BUKU PETUNJUK (MODUL) PRAKTIKUM

SISTEM PENGOPERASIAN
(MII 2612)

TIM PENYUSUN:

1. Arif Nurwidyantoro, S.Kom., M.Cs.
2. Roghib Muhammad Hujja, S.Si., M.Cs.
3. Lukman Awaludin, S.Si., M.Cs.
4. Guntur Budi Herwanto, S.Kom., M.Cs.

LABORATORIUM KOMPUTER DASAR
DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
YOGYAKARTA
2018

TATA TERTIB PRAKTIKAN

1. Masuk sesuai jadwal/waktu yang telah ditentukan/disepakati bersama, terlambat lebih dari 15 menit dipersilahkan tidak memasuki ruang LABORATORIUM KOMPUTER DASAR.
2. Praktikan yang tidak mengikuti praktikum lebih dari 3 kali tanpa keterangan resmi, tidak berhak mengikuti responsi/ujian.
3. Tidak diadakan ujian susulan baik ujian tengah semester maupun akhir semester, kecuali atas persetujuan dosen/pengampu mata kuliah bersangkutan.
4. Memakai Pakaian rapi dan sopan:
   Pria :
   - Kemeja Lengan panjang atau pendek
   - Celana panjang Rapi
   - Bersepatu
   - Tidak boleh memakai T-shirt tanpa krah atau tanpa lengan

   Wanita :
   - Kemeja Lengan panjang/pendek (tidak ketat dan atau transparan)
   - Rok atau Celana panjang (tidak ketat dan atau transparan)
   - Bersepatu
   - Tidak boleh memakai T-shirt tanpa krah atau tanpa lengan
5. Mahasiswa tidak diperbolehkan merokok, makan dan minum pada saat kuliah praktikum.
6. Tas dan perlengkapan lain harus diletakkan pada tempat yang telah disediakan (hanya diperbolehkan membawa barang berharga, Ex: Handphone, dompet dan alat yang diperlukan untuk praktikum)
7. Barang berharga milik peserta kuliah praktikum menjadi tanggung jawab sendiri (laboran tidak bertanggungjawab atas kehilangan barang tersebut).
8. Dering HP harus dimatikan (silent) pada saat kuliah praktikum.
9. Selesai Praktikum, Komputer dimatikan dan kursi dirapikan kembali.
10. Mahasiswa diwajibkan menjaga kebersihan dan ketertiban serta ketenangan belajar.
11. Mahasiswa tidak diperbolehkan menggunakan komputer untuk bermain games.
12. Mahasiswa tidak diperkenankan men-install program/software tanpa petugas lab.
13. Mahasiswa tidak diperkenankan memindah posisi hardware (mouse, keybord, monitor, CPU)
14. Mahasiswa tidak diperbolehkan membawa atau mengambil (secara sengaja atau tidak sengaja) perlengkapan praktikum yang ada di Laboratorium komputer).
15. Mahasiswa wajib menjaga keutuhan semua peralatan yang ada di Laboratorium Komputer serta tidak diperbolehkan memakai komputer Pengajar/Instruktur.
16. Mengisi daftar hadir dan mencatat nomor kursi/komputer yang digunakan selama praktikum berlangsung ke dalam form daftar hadir yang telah disediakan.
17. Melaporkan keadaan komputer dan atau peralatan yang digunakan (yang rusak/tidak berfungsi) sebelum, sesaat dan atau sesudah penggunaan ke Pengajar/Instruktur atau kepada laboran.
18. Praktikan wajib mematikan Komputer yang telah selesai digunakan dan merapikan kembali kursi, meja dan perlengkapan pendukung praktikum lainnya setelah praktikum selesai dilaksanakan/berakhir.
19. Laboran berhak mencatat, memberikan sanksi atau melakukan tindakan seperlunya terhadap praktikan yang melanggar tata tertib.

LABORATORIUM KOMPUTER DASAR
DIKE F.MIPA UGM

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Alhamdullilah, atas berkat rahmat Allah Yang Maha Kuasa dengan didorongkan oleh keinginan luhur memperluas wawasan dalam pengembangan pengetahuan tentang Sistem Pengoperasian, maka buku petunjuk (modul) praktikum ini disusun/dibuat.

