



# Operating System

## Training Module

May 2011

For the latest information, please see [bluejack.binus.ac.id](http://bluejack.binus.ac.id)



Information in this document, including URL and other Internet Web site references, is subject to change without notice. This document supports a preliminary release of software that may be changed substantially prior to final commercial release, and is the proprietary information of Binus University.

This document is for informational purposes only. BINUS UNIVERSITY MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

The entire risk of the use or the results from the use of this document remains with the user. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Binus University.

Binus University may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Binus University, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2010 Binus University. All rights reserved.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# TABLE OF CONTENT

TABLE OF CONTENT .....	3
OVERVIEW .....	4
OBJECTIVE .....	5
SOFTWARE/HARDWARE REQUIREMENT .....	7
OPERATING SYSTEM - INTRODUCTION .....	8
OPERATING SYSTEM - FILE AND FOLDER MANAGEMENT .....	10
OPERATING SYSTEM - REDIRECTION INPUT, OUTPUT DAN FILE .....	13
OPERATING SYSTEM - PROCESS MANAGEMENT .....	15
OPERATING SYSTEM – PARENT AND CHILD PROCESS .....	17
OPERATING SYSTEM – PIPE .....	20
OPERATING SYSTEM – FIFO .....	22
OPERATING SYSTEM – SHARE MEMORY .....	24
OPERATING SYSTEM – SEMAPHORE .....	28
OPERATING SYSTEM – THREAD .....	32

# OVERVIEW

MODULE 01 .....	8
• INTRODUCTION OF OPERATING SYSTEM .....	8
MODULE 02 .....	10
• FILE AND FOLDER MANAGEMENT IN LINUX .....	10
MODULE 03 .....	13
• REDIRECTION INPUT, OUTPUT DAN FILE IN LINUX .....	13
MODULE 04 .....	15
• PROCESS MANAGEMENT IN LINUX .....	15
MODULE 05 .....	17
• PARENT AND CHILD PROCESS .....	17
MODULE 06 .....	20
• PIPE .....	20
MODULE 07 .....	22
• FIFO .....	22
MODULE 08 .....	24
• SHARE MEMORY .....	24
MODULE 09 .....	28
• SEMAPHORE .....	28
MODULE 10 .....	32
• THREAD .....	32

# OBJECTIVE

## Module 01:

- Setelah mempelajari materi ini, diharapkan dapat mengerti dan menguasai pengertian sistem operasi, kelebihan dan kekurangan Linux.

## Module 02:

- Setelah mempelajari materi ini, dapat menggunakan konsol pada sistem operasi linux.
- Setelah mempelajari materi ini, dapat menggunakan sintaks-sintaks konsol dalam manajemen file dan direktori.

## Module 03:

- Setelah mempelajari materi ini, dapat menggunakan sintaks-sintaks konsol dalam redirection input, output dan file.

## Module 04:

- Setelah mempelajari materi ini, dapat menggunakan sintaks-sintaks konsol dalam manajemen proses.

## Module 05:

- Setelah mempelajari materi ini, dapat menerapkan konsep proses parent dan child dalam program.
- Setelah mempelajari materi ini, dapat membuat program dengan menggunakan parent dan child.
- Setelah mempelajari materi ini, dapat memahami jalannya proses parent dan child.
- Setelah mempelajari materi ini, dapat memahami proses id dan parent proses id.
- Setelah mempelajari materi ini, dapat mengerti pengiriman signal ke proses.

## Module 06:

- Setelah mempelajari materi ini, dapat menerapkan penggunaan pipe dalam pembuatan program.
- Setelah mempelajari materi ini, dapat menerapkan konsep pipe dalam proses parent dan child.

## Module 07:

- Setelah mempelajari materi ini, dapat menerapkan penggunaan fifo dalam pembuatan program.
- Setelah mempelajari materi ini, dapat menerapkan konsep fifo dalam dua proses.

## Module 08:

- Setelah mempelajari materi ini, dapat menerapkan penggunaan fifo dalam pembuatan program.
- Setelah mempelajari materi ini, dapat menerapkan konsep fifo dalam dua proses.

## Module 09:


- Setelah mempelajari materi ini, dapat menerapkan konsep semaphore dalam program.
- Setelah mempelajari materi ini, dapat memahami penggunaan semaphore dalam proses.

Module 10:

- Setelah mempelajari materi ini, dapat menerapkan konsep thread dengan semaphore.
- Setelah mempelajari materi ini, dapat Memahami konsep thread.
- Setelah mempelajari materi ini, dapat Memahami perbedaan thread dan semaphore.
- Setelah mempelajari materi ini, dapat Menerapkan thread dalam program.
- Setelah mempelajari materi ini, dapat Menerapkan thread dan semaphore.

# SOFTWARE/HARDWARE REQUIREMENT

- openSUSE supports most PC hardware components.
- Pentium\* III 500 MHz or higher processor (Pentium 4 2.4 GHz or higher or any AMD64 or Intel\* EM64T processor recommended)
- Main memory: 512 MB physical RAM (1 GB recommended)
- Hard disk: 3 GB available disk space (more recommended)
- Sound and graphics cards: supports most modern sound and graphics cards, 800 x 600 display resolution (1024 x 768 or higher recommended)
- Booting from CD/DVD drive or USB-Stick for installation, or support for booting over network (you need to setup PXE by yourself, look also at Network install) or an existing installation of openSUSE, more information at Installation without CD
- The GRUB bootloader co-operates with other operating systems on the same machine. openSUSE can be installed on one free harddisk partition, while preserving existing installations on other partitions.

