Universitas Pertamina

Fakultas Sains dan Komputer

Program Studi Ilmu Komputer

Modul Praktikum Sistem Operasi

Disusun oleh:

Muhamad Koyimatu (116108)

Zuki Pristiantoro Putro (219056)

Februari 2023

Daftar Isi

Modul 1

Operasi Dasar Linux

1.1 Tujuan

1. Mampu mengoperasikan terminal Linux.
2. Mampu menggunakan beberapa perintah dalam file/directory.

1.2 Pendahuluan

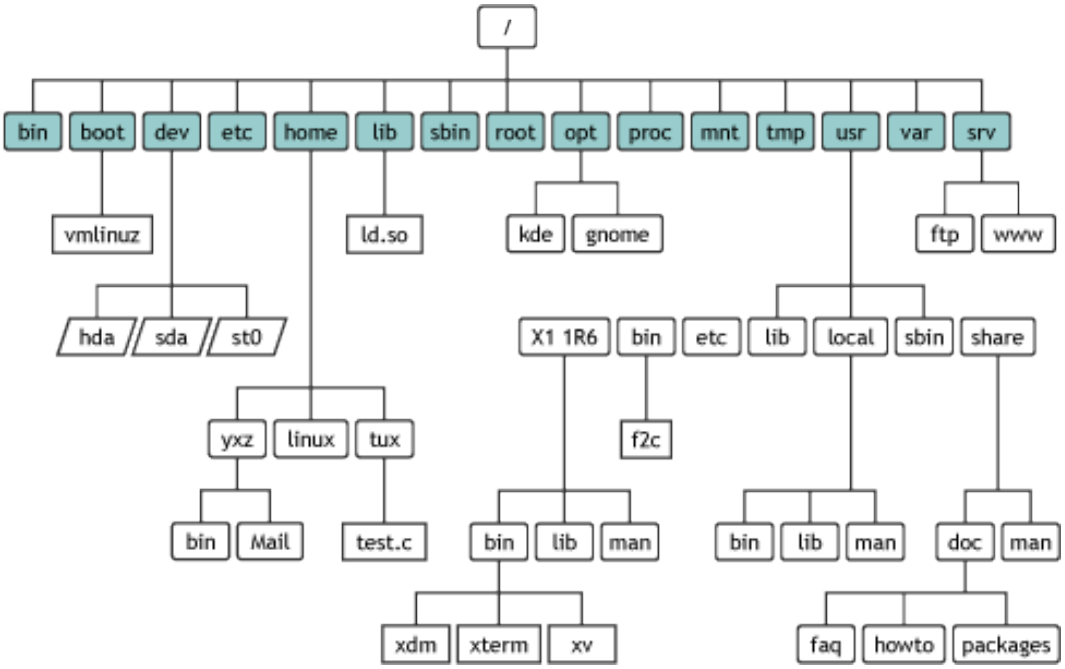
Linux adalah sistem operasi berbasis *open source* di bawah lisensi GNU. Dengan lisensi GNU (Gnu Not Unix), program dan sumber kodenya(source code) dapat diperoleh dengan lengkap. Termasuk hak untuk mengkopi dengan bebas, atau bahkan mengubah kode sumbernya. Hal tersebut bersifat legal dibawah lisensi. Meskipun gratis, lisensi GNU memperbolehkan pihak yang ingin menarik biaya untuk penggandaan maupun pengiriman program.

Linux pada awalnya dibuat oleh seorang mahasiswa Finlandia yang bernama Linus Torvalds. Dulunya Linux merupakan proyek hobi yang diinspirasikan dari Minix, yaitu sistem UNIX kecil yang dikembangkan oleh Andrew Tanenbaum. Linux versi 0.01 dikerjakan sekitar bulan Agustus 1991. Kemudian pada tanggal 5 Oktober 1991, Linus mengumumkan versi resmi Linux, yaitu versi 0.02 yang hanya dapat menjalankan shell bash (GNU Bourne Again Shell) dan gcc (GNU C Compiler).

Struktur direktori Linux berbentuk pohon. Dasar dari sistem hirarki dimulai dari root. Pemisah antar direktori pada Linux menggunakan simbol *forward slash* (`/`).

Berikut adalah beberapa direktori tingkat atas yang harus diketahui:

1. `/`"Root" - merupakan puncak dari hirarki sistem file. Semuanya berawal dari root. Ada istilah "cari di slash", memiliki arti cari di direktori root.
2. `/bin` - Binaries dan program lain yang dapat dieksekusi.
3. `/etc` - System configuration files.
4. `/home` - Home directories.
5. `/opt` - Optional atau third party software.
6. `/tmp` - Temporary space.



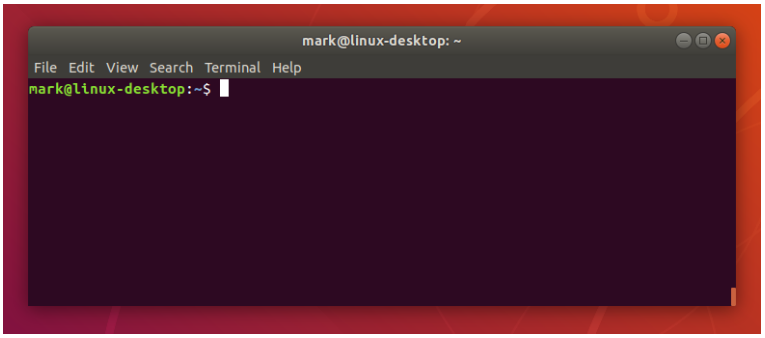
Gambar 1.1: Struktur direktori Linux

7. `/usr` - call User.
8. `/var` - Variable data.

1.3 Cara Kerja

1.3.1 Membuat folder kerja

Masuk ke sistem Linux yang digunakan. Kemudian jalankan aplikasi "Terminal"



Gambar 1.2: Tampilan awal Terminal pada Linux Ubuntu

Pada aplikasi terminal, silahkan masuk ke direktori kerja dan memastikan lokasi dengan perintah `pwd`.

Setelah masuk ke folder kerja, buat folder "sisop", diikuti dengan perintah untuk masuk ke folder "sisop"

```
$ mkdir sisop
$ cd sisop
```

Kemudian buat folder sesuai dengan nim peserta praktikum diikuti nama. misal

```
$ mkdir 105222000_asep
```

Masuk ke dalam folder tersebut, kemudian buat folder dengan nama "Modul 1". Kita akan bekerja di dalam folder Modul 1. Hirarkinya kerja kita adalah `~/sisop/105222xxx_nama/Modul 1/`. Sebagai catatan, untuk modul 2 dan seterusnya dikerjakan dengan membuat folder "Modul 2" dan seterusnya di bawah forlder NIM_nama yang sama.

1.3.2 Command dasar Linux

Linux memiliki command dasar yang dapat diakses langsung lewat terminal.

Basic Command

1. Date Command

   Perintah ini untuk menunjukkan tanggal dan jam saat ini

   ```
   $ date
   $ date +%ch
   ```

   Options:

   - a = Abbrevated weekday.
   - A = Full weekday.
   - b = Abbrevated month.
   - B = Full month.
   - c = Current day and time.
   - C = Display the century as a decimal number.
   - d = Day of the month.
   - D = Day in "mm/dd/yy" format
   - h = Abbrevated month day.
   - H = Display the hour.
   - L = Day of the year.
   - m = Month of the year.
   - M = Minute.
   - P = Display AM or PM
   - S = Seconds
   - T = HH:MM:SS format
   - u = Week of the year.
   - y = Display the year in 2 digit.
   - Y = Display the full year.
   - Z = Time zone .

2. Calender Command

   Digunakan untuk menunjukkan kalender.

```
$ cal <year>
$ cal <month> <year>
```

3. Echo Command

   Perintah untuk print suatu argumen pada layar.

```
$ echo <text>
$ echo ‘‘text
  >line2
  >line3’’
```

4. ‘who’ Command:

   Perintah untuk untuk menunjukkan pengguna yang terhubung pada sistem komputer saat ini.

```
$ who - options
```

   Options:

   - H – Display the output with headers.
   - b – Display the last booting date or time or when the system was lastely rebooted.

5. ‘who am i’ Command:

   Menunjukkan detail dari yang sedang bekerja

```
$ who am i
```

6. ‘CLEAR’ Command

   Digunakan untuk menghapus/membersihkan layar

```
$ clear
```

7. ‘MAN’ Command

   Digunakan untuk membantu melihat pilihan command yang tersedia. Mirip seperti help command.

```
$ man <command name>
```

8. LIST Command

   Digunakan untuk melihat semua file yang ada pada direktori.

```
$ ls -options <arguments>
```

   Jika tidak ada argumen, artinya mencari pada direktori yang sama.

   Options:

- a– used to list all the files including the hidden files.
- c– list all the files columnwise.
- d- list all the directories.
- m- list the files separated by commas.
- p- list files include "/" to all the directories.
- r- list the files in reverse alphabetical order.
- f- list the files based on the list modification date.
- x-list in column wise sorted order.

Directory Related Commands

1. Present Working Directory Command

   Untuk mencetak path lengkap dari working direktori.

   ```
   $ pwd
   ```

2. MKDIR Command

   Untuk membuat direktori baru.

   ```
   $ mkdir <directory name>
   ```

3. CD Command

   Untuk berpindah direktori.

   ```
   $ mkdir <directory name>
   ```

4. RMDIR Command

   Untuk menghapus direktori, namun bukan direktori yang sedang aktif.

   ```
   $ rmdir <directory name>
   ```

File Related Commands

1. Membuat File

   Untuk membuat file pada direktori aktif bisa menggunakan perintah CAT.

   ```
   $ cat ><filename>
   ```

   Simbol > menunjukkan redirektori pada perintah CAT. Contoh:

   ```
   $ cat coba.txt
   ```

   Akan menunggu input dari pengguna. silahkan ketikkan apapun (dipisahkan dengan ENTER) dan **CTRL+D** untuk keluar.

   ```
   Nama: Asep
   TTL: Jakarta, 1 Januari 2000
   Hobi: membaca
   ```

2. Menampilkan file

   Untuk menampilkan isi dari suatu file, bisa menggunakan CAT, tanpa operator "¿".

```
$ cat <filename>
```

   Option -s untuk mengabaikan error atau warning.

3. Menyalin isi file

   Untuk menyalin isi dari suatu file ke file lain. Jika file tujuan tidak ada, akan dibuat. Jika file tujuan sudah ada, akan ditimpa.

```
$ cat <filename source> >> <destination filename>
```

   Option -n isi dari file akan memasukkan baris kosong.

```
$ cat -n <filename>
```

   Selain itu, bisa juga menggunakan `cp`

```
$ cp <filename source> >> <destination filename>
```

4. Mengurutkan file

   Mengurutkan isi dengan urutan normal atau terbalik secara alfabet

```
$ sort <filename>
$ sort -r <filename>
```

5. Pindah file

   Memindahkan file ke lokasi yang dituju

```
$ mv <source filename> <destination filename>
```

6. Menghapus file

   Menghapus file secara permanen

```
$ rm <filename>
```

7. WORD command

   Menghitung jumlah baris, kata, dan karakter

```
$ wc<filename>
```

   Option:

   - -c - menampilkan jumlah karakter.
   - -l - menampilkan jumlah baris.
   - -w - menampilkan jumlah kata.

Filters and Pipes

1. HEAD

   Menampilkan 10 baris pertama dari suatu file

   ```
   $ head <filename>
   ```

2. TAIL

   Menampilkan 10 baris terakhir dari suatu file

   ```
   $ tail<filename>
   ```

3. MORE

   Menampilkan file perhalaman. Untuk melanjutkan scroll, menggunakan space bar.

