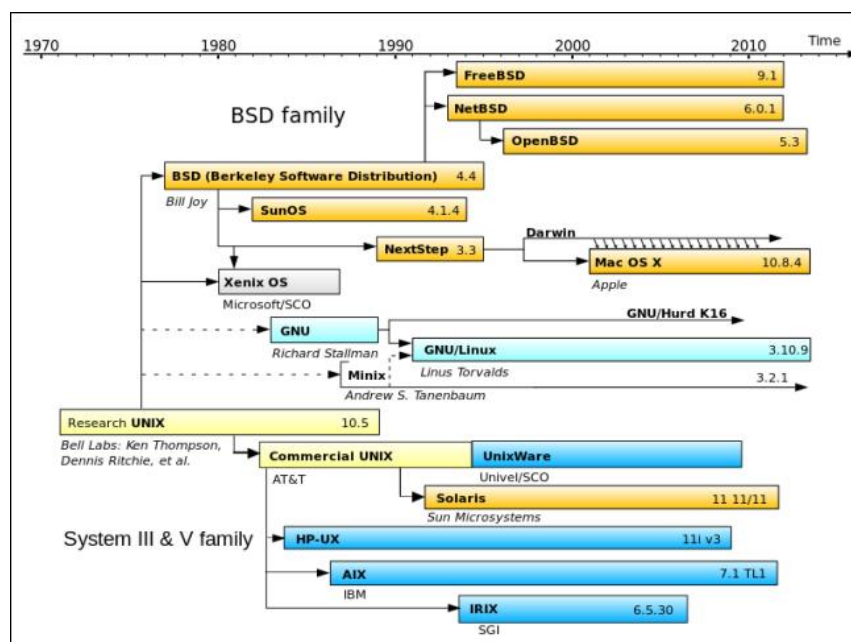


**Gambar 10-5 Proses Instalasi**

## 10.3 Sistem Operasi Linux

### 10.3.1 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.

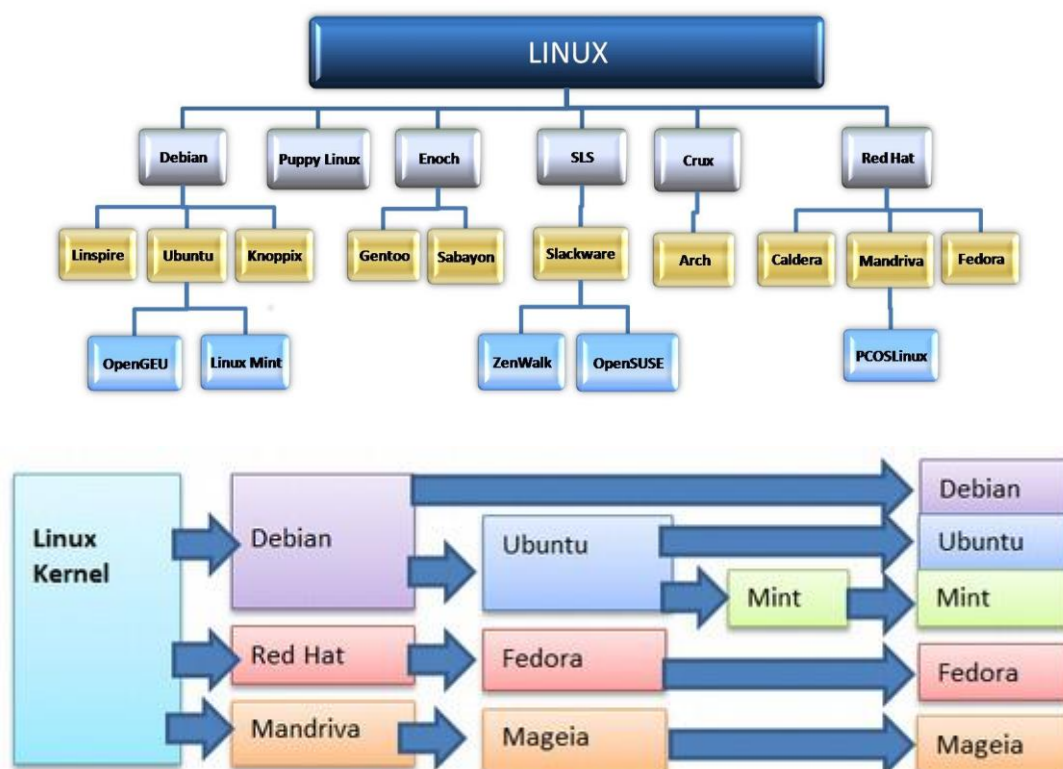


**Gambar 10-6 Pohon Keluarga Ubuntu**

### 10.3.2 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 *portofolio* layanan yang disediakan oleh Canonical yang pada akhirnya mendanai perbaikan *platform*.



Gambar 10-7 Beberapa Linux Distro