<h1>Operating System - Introduction</h1>	
<b>Module 01</b> <ul style="list-style-type: none"> <li>• Introduction of Operating System</li> </ul>	
Last Update 25 May 2011	Revision 1

## 1. Module Description

1. Pengenalan Sistem Operasi dan Linux
2. Kelebihan Linux
3. Kekurangan Linux
4. Distro Linux

## 2. Learning Outcomes

- Setelah mempelajari materi ini, diharapkan dapat mengerti dan menguasai pengertian sistem operasi, kelebihan dan kekurangan Linux.

## 3. Material

### 1. Pengenalan Sistem Operasi dan Linux

Apakah yang diperlukan untuk menjalankan sebuah komputer? Pemrograman Assembly? Apakah banyak orang yang bisa? Terlalu rumit sepertinya. Sekarang ada yang namanya Sistem Operasi. Apakah Sistem Operasi itu? Sebuah *extended machine*, yang artinya mempermudah kekompleksan pemrograman assembly, sehingga memudahkan kita dalam penggunaan komputer. Selain itu Sistem operasi juga *resource manager*, resource yang dimaksudkan di sini adalah semua komponen yang terhubung dengan komputer, harddisk, mouse, keyboard, dll.

Sistem Operasi yang biasa dipakai dan sudah merakyat di Indonesia adalah Windows, sebenarnya tidak hanya Windows, ada banyak Sistem Operasi lainnya, ada Novell, UNIX, Mac OS, Haiku, Linux, dll. Yang akan kita pelajari di sini adalah Linux.

Linux diambil dari 2 nama yaitu Linus Torvalds dan Unix. Linux adalah sistem operasi turunan UNIX. Untuk sejarah lengkapnya silakan lihat di link ini <http://tldp.org/LDP/intro-linux/html/>. Linux merupakan Sistem Operasi yang bersifat Open Source, sehingga bebas dimodifikasi, disebarluaskan, atau bahkan dijual kembali dengan syarat tetap melampirkan source code asli dalam distribusinya. Kernel merupakan inti dari sistem operasi yang mengatur penggunaan memory, piranti I/O, proses-proses, pemakaian file dan lain-lain. Yang dibuat Torvald sebetulnya hanyalah kernel, belum sistem operasi secara keseluruhan. Pengembang GNU yang sedangkan mengembangkan sistem operasi pada saat itu tinggal satu komponen yang kurang yakni kernel. Oleh karenanya mereka mengimplan kernel Linux ke dalam sistem operasi mereka menghasilkan sistem operasi yang dikenal dengan GNU/Linux.



## **2. Kelebihan Linux**

Berikut kelebihan Linux:

- Tidak memerlukan perangkat keras yang mahal.
- Linux merupakan sistem operasi FULL 32 – bit.
- Preemptive multitasking.
- Multiuser.
- Multiconsole.
- Linux memiliki grafis antarmuka (GUI) sebagaimana Operating System lainnya.
- Program-program maupun aplikasi-aplikasi networking tersedia dalam semua distribusi Linux (dalam CD atau disket), sehingga tak perlu mencari/membeli/mendownload aplikasi tambahan lagi.
- Tidak memerlukan defragment.
- Dukungan akses 33 macam sistem file yang berbeda.
- Pertambahan pengguna Linux yang pesat.
- Remote Control.


## **3. Kekurangan Linux**

Linux juga memiliki kekurangan:

- Pengembang Game belum banyak.
- Hampir semua vendor hardware menyediakan driver untuk Windows.
- Pengguna Linux masih sedikit.
- Masih banyak pengguna yang masih “Windows Minded”.
- Proses instalasi software akan tidak semudah windows jika komputer tersebut tidak memiliki sambungan internet.

## **4. Distro Linux**

Distro adalah distribusi Linux yang berisi kernel Linux beserta kumpulan aplikasi yang dibuat menjadi satu paket. Contohnya : Redhat, Suse, Mandriva, Debian, Slackware, Ubuntu, Gentoo.

<h1>Operating System - File and Folder Management</h1>	
<h2>Module 02</h2> <ul style="list-style-type: none"> <li>File and Folder Management in Linux</li> </ul>	
<p>Last Update 25 May 2011</p>	<p>Revision 1</p>

### 1. Module Description

1. Penggunaan manual pada linux
2. File dan Folder Management
3. Menggunakan editor vi
4. Melihat penggunaan space storage
5. Mengganti hak akses suatu file

### 2. Learning Outcomes

- Setelah mempelajari materi ini, dapat menggunakan konsol pada sistem operasi linux.
- Setelah mempelajari materi ini, dapat menggunakan sintaks-sintaks konsol dalam manajemen file dan direktori.

### 3. Material

#### 1. Penggunaan manual pada linux

Manual pada linux digunakan sebagai bantuan untuk user. Terdapat 4 sintaks dasar yang digunakan untuk membuka bantuan:

- **man**  
contoh : man ls
- **--help**  
contoh : ls --help
- **info**  
contoh : info ls
- **whatis**  
contoh : whatis ls

#### 2. File dan Folder Management

Berikut ini beberapa sintaks yang digunakan untuk manage file dan folder:

- **ls**  
Untuk melihat file/folder yang ada pada direktori aktif.
  - ✓ **ls <nama file>**: menampilkan file dengan nama tersebut bila ada. Bila tidak ada, ditampilkan pesan bahwa file / folder tersebut tidak ada.
  - ✓ **ls -a**: menampilkan semua file pada directory yang sedang ada, termasuk nama file yang berawalan "." (hidden).
  - ✓ **ls -l** atau **dir**: menampilkan file dengan format lengkap.
  - ✓ **ls -full-time**: menampilkan file dengan format waktu yang lengkap.