   ```
   $ more <filename>
   ```

4. SORT

   Digunakan untuk mengurutkan data.

   ```
   $ sort <filename>
   ```

5. PIPE

   It is a mechanism by which the output of one command can be channelled into the input of another command.

   ```
   $ who | wc -l
   ```

6. TR

   `tr` digunakan untuk mentranslasikan serangkaian karakter ke bentuk standar yang lain.

   ```
   $ tr ‘‘[a-z]’’ ‘‘[A-Z]’’
   ```

   Contoh:

   ```
   $ cat coba.txt | tr ‘‘[a-z]’’ ‘‘[A-Z]’’
   ```

1.4 Tugas Pendahuluan

1. Buatlah catatan sederhana (maksimal 1 lembar, 2 halaman bolak-balik) berisi semua perintah dasar pada terminal yang sering digunakan atau bermanfaat.
2. Tuliskan pengucapan/pelafazan dalam bahasa Indonesia atau bahasa Inggris simbol-simbol yang ada pada keyboard. Misal: `~, @, $, ?, *`, dan lain-lain!

Modul 2

Operasi Dasar Terminal Linux

2.1 Pendahuluan

Ada berbagai pilihan text editor pada terminal Linux, yang paling umum adalah Vi atau Vim, Gedit, dan Nano.Vi editor adalah salah satu visual editor yang digunakan untuk membuat dan mengedit teks, file, dokumen dan program. Vi menampilkan isi dari file pada layar dan memungkinkan pengguna untuk menambah, menghapus atau mengganti bagian pada text. Ada 2 mode yang tersedia pada Vi text editor, yaitu:

1. Command mode, otomatis saat memulai Vi. Pada mode ini, Vi menginterpretasikan setiap karakter yang diketikkan sebagai command. Memungkinkan pengguna untuk menjelajahi file, delete, copy, atau paste suatu text. Untuk masuk ke Command mode, tekan tombol ESC.

2. Input (atau) insert mode, yang meungkinkan untuk mengetik pada file. Setiap yang diketikkan dianggap sebagai input text. Untuk masuk ke mode insert, cukup tekan tombol "i".

3. Last line mode (Escape mode), dipanggil dengan tombol titik dua ":". Kursor akan langsung berpindah ke baris terakhir dan menunggu suatu command. Biasa digunkan untuk save atau keluar dari file.

Compiler adalah software yang mampu mengubah source code yang ditulis dalam suatu bahasa komputer (source language) ke bahasa komputer lainnya (target language, biner atau object code). Kompiler standar pada kebanyakan sistem Unix adalah GNU Compiler Collection (gcc). Secara umum, gcc/g++ sangatlah felsibel karena dapat berjalan pada berbagai platform komputer. Banyak orang menggunakan gcc/g++ karena kemudahan untuk porting ke platform lainnya.

Praktikum Sistem Operasi sepenuhnya menggunakan Vi yang terinstall pada sistem Linux, yang tersedia pada Laboratorium Komputer 2404.

2.2 Cara Kerja

2.2.1 Membuat folder kerja

Masuk ke sistem Linux yang tersedia di Laboratorium Komputer 2404. Kemudian jalankan aplikasi "Terminal"

Pada aplikasi terminal, cek apakah sudah berada di Desktop dengan perintah `pwd`. jika belum masuk dengan menggunakan perintah

```
$ cd ~/Desktop
```

Setelah masuk ke Desktop, buat folder "sisop", diikuti dengan perintah untuk masuk ke folder "sisop"

```
$ mkdir sisop
$ cd sisop
```

Kemudian buat folder sesuai dengan nim peserta praktikum diikuti nama. misal

```
$ mkdir 105222000_asep
```

Masuk ke dalam folder tersebut, kemudian buat folder dengan nama "Modul 2". Kita akan bekerja di dalam folder Modul 2. Hirarkinya kerja kita adalah `~/sisop/105222xxx_nama/Modul 2/`.

2.2.2 Command dasar Vi

Memulai Vi

Vi editor dapat dipanggil dengan menggunakan command berikut pada terminal

```
$ vi <filename> #membuat file baru jika belum ada file sebelumnya, jika sudah ada akan
    membuka file tersebut.
$ vi - R <filename> #membuka file dalam read only mode
$ view <filename> #membuka file dalam read only mode
```

Command ini akan menampilkan layar dengan 25 baris dan simbol tilt( ) pada awal setiap baris. Syntax pertama akan menyimpan file sesuai dengan filename yang dituliskan jika file belum ada. Jika file sudah ada, akan membuka file tersebut untuk mengedit.

Option:

- `vi +n < filename>` untuk menentukan posisi kursor pada baris ke-n.

Navigation

Untuk bergerak pada file tanpa mengubah teks, harus berada pada command mode (tekan ESC dua kali). Berikut adalah beberapa command yang dapat digunakan untuk pergerakan kursor pada satu karakter.

- k : Memindahkan kursor ke atas satu baris.
- j : Memindahkan kursor ke bawah satu baris.
- h : Memindahkan kursor ke kiri satu karakter.

- l : Memindahkan kursor ke kanan satu karakter.
- 0 atau | : Memindahkan posisi kursor ke awal baris.
- $: Memindahkan posisi kursor ke akhir baris.
- W : Memindahkan posisi kursor ke kata berikutnya.
- B : Memindahkan posisi kursor ke kata sebelumnya.
- (: Memindahkan posisi kursor ke awal kalimat saat ini.
-) : Memindahkan posisi kursor ke awal kalimat berikutnya.
- H : Berpindah ke paling atas layar.
- nH : Berpindah sebanyak n garis dari atas layar.
- M : Berpindah ke tengah layar.
- L : Berpindah ke paling bawah layar
- nL : Berpindah sebanyak n garis dari bawah layar.
- colon/titik dua (:) diikuti x : titik dua diikuti dengan angka akan memindahkan kursor ke baris sekian yang direpresentasikan dengan x.

Scrolling

Command yang sangat berguna, dikombinasikan dengan CTRL

- CTRL+d : Berpindah 1/2 layar.
- CTRL+f : Berpindah satu layar penuh.
- CTRL+u : Berpindah mundur 1/2 layar.
- CTRL+b : Berpindah mundur satu layar penuh.
- CTRL+e : Berpindah layar ke atas satu baris.
- CTRL+y : Berpindah layar ke bawah satu baris.
- CTRL+u : Berpindah layar ke atas 1/2 halaman.
- CTRL+d : Berpindah layar ke bawah 1/2 halaman.
- CTRL+b : Berpindah layar ke atas satu halaman.
- CTRL+f : Berpindah layar ke bawah satu halaman.
- CTRL+I : Memuat ulang (redraw) layar.

Entering and Replacing Text

Untuk mengedit file, kita harus masuk ke insert mode. Ada beberapa cara masuk ke insert mode dengan menggunakan command mode.

- i : Insert text sebelum lokasi kursor.
- I : Insert text pada awal baris saat ini.

- a : Insert text setelah lokasi kursor.
- A : Insert text pada akhir baris saat ini.
- o : Membuat barus baru untuk input di bawah lokasi kursor.
- O : Membuat barus baru untuk input di atas lokasi kursor.
- r : Mengganti satu karakter pada kursor dengan karakter yang akan diketikkan berikutnya.
- R : Mengganti teks dari posisi kursor ke kanan.
- s : Mengganti satu karakter pada kursor dengan beberapa karakter.
- S : Mengganti seluruh baris.

Deleting Characters

Berikut adalah beberapa command penting yang dapat digunakan untuk menghapus karakter dan baris pada file yang dibuka.

- x : Menghapus satu karakter pada lokasi kursor.
- X : Menghapus satu karakter pada lokasi sebelum kursor.
- dw : Menghapus kata.
- d^ Menghapus dari posisi kursor hingga awal baris.
- d$: Menghapus dari posisi kursor hingga akhir baris.
- dd : Menghapus satu baris pada posisi kursor.

Copy Paste Command

Menyalin baris atau kata dari satu tempat ke tempat lainnya dengan menggunakan command berikut.

- yy : copy (yank, cut) baris saat ini ke buffer.
- 9yy : copy (yank, cut) 9 baris ke bawah termasuk baris saat ini ke buffer.
- p : put (paste) sejumlah baris yang ada di buffer ke teks setelah baris saat ini.
- P : put (paste) sejumlah baris yang ada di buffer ke teks sebelum baris saat ini.

Save and Exit Command

Perlu menekan ESC diikuti dengan titik dua sebelum mengetikkan command berikut:

- q : Quit
- q! : Quit tanpa save.
- r filename : membaca data dari file yang bernama filename.
- wq : Write and quit (save and exit).
- w filename : Write ke file yang bernama filename (save as).
- w! filename : Overwrite ke file yang bernama filename (save as forcefully).

- !cmd : Runs shell commands dan kembali ke Command mode.

Searching and Replacing

vi memiliki kemampuan search dan replace yang sangat baik. Syntax yang digunakan search adalah:

```
:s/string
```

Syntax yang digunakan untuk replace satu string adalah:

```
:s/pattern/replace
:s/yang akan diganti/pengganti
```

Syntax yang digunakan untuk replace seluruh string pada satu dokumen adalah:

```
:%s/pattern/replace
:%s/yang akan diganti/pengganti
```

2.2.3 Menulis Program Sederhana dengan vi

Tuliskan kode berikut pada vi dengan nama file hello.cpp

```
#include<iostream>
using namespace std;
int main()
{
cout<<"Hello World\n";
return 0;
}
```

2.3 Tugas Pendahuluan

1. Buatlah catatan sederhana (maksimal 1 halaman) berisi semua perintah dasar pada vi yang sering digunakan atau bermanfaat.
2. Tuliskan beberapa editor yang umum pada linux. Tuliskan kelebihan dan kekurangan dari masing-masing teks editor yang disebutkan!

Modul 3

Unix Shell Programming

3.1 Tujuan

1. Mempelajari Unix Shell Programming Commands
2. Mampu menulis program shell untuk konkatenasi dua string.
3. Mampu menulis program shell untuk membandingkan dua buah string

3.2 Pendahuluan

Shell adalah program yang menjembatani user dengan sistem operasi (kernel), umumnya shell menyediakan prompt sebagai user interface, tempat dimana user mengetikkan perintah-perintah yang diinginkan baik berupa perintah internal shell (internal command), ataupun perintah eksekusi suatu file progam (eksternal command), selain itu shell memungkinkan user menyusun sekumpulan perintah pada sebuah atau beberapa file untuk dieksekusi sebagai program.

Sistem operasi linux dilengkapi dengan banyak shell dengan kumpulan perintah yang banyak, sehingga memungkinkan pemakai memilih shell yang paling sesuai, dan juga dapat berpindah pindah dasi shell yang satu ke shell yang lain. Beberapa shell yang umum pada sistem linux antara lain:

1. C-shell -(csh), di ciptakan oleh Bill Joy, bahasa pemograman shell ini lebih sulit digunakan oleh pemula karena memiliki sintaks mirip bahasa C dan oleh karena itulah shell ini dinamakan C Shell. Kelebihan shell ini memiliki kemampuan interaktivitas yang lebih complatition untuk dapat melengkapi perintah yang belum lengkap dapat dilakukan dengan menekan tombol Tab.
2. Bourne Shell (bsh atau sh), di ciptakan oleh Steven Bourne, merupakan shell UNIX yang pertama dan tercepat pada semua syistem UNIX Bourne Shell memiliki bahasa pemograman shell yang baik tetapi kurang nyaman dalam hal Interaktivitas.
3. Korn Shell (ksh) diciptakan oleh Dave Korn, Shell ini diciptakan dengan mengabungkan kelebihan Bourne Shell dan C Shell sehingga shell ini memiliki Interaktivitas yang baik dan juga gaya pemograman shell yang mudah.