Buku petunjuk (modul) Praktikum Sistem Pengoperasian ini dibuat untuk membantu jalannya Praktikum Sistem Pengoperasian prodi S1 Ilmu Komputer. Buku petunjuk (modul) ini dibuat sedemikian rupa sehingga dapat dengan mudah dipahami dan dipelajari oleh mereka yang belum pernah mengenal Sistem Pengoperasian sekalipun, dan bagi mereka yang pernah mengenal membaca buku ini akan menyegarkan ingatan.

Terima kasih kami ucapkan kepada semua pihak yang telah membantu dan mendukung pembuatan buku ini.

Wassalamu'alaikum Wr. Wb.

LABORATORIUM KOMPUTER DASAR
DIKE F.MIPA UGM

DAFTAR ISI

PERTEMUAN 1
VMWARE DAN INSTALASI SISTEM PENGOPERASIAN

A. Tujuan Praktikum

- Praktikan dapat mengetahui cara menginstallasi VMWare dan system operasi
- Praktikan dapat memahami fungsi system operasi Windows 7
- Praktikan dapat menguasai dan mempraktikkan cara menginstall Windows 7
- Praktikan dapat memahami fungsi dynamic link libraries (.dll)

B. Materi

Virtual Machine

Virtual Machine (Indonesia : mesin virtual) pada mulanya didefinisikan oleh Gerard J. Popek dan Robert P. Goldberg pada tahun 1974 sebagai sebuah duplikat yang efisien dan terisolasi dari suatu mesin asli. Pada masa sekarang ini, virtual machine dapat mensimulasikan perangkat keras walaupun tidak ada perangkat keras aslinya sama sekali. Virtual machine terdiri dari dua kategori besar, dipisahkan menurut cara penggunaan dan tingkat keterhubungannya dengan mesin-mesin aslinya. Sebuah System Virtual Machine adalah perangkat yang berupa platform sistem yang lengkap dan dapat menjalankan sebuah sistem operasi yang lengkap. Sebaliknya, Process Virtual Machine didesain untuk menjalankan sebuah program komputer tertentu (tunggal), yang berarti mesin virtual ini mendukung proses tertentu juga.

Disini akan dikenalkan bagaimana cara instalasi dan menggunakan suatu System Virtual Machine. Yaitu VMware Workstations 12 Player (VMware). VMware adalah salah satu System Virtual Machine yang dapat menjalankan suatu sistem operasi secara lengkap.

Windows 7

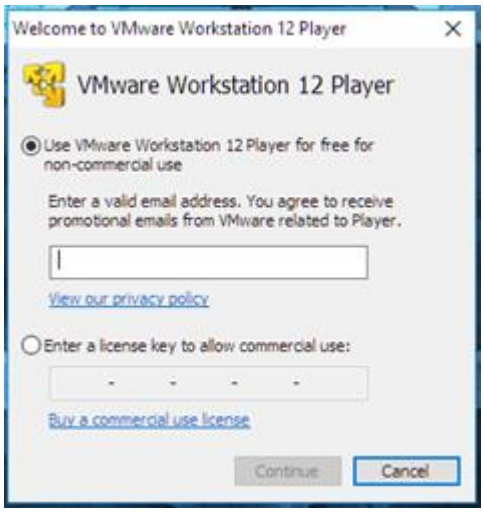
Windows 7 adalah suatu sistem opeasi yang banyak dipakai sampai saat ini karena kemudahan dalam pemakaiannya. Untuk melakukan penginstalan Windows 7 diperlukan ketelitian dan kesabaran dalam prosesnya karena memerlukan waktu yang lumayan lama. Ada beberapa jenis Windows 7 diantaranya Windows 7 Starter, Home Basic, Home Premium, Professional, Ultimate, dan Enterprise.

Penggunaan dan fitur- fitur bawaan dari Windows 7 cukup lengkap dan mudah dioperasikan, hal ini membuat Windows 7 masih tetap digemari dan masih layak konsumsi bagi masyarakat banyak. Walaupun sudah terbit versi – versi Windows terbaru seperti Windows 8 ataupun Windows 10.