- **cd**  
Untuk mengubah direktori yang sedang aktif.  
✓ **cd** <nama direktori>: berpindah dari direktori aktif ke direktori tujuan.  
✓ **cd** /: berpindah dari direktori aktif ke direktori / (root).  
✓ **cd** ..: berpindah ke direktori parent.  
✓ **cd** ../**sisop**: berpindah dari direktori parent kemudian masuk ke direktori sisop.
- **pwd**  
Mencetak direktori yang aktif.
- **mkdir**  
Membuat direktori baru  
✓ **mkdir** sisop: membuat direktori baru dengan nama sisop pada direktori aktif.
- **touch**  
Memodifikasi waktu terakhir pengaksesan file. Jika file belum ada, maka akan langsung dibuat file dengan nama tersebut.
- **rm**  
Menghapus file/folder.  
✓ **rm** sisop: menghapus file sisop.  
✓ **rm -r** sisop: menghapus folder sisop.
- **mv**  
Memindahkan file / folder atau merename file / folder tergantung kondisi.
- **cp**  
Mengcopy file / folder.
- **cat**  
Menampilkan isi suatu file.  
✓ **cat** sisop: isi file sisop akan ditampilkan ke layar.

### 3. Menggunakan editor vi

vi adalah salah satu text editor pada Linux. Dalam vi terdapat 2 mode: *command mode* dan *edit mode*.

- ✓ Command mode: mode di mana kita dapat melakukan berbagai perintah seperti save, quit, copy, paste, cut, dll. Untuk berpindah ke command mode, tekan tombol **ESC**.
- ✓ Edit mode: mode yang digunakan untuk mengedit isi dari suatu file. Untuk berpindah ke edit mode, tekan tombol **i**.

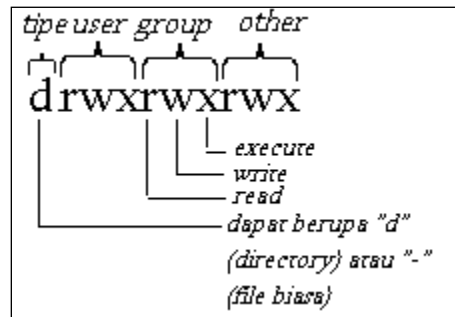
No	Command	Kegunaan	Contoh
1	w <nama file>	Save	w test.txt
2	q!	Exit without save	q!
3	wq	Save and Exit	wq
4	Y <baris> y	Copy beberapa baris	Y2y
5	C <baris> c	Cut beberapa baris	C4c
6	p	Paste	p

#### 4. Melihat penggunaan space storage

- **du**  
Menampilkan besar pemakaian space pada hard disk dalam byte.
- **df**  
Menampilkan besar space yang tersedia pada harddisk.
- **free**  
Menampilkan jumlah memory yang tersedia dan dipakai system. Baik memory fisik maupun memory swap.
- **vmstat**  
Menampilkan status dari virtual memory.


#### 5. Mengganti hak akses suatu file

- **chmod**  
Mengubah hak akses suatu file.



Contoh pemakaian:

- ✓ **chmod** u=rwx, o=rx, g=x sisop.
  - ✓ **chmod** 0751 sisop
- Artinya, user dapat membaca (r) menulis (w) dan mengeksekusi(x) file sisop, group dapat membaca (r) dan mengeksekusi (x) file sisop, sedangkan other hanya dapat mengeksekusi (x) file sisop.

<h1>Operating System - Redirection input, output dan file</h1>	
<h2>Module 03</h2> <ul style="list-style-type: none"> <li>• <b>Redirection input, output dan file in Linux</b></li> </ul>	
<p>Last Update 25 May 2011</p>	<p>Revision 1</p>

### 1. Module Description

1. Pipe, redirection, grep, find, dan bahasa awk dalam konsole linux

### 2. Learning Outcomes

- Setelah mempelajari materi ini, dapat menggunakan sintaks-sintaks konsole dalam redirection input, output dan file.

### 3. Material

#### 1. Pipe, redirection, grep, find, dan bahasa awk dalam konsole

- **pipe**

Hasil dari sebuah operasi dijadikan input bagi operasi yang lain. Operator yang digunakan "|"

- ✓ **ls -l | sort:** output dari ls -l akan menjadi inputan dari sort. Hasilnya akan ditampilkan isi direktori lengkap secara terurut.

- **redirection**

Menggunakan >, >>, dan <

- ✓ **ls -l > sisop.txt:** output dari ls -l akan dimasukkan ke dalam file sisop.txt. Jika file tersebut sudah ada, maka akan di overwrite.
- ✓ **ls -l >> sisop.txt:** output dari ls -l akan dimasukkan ke dalam file sisop.txt. Jika file tersebut sudah ada, maka akan ditambahkan di bawahnya.

- **grep**

Menampilkan / mencari pattern tertentu dari suatu file atau sekelompok tulisan / kalimat.


- ✓ **grep "a" test.txt:** mencari pada tiap baris dari file "test.txt", bila ditemukan kata "a" pada suatu baris dari isi file tersebut, maka baris tersebut akan ditampilkan ke layar.

- **find**

Mencari suatu file.

- ✓ **find nama:** menampilkan file di direktori nama tersebut.
- ✓ **find a\*:** menampilkan file di direktori yang berawalan a.