4. Buorne Again Shell (bash), shell ini dikembangkan oleh steven Bourne (Pencipta SH) di mana shell ini adalah pengembangan Bourne Shell (sh) yang sudah di lengkapi dengan berbagai kelebihan yang tidak terdapat pada versi sebelumnya. Shell ini juga di lengkapi dengan kelebihan pada C shell dan juga Korn Shell. Bash juga memiliki bahasa pemograman yang baik serta interaktivitas yang mudah di pahami. Bash adalah jenis shell yang paling banyak di gunakan pada saat ini.

5. TENEX C-Shell (tcsh), dikembangkan dari C-shell ditambahkan kemampuan emacs untuk mengedit command line Word completion dan speel correction.

Praktikum Sistem Operasi sepenuhnya menggunakan Vi yang terinstall pada sistem Linux, yang tersedia pada Laboratorium Komputer 2404.

3.3 Cara Kerja

3.3.1 Membuat folder kerja

Masuk ke sistem Linux yang tersedia di Laboratorium Komputer 2404. Kemudian jalankan aplikasi "Terminal"

Pada aplikasi terminal, cek apakah sudah berada di Desktop dengan perintah `pwd`. jika belum masuk dengan menggunakan perintah

```
$ cd ~/Desktop
```

Setelah masuk ke Desktop, buat folder "sisop", diikuti dengan perintah untuk masuk ke folder "sisop"

```
$ mkdir sisop
$ cd sisop
```

Kemudian buat folder sesuai dengan nim peserta praktikum diikuti nama. misal

```
$ mkdir 105222000_asep
```

Masuk ke dalam folder tersebut, kemudian buat folder dengan nama "Modul 3". Kita akan bekerja di dalam folder Modul 3. Hirarkinya kerja kita adalah `~/sisop/105222xxx_nama/Modul 3/`.

3.3.2 Shell Commands

Shell Keyword

```
echo, read, if fi, case, esac, for, while, do , done, until , set, unset, readonly, shift,
export, break, continue, exit, return, trap , wait, eval ,exec, ulimit , umask.
```

Hal umum Shell

The shbang line	shbang line merupakan baris awal pada script yang memungkinkan kernel untuk mengenali jenis shell yang digunakan. shbang line terdiri dari sebuah `#!` dan diikuti dengan pathname lengkap untuk shell. Kemudian diikuti dengan option. Contoh: `#!/bin/sh`
Comment	Komentar dapat dilakuka dengan menggunakan tanda `#` Contoh: `#ini adalah teks komentar`
Wildcard	Ada beberapa karakter yang mendapat perlakuan khusus untuk shell, dikenal sebagai metacharacter atau wildcards. Karakter ini bukan angka atau huruf. Misal `*,?,[]` digunakan untuk ekspansi filename, `<,>,2>,>>, |` merupakan standar I/O redirection dan pipes. Untuk melindungi metacharacter, bisa digunakan quote `''How are you?''` Contoh: `#rm *; ls ??; cat file[1-3]`

Variabel Shell

Variabel Shell dapat berubah pada saat eksekusi program. C shell menawarkan command "set" untuk menentukan value suatu variabel. Contoh:

```
% set myname= Fred
% set myname = "Fred Bloggs"
% set age=20
```

Tanda dollar"$" digunakan untuk memanggil nilai variabel. Contoh:

```
% echo $myname akan memunculkan Fred Bloggs pada layar
```

Tanda at "@" digunakan untuk menentukan nilai integer konstan. Contoh:

```
%@myage=20
%@age1=10
%@age2=20
%@age=$age1+$age2
%echo $age
```

Variabel lokal	variabel lokal pada scope shell yang sedang digunakan. Ketika script selesai, tidak tersimpan. Variabel lokal merupakan himpunan dan nilai. Contoh:
	`#variable namevalue`
	`#name=''Tohn Jerry''`
	`#x=5`

Variabel global	Variabel lingkingan, merupakan himpunan untuk shell yang sedang berjalan dan proses-proses lain yang muncul dari proses tersebut. Berakhir jika script selesai. Contoh:
	`#VARIABLE_NAME=value`
	`#export VARIABLE_NAME`
	`#PATH=/bin:/usr/bin:.`
	`#export PATH`

Mengekstrak nilai variabel	Dengan menggunakan simbol dollar "$". Contoh `echo` $ *variablename* `echo` $ *name*

Expression Command

Untuk semua operasi aritmatika

```
Var = ‘expr$value1 + $ value2’
```

Aritmatika	Bourne shell tidak mendukung aritmatika. Unix/Linux command digunakan untuk melakukan perhitungan. Contoh: `n=expr 5+5’` `echo` $ n
Operator	Bourne shell menggunakan built-in test command operator untuk mengetes angka dan string. Contoh Equality `=; !=; -eq; -ne` Logika `-a; -o; !` Logika `AND &&; OR ||` Relational `-gt; -ge; -lt; -le` Aritmatika `+; -; \*; /; %`
Argumen	Argumen dapat diberikan ke script melalui command line. Contoh Pada command line: $ `scriptname arg1 arg2 arg3 ...` Pada script: echo $1 $2 $3 Pada script: echo $* Pada script: echo $#

READ Statement

Untuk mendapatkan input dari user

```
read x y #tidak perlu ada koma diantara variabel
```

ECHO Statement

Mirip dengan output statement, untuk mengeprint sesuatu di layar. Wildcard bisa menggunakan simbol backslash atau quote.

```
echo ‘‘String’’ (atau) echo $ b(untuk variabel)
echo ‘‘What is your name?’’
```

Conditional Statement

If diikuti dengan suatu command. Jika suatu ekpresi dites, ditutup dengan square bracket. Keyword “then” diletakkan setelah closing parenthesis. Suatu if harus diakhiri dengan fi.

1. if, digunkan untuk mengecek kondisi dan jika memenuhi, akan melakukan aksi berikutnya. Jika tidak akan langsung pindah ke bagian selanjutnya.
2. if...else

```
If cp $ source $ target
Then
          Echo File copied successfully
Else
```

```
        Echo Failed to copy the file
```

3. nested if

```
if condition
then
        command
    if condition
    then
        command
    else
        command
   fi
fi
```

4. case...esac, Digunakan untuk mengeksekusi shell script yang berdasarkan pilihan.

Contoh if

```
if command
then
   block of statements
elif command
then
   block of statements
elif command
then
   block of statements
else
   block of statements
fi
-----------------------------
if [ expression ]
then
   block of statements
elif [ expression ]
then
   block of statements
elif [ expression ]
then
   block of statements
else
   block of statements
fi
---------------------------------
```

Contoh case

```
case variable_name in
   pattern1)
      statements
        ;;
   pattern2)
```

```
      statements
         ;;
   pattern3)
      ;;
   *) default value
      ;;
esac
case "$color" in
   blue)
      echo $color is blue
      ;;
   green)
      echo $color is green
      ;;
   red|orange)
      echo $color is red or orange
      ;;
   *) echo "Not a color" # default
esac
```

Contoh if/else

```
if [ expression ]
then
   block of statements
else
   block of statements
fi
-------------------------------------
```

LOOP

Ada 3 jenis loop, while, until, dan for. Wjile loop diikuti dengan command atau ekspresi yang ditutup square bracket, suatu do keyword, suatu blok statement, dan diakhiri dengan done keyword. Selama statement benar, body statement antara do dan done akan terus dieksekusi

Until loop mirip dengan while loop, kecuali pada body loop akan dieksekusi selama ekspresi bernilai salah.

For loop digunakan untuk mengiterasi sekumpulan kata, memproses kata kemudian selesai, untuk memproses kata dan menggantinya. Berhenti ketika semua kata telah berganti. For loop diikuti variabel nama, in keyword, list kata kemudian blok statement, dan diakhiri dengan done keyword.

Contoh:

```
while command
do
   block of statements
done
-----------
while [ expression ]
do
   block of statements
```

```
done
until command         for variable in word1 word2 word3 ...
do                 do
   block of statements     block of statements
done               done
-----------
until [ expression ]
do
   block of statements
done
------------
until control command
do
   commands
done
```

Eksekusi Shell script

1. Menggunakan change mode command
2. $ chmod u + x sum.sh
3. $ sum.sh atau $ sh sum.sh

3.4 Tugas Pendahuluan

1. Tuliskan semua jenis shell yang kalian ketahui, berikan sedikit penjelasan atau kelebihan atau perbedaan dengan shell yang lain.
2. Perhatikan shell script sederhana berikut:

   ```
   #!/bin/sh
   for i in 1 2 3 4 5
   do
     echo "Loop ... nomor $i"
   done
   ```

   Jelaskan maksud dari setiap baris pada shell di atas. Kemudian perkirakan output yang akan dikeluarkan?

Modul 4

Shell Programming

4.1 Tujuan

1. Mampu menulis program shell untuk konkatenasi dua string.
2. Mampu menulis program shell untuk membandingkan dua buah string.
3. Mampu menulis program shell untuk menentukan nilai terbesar dari tiga angka.
4. Mampu menulis program shell untuk menghasilkan deret fibonacci.
5. Mampu menulis program shell untuk melakukan operasi aritmatika dengan menggunakan `case`.

4.2 Algoritma

4.2.1 Konkatenasi dua string

1. Masuk ke vi editor dan masuk ke insert mode.
2. Baca string pertama.
3. Baca string kedua.
4. Konkatenasi dua string.
5. Masuk ke escape mode dan eksekusi dan verifikasi output yang dihasilkan.

4.2.2 Membandingkan dua string

1. Masuk ke vi editor dan masuk ke insert mode.
2. Baca string pertama.
3. Baca string kedua.
4. Bandingkan dua string dengan menggunakan loop `if`.

5. Jika kondisi memenuhi, maka print bahwa kedua string tersebut equal. Selain itu print bahwa kedua string tersebut tidak equal.
6. Masuk ke escape mode dan eksekusi dan verifikasi output yang dihasilkan.

4.2.3 Nilai maksimum dari tiga angka

1. Deklarasikan tiga variabel.
2. Cek jika A lebih besar dari B dan C.
3. Jika iya, print A is greater.
4. Else cek jika B lebih besar dari C.
5. Jika iya, print B is greater.
6. Else print C is greater.

4.2.4 Deret Fibonacci

1. Set nilai awal a=0 dan b=1.
2. Print nilai a dan b.
3. Jumlahkan nilai a dan b. Simpan nilai yang telah dijumlahkan ke variabel c.
4. Print nilai dari c.
5. Inisialisasi a adalah b dan b adalah c.
6. Ulangi langkah ke-3,4,5 hingga nilai a kurang dari n.

4.2.5 Operasi aritmatika dengan `case`

1. Read input variabel dan berikan value-nya.
2. Print berbagai operasi aritmatika yang akan digunakan.
3. Menggunakan operator `case` beri berbagai fungsi untuk operator aritmatika.
4. Print hasilnya dan hentikan eksekusi.