### 10.3.3 Instalasi Ubuntu

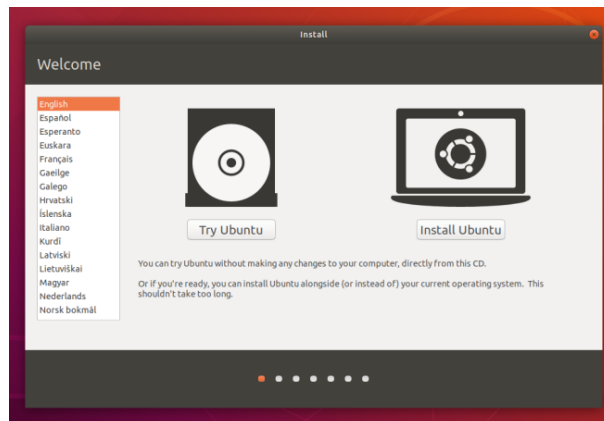
Sebelum melangkah ke proses instalasi, sebaiknya memperhatikan kebutuhan sistem operasi atau aplikasi. Berikut adalah kebutuhan minimal dan rekomendasi kebutuhan dari Ubuntu.

Tabel 10-3 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

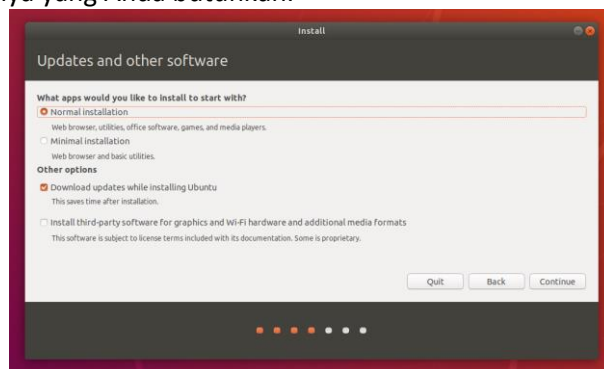
Berikut adalah langkah-langkah instalasi Ubuntu:

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*'.