- **sort**  
Mengurutkan data.
  - ✓ **ls -l | sort**: Tampilan ls -l akan disort berdasarkan field paling depan, kemudian ditampilkan ke layar.
  - ✓ **ls -l | sort -k3**: Sorting berdasarkan kolom ke-3.
- **head**  
Menampilkan baris teratas dari kalimat.
  - ✓ **cat sisop.txt | head -3**: menampilkan 3 baris teratas dari file sisop.txt.
- **tail**  
Menampilkan baris terbawah dari kalimat.
  - ✓ **cat sisop.txt | tail -3**: menampilkan 3 baris terbawah dari file sisop.txt.
- **awk**  
Merupakan bahasa pemrograman. Dapat dipakai untuk membaca dan memproses hasil output dari suatu sintaks.
  - ✓ **ls -l | awk '{print \$1}'**: mencetak kolom pertama dari hasil ls -l.

<h1>Operating System - Process Management</h1>	
<h2>Module 04</h2> <ul style="list-style-type: none"> <li>• <b>Process Management in Linux</b></li> </ul>	
<p>Last Update 25 May 2011</p>	<p>Revision 1</p>

### 1. Module Description

1. Melihat proses yang ada
2. Menghentikan suatu proses
3. Memahami proses background dan foreground

### 2. Learning Outcomes

- Setelah mempelajari materi ini, dapat menggunakan sintaks-sintaks konsole dalam manajemen proses.

### 3. Material

#### 1. Melihat proses yang ada

Ada beberapa cara untuk melihat proses yang sedang aktif diantaranya:

- **ps**  
Menampilkan snapshot dari proses aktif.
- **top**  
Menampilkan proses yang sedang berjalan dalam system secara real time.

Dalam management proses juga terdapat beberapa sintaks pendukung antara lain:

- **uptime**  
Untuk melihat keterangan pada system, telah berapa lama system berjalan dll.
- **renice**  
Mengatur nice value dari proses yang sedang berjalan.
- **shutdown**  
Mematikan system.  
✓ **/sbin/shutdown -h now**: langsung mematikan system.

#### 2. Menghentikan suatu proses

Untuk menghentikan suatu proses dapat digunakan signal dengan menggunakan perintah kill.

- **kill**  
Mengirimkan signal ke suatu proses. Default signal: SIGTERM(15).  
✓ **kill 1761**: mengirimkan signal SIGTERM ke proses dengan pid 1761.  
✓ **kill -9 1761**: mengirimkan signal SIGKILL ke proses dengan pid 1761.


- ✓ **killall** firefox: menghentikan semua proses dengan nama firefox.

### **3. Memahami proses background dan foreground**

Proses background

Proses foreground



<h1>Operating System – Parent and Child Process</h1>	
<h2>Module 05</h2> <ul style="list-style-type: none"> <li>• Parent and Child Process</li> </ul>	
<p>Last Update 25 May 2011</p>	<p>Revision 1</p>

### 1. Module Description

1. Pengertian GCC
2. Pengertian Proses
3. Memahami Proses Parent and Child

### 2. Learning Outcomes

- Setelah mempelajari materi ini, dapat menerapkan konsep proses parent dan child dalam program.
- Setelah mempelajari materi ini, dapat membuat program dengan menggunakan parent dan child.
- Setelah mempelajari materi ini, dapat memahami jalannya proses parent dan child.
- Setelah mempelajari materi ini, dapat memahami proses id dan parent proses id.
- Setelah mempelajari materi ini, dapat mengerti pengiriman signal ke proses.

### 3. Material

#### 1. Pengertian GCC

GCC (GNU Compiler Collection) merupakan kompiler multiplatform dari GNU yang mendukung beberapa bahasa pemrograman : C, C++, Objective-C, Fortran, Java, dan Ada.

Dalam training kali ini kita hanya membahas bahasa C. Pada GCC, struktur programmingnya bisa dibilang hampir sama dengan dengan Borland C Compiler. GCC juga mengacu pada standar ANSI C.

Setelah kita membuat program untuk mengcompilanya kita bisa gunakan sintaks:

***gcc <nama file yang akan dicompile> -o <nama file output>***

Contoh: gcc coba.c -o coba

dalam hal ini compiler akan menghasilkan file coba yang dapat dieksekusi. Untuk mengeksekusi program gunakan './'. Caranya: ***./coba***.

#### 2. Pengertian Proses

Sebuah proses adalah instance dari program yang dieksekusi dan juga merupakan "operating system's basic scheduling unit". Sebuah proses dianggap sebagai sebuah program yang sedang dijalankan dan terdiri atas elemen – elemen sebagai berikut :

- Context dari program dan status program.

- Working directory dari program tersebut.
- File dan directory yang diakses oleh program.
- Hak akses dari program.
- Memory dan resource system yang lain yang dialokasikan untuk proses itu.
- Mempunyai ID (PID , PPID)

#### Contoh Code:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  int main(){
6      printf("PID dari proses ini: %d\n", getpid());
7      printf("PPID dari proses ini: %d\n", getppid());
8      return 0;
9  }
```

Program akan mencetak PID (Process ID) dan PPID (Parent Process ID) dari program tersebut.

### 3. Memahami Proses Parent and Child


Parent and Child adalah suatu istilah untuk membuat proses baru dari sebuah proses yang telah ada. Hubungan antara proses parent dan proses child: **PID dari proses parent adalah PPID dari proses child.**

Syntax untuk membuat child proses adalah **fork()**. fungsi yang terdapat dalam "**unistd.h**" ini akan mengembalikan nilai yang merupakan pid dari child proses sekaligus memecah proses ini menjadi 2 proses. Setelah syntax fork dijalankan , maka proses akan terbelah menjadi 2 dan masing masing proses akan menjalankan syntax yang sama yaitu syntax dibawah syntax fork . Kedua proses tersebut (parent and child) akan berjalan bersama sama.

### Contoh Code:

Program berikut akan mencetak nilai i sebanyak 10. Child akan mencetak 5 i, dan parent akan mencetak 5 i. Fungsi **sleep(1)** berguna untuk member delay selama 1 detik. Fungsi **wait(&pid)** akan menunggu proses child selesai terlebih dulu, kemudian proses parent selesai.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  int main(){
6      pid_t pid;
7      int i;
8
9      pid = fork();
10     if (pid < 0){
11         perror("Error Fork\n");
12         exit(1);
13     }else if (pid == 0){ //Proses Child
14         printf("[CHILD]\n");
15         for (i=0; i<5; i++) {
16             printf("Child : %d\n", i);
17             sleep(1);
18         }
19     }else{
20         printf("[PARENT]\n");
21         for (i=0; i<5; i++){
22             printf("Parent : %d\n", i);
23             sleep(1);
24         }
25         wait(&pid);
26     }
27     return 0;
28 }
```