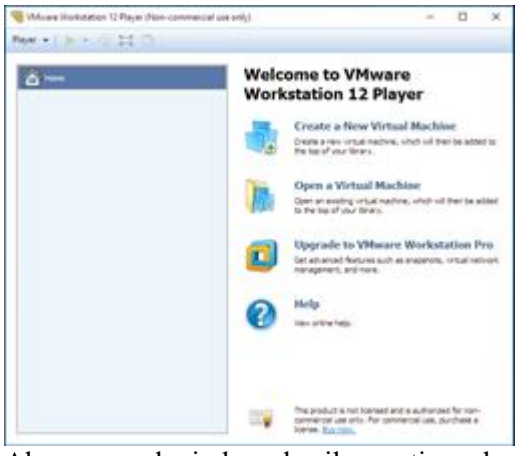
C. Aktivitas Praktikum

Instalasi Sistem Operasi pada VMWARE

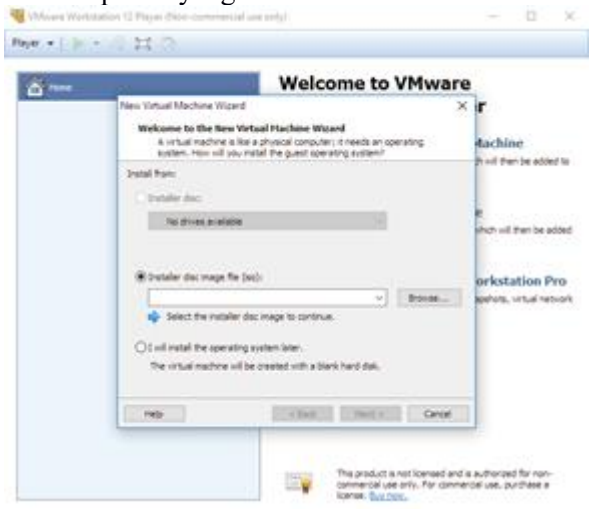
1. Install VMware Workstation 12 Player.
2. Pilih “Use VMware Workstation 12 Plater for free for non – commercial use” dan input alamat email.



3. Pilih “Create a New Virtual Machine”.

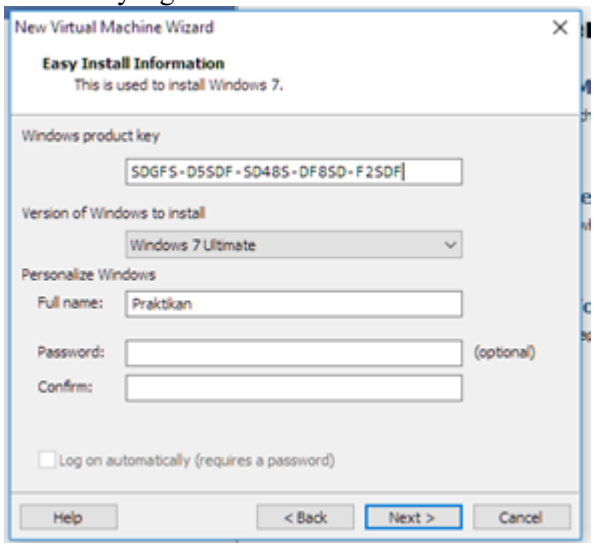


4. Akan muncul windows kecil seperti gambar di bawah. Klik Browse… dan pilih file .iso sistem operasi yang akan diinstal.