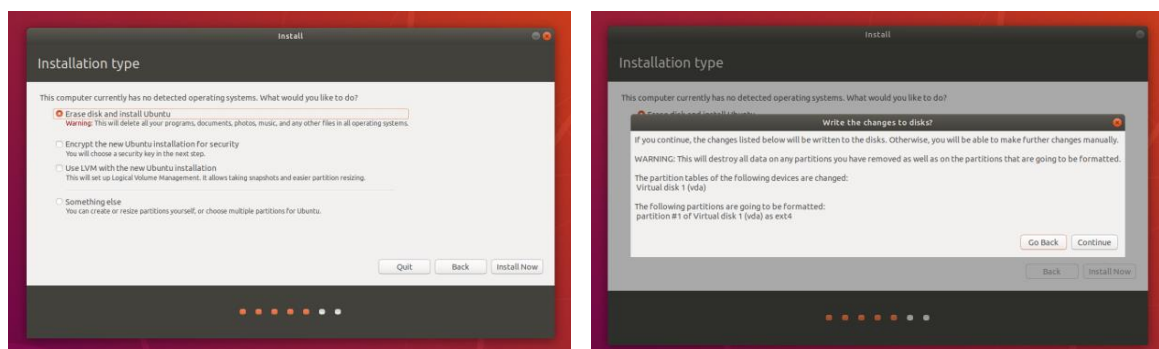
**Gambar 10-8 Pilihan Bahasa**

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 10-9 Pilihan Instalasi**

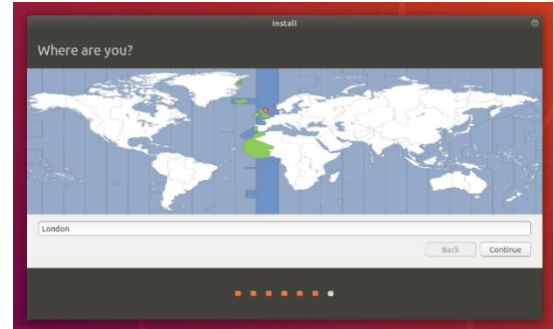
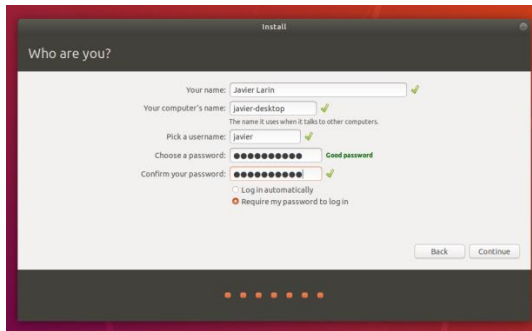
Di bawah pertanyaan tipe instalasi adalah dua kotak centang, satu untuk mengaktifkan pembaruan ketika menginstal dan yang lain untuk mengaktifkan perangkat lunak pihak ketiga. Alokasikan ruang *drive*. Gunakan kotak centang untuk memilih apakah Anda ingin menginstal Ubuntu di samping sistem operasi lain,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



**Gambar 10-10 Tipe Instalasi dan Pemilihan Partisi**

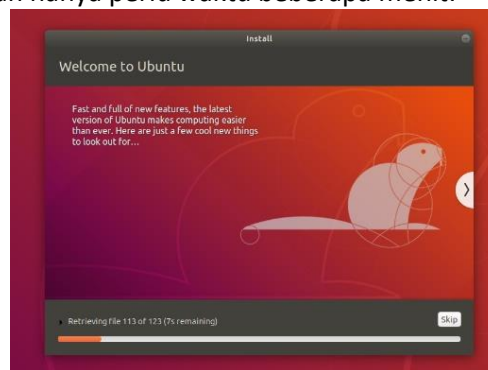


6. Setelah mengonfigurasi penyimpanan, klik tombol *'Install Now'*. Sebuah panel kecil akan muncul dengan gambaran tentang opsi penyimpanan yang Anda pilih, dengan kesempatan untuk kembali jika rinciannya salah. Klik *'Continue'* untuk memperbaiki perubahan tersebut di tempat dan memulai proses instalasi.
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.



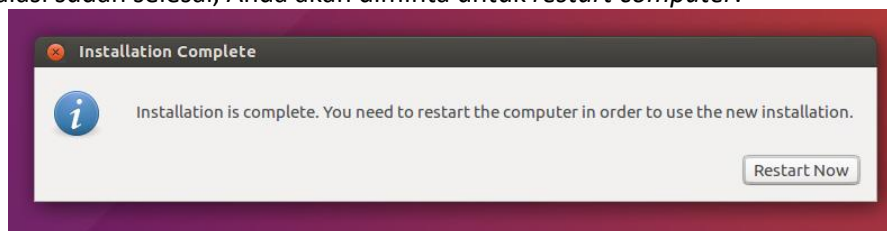
**Gambar 10-11 Pilih Kota dan Pengisian Data**

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 10-12 Proses Instalasi**

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



**Gambar 10-13 Proses Instalasi Selesai**

## Modul 11 Perintah Dasar Linux

### Tujuan Praktikum

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

### 11.1 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 11-1 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"><li>• # (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>.</li><li>• \$ 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>.</li></ul>
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
--	---

### 11.1.1 ls (List Dictionary)

\$ **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 11-2 Sebagian Daftar Opsi Perintah ls**