<b>Operating System – Pipe</b>	
<b>Module 06</b> <ul style="list-style-type: none"> <li>• Pipe</li> </ul>	
Last Update 25 May 2011	Revision 1

### 1. Module Description

1. Implementasi Pipe
2. Implementasi Unnamed Pipe

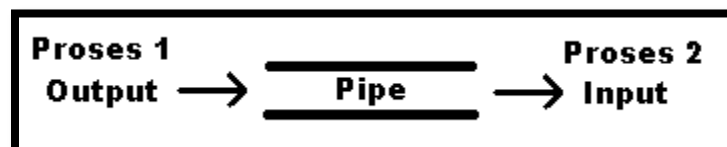
### 2. Learning Outcomes

- Setelah mempelajari materi ini, dapat menerapkan penggunaan pipe dalam pembuatan program.
- Setelah mempelajari materi ini, dapat menerapkan konsep pipe dalam proses parent dan child.

### 3. Material

#### 1. Implementasi Pipe

Pipe adalah salah satu media untuk IPC (interprocess communication). Dimana output suatu proses menjadi input bagi proses lain.



Pada pipe komunikasi terjadi secara 1 arah (half duplex), artinya pada saat proses 1 melakukan input pada PIPE, maka proses 2 tidak dapat melakukan input pada pipe . Pada system pipe ini bila suatu proses menginput ke dalam pipe, maka proses tersebut akan melakukan wait sampai data didalam pipe tersebut terbaca.

#### 2. Implementasi Unnamed Pipe

Diberi nama unnamed pipe Karena pipe ini tidak terdapat di dalam directory dalam file system sehingga user tidak dapat melihat pipe ini. Pada unnamed pipe ini menggunakan int fd[2]. fd adalah sebuah file descriptor di mana:


- Index 0 untuk read
- Index 1 untuk write

Karena pipe adalah komunikasi 1 arah , maka biasanya bila suatu proses berfungsi untuk menulis kedalam pipe maka file descriptor untuk read nya ditutup (close fd [0]) begitu juga sebaliknya bila berfungsi untuk membaca maka file descriptor untuk writenya ditutup (close fd[1]).

### Contoh Code:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  int main(){
6      pid_t pid;
7      char kata[100], fflush;
8      int fd[2];
9
10     system("clear");
11     printf("Program unnamed pipe sederhana\n");
12     if ((pipe(fd)) < 0){
13         perror("Error pipe\n");
14         exit(1);
15     }
16
17     pid = fork();
18     if (pid < 0){
19         perror("Error fork\n");
20         exit(1);
21     }else if (pid == 0){ //Proses Child
22         close(fd[1]);
23         read(fd[0], (char *)&kata, sizeof(kata));
24         printf("Kata yang ditulis adalah: %s\n", kata);
25     }else{ //Proses Parent
26         sleep(1);
27         close(fd[0]);
28         printf("Masukkan kata: ");
29         scanf("%s", kata); scanf("%c", &fflush);
30         write(fd[1], (char *)&kata, sizeof(kata));
31         wait(&pid);
32     }
33
34     return 0;
35 }
```

Pada Code diatas **Proses Child** akan mencetak kata yang dimasukan pada **Proses Parent**.

<b>Operating System – Fifo</b>	
<b>Module 07</b> <ul style="list-style-type: none"> <li><b>Fifo</b></li> </ul>	
Last Update 25 May 2011	Revision 1

### 1. Module Description

1. Implementasi Fifo (Named Pipe)

### 2. Learning Outcomes

- Setelah mempelajari materi ini, dapat menerapkan penggunaan fifo dalam pembuatan program.
- Setelah mempelajari materi ini, dapat menerapkan konsep fifo dalam dua proses.


### 3. Material

#### 1. Implementasi Fifo (Named Pipe)

Diberi nama named pipe Karena pipe ini terdapat di dalam directory dalam file system sehingga user dapat melihat pipe ini. Pipe jenis ini dapat dilihat sebagai file dengan attribute **p** yang artinya adalah **pipe**.

### Contoh Code:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/stat.h>
6
7  int main(){
8      pid_t pid;
9      system("rm pipe");
10     system("clear");
11     printf("Named Pipe\n");
12
13     pid = fork();
14     if (pid < 0){
15         perror("Error fork\n");
16         exit(1);
17     }else if (pid == 0){
18         if (mkfifo("pipe", 0666) < 0){
19             perror("Error mkfifo\n");
20             exit(1);
21         }
22         printf("Child menulis ke dalam pipe\n");
23         FILE *tulis = fopen("pipe", "w");
24         fprintf(tulis, "Kata yang ditulis");
25         fclose(tulis);
26     }else{
27         sleep(1);
28         char kata[100];
29         printf("Parent membaca dari dalam pipe\n");
30         FILE *baca = fopen("pipe", "r");
31         fscanf(baca, "%s", kata);
32         printf("Kata yg ada di pipe: %s\n", kata);
33         fclose(baca);
34         wait(&pid);
35     }
36     return 0;
37 }
```

<h1>Operating System – Share Memory</h1>	
<h2>Module 08</h2> <ul style="list-style-type: none"> <li>• Share Memory</li> </ul>	
<p>Last Update 25 May 2011</p>	<p>Revision 1</p>

### 1. Module Description

#### 1. Share Memory

### 2. Learning Outcomes

- Setelah mempelajari materi ini, dapat menerapkan penggunaan fifo dalam pembuatan program.
- Setelah mempelajari materi ini, dapat menerapkan konsep fifo dalam dua proses.