4.3 Cara Kerja

4.3.1 Konkatenasi dua string

1. Program

```
echo ‘‘enter the first string’’
read str1
echo ‘‘enter the second string’’
read str2
echo ‘‘the concatenated string is’’ $str1$str2
```

2. Contoh I/P

```
Enter first string: Hello
Enter first string: World
```

3. Contoh O/P

```
The concatenated string is HelloWorld
```

4.3.2 Membandingkan dua string

1. Program

```
echo ‘‘enter the first string’’
read str1
echo ‘‘enter the second string’’
read str2
if [ $str1 = $str2 ]
then
echo ‘‘strings are equal’’
else
echo ‘‘strings are unequal’’
fi
```

2. Contoh I/P:1

```
Enter first string: hai
Enter first string: hai
```

3. Contoh O/P:1

```
The two strings are equal
```

4. Contoh I/P:2

```
Enter first string: hai
Enter first string: halo
```

5. Contoh O/P:2

```
The two strings are not equal
```

4.3.3 Nilai maksimum dari tiga angka

(a) Program

```
echo ‘‘enter A’’
read a
echo ‘‘enter B’’
read b
echo ‘‘enter C’’
read c
if [ $a -gt $b -a $a -gt $c ]
```

```
then
echo ‘‘A is greater’’
elif [ $b -gt $a -a $b -gt $c ]
then
echo ‘‘B is greater’’
else
echo ‘‘C is greater’’
fi
```

(b) Contoh I/P

```
Enter A:23
Enter B:45
Enter C:67
```

(c) Contoh O/P

```
C is greater.
```

4.3.4 Deret Fibonacci

(a) Program

```
echo enter the number
read n
a=-1
b=1
i=0
while [ $i -le $n ]
do
t=’expr $a + $b’
echo $t
a=$b
b=$t
i=’expr $i + 1
done
```

(b) Contoh I/P

```
Enter the no: 5
```

(c) Contoh O/P

```
0
1
1
2
3
5
```

Operasi aritmatika dengan `case`

(a) Program

```
echo 1.Addition
echo 2.Subraction
echo 3.Multiplication
echo 4.Division
echo enter your choice
read a
echo enter the value of b
read b
echo enter the value of c
read c
echo b is $b c is $c
case $a in
1)d='expr $b + $c'
echo the sum is $d
;;
2)d='expr $b - $c'
echo the difference is $d
;;
3)d='expr $b \* $c'
echo the product is $d
;;
4)d='expr $b / $c'
echo the quotient is $d
;;
esac
```

(b) Contoh I/P

```
   1.Addition
   2.Subraction
   3.Multiplication
   4.Division
Enter your choice:1
Enter the value of b:3
Enter the value of c:4
b is 3 c is 4
the sum is 7
```

(c) Contoh O/P

```
b is 3 c is 4
the sum is 7
```

4.4 Tugas Pendahuluan

1. Tuliskan operasi-operasi dasar pada string (khususnya pada shell)!
2. Tuliskan operasi-operasi dasar aritmatika pada shell dan cara menggunakannya?

Modul 5

System Calls (Proses)

5.1 Tujuan

1. Melihat proses id dari suatu proses.
2. Menghentikan suatu proses dengan `kill`
3. Membuat proses baru dengan `fork` dan `exec`
4. Mengimplementasikan `wait` dalam pembuatan proses baru.
5. Membuat proses *zombie* dan *orphan*

5.2 Teori dan Cara Kerja

Setiap program yang sedang berjalan disebut dengan proses. Proses dapat berjalan secara *foreground* atau *background*. Jalankan

```
$ ps -e
```

untuk melihat semua proses yang sedang berjalan.

5.2.1 Proses ID

Proses-proses dapat diibaratkan seperti orang tua (*parent*) dengan anak (*child*) yang turun temurun.

- Setiap proses memiliki parent dan child.
- Setiap proses memiliki ID (*pid*) dan parent ID (*ppid*), kecuali proses `init` atau `systemd`.
- ppid dari sebuah proses adalah ID dari parent proses tersebut. Perhatikan *ascii art* dibawah:

```
 [Parent]           [Child]
+--------+        +--------+
| pid=7  |        | pid=10 |
| ppid=4 |------->| ppid=7 |
```

```
| bash   |        | nano   |
+--------+        +--------+
```

Perhatikan, ppid dari proses `nano` adalah pid dari proses `bash`.

Untuk memvisualisasikan semua hierarki parent-child, jalankan

```
$ pstree | less
```

pstree berfungsi untuk menampilkan suatu proses dalam bentuk tree. Tree dari proses yang ditampilkan memiliki pid atau init (jika pid dihilangkan) sebagai root.

Contoh program

```c
#include <stdio.h>
#include <unistd.h>

int main() {
  printf("The process ID is %d\n", (int) getpid());
  printf("The parent process ID is %d\n", (int) getppid());
  return 0;
}
```

Hasil

```
The process ID is 8295
The parent process ID is 29043
```

- getpid(): digunakan untuk memunculkan *process ID* dari proses yang dipanggil.
- getppid(): digunakan untuk memunculkan *process ID* dari **parent** proses yang dipanggil.

5.2.2 Melihat Proses

Jalankan program

```
$ ps -e -o pid,ppid,command
```

```
  PID  PPID COMMAND
    1     0 /sbin/init splash
    2     0 [kthreadd]
    6     2 [ksoftirqd/0]
    7     2 [rcu_sched]
    8     2 [rcu_bh]
    9     2 [migration/0]
   10     2 [lru-add-drain]
   11     2 [watchdog/0]
 .......
 (long list)
 .......
 3760  2684 /usr/lib/x86_64-linux-gnu/notify-osd
 3789  2684 /opt/google/chrome/chrome
25793  9789 ps -e -o pid,ppid,command
```

- ps : menampilkan rincian dari proses yang sedang berjalan.
- -e : memilih seluruh proses yang sedang berjalan.
- -o : format yang ditentukan user.

5.2.3 Membunuh Proses

Membunuh proses menggunakan `$ kill {pid}` Contoh:

```
$ kill 3789
```

Terdapat beberapa macam signal yang digunakan dalam command kill, antara lain sebagi berikut :

Signal name	Signal value	Effect
SIGHUP	1	Hangup
SIGINT	2	Interupt from keyboard
SIGKILL	9	Kill signal
SIGTERM	15	Termination signal
SIGSTOP	17,19,23	Stop the process

Secara *default*, `$ kill` menggunakan signal SIGTERM. Untuk menggunakan signal tertentu, gunakan `$ kill -{signal value} {pid}`. Contoh, `$ kill -9 3789` untuk menggunakan SIGKILL.

5.2.4 Membuat proses

Proses dapat dibuat menggunakan dua cara (pada C), yaitu dengan `system()` atau `fork` & `exec`. `fork` dan `exec` adalah bagian dari system call, sedangkan `system` bukan.

- `fork` digunakan untuk menduplikasi program yang sedang berjalan.
- `exec` digunakan untuk mengganti program yang sedang berjalan dengan program yang baru.

Menggunakan fork

`fork` digunakan untuk menduplikasi proses. Proses yang baru disebut dengan *child* proses, sedangkan proses pemanggil disebut dengan *parent* proses. Spesifikasi `fork` bisa dilihat dengan `$ man 2 fork`.

Setelah `fork` dipanggil, kita tidaklah tahu proses manakah yang pertama selesai.

```
$ man 2 fork
....
RETURN VALUE
       On success, the PID of the child process is returned in the parent,
       and 0 is returned in the child. On failure, -1 is returned in the
       parent, no child process is created, and errno is set appropriately.
....
```

```
int main() {
                          pid: 23, ppid: 10
                           [Main process]
                               |
```

```
  fork();            > Child process created <
                               +
                             /   \
                            /     \
              pid: 23, ppid: 10  pid: 30, ppid: 23
               [Parent Process]  [Child Process]

  return 0;
}
```

Perhatikan, bahwa:

- `pid` Parent Process == `ppid` Child
- `child_id` Parent Process == `pid` Child Process

Menggunakan exec

Exec adalah function yang digunakan untuk menjalankan program baru dan mengganti program yang sedang berlangsung. `exec` adalah program family yang memiliki berbagai fungsi variasi, yaitu `execvp`, `execlp`, `execv`, dan lain lain.

Manual: `$ man 3 exec`

```c
#include <stdio.h>
#include <unistd.h>

int main () {

  // argv[n] = { {your-program-name}, {argument[1]}, {argument[2]},.....,{argument[n-2]},
      NULL }
  char *argv[4] = {"list", "-l", "/", NULL};

  execv("/bin/ls", argv);

  printf("This line will not be executed\n");

  return 0;
}
```

Menggunakan fork dan exec

Permasalahan: Bagaimana cara menjalankan dua proses dalam satu program?

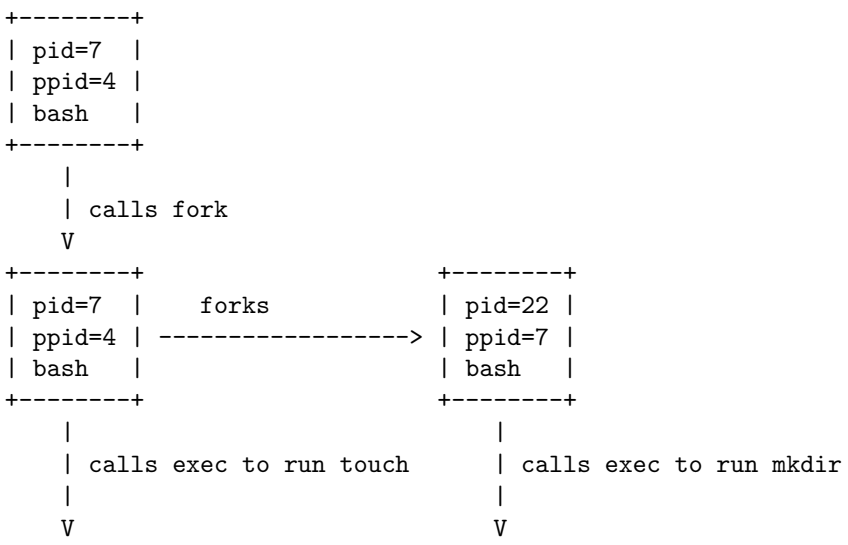
Contoh Permasalahan: Bagaimana cara membuat folder `/sisop` dan membuat file kosong bernama `/process.c`?

Maka, bagaimana cara menjalankan `mkdir` dan `touch` dalam satu program?

Solusi: Gunakan fork dan exec!

Buat sebuah program dengan:

Buat proses baru dengan `fork` Jalankan `exec` yang memanggil `mkdir` pada child process Jalankan `exec` yang memanggil `touch` pada parent process Visualisasi



Program

```c
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
  pid_t child_id;

  child_id = fork();

  if (child_id < 0) {
    exit(EXIT_FAILURE);
  }

  if (child_id == 0) {
    // this is child

    char *argv[] = {"mkdir", "sample-dir", NULL};
    execv("/bin/mkdir", argv);
  } else {
    // this is parent

    char *argv[] = {"touch", "sample-touch.txt", NULL};
    execv("/usr/bin/touch", argv);
  }
}
```

Menggunakan wait

`wait` adalah function yang digunakan untuk mendapatkan informasi ketika child proses berganti *state*-nya. Pergantian state dapat berupa *termination*, *resume*, atau *stop*. Pada modul ini, kita hanya menggunakan wait untuk menangkap state *termination*.

Fungsi `wait` pada parent process juga berguna untuk menangkap *exit* status dari child.

Permasalahan: Bagaimana cara membuat program yang menjalankan suatu proses tanpa menghentikan program?