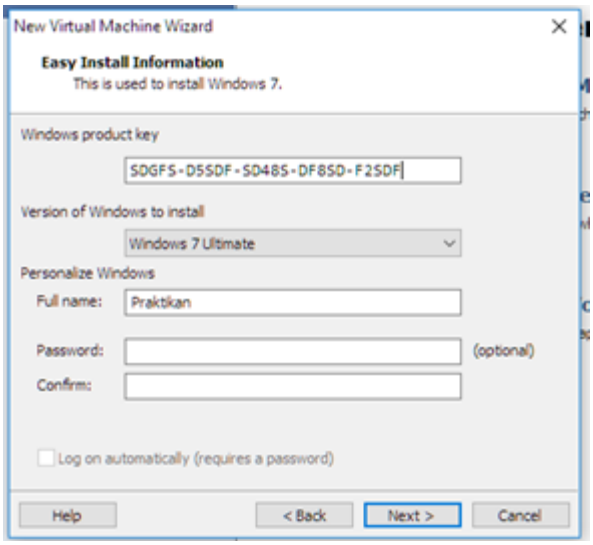


5. Jika berhasil, Sistem Operasi yang akan diinstal akan langsung terdeteksi.

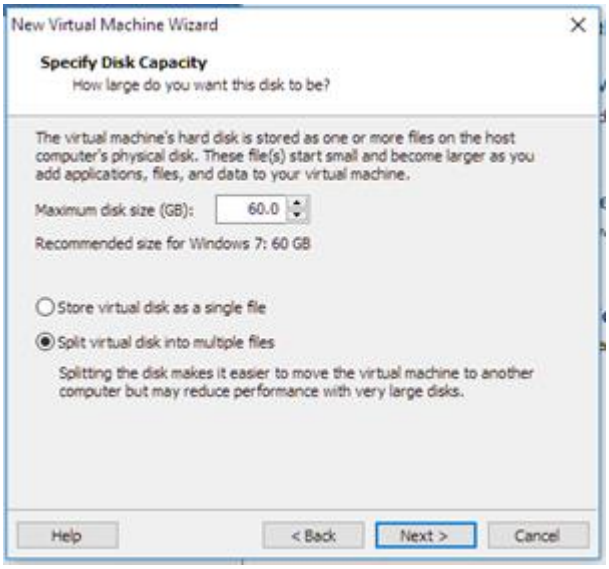
6. Tuliskan serial number Windows dan username yang diinginkan. Dan pilih juga versi windows yang sesuai.



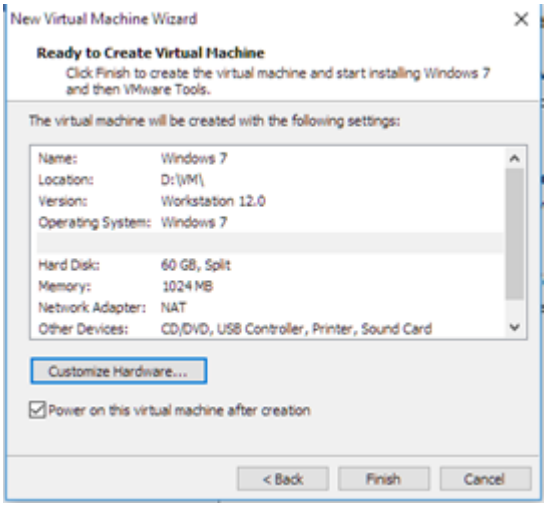
7. Tuliskan Nama Virtual Machine dan lokasi dimana sistem operasi akan di-install.



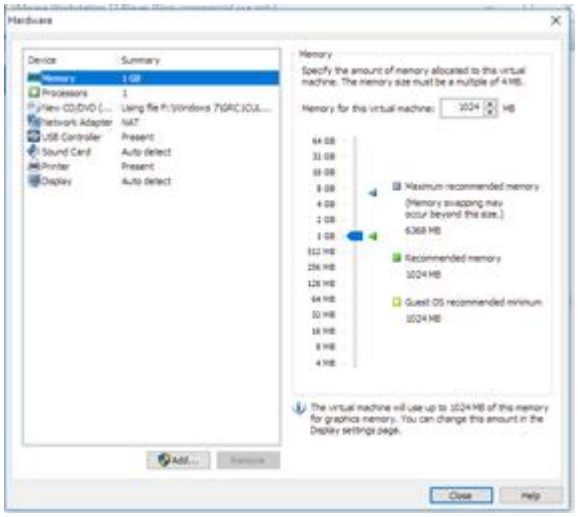
8. Sesuaikan ukuran Hard Disk untuk sistem operasi.



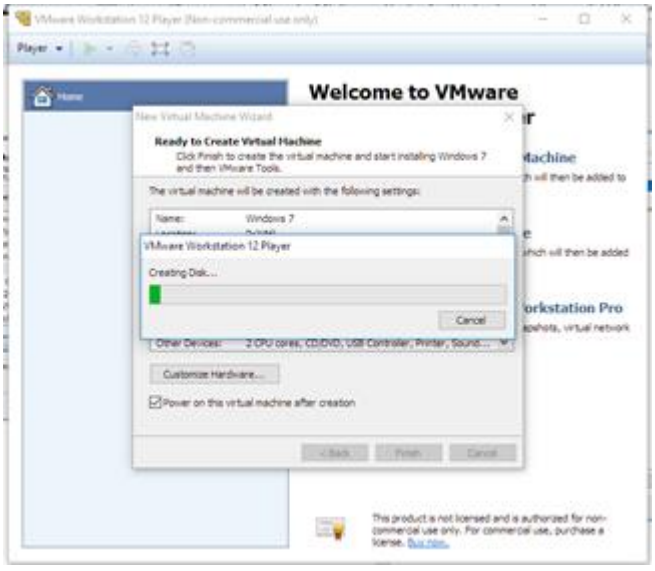
9. Dapat dilakukan pengubahan hardware untuk virtual machine dengan men-klik “Customize Hardware…” atau jika hardware dirasa sudah sesuai.