### 3. Material

#### 1. Share Memory

Shared memory adalah sebuah segment memory yang dibuat oleh kernel, yang bisa digunakan oleh beberapa proses agar proses-proses tersebut bisa saling berkomunikasi. Jika sebuah proses mengubah data yang ada maka secara otomatis proses lain juga dapat melihat perubahan yang ada . Segment yang dibuat oleh sebuah proses dapat ditulis dan dibaca oleh proses lain.

Sintaks-sintaks yang digunakan:

- **int shmget(key\_t key, size\_t size, int shmflg);**

Berfungsi untuk membuat shared memory

- ✓ **key\_t key:** merupakan key dari shared memory. Jika dua proses menggunakan key yang sama, maka kedua proses tersebut akan mendapatkan shmid yang sama. Jika diset 0 / IPC\_PRIVATE, system yang menentukan segment yang dipesan, sehingga shmid yang dikembalikan bisa berbeda-beda.
- ✓ **size\_t size:** ukuran dari shared memory yang akan dibuat (dalam bytes).
- ✓ **int shmflg:** diisi dengan permission dari shared memory (seperti permission pada file, ada hak akses untuk user,group dan other) kemudian bisa dipipe dengan IPC\_CREAT dan IPC\_EXCL
  - IPC\_CREAT: digunakan jika ingin membuat shared memory dengan key tertentu.
  - IPC\_EXCL: gunanya jika kita ingin membuat shared memory dengan key tertentu dan jika ternyata pada key tersebut ada isinya maka shmget akan mengembalikan -1 (error)



- **int shmctl(int shmid, int cmd, struct shmid\_ds \*buf);**  
 Berfungsi untuk mengontrol shared memory.
    - ✓ **int shmid:** ID dari shared memory yang akan dikontrol.
    - ✓ **int cmd:** command yang akan dijalankan terhadap shared memory tersebut. Ada 3 cmd untuk user selain super user:
      - IPC\_STAT: Ambil keterangan dari shared memory dimasukkan dalam struct shmid\_ds
      - IPC\_SET: Mengeset shared memory pada bagian:
        - shm\_perm.uid
        - shm\_perm.gid
        - shm\_perm.mode //Lower order nine bits.
      - IPC\_RMID: Menghapus shared memory yang dibuat

Untuk super user terdapat tambahan cmd:

    - SHM\_LOCK
    - SHM\_UNLOCK  - ✓ **struct shmid\_ds\* buf:** struct yang berfungsi sebagai buffer yang digunakan untuk mengontrol shared memory.
- 
- **void \*shmat(int shmid, const void \*shmaddr, int shmflg);**  
 Berfungsi untuk mengattach suatu pointer ke shared memory agar kita bisa mengakses isi shared memory tersebut melalui pointer tersebut.
    - ✓ **int shmid:** ID dari shared memory yang akan di-attach.
    - ✓ **const void \*shmaddr:** diisi dengan address dari shared memory. biasanya diisi NULL, agar system yang akan menentukan sendiri alamat memorinya.
    - ✓ **int shmflg:** berisi hak akses dari shared memory

### Contoh Code server.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/types.h>
5  #include <sys/ipc.h>
6  #include <sys/shm.h>
7
8  struct data{
9      char kata[100];
10     int flag;
11 };
12
13 int main(){
14     system("clear");
15     printf("SERVER\n");
16
17     int shmid = shmget(123456, sizeof(struct data), IPC_CREAT|0666);
18     struct data *p = (struct data*)shmat(shmid, 0, 0666);
19
20     strcpy(p->kata, "");
21     p->flag = 0;
22     do{
23         if (p->flag == 1){
24             printf("Kata yang ditulis: %s\n", p->kata);
25             p->flag = 0;
26         }
27     }while(strcasecmp(p->kata, "exit") != 0);
28
29     shmdt(p);
30     shmctl(shmid, IPC_RMID, 0);
31     return 0;
32 }
```

### Contoh Code client.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/types.h>
5  #include <sys/ipc.h>
6  #include <sys/shm.h>
7
8  struct data{
9      char kata[100];
10     int flag;
11 };
12
13 int main(){
14     system("clear");
15     printf("SERVER\n");
16
17     char fflush;
18     int shmid = shmget(123456, sizeof(struct data), IPC_CREAT|0666);
19     struct data *p = (struct data*)shmat(shmid, 0, 0666);
20
21     strcpy(p->kata, "");
22     p->flag = 0;
23     do{
24         printf("Masukkan kata: ");
25         scanf("%s", p->kata); scanf("%c", &fflush);
26         p->flag = 1;
27     }while(strcasecmp(p->kata, "exit") != 0);
28
29     shmdt(p);
30     shmctl(shmid, IPC_RMID, 0);
31     return 0;
32 }
```

# Operating System – Semaphore

## Module 09

- Semaphore



Last Update 25 May 2011

Revision 1

### 1. Module Description

#### 1. Implementasi Semaphore

### 2. Learning Outcomes

- Setelah mempelajari materi ini, dapat menerapkan konsep semaphore dalam program.
- Setelah mempelajari materi ini, dapat memahami penggunaan semaphore dalam proses.