Contoh permasalahan: Bagaimana cara membuat folder `/sisop` dan membuat file kosong bernama `/process.c` di dalamnya?

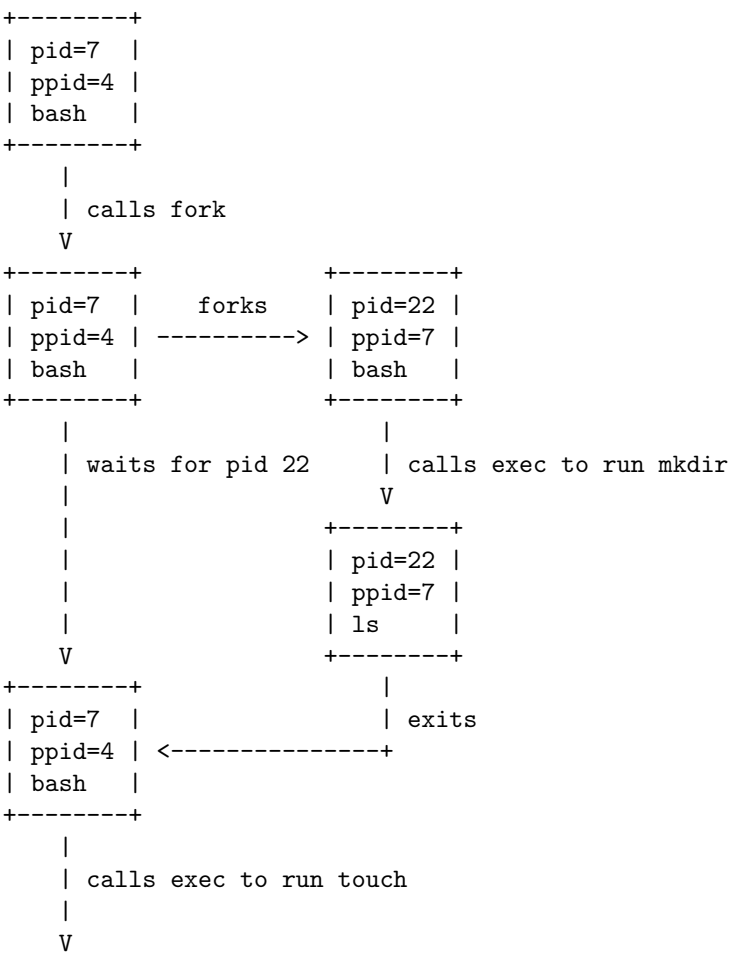
Maka, bagaimana cara menjalankan `mkdir` lalu menjalankan `touch` dalam satu program?

Solusi: Gunakan `fork`, `exec`, dan `wait`!

Buat sebuah program dengan:

1. Buat proses baru dengan `fork`
2. Jalankan `exec` yang memanggil `mkdir` pada child process
3. Buat parent process menunggu (`wait`) hingga proses pada child selesai
4. Setelah child selesai, jalankan `exec` yang memanggil `touch` pada parent

Visualisasi



Contoh:

```
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
  pid_t child_id;
  int status;

  child_id = fork();

  if (child_id == 0) {
    // this is child

    char *argv[4] = {"mkdir", "-p", "sample-dir", NULL};
    execv("/bin/mkdir", argv);
  } else {
    // this is parent

    // the parent waits for all the child processes
    while ((wait(&status)) > 0);

    char *argv[3] = {"touch", "sample-dir/sample-touch.txt", NULL};
    execv("/usr/bin/touch", argv);
  }
}
```

5.2.5 Menggunakan system

Ketika system dijalankan, program hanya memanggil `external command` (kalau di Ubuntu berupa program `/bin/bash`). Penggunaan `system` sangat tergantung pada environment.

Contoh, ketika user menjalankan `system("ls -l")`, ini sama seperti menjalankan `$ ls -l` pada bash.

Meskipun mudah digunakan, tidak disarankan menggunakan fungsi `system` karena beberapa asalan:

```
$ man system

  ....
  NOTES
  system() provides simplicity and convenience: it handles  all
  of  the details of calling fork(2), execl(3), and waitpid(2),
  as well as the necessary manipulations of signals.

  Do not use system() from a program with set-user-ID  or  set-
  group-ID privileges, because strange values for some environ
  ment variables might be used  to  subvert  system  integrity.
  ...
```

Contoh

```c
#include <stdlib.h>

int main() {
  int return_value;
  return_value = system("ls -l /");
  return return_value;
}
```

Hasil

```
total 156
drwxr-xr-x   2 root root  4096 Sep 14 06:35 bin
drwxr-xr-x   4 root root  4096 Sep 20 00:24 boot
drwxrwxr-x   2 root root  4096 Agu 14 14:05 cdrom
drwxr-xr-x   3 root root  4096 Sep 12 19:11 data
(long list)
```

5.2.6 Jenis-Jenis Proses

Proses Zombie

Zombie proses adalah child proses yang telah selesai mengerjakan tugasnya, namun belum ditangkap oleh parentnya dengan fungsi wait.

```
PID     PPID    STAT  COMMAND
28621   31403   S+    ./sample-zombie-process
28622   28621   Z+    [sample-zombie-p] <defunct>
```

Status dari zombie process adalah Z atau terdapat ¡defunct¿ di akhir commandnya. Contoh:

```c
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main () {
  pid_t child_pid;

  /* Create a child process. */
  child_pid = fork ();

  if (child_pid < 0) {
    exit(EXIT_FAILURE);
  }

  if (child_pid == 0) {
    /* This is the child process. Exit immediately. */
    exit (0);
  } else {
    /* This is the parent process. Sleep for a minute. */
    sleep (60);
  }

  return 0;
}
```

Proses Orphan

Orphan process adalah child process yang yang parent processnya telah berhenti.

```
PID    PPID     STAT  COMMAND
    1     0     Ss    /sbin/init
28369     1     S     ./sample-orphan-process
```

Contoh:

```c
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main () {
  pid_t child_pid;

  /* Create a child process. */
  child_pid = fork ();

  if (child_pid < 0) {
    exit(EXIT_FAILURE);
  }
    if (child_pid == 0) {
    /* This is the child process. Sleep for a minute. */
    sleep (60);
  } else {
    /* This is the parent process. Exit immediately. */
    exit (0);
  }

  return 0;
}
```

5.3 Tugas Pendahuluan

1. Tuliskan perbedaan antara proses dan thread!
2. Apakah yang dimaksud dengan proses ID, mengapa proses ID penting?
3. `top` dan `ps` memberikan informasi persentase penggunaan CPU. Namun untuk proses yang sama, nilainya tidak sama. Mengapa hal ini bisa terjadi?

Modul 6

Algoritma Penjadwalan CPU

6.1 Tujuan

1. Membuat program C yang mensimulasikan algoritma penjadwalan CPU.
2. Menentukan nilai turnaround time (TAT) dan waiting time (WT) untuk algoritma penjadwalan FCSF.
3. Menentukan nilai turnaround time (TAT) dan waiting time (WT) untuk algoritma penjadwalan SJF.
4. Menentukan nilai turnaround time (TAT) dan waiting time (WT) untuk algoritma penjadwalan Round Robin.
5. Menentukan nilai turnaround time (TAT) dan waiting time (WT) untuk algoritma penjadwalan Prioritas.

6.2 Teori dan Cara Kerja

Multiprogramming bertujuan untuk memaksimalkan kinerja CPU dengan cara mengatur alokasi waktu yang digunakan oleh CPU, sehingga CPU terus bekerja dan meminimalkan CPU idle. Untuk mengakomodir hal tersebut, diperlukan adanya penjadwalan proses-proses yang ada pada sistem. Untuk sistem dengan prosesor tunggal, hanya ada satu proses yang dapat berjalan setiap waktunya. Jika ada lebih dari satu proses, maka proses yang lain harus menunggu mendapatkan giliran masuk dan menggunakan CPU.

6.2.1 FCFS

Algoritma penjadwalan FCFS bekerja dengan membaca jumlah proses pada sistem dan juga CPU burst time. Penjadwalan dilakukan dengan dasar waktu kedatangan proses. Setiap proses akan dieksekusi sesuai urutan kedatangan.

6.2.2 SJF

Algoritma penjadwalan FCFS bekerja dengan membaca jumlah proses pada sistem dan juga CPU burst time. Semua proses/job diurutkan berdasarkan nilai burst time. Jika dua proses atau lebih memiliki nilai burst time yang sama, maka diurutkan berdasarkan waktu kedatangannya (FCFS). Setiap proses dieksekusi berdasarkan nilai burst time.

6.2.3 Round Robin

Algoritma penjadwalan FCFS bekerja dengan membaca jumlah proses pada sistem, CPU burst time, dan besar time slice. Semua proses/job diurutkan berdasarkan nilai burst time. Jika dua proses atau lebih memiliki nilai burst time yang sama, maka diurutkan berdasarkan waktu kedatangannya (FCFS). Setiap proses dieksekusi berdasarkan nilai burst time.

6.2.4 Penjadwalan Prioritas

Algoritma penjadwalan FCFS bekerja dengan membaca jumlah proses pada sistem, CPU burst time, dan nilai prioritasnya. Semua proses diurutkan berdasarkan prioritas. Jika dua proses atau lebih memiliki nilai prioritas yang sama, maka diurutkan berdasarkan waktu kedatangannya (FCFS). Setiap proses dieksekusi berdasarkan nilai burst time.

6.3 Cara Kerja

6.3.1 FCFS

1. Program

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

```
   printf("\nAverage Waiting Time -- %f", wtavg/n);
   printf("\nAverage Turnaround Time -- %f", tatavg/n);
   getchar();
}
```

2. Contoh Input

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

3. Contoh Output

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

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

6.3.2 SJF

1. Program

```
#include<stdio.h>
int main()
{
   int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
   float wtavg, tatavg;
   fflush(stdin);
   printf("\nEnter the number of processes -- ");
   scanf("%d", &n);
   for(i=0;i<n;i++)
   {
      p[i]=i;
      printf("Enter Burst Time for Process %d -- ", i);
      scanf("%d", &bt[i]);
   }
   for(i=0;i<n;i++)
      for(k=i+1;k<n;k++)
         if(bt[i]>bt[k])
         {
            temp=bt[i];
            bt[i]=bt[k];
            bt[k]=temp;

            temp=p[i];
            p[i]=p[k];
            p[k]=temp;
         }
```

```
   wt[0] = wtavg = 0;
   tat[0] = tatavg = bt[0];
   for(i=1;i<n;i++)
   {
      wt[i] = wt[i-1] +bt[i-1];
      tat[i] = tat[i-1] +bt[i];
      wtavg = wtavg + wt[i];
      tatavg = tatavg + tat[i];
   }
   printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
   for(i=0;i<n;i++)
      printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
   printf("\nAverage Waiting Time -- %f", wtavg/n);
   printf("\nAverage Turnaround Time -- %f", tatavg/n);
   getchar();
}
```

2. Contoh Input

```
Enter the number of processes -- 4
Enter Burst Time for Process 0 -- 6
Enter Burst Time for Process 1 -- 8
Enter Burst Time for Process 2 -- 7
Enter Burst Time for Process 3 -- 3
```

3. Contoh Output

```
PROCESS  BURST TIME   WAITING TIME    TURNAROUND TIME
P0           3            0              3
P1           6            3              9
P2           7            9              16
P3           8            16             24

Average Waiting Time-- 7.000000
Average Turnaround Time -- 13.000000
```

6.3.3 Round Robin

1. Program

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

   for(i=0;i<n;i++)
   {
      printf("\nEnter Burst Time for process %d -- ", i+1);
```