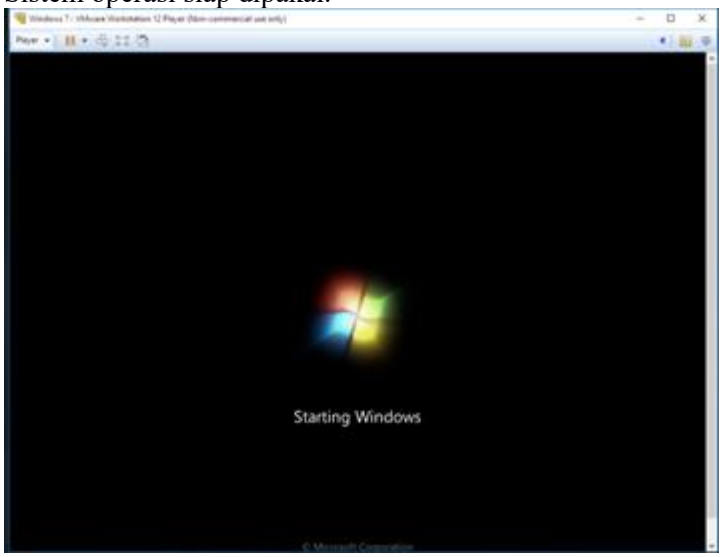
10. Sesuaikan hardware dengan keinginan.



11. Tunggu proses instalasi selesai.



12. Sistem operasi siap dipakai.



D. Tugas

Buatlah laporan Instalasi Sistem Operasi Linux dengan Distro yang anda tentukan sendiri, pada VMWare ataupun Virtual Machine Lainnya

PERTEMUAN 2
COMMAND PROMPT

A. Tujuan Praktikum

- Praktikan dapat mengetahui cara menggunakan windows command prompt
- Praktikan dapat mengetahui command dasar dan menengah dari windows command prompt
- Praktikan dapat mengetahui cara menggunakan linux terminal
- Praktikan dapat mengetahuin command dasar dan menengah dari linux terminal

B. Materi dan Aktivitas Praktikum

Windows Command Prompt

Command Prompt memungkinkan kita bekerja di lingkungan yang lebih mirip sistem operasi tradisional dibandingkan dengan lingkungan berbasis Graphical User Interface (GUI) Windows. Di Command Prompt, kita akan menggunakan keyboard untuk menjalankan berbagai perintah. Command Prompt bekerja pada low level dibandingkan GUI Windows. Ini berarti kita akan memiliki kontrol lebih besar terhadap komputer. Kerugiannya adalah bahwa command prompt kurang user-friendly.

Untuk menjalankan Command Prompt klik Start -> Run dan ketik cmd lalu enter.
Command Prompt tampil dalam terminal berwarna hitam sebagai default. Tampilan pertama command prompt pada umumnya akan seperti ini:
C:\>

Disinilah tempat kita mengetik instruksi command prompt. Instruksi dalam command prompt tidak case-sensitive, itu berarti kita mengetikkan cd sama dengan CD.

Bekerja dengan Files dan Direktori

Anda juga bisa menggunakan perintah Command Prompt untuk mengatur file ke dalam hirarki direktori. Perintah ini setara dengan perintah yang sesuai yang Anda akses melalui antarmuka titik-dan-klik Windows. Hal ini berguna untuk mengenal kedua antarmuka untuk mengelola file.

dir: Untuk melihat isi sebuah direktori, ketik dir. Perintah ini akan mencantumkan semua file dan direktori dalam direktori saat ini. Ini sama dengan mengklik folder Windows untuk melihat apa yang ada di dalamnya.

```
C:\> dir
        Volume in drive C has no label.
        Volume Serial Number is C8C7-BDCD
 Directory of C:\
10/26/2004  01:36 PM             0 AUTOEXEC.BAT
10/26/2004  01:36 PM             0 CONFIG.SYS
02/10/2005  01:36 PM       126 HelloWorld.java
12/09/2004  12:11 AM  DIR       Documents and Settings
02/10/2005  08:59 PM  DIR       introcs
11/02/2004  08:31 PM  DIR       j2sdk1.4.2_06
12/29/2004  07:15 PM    DIR     Program Files
01/13/2005  07:33 AM   DIR      WINDOWS
        3 File(s)      126 bytes
        5 Dir(s)  32,551,940,096 bytes free
```