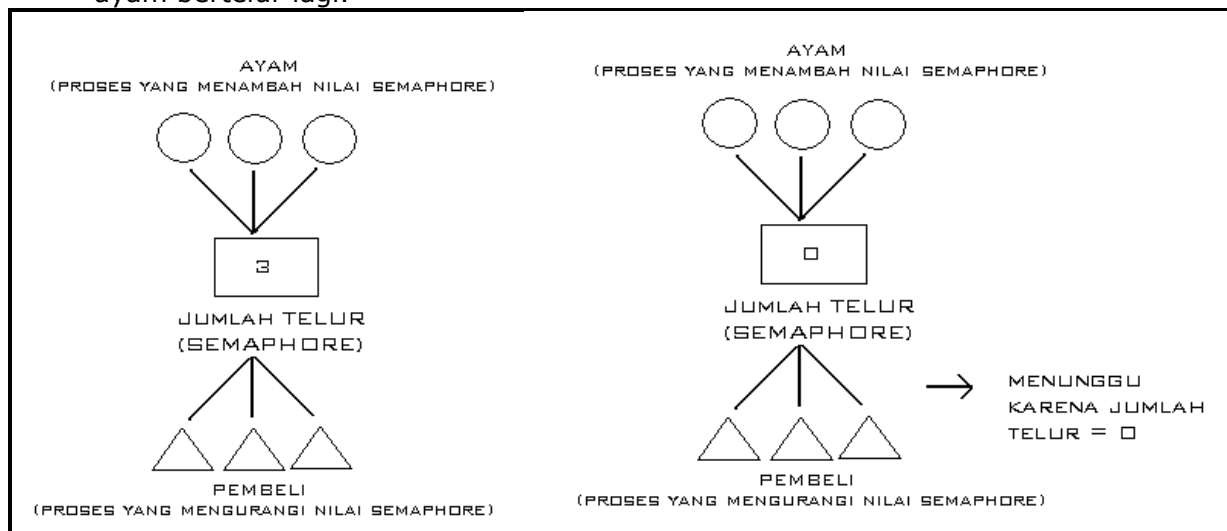
### 3. Material

#### 1. Implementasi Semaphore

Semaphore biasa digunakan untuk mengatur jalannya beberapa proses yang berjalan bersamaan. Wujud semaphore berupa counter (angka) yang dikenali oleh beberapa proses. Counter ini bisa ditambahkan atau dikurangi nilainya oleh proses. Untuk pengaturan jalannya proses, pengaturannya tergantung logika Anda.

Contoh:

Di dalam peternakan ayam, peternak mendapatkan telur dari ayam dan telur dibeli oleh pembeli. Jika suatu saat telur habis terjual dan ayam belum bertelur lagi, maka pembeli tidak dapat membeli telur tersebut, pembeli harus menunggu sampai ayam bertelur lagi.



Dalam contoh di atas, semaphore berfungsi untuk mengatur kapan suatu proses boleh berjalan dan kapan tidak (mengatur pembelian telur). Bila ternyata nilai dari semaphore itu adalah 0 dan terdapat proses yang mengurangnya (pembelian telur) maka proses yang mengurangi tersebut akan berhenti beberapa saat sampai ada proses lain yang menambah nilai dari semaphore tersebut (ayam bertelur).

Sintaks-sintaks yang digunakan:

- **int semget(key\_t key, int nsems, int semflg);**  
Berfungsi untuk membuat semaphore.
  - ✓ **key\_t key**: sebagai key dari semaphore. Penggunaan key pada semaphore sama seperti pada shared memory.
  - ✓ **int nsems**: Jumlah dari semaphore yang dibuat. Jika  $nsems > 1$  maka semaphore akan berupa array.
  - ✓ **int semflg**: diisi dengan permission dari semaphore. Penggunaan shmflag pada semaphore sama seperti shared memory.
- **int semctl(int semid, int semnum, int cmd, ...);**  
Berfungsi untuk mengontrol semaphore. Semctl bisa dipanggil dengan 3 atau 4 parameter.
  - ✓ **int semid**: ID dari semaphore yang akan dikontrol.
  - ✓ **int semnum**: menentukan semaphore mana yang akan digunakan . Bila hanya ada 1 semaphore maka diisi dengan 0. penggunaanya sama seperti indeks pada array.
  - ✓ **int cmd**: command yang akan dijalankan terhadap semaphore. Command yang sering digunakan:
    - GETVAL: Mengembalikan nilai val dari semaphore
    - SETVAL: Mengeset nilai dari semaphore sesuai dengan nilai val pada parameter semun.
    - GETPID: Mengembalikan nilai dari semaphore id (sempid)
    - IPC\_STAT: Ambil keterangan dari semaphore dimasukkan dalam struct shmid\_ds
    - IPC\_SET: Mengeset semaphore pada bagian:
      - sem\_perm.uid
      - sem\_perm.gid
      - sem\_perm.mode //low order nine bits
    - IPC\_RMID: Menghapus semaphore yang dibuat
  - ✓ **arg [optional]**: merupakan objek dari union semun. Biasa dipakai pada cmd SETVAL dan IPC\_STAT, IPC\_SET.
- **int semop(int semid, struct sembuf \*sops, size\_t nsops);**  
Digunakan untuk mengoperasikan semaphore (menambah atau mengurangi nilai dari semaphore).
  - ✓ **int semid**: ID dari semaphore yang akan dioperasikan.
  - ✓ **struct sembuf \*sops**: Struct yang digunakan untuk mengoperasikan semaphore. Struct sembuf terdiri dari:
    - **sem\_num**: Semaphore mana yang akan dioperasikan
    - **sem\_op**: Nilai yang mengubah semaphore
    - **sem\_flg**: Flag untuk menentukan operasi apa yang digunakan


- ✓ **size\_t nsops:** jumlah operasi yang dijalankan.