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

### 11.1.2 cd (Change Dictionary)

\$ **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
```

### 11.1.3 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 11-3 Sebagian Daftar Opsi Perintah cp**

Opsi	Keterangan
<b>-R</b>	Digunakan untuk meng- <i>copy</i> suatu <i>directory</i> beserta dengan isinya
<b>-v</b>	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.

#### 11.1.4 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 11-4 Sebagian Daftar Opsi Perintah rm**

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

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*.

#### 11.1.5 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 11-5 Sebagian Daftar Opsi Perintah mv**

Opsi	Keterangan
<b>-R</b>	Digunakan untuk meng-copy suatu <i>directory</i> beserta dengan isinya
<b>-v</b>	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*.

### 11.1.6 mkdir (Make Dictionary)

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

*Tabel 11-6 Sebagian Daftar Opsi Perintah mkdir*

Opsi	Keterangan
<b>-p</b>	Opsi untuk membuat <i>directory</i> secara bertingkat dengan hanya menggunakan satu buah <i>command</i> .
<b>-m</b>	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}
```

### 11.1.7 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 ditunjukkan setelah perintah tersebut di jalankan, dan saat ini *directory* kerja aktif nya sedang berada di **/home/praktikan**.

### 11.1.8 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*.

### 11.1.9 | (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*".

### 11.1.10 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

## 11.2 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 (Gnat), 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.

### 11.2.1 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++".

### 11.2.2 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 12 Scripting

### Tujuan Praktikum

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

## 12.1 Shell

### 12.1.1 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@karaswain: /usr/portage/app-shells/bash $ grep -i /dev/sda /etc/fstab | cut --fields=3
/dev/sda1      /boot        none
/dev/sda2      /             /
kar@karaswain: /usr/portage/app-shells/bash $ date
Sat Aug 8 02:42:24 MSD 2009
kar@karaswain: /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
l100_wdt         12272   0
l2c_i801         3380    0
kar@karaswain: /usr/portage/app-shells/bash $
```

Gambar 12-1 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!
```

### 12.1.2 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 12-2 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.

## 12.2 Scripting Dasar

### 12.2.1 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
```

### 12.2.2 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*

### 12.2.3 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
```

### 12.2.4 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
```



## Operator

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

$\text{Var}++$  = *increment*

$\text{Var}--$  = *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
```

## 12.2.5 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:

$\text{String1} = \text{String2}$       *string* 1 sama dengan *string* 2

$\text{String1} \neq \text{String2}$       *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 <i>file</i>	<i>file</i> 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
```

## 12.2.6 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.

### 12.2.7 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 13 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.

## 13.1 Access Control (Permission)

### 13.1.1 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 penggunaanya. 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 banyak 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:

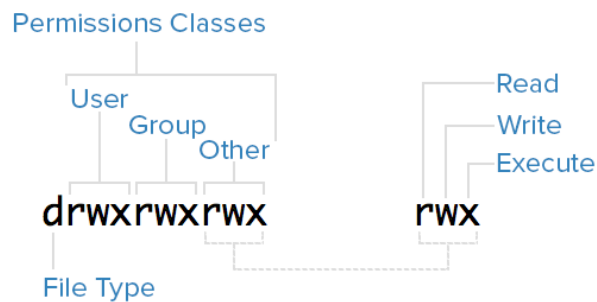
```
ls -l /etc
```

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 13-1 Daftar File

## 5. Mode



Gambar 13-2 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

## 13.1.2 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
```

## 13.2 Integrity (SHA-256)

### 13.2.1 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:



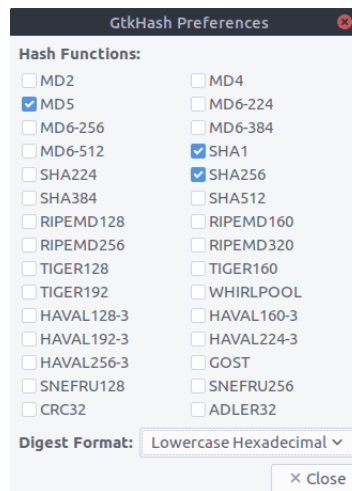
```
Fedora-19-i386-netinst.iso: FAILED
sha256sum: WARNING: 1 computed checksum did NOT match
```