```
      scanf("%d",&bu[i]);
      ct[i]=bu[i];
   }
   printf("\nEnter the size of time slice -- ");
   scanf("%d",&t);
   max=bu[0];
   for(i=1;i<n;i++)
      if(max<bu[i])
         max=bu[i];
   for(j=0;j<(max/t)+1;j++)
      for(i=0;i<n;i++)
         if(bu[i]!=0)
            if(bu[i]<=t)
            {
               tat[i]=temp+bu[i];
               temp=temp+bu[i];
               bu[i]=0;
            }
            else
            {
               bu[i]=bu[i]-t;
               temp=temp+t;
            }
   for(i=0;i<n;i++)
   {
      wa[i]=tat[i]-ct[i];
      att+=tat[i];
      awt+=wa[i];
   }
   printf("\nThe Average Turnaround time is -- %f",att/n);
   printf("\nThe Average Waiting time is -- %f ",awt/n);
   printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
   for(i=0;i<n;i++)
      printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
   getchar();
}
```

2. Contoh Input

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

Enter the size of time slice -- 3
```

3. Contoh Output

```
The Average Turnaround Time -- 15.666667
The Average Waiting Time-- 5.666667

PROCESS  BURST TIME   WAITING TIME    TURNAROUND TIME
```

1	24	6	30
2	3	4	7
3	3	7	10

6.3.4 Penjadwalan Prioritas

1. Program

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

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

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

            temp=pri[i];
            pri[i]=pri[k];
            pri[k]=temp;
         }
   wtavg = wt[0] = 0;
   tatavg = tat[0] = bt[0];
   for(i=1;i<n;i++)
   {
      wt[i] = wt[i-1] + bt[i-1];
      tat[i] = tat[i-1] + bt[i];

      wtavg = wtavg + wt[i];
      tatavg = tatavg + tat[i];
   }

   printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
   for(i=0;i<n;i++)
      printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
```

```
    printf("\nAverage Waiting Time is --- %f",wtavg/n);
    printf("\nAverage Turnaround Time is --- %f",tatavg/n);
    getchar();
}
```

2. Contoh Input

```
Enter the number of processes -- 5
Enter the Burst Time & Priority of Process 0 --- 10  3
Enter the Burst Time & Priority of Process 1 --- 1   1
Enter the Burst Time & Priority of Process 2 --- 2   4
Enter the Burst Time & Priority of Process 3 --- 1   5
Enter the Burst Time & Priority of Process 4 --- 5   2
```

3. Contoh Output

```
PROCESS  PRIORITY     BURST TIME      WAITING TIME    TURNAROUND TIME
1            1            1               0               1
4            2            5               1               6
0            3            10              6               16
2            4            2               16              18
3            5            1               18              19

Average Waiting Time is --- 8.200000
Average Turnaround Time is --- 12.000000
```

6.4 Tugas Pendahuluan

1. Berikan kelebihan dan kekurangan dari 4 algoritma penjadwalan CPU yang akan disimulasikan pada percobaan ini?
2. Berikan definisi untuk:
   - arrival time
   - burst time
   - waiting time
   - turnaround time

Modul 7

Manajemen Memory

7.1 Tujuan

1. Membuat program C yang mensimulasikan teknik manajemen memory MVT dan MFT.
2. Membuat program C yang mensimulasikan teknik alokasi memory secara berkelanjutan (contiguous).
3. Menentukan metode alokasi terbaik untuk memory.

7.2 Teori

7.2.1 Teknik Manajemen Memory

MFT (Multiprogramming with a Fixed number of Tasks) adalah salah satu teknik manajemen memory klasik dimana memory dipartisi ke ukuran patisi yang tetap dan setiap job/proses ditempatkan pada salah satu partisi. Memory ditugaskan untuk suatu partisi tidak akan berubah ukurannya. MVT (Multiprogramming with a Variable number of Tasks) adalah teknik manajemen memory dimana setiap job/proses mendapatkan sejumlah memory sesuai dengan kebutuhannya. Partisi memory yang dinamis dan berubah saat job/proses keluar dan masuk ke sistem. MVT lebih "efisien" untuk sumber daya user. MFT mengalami internal fragmentation dan MVT mengalami eksternal fragmentation

7.2.2 Teknik Alokasi Memory

Salah satu metode yang paling sederhana pada untuk mengalokasikan memory adalah dengan membagi memory ke beberapa partisi yang ukurannya tetap. Pada metode multi-partisi, ketika partisi kosong, sebuah proses dipilih dari antrian input dan dimuat ke partisi yang kosong. Ketika proses selesai, partisi akan dikosongkan/ dikembalikan untuk proses lainnya. Sistem Operasi memiliki tabel yang dapat memantau ketersediaan memory. Ketika proses datang dan memerlukan memory, harus disediakan suatu bagian dari memory untuk proses tersebut. Saat akan melakukan proses load dan swap ke main memory, dan jika ada lebih dari satu blok dengan ukuran yang sesuai, maka Sistem Operasi harus menentukan siapa yang menempati blok kosong terlebih dahulu.

Best-fit memilih block dengan ukuran yang mendekati ukuran permintaan. First-fit memilih blok pertama yang bisa ditempati. Worst-fit memilih blok dengan ukuran terbesar.

7.3 Cara Kerja

7.3.1 MFT

1. Program

```
#include<stdio.h>
int main()
{
int ms, bs, nob, ef,n, mp[10],tif=0;
int i,p=0;

fflush(stdin);
printf("Enter the total memory available (in Bytes) -- ");
scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- ");
scanf("%d", &bs);
nob=ms/bs;
ef=ms - nob*bs;
printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
   printf("Enter memory required for process %d (in Bytes)-- ",i+1);
   scanf("%d",&mp[i]);
}

printf("\nNo. of Blocks available in memory -- %d",nob);
printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
   printf("\n %d\t\t%d",i+1,mp[i]);
   if(mp[i] > bs)
      printf("\t\tNO\t\t---");
   else
   {
      printf("\t\tYES\t%d",bs-mp[i]);
      tif = tif + bs-mp[i];
      p++;
   }
}
if(i<n)
   printf("\nMemory is Full, Remaining Processes cannot be accomodated");

printf("\n\nTotal Internal Fragmentation is %d",tif);
printf("\nTotal External Fragmentation is %d",ef);
getchar();
}
```

2. Contoh Input

```
Enter the total memory available (in Bytes) -- 1000
Enter the block size (in Bytes) -- 300
Enter the number of processes -- 5
Enter memory required for process 1 (in Bytes) -- 275
Enter memory required for process 2 (in Bytes) -- 400
Enter memory required for process 3 (in Bytes) -- 290
Enter memory required for process 4 (in Bytes) -- 293
Enter memory required for process 5 (in Bytes) -- 100
No. of Blocks available in memory -- 3
```

3. Contoh Output

```
PROCESS  MEMORY REQUIRED   ALLOCATED     INTERNAL FRAGMENTATION
1             275             YES             25
2             400             NO              ---
3             290             YES             10
4             293             YES             7

Memory is Full, Remaining Processes cannot be accommodated
Total Internal Fragmentation is 42
Total External Fragmentation is 100
```

7.3.2 MVT

1. Program

```
#include<stdio.h>
int main()
{
int ms,mp[10],i, temp,n=0;
char ch = 'y';

fflush(stdin);
printf("\nEnter the total memory available (in Bytes)-- ");
scanf("%d",&ms);
temp=ms;
for(i=0;ch=='y';i++,n++)
{
   printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);
   scanf("%d",&mp[i]);
   if(mp[i]<=temp)
   {
      printf("\nMemory is allocated for Process %d ",i+1);
      temp = temp - mp[i];
   }
   else
   {
      printf("\nMemory is Full");
      break;
   }
   printf("\nDo you want to continue(y/n) -- ");
```

```
   scanf(" %c", &ch);
}
printf("\n\nTotal Memory Available -- %d", ms);

printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");
for(i=0;i<n;i++)
   printf("\n \t%d\t\t%d",i+1,mp[i]);
printf("\n\nTotal Memory Allocated is %d",ms-temp);
printf("\nTotal External Fragmentation is %d",temp);
getchar();
}
```

2. Contoh Input

```
Enter the total memory available (in Bytes) -- 1000
Enter memory required for process 1 (in Bytes) -- 400
Memory is allocated for Process 1
Do you want to continue(y/n) -- y
Enter memory required for process 2 (in Bytes) -- 275
Memory is allocated for Process 2
Do you want to continue(y/n) -- y
Enter memory required for process 3 (in Bytes) -- 550
```

3. Contoh Output

```
Memory is Full
Total Memory Available -- 1000

PROCESS    MEMORY ALLOCATED
1               400
2               275

Total Memory Allocated is 675
Total External Fragmentation is 325
```

7.3.3 WORST-FIT

1. Program

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

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

```
   printf("\nEnter the size of the blocks:-\n");
   for(i=1;i<=nb;i++)
   {
      printf("Block %d:",i);
      scanf("%d",&b[i]);
   }
   printf("Enter the size of the files :-\n");
   for(i=1;i<=nf;i++)
   {
      printf("File %d:",i);
      scanf("%d",&f[i]);
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(bf[j]!=1)
         {
            temp=b[j]-f[i];
            if(temp>=0)
            {
               ff[i]=j;
               break;
            }
         }
      }
      frag[i]=temp;
      bf[ff[i]]=1;
   }
   printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
   for(i=1;i<=nf;i++)
      printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
   getchar();
}
```

2. Contoh Input

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

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

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

3. Contoh Output

```
File No    File Size     Block No      Block Size    Fragment
1               1          1             5             4
```

```
2              4          3            7            3
```

7.3.4 BEST-FIT

1. Program

```
#include<stdio.h>
#define max 25
int main()
{
   int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
   static int bf[max],ff[max];
   fflush(stdin);

   printf("\nEnter the number of blocks:");
   scanf("%d",&nb);
   printf("Enter the number of files:");
   scanf("%d",&nf);
   printf("\nEnter the size of the blocks:-\n");
   for(i=1;i<=nb;i++)
      printf("Block %d:",i);scanf("%d",&b[i]);
   printf("Enter the size of the files :-\n");
   for(i=1;i<=nf;i++)
   {
      printf("File %d:",i);
      scanf("%d",&f[i]);
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(bf[j]!=1)
         {
            temp=b[j]-f[i];
            if(temp>=0)
               if(lowest>temp)
               {
                  ff[i]=j;
                  lowest=temp;
               }
         }
      }
      frag[i]=lowest;
      bf[ff[i]]=1;
      lowest=10000;
   }
   printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
   for(i=1;i<=nf && ff[i]!=0;i++)
      printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
   getchar();
}
```

2. Contoh Input

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

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

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

3. Contoh Output

```
File No    File Size     Block No      Block Size    Fragment
1               1          2             2             1
2               4          1             5             1
```

7.3.5 FIRST-FIT

1. Program

```
#include<stdio.h>
#define max 25
int main()
{
   int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
   static int bf[max],ff[max];
   fflush(stdin);

   printf("\n\tMemory Management Scheme - Worst Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d",&nb);
   printf("Enter the number of files:");
   scanf("%d",&nf);
   printf("\nEnter the size of the blocks:-\n");
   for(i=1;i<=nb;i++)
   {
      printf("Block %d:",i);
      scanf("%d",&b[i]);
   }
   printf("Enter the size of the files :-\n");
   for(i=1;i<=nf;i++)
   {
      printf("File %d:",i);
      scanf("%d",&f[i]);
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
```