Ada 7 item dalam direktori ini. Beberapa di antaranya adalah file, seperti HelloWorld.java. Lainnya adalah direktori, seperti introcs.

cd: Hal ini sering berguna untuk mengetahui di direktori mana Anda saat ini bekerja. Untuk mengetahui, ketik cd pada command prompt.

```
C:\> cd
C:\
```

Untuk mengubah directories, gunakan cd command dengan nama directory.

```
C:\> cd introcs
```

Command prompt akan terlihat seperti berikut:

```
C:\introcs>
```

Untuk melihat isi directory::

```
C:\introcs> dir
 Volume in drive C has no label.
 Volume Serial Number is C8C7-BDCD
 Directory of C:\introcs
02/10/2005  08:59 PM    DIR              .
02/10/2005  08:59 PM    DIR              ..
02/03/2005  11:53 PM               126 HelloWorld.java
01/17/2005  01:16 AM               256 readme.txt
               2 File(s)            382 bytes
               2 Dir(s)
```

Untuk kembali ke direktori sebelumnya, gunakan perintah cd, tapi kali ini diikuti oleh spasi dan dua titik.

```
C:\introcs> cd ..
C:\>
```

mkdir: Untuk membuat direktori baru, gunakan perintah mkdir. Perintah berikut membuat sebuah direktori bernama halo.

```
C:\introcs> mkdir hello
```

To see that it actually worked, use the dir command.

```
C:\introcs> dir
 Volume in drive C has no label.
 Volume Serial Number is C8C7-BDCD
 Directory of C:\introcs
02/10/2005  08:59 PM    DIR              .
02/10/2005  08:59 PM    DIR              ..
02/11/2005  02:53 PM    DIR              hello
02/03/2005  11:53 PM               126 HelloWorld.java
```

```
01/17/2005  01:16 AM               256 readme.txt
               2 File(s)          382 bytes
               3 Dir(s)
```

move: Sekarang, pindahkan dua file HelloWorld.java dan readme.txt ke dalam direktori hello menggunakan perintah move.

```
C:\introcs> move HelloWorld.java hello
C:\introcs> move readme.txt hello
C:\introcs> dir
 Volume in drive C has no label.
 Volume Serial Number is C8C7-BDCD

 Directory of C:\introcs

02/10/2005  08:59 PM   DIR              .
02/10/2005  08:59 PM   DIR              ..
02/11/2005  02:53 PM   DIR              hello
               0 File(s)          0 bytes
               3 Dir(s)
```

Kedua file tersebut tidak lagi terlihat dari direktori saat ini.

Untuk mengakses dua file, ganti direktori dengan perintah cd. Kemudian gunakan perintah dir untuk melihat apa yang ada di direktori baru ini.

```
C:\introcs> cd hello
C:\introcs\hello> dir
 Volume in drive C has no label.
 Volume Serial Number is C8C7-BDCD

 Directory of C:\introcs\hello

02/10/2005  08:59 PM   DIR              .
02/10/2005  08:59 PM   DIR              ..
02/03/2005  11:53 PM               126 HelloWorld.java
01/17/2005  01:16 AM               256 readme.txt
               2 File(s)          382 bytes
               2 Dir(s)
```

Anda juga dapat menggunakan pindah untuk mengubah nama file. Cukup tentukan nama file baru, bukan nama direktori. Misalkan Anda secara tidak sengaja mengacaukan kasus atas dan bawah dan telah menyelamatkan HelloWorld.java sebagai helloworld.java. Gunakan dua perintah bergerak untuk memperbaikinya.

```
C:\introcs\hello> dir
 Volume in drive C has no label.
 Volume Serial Number is C8C7-BDCD

 Directory of C:\introcs\hello

02/10/2005  08:59 PM   DIR              .
```

```
02/10/2005  08:59 PM   DIR              ..
02/03/2005  11:53 PM              126 helloworld.java
01/17/2005  01:16 AM              256 readme.txt
          2 File(s)          382 bytes