### 13.2.2 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 13-3** 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*:

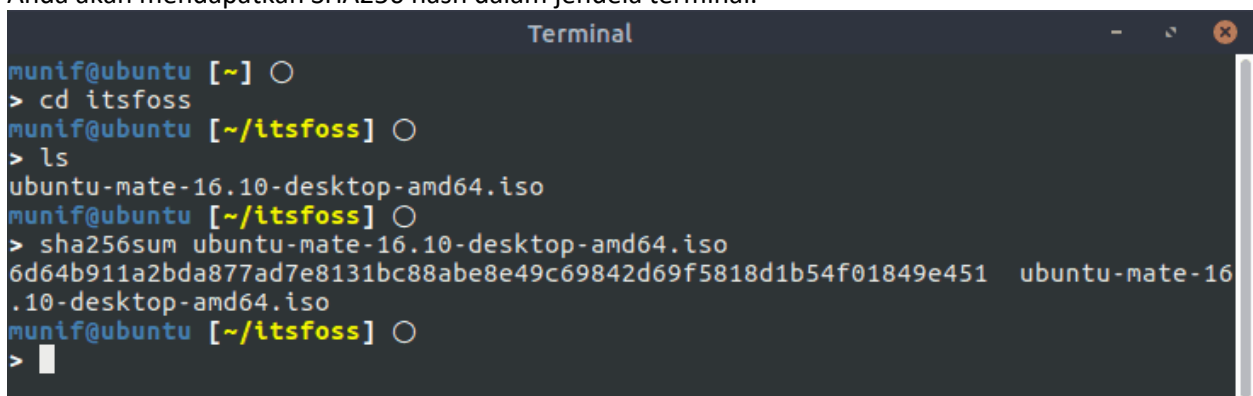
Pertama pergi ke direktori di mana gambar *.iso* disimpan:

```
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.



```
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 13-4 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.

## 13.3 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.

### 13.3.1 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 dienkrpsi

```
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, *unmount* 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
```

## 13.4 GPG

### 13.4.1 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.

### 13.4.2 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/>

### 13.4.3 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
9oxeLQpN+r75Ko39RVE88dRcW710fPY0+fjSXBKhpN+raRMUKJp4AX9BJd00YA/4
EpD+8cDK4DuLlLdn1x0q41VUsznXrnMpQedRmAL9f9bL6pbLTJhaKeorTokTvdn6
5VT3pb2o+jr6NETaUxd99ZG/osPar9tNThVLIIzG1nDabcTFbMB+w7w0JuhXyTLQ
JBU9xmavTM71PfV6Pkh4j1pfWImXc1D8dS+jcvKeXInBfm2XZsfOCesk12YnK3Nc
u1Xe1lxzSt7Cegum4S/YuxmYoh462oGZ7FA4Cr2lvAPVp09zmgQ8JITXiYg2wB3
. . .
```

**Gambar 13-5 Public Key**

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

### 13.4.4 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.

### 13.4.5 Dekripsi dengan PGP

Untuk dekripsi gunakan perintah:

```
gpg file_name.asc
```



## Daftar Pustaka

1. D. Comer, "Operating System Design - The Xinu Approach", Second Edition CRC Press, 2015.
2. The Xinu Page. Xinu Team. [Online] <https://xinu.cs.purdue.edu/>.
3. Sourcetrail Free and open-source cross-platform source explore. SourceTrail. [Online] <https://www.sourcetrail.com/>.
4. 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>.
5. 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/>.
6. 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/>.
7. 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>.
8. 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/>.
9. 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>.
10. 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>.
11. bad\_crow. vim-basics. howtoforge. [Online] <https://www.howtoforge.com/vim-basics>.
12. openvim. [Online] <https://www.openvim.com/>.



**Kontak Kami :**



@fiflab



Praktikum IF LAB



informaticslab@telkomuniversity.ac.id