```
            if(bf[j]!=1) //if bf[j] is not allocated
            {
               temp=b[j]-f[i];
               if(temp>=0)
                  if(highest<temp)
                  {
                     ff[i]=j;
                     highest=temp;
                  }
            }
         }
         frag[i]=highest;
         bf[ff[i]]=1;
         highest=0;
      }
      printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
      for(i=1;i<=nf;i++)
         printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
      getchar();
}
```

2. Contoh Input

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

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

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

3. Contoh Output

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

7.4 Tugas Pendahuluan

1. Apakah yang dimaksud dengan internal fragmentation dan external fragmentation pada memory?
2. Jelaskan perbedaan antara MVT dan MFT!
3. Apakah keuntungan dari penggunaan skema alokasi memory secara noncontiguous?
4. Berikan definisi untuk:(1) base address dan (2) offset?

Modul 8

Virtual Memory (Algoritma Page Replacement)

8.1 Tujuan

1. Membuat program C yang mensimulasikan algoritma page replacement.

8.2 Teori

8.2.1 Page Replacement

Page replacement merupakan dasar dari demand paging, yang melengkapai pemisahan antara logical memory dan physical memory. Dengan mekanisme ini, virtual memory yang sangat besar dapat disediakan untuk programmer dengan physical memory yang kecil. Terdapat beberapa algoritma page replacement. Setiap Sistem Operasi memiliki skema page replacement tersendiri.

1. FIFO

   Algoritma page replacement FIFO mengganti setiap page berdasarkan urutan waktu kedatangan. Jika harus dilakukan page replacement, dipilih page yang paling tua (lama).

2. LRU

   LRU mirip dengan FIFO menggunakan urutan kedatangan, namun ditambahkan informasi urutan dibaca/digunakan. Jika harus dilakukan page replacement, LRU memilih page yang paling lama tidak digunakan/dibaca.

3. LFU

   Algoritma page replacement LFU memerlukan counter berapa kali page tersebut telah diakses. Asumsi LFU adalah, jika sering digunakan saat ini, maka akan sering digunakan juga di kemudian. Page yang akan diganti adalah page dengan counter yang paling kecil, atau yang jarang diakses.

4. Optimal

Algoritma page replacement optimal memiliki jumlah page fault paling sedikit dibandingkan algoritma lainnya dan tidak akan mengalami anomali Belady. Ide dasarnya adalah dengan mengganti page yang tidak akan digunakan dalam waktu lama. Namun algoritma optimal cukup sulit diimplementasikan, karena memerlukan informasi reference string yang akan digunakan di masa depan.

8.3 Cara Kerja

8.3.1 FIFO

1. Program

```
#include<stdio.h>
main()
{
   int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
   fflush(stdin);
   printf("\n Enter the length of reference string -- ");
   scanf("%d",&n);
   printf("\n Enter the reference string -- ");
   for(i=0;i<n;i++)
      scanf("%d",&rs[i]);
   printf("\n Enter no. of frames -- ");
   scanf("%d",&f);
   for(i=0;i<f;i++)
      m[i]=-1;
   printf("\n The Page Replacement Process is -- \n");
   for(i=0;i<n;i++)
   {
      for(k=0;k<f;k++)
      {
         if(m[k]==rs[i])
         break;
      }
      if(k==f)
      {
         m[count++]=rs[i];
         pf++;
      }
      for(j=0;j<f;j++)
      printf("\t%d",m[j]);
      if(k==f)
      printf("\tPF No. %d",pf);
      printf("\n");
      if(count==f)
      count=0;
   }
   printf("\n The number of Page Faults using FIFO are %d",pf);
   getchar();
}
```

2. Contoh Input

```
Enter the length of reference string -- 20
Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter no. of frames -- 3
```

3. Contoh Output

```
The Page Replacement Process is --
   7  -1   -1           PF No. 1
   7  0    -1           PF No. 2
   7  0    1            PF No. 3
   2  0    1            PF No. 4
   2  0    1
   2  3    1            PF No. 5
   2  3    0            PF No. 6
   4  3    0            PF No. 7
   4  2    0            PF No. 8
   4  2    3            PF No. 9
   0  2    3            PF No. 10
   0  2    3
   0  2    3
   0  1    3            PF No. 11
   0  1    2            PF No. 12
   0  1    2
   0  1    2
   7  1    2            PF No. 13
   7  0    2            PF No. 14
   7  0    1            PF No. 15

The number of Page Faults using FIFO are 15
```

8.3.2 LRU

1. Program

```
#include<stdio.h>
main()
{
   int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;
   fflush(stdin);
   printf("Enter the length of reference string -- ");
   scanf("%d",&n);
   printf("Enter the reference string -- ");
   for(i=0;i<n;i++)
   {
      scanf("%d",&rs[i]);
      flag[i]=0;
   }
   printf("Enter the number of frames -- ");
   scanf("%d",&f);
   for(i=0;i<f;i++)
   {
      count[i]=0;
```

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

2. Contoh Input

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

3. Contoh Output

```
The Page Replacement Process is --
```

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

The number of Page Faults using LRU are 12
```

8.3.3 LFU

1. Program

```
#include<stdio.h>
int main()
{
   int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;
   fflush(stdin);
   printf("\nEnter number of page references -- ");
   scanf("%d",&m);
   printf("\nEnter the reference string -- ");
   for(i=0;i<m;i++)
      scanf("%d",&rs[i]);
   printf("\nEnter the available no. of frames -- ");
   scanf("%d",&f);
   for(i=0;i<f;i++)
   {
      cntr[i]=0;
      a[i]=-1;
   }
   printf(``\nThe Page Replacement Process is -- \n'');
   for(i=0;i<m;i++)
   {
      for(j=0;j<f;j++)
      if(rs[i]==a[j])
      {
         cntr[j]++;
         break;
```

```
      }
      if(j==f)
      {
         min = 0;
         for(k=1;k<f;k++)
            if(cntr[k]<cntr[min])
               min=k;
         a[min]=rs[i];
         cntr[min]=1;
         pf++;
      }
      printf(``\n'');
      for(j=0;j<f;j++)
      printf(``\t%d'',a[j]);
      if(j==f)
      printf(``\tPF No. %d'',pf);
   }
   printf("\n\n Total number of page faults -- %d",pf);
   getchar();
}
```

2. Contoh Input

```
Enter number of page references -- 10
Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3
Enter the available no. of frames -- 3
```

3. Contoh Output

```
The Page Replacement Process is --

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

Total number of page faults -- 8
```

8.3.4 Optimal

1. Program

```
#include<stdio.h>
int n;
main()
{
```

```
int seq[30],fr[5],pos[5],find,flag,max,i,j,m,k,t,s;
int count=1,pf=0,p=0;
float pfr;
fflush(stdin);
printf("Enter maximum limit of the sequence: ");
scanf("%d",&max);
printf("\nEnter the sequence: ");
for(i=0;i<max;i++)
   scanf("%d",&seq[i]);
   printf("\nEnter no. of frames: ");
   scanf("%d",&n);
   fr[0]=seq[0];
   pf++;
   printf("%d\t",fr[0]);
   i=1;
   while(count<n)
   {
      flag=1;
      p++;
      for(j=0;j<i;j++)
      {
         if(seq[i]==seq[j])
         flag=0;
      }
      if(flag!=0)
      {
         fr[count]=seq[i];
         printf("%d\t",fr[count]);
         count++;
         pf++;
      }
      i++;
   }
   printf("\n");
   for(i=p;i<max;i++)
   {
      flag=1;
      for(j=0;j<n;j++)
      {
         if(seq[i]==fr[j])
            flag=0;
      }
      if(flag!=0)
      {
         for(j=0;j<n;j++)
         {
            m=fr[j];
            for(k=i;k<max;k++)
            {
               if(seq[k]==m)
               {
                  pos[j]=k;
                  break;
```

```
                }
                else
                   pos[j]=1;
             }
          }
          for(k=0;k<n;k++)
          {
             if(pos[k]==1)
             flag=0;
          }
          if(flag!=0)
          s=findmax(pos);
          if(flag==0)
          {
             for(k=0;k<n;k++)
             {
                if(pos[k]==1)
                {
                   s=k;
                   break;
                }
             }
          }
          fr[s]=seq[i];
          for(k=0;k<n;k++)
          printf("%d\t",fr[k]);
          pf++;
          printf("\n");
       }
    }
    pfr=(float)pf/(float)max;
    printf("\nThe no. of page faults are %d",pf);
    printf("\nPage fault rate %f",pfr);
    getchar();
}
int findmax(int a[])
{
   int max,i,k=0;
   max=a[0];
   for(i=0;i<n;i++)
   {
      if(max<a[i])
      {
         max=a[i];
         k=i;
      }
   }
   return k;
}
```

2. Contoh Input

```
Enter number of page references -- 10
```

```
Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3
Enter the available no. of frames -- 3
```

3. Contoh Output

```
The Page Replacement Process is --

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

Total number of page faults -- 8
```

8.4 Tugas Pendahuluan

1. Definisikan konsep dari virtual memory?
2. Jelaskan semua algoritma page replacement yang Anda ketahui? Berikan contoh untuk masing-masing algoritma dengan menggunakan suatu reference string menggunakan NIM Anda!
3. Apakah yang dimaksud dengan demand paging?
4. Apa yang dimaksud dengan page fault? Mengapa bisa terjadi page fault?

Modul 9

Penjadwalan Disk

9.1 Tujuan

1. Membuat program C yang mensimulasikan algoritma penjadwalan disk.

9.2 Teori

Salah satu peran sistem operasi adalah mengatur penggunaan perangkat keras secara efisien. Untuk disk drive, harus memberikan waktu akses (access time) cepat dan disk bandwidth besar. Access time dan bandwidth dapat ditingkatkan dengan mengatur urutan disk I/O request dengan menggunakan penjadwalan. Ada beberapa algoritma penjadawalan untuk disk, seperti FCFC, SCAN, dan C-SCAN.

1. FCFS

   Bentuk algoritma penjadwalan disk yang paling sederhana adalah First Come First Served (FCFS). Sistem kerja dari algoritma ini melayani permintaan yang lebih dulu datang di queue. Algoritma ini pada hakekatnya adil bagi permintaan I/O yang mengantri di queue karena penjadwalan ini melayani permintaan sesuai waktu tunggunya di queue. Tetapi yang menjadi kelemahan algoritma ini adalah bukan merupakan algoritma dengan layanan yang tercepat. Sebagai contoh, misalnya di queue disk terdapat antrian permintaan blok I/O di silinder

2. SCAN

   Pada algoritma SCAN, head bergerak ke silinder paling ujung dari disk. Setelah sampai disana maka head akan berbalik arah menuju silinder di ujung yang lainnya. Head akan melayani permintaan yang dilaluinya selama pergerakannya ini. Algoritma ini disebut juga sebagai Elevator Algorithm karena sistem kerjanya yang sama seperti yang digunakan elevator di sebuah gedung tinggi dalam melayani penggunanya. Elevator akan melayani pengguna yang akan menuju ke atas dahulu sampai lantai tertinggi, baru setelah itu dia berbalik arah menuju lantai terbawah sambil melayani penggunanya yang akan turun atau sebaliknya. Jika melihat analogi yang seperti itu maka dapat dikatakan head hanya melayani permintaan yang berada di depan arah pergerakannya. Jika ada permintaan yang berada di belakang

arah geraknya, maka permintaan tersebut harus menunggu sampai head menuju silinder di salah satu disk, lalu berbalik arah untuk melayani permintaan tersebut.

3. C-SCAN

   Algoritma C-SCAN atau Circular SCAN merupakan hasil modifikasi dari SCAN untuk mengurangi kemungkinan banyak permintaan yang menunggu untuk dilayani. Perbedaan yang paling mendasar dari kedua algoritma ini adalah pada behavior saat pergerakan head yang berbalik arah setelah sampai di ujung disk. Pada C-SCAN, saat head sudah berada di silinder terujung disk, head akan berbalik arah dan bergerak secepatnya menuju silinder di ujung disk yang satu lagi, tanpa melayani permintaan yang dilalui dalam pergerakannya. Sedangkan pada SCAN akan tetap melayani permintaan saat bergerak berbalik arah menuju ujung yang lain.

9.3 Cara Kerja

9.3.1 FCFS

1. Program

```
#include<stdio.h>
int main()
{
   int t[20], n, i, j, tohm[20], tot=0;
   float avhm;
   fflush(stdin);
   printf(``enter the no.of tracks'');
   scanf(``%d'',&n);
   printf(``enter the tracks to be traversed'');
   for(i=2;i<n+2;i++)
      scanf(``%d'',&t[i]);
   for(i=1;i<n+1;i++)
   {
      tohm[i]=t[i+1]-t[i];
      if(tohm[i]<0)
      tohm[i]=tohm[i]*(-1);
   }
   for(i=1;i<n+1;i++)
      tot+=tohm[i];
   avhm=(float)tot/n;
   printf(``Tracks traversed\tDifference between tracks\n'');
   for(i=1;i<n+1;i++)
      printf(``%d\t\t\t%d\n'',t[i],tohm[i]);
   printf(``\nAverage header movements:%f'',avhm);
   getchar();
}
```

2. Contoh Input

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

3. Contoh Output

```
Tracks traversed  Difference between tracks
     55                  45
     58                  3
     60                  2
     70                  10
     18                  52
     90                  72
     150                 60
     160                 10
     184                 24
Average header movements:30.888889
```

9.3.2 SCAN

1. Program

```
#include<stdio.h>
int main()
{
   int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
   fflush(stdin);
   printf("enter the no of tracks to be traveresed");
   scanf("%d'",&n);
   printf("enter the position of head");
   scanf("%d",&h);
   t[0]=0;t[1]=h;
   printf("enter the tracks");
   for(i=2;i<n+2;i++)
      scanf("%d",&t[i]);
   for(i=0;i<n+2;i++)
   {
      for(j=0;j<(n+2)-i-1;j++)
      {  if(t[j]>t[j+1])
         {
            temp=t[j];
            t[j]=t[j+1];
            t[j+1]=temp;
   } } }
   for(i=0;i<n+2;i++)
      if(t[i]==h)
         j=i;k=i;
   p=0;
   while(t[j]!=0)
   {
      atr[p]=t[j];
      j--;
      p++;
   }
   atr[p]=t[j];
   for(p=k+1;p<n+2;p++,k++)
      atr[p]=t[k+1];
```

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

2. Contoh Input

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

3. Contoh Output

```
Tracks traversed  Difference between tracks
      150                   50
      160                   10
      184                   24
      90                    94
      70                    20
      60                    10
      58                    2
      55                    3
      18                    37
Average header movements: 27.77
```

9.3.3 C-SCAN

1. Program

```
#include<stdio.h>
int main()
{
   int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
   fflush(stdin);
   printf("enter the no of tracks to be traveresed");
   scanf("%d'",&n);
   printf("enter the position of head");
   scanf("%d",&h);
   t[0]=0;t[1]=h;
   printf("enter total tracks");
   scanf("%d",&tot);
   t[2]=tot-1;
   printf("enter the tracks");
   for(i=3;i<=n+2;i++)
      scanf("%d",&t[i]);
   for(i=0;i<=n+2;i++)
```

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

2. Contoh Input

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

3. Contoh Output

```
Tracks traversed  Difference between tracks
      150                 50
      160                 10
      184                 24
      18                  240
      55                  37
      58                  3
```

```
      60                    2
      70                    10
      90                    20
Average seek time : 35.7777779
```

9.4 Tugas Pendahuluan

1. Mengapa harus ada penjadwalan disk?
2. Sebutkan dan jelaskan semua algoritma penjadwalan disk yang Anda ketahui!
3. Definisikan istilah berikut: (1) disk seek time, (2) disk access time, dan (3) rotational latency!
4. Jelaskan perbedaan antara HDD dan NVM!

Modul 10

Metode Alokasi File

10.1 Tujuan

1. Membuat program C yang mensimulasikan metode pengalokasian file.

10.2 Teori

File adalah sekumpulan data, biasanya beruruta pada disk. Sebagai entitas logical, file memungkin untuk membagi data ke beberapa kelompok. Sebagai entitas fisik, file harus diatur di-organisasikan. Istilah file diorganisasikan merujuk bagaimana data menyimpan file, dan bagaimana metode untuk mengaksesnya.

1. Sequential File Allocation

   Organisasi file ini mencatat penyimpanan file satu persatu pada fisik dan logic. Proses pencatatan dilakukan dengan urut, penempatan blok ke-15 berada setelah blok ke-14. Sequential file hanya bisa diakses dengan membaca catatan sebelumnya.

2. Linked File Allocation

   Linked allocation memungkinkan setiap file terhubung (link) pada blok disk. Blok disk bisa tersebar dimanapun pada disk. Direktori yang memuat pointer blok pertama dan terakhir dari suatu file. Setiap blok memiliki pointer yang menunjukkan lokasi blok selanjutnya.

3. Indexed File Allocation

   Indexed allocation mengimpulkan semua pointer pada satu lokasi: sebuah indeks blok. Setiap file memiliki indeks blok masing-masing, yang berisi arrat dari disk-block addresses. Indeks ke-i menunjukkan blok ke-i dari suatu file. Direktori memuat alamat dari indeks blok. Untuk menemukan dan membaca blok ke-i, digunakan pointer pada indeks ke-i.

10.3 Cara Kerja

10.3.1 Sequential File Allocation

1. Program

```
#include<stdio.h>
#include<scstring>
{
        char name[20];
        int sb, nob;
}ft[30];
int main()
{
        int i, j, n;
        char s[20];
        fflush(stdin);
        printf("Enter no of files :");
        scanf("%d",&n);

        for(i=0;i<n;i++)
        {
                printf("\nEnter file name %d :",i+1);
                scanf("%s",ft[i].name);
                printf("Enter starting block of file %d :",i+1);
                scanf("%d",&ft[i].sb);
                printf("Enter no of blocks in file %d :",i+1);
                scanf("%d",&ft[i].nob);
        }
        printf("\nEnter the file name to be searched-- ");
        scanf("%s",s);
        for(i=0;i<n;i++)
                if(strcmp(s, ft[i].name)==0)
                        break;
        if(i==n)
        printf("\nFile Not Found");
        else
        {
                printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED\n");
        printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);
        for(j=0;j<ft[i].nob;j++)
                printf("%d, ",ft[i].sb+j);
        printf("\n");
        }
        getchar();
}
```

2. Contoh Input

```
Enter no of files :3

Enter file name 1 :A
Enter starting block of file 1 :85
Enter no of blocks in file 1 :6

Enter file name 2 :B
Enter starting block of file 2 :102
Enter no of blocks in file 2 :4
```

```
Enter file name 3 :C
Enter starting block of file 3 :60
Enter no of blocks in file 3 :4
Enter the file name to be searched -- B
```

3. Contoh Output

```
FILE NAME     START BLOCK     NO OF BLOCKS BLOCKS        OCCUPIED
     B          102                  4               102, 103, 104, 105
```

10.3.2 Linked File Allocation

1. Program

```
#include<stdio.h>
#include<scstring>
#include<cstdlib>
{
char name[20];
int nob;
struct block *sb;
}ft[30];
struct block
{
        int bno;
        struct block *next;
};

void main()
{
        int i, j, n;
        char s[20];
        struct block *temp;
        fflush(stdin);
        printf("Enter no of files :");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter file name %d :",i+1);
                scanf("%s",ft[i].name);
                printf("Enter no of blocks in file %d :",i+1);
                scanf("%d",&ft[i].nob);
                ft[i].sb=(struct block*)malloc(sizeof(struct block));
                temp = ft[i].sb;
                printf("Enter the blocks of the file :");
                scanf("%d",&temp->bno);
                temp->next=NULL;

                for(j=1;j<ft[i].nob;j++)
                {
                        temp->next = (struct block*)malloc(sizeof(struct block));
                        temp = temp->next;
```

```
                scanf("%d",&temp->bno);
            }
            temp->next = NULL;
        }
        printf("\nEnter the file name to be searched -- ");
        scanf("%s",s);
        for(i=0;i<n;i++)
            if(strcmp(s, ft[i].name)==0)
                break;
        if(i==n)
            printf("\nFile Not Found");
        else
        {
            printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
            printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);
            temp=ft[i].sb;
            for(j=0;j<ft[i].nob;j++)
            {
                printf("%d -> ",temp->bno);
                temp = temp->next;
            }
        printf("\n");
        }
        getchar();
}
```

2. Contoh Input

```
Enter no of files : 2

Enter file 1 : A
Enter no of blocks in file 1 : 4
Enter the blocks of the file 1 : 12 23 9 4

Enter file 2 : G
Enter no of blocks in file 2 : 5
Enter the blocks of the file 2 : 88 77 66 55 44
Enter the file to be searched : G
```

3. Contoh Output

```
FILE NAME       NO OF BLOCKS BLOCKS                OCCUPIED
   G                 5                        88 -> 77 -> 66 --> 55 --> 44
```

10.3.3 Indexed Allocation

1. Program

```
#include<stdio.h>
#include<string.h>
struct fileTable
{
```

```
char name[20];
int nob, blocks[30];
}ft[30];
void main()
{
        int i, j, n;
        char s[20];
        fflush(stdin);
        printf("Enter no of files :");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter file name %d :",i+1);
                scanf("%s",ft[i].name);
                printf("Enter no of blocks in file %d :",i+1);
                scanf("%d",&ft[i].nob);
                printf("Enter the blocks of the file :");
                for(j=0;j<ft[i].nob;j++)
                        scanf("%d",&ft[i].blocks[j]);
        }
        printf("\nEnter the file name to be searched-- ");
        scanf("%s",s);
        for(i=0;i<n;i++)
                if(strcmp(s, ft[i].name)==0)
        break;
        if(i==n)
                printf("\nFile Not Found");
        else
        {
                printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
                printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);
                for(j=0;j<ft[i].nob;j++)
                        printf("%d, ",ft[i].blocks[j]);
        }
        printf("\n");
        getchar();
}
```

2. Contoh Input

```
Enter no of files : 2

Enter file 1 : A
Enter no of blocks in file 1 : 4
Enter the blocks of the file 1 : 12 23 9 4

Enter file 2 : G
Enter no of blocks in file 2 : 5
Enter the blocks of the file 2 : 88 77 66 55 44
Enter the file to be searched : G
```

3. Contoh Output

```
FILE NAME       NO OF BLOCKS BLOCKS              OCCUPIED
   G                 5                       88, 77, 66, 55, 44
```

10.4 Tugas Pendahuluan

1. Apakah yang dimaksud file dan directory?
2. Tuliskan jenis-jenis file yang Anda ketahui!
3. Apa tujuan dari strategi pengalokasian file?
4. Jelaskan mengenai FAT dan NTFS? Manakah yang lebih baik menurut Anda, berikan alasannya!