#### Contoh Code server.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/types.h>
5  #include <sys/ipc.h>
6  #include <sys/sem.h>
7
8  struct data {
9      char kata[100];
10 };
11
12 int main(){
13     system("clear");
14     printf("SERVER\n");
15
16     int shmid = shmget(IPC_PRIVATE, sizeof(struct data), IPC_CREAT|0666);
17     struct data *p = (struct data*)shmat(shmid, 0, 0666);
18
19     int semid = semget(IPC_PRIVATE, 1, IPC_CREAT|0666);
20     semctl(semid, 0, SETVAL, 0);
21
22     printf("SHMID: %d\n", shmid);
23     printf("SEMID: %d\n", semid);
24
25     struct sembuf op;
26     op.sem_num = 0;
27     op.sem_op = -1;
28     op.sem_flg = 0;
29
30     do{
31         semop(semid, &op, 1);
32         printf("Kata yang diketik: %s\n", p->kata);
33     }while(strcasecmp(p->kata, "exit") != 0);
34
35     shmctl(shmid, IPC_RMID, 0);
36     shmdt(p);
37     semctl(semid, 0, IPC_RMID);
38     return 0;
39 }
```

### Contoh Code client.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/types.h>
5  #include <sys/ipc.h>
6  #include <sys/sem.h>
7
8  struct data {
9      char kata[100];
10 };
11
12 int main(){
13     system("clear");
14     printf("SERVER\n");
15
16     char fflush;
17     int shmid, semid;
18
19     printf("Masukkan SHMID: ");
20     scanf("%d", &shmid); scanf("%c", &fflush);
21
22     printf("Masukkan SEMID: ");
23     scanf("%d", &semid); scanf("%c", &fflush);
24
25     struct data *p = (struct data*)shmat(shmid, 0, 0666);
26
27     semctl(semid, 0, SETVAL, 0);
28
29     struct sembuf op;
30     op.sem_num = 0;
31     op.sem_op = 1;
32     op.sem_flg = 0;
33
34     do{
35         printf("Masukkan kata: ");
36         scanf("%s", p->kata); scanf("%c", &fflush);
37         semop(semid, &op, 1);
38     }while(strcasecmp(p->kata, "exit") != 0);
39
40     shmctl(shmid, IPC_RMID, 0);
41     shmdt(p);
42     semctl(semid, 0, IPC_RMID);
43     return 0;
44 }
```

<b>Operating System – Thread</b>	
<b>Module 10</b> <ul style="list-style-type: none"> <li>• Thread</li> </ul>	
Last Update 25 May 2011	Revision 1

## 1. Module Description

### 1. Thread

## 2. Learning Outcomes

- Setelah mempelajari materi ini, dapat menerapkan konsep thread dengan semaphore.
- Setelah mempelajari materi ini, dapat Memahami konsep thread.
- Setelah mempelajari materi ini, dapat Memahami perbedaan thread dan semaphore.
- Setelah mempelajari materi ini, dapat Menerapkan thread dalam program.
- Setelah mempelajari materi ini, dapat Menerapkan thread dan semaphore.

## 3. Material

### 1. Thread

Apakah itu Thread? Thread merupakan sistem yang memungkinkan agar suatu proses dapat menjalankan beberapa hal sekaligus, atau melakukan banyak proses didalam satu proses. Definisi yang lebih jelas adalah bahwa thread itu adalah pengaturan kontrol pengeksekusian program didalam satu proses. Jadi setiap proses minimal menjalankan satu buah thread.

Penting bagi kita untuk mengetahui bedanya konsep thread dengan konsep Parent and Child. Didalam parent and child, ketika sebuah proses memanggil fungsi fork. Proses tersebut membuat kopi dan dirinya sendiri dengan PID dan variable yang berbeda, juga berjalan secara independent dari proses yang membuatnya. Sedangkan didalam thread, proses (thread) yang baru berjalan secara bergantian dengan proses yang membuatnya (secara cepat) tapi mempunyai stack memory ( dan variable lokal ) sendiri. Tapi variable global, file descriptor, handle signal, dan direktori aktifnya sama dengan yang digunakan oleh proses induk.



### Contoh Code thread.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <pthread.h>
5
6  void *thread1_function(){
7      int i;
8      for (i=0; i<5; i++){
9          sleep(1);
10         printf("Thread 1: %d\n", i);
11     }
12     pthread_exit(NULL);
13 }
14
15 void *thread2_function(void *arg){
16     int i;
17     for (i=0; i<5; i++){
18         sleep(1);
19         printf("Thread 2 - %s: %d\n", (char *)arg, i);
20     }
21     pthread_exit(NULL);
22 }
23
24 int main(){
25     pthread_t thread[2];
26     char message[] = "Message";
27     int res;
28
29     //THREAD 1
30     res = pthread_create(&thread[0], NULL, thread1_function, NULL);
31     if (res != 0) {
32         perror("Error thread 1\n");
33         exit(1);
34     }
35
36     //THREAD 2
37     res = pthread_create(&thread[1], NULL, thread2_function, (void *)message);
38     if (res != 0) {
39         perror("Error thread 2\n");
40         exit(1);
41     }
42
43     printf("Joining thread\n");
44     pthread_join(thread[0], NULL);
45     pthread_join(thread[1], NULL);
46
47     return 0;
48 }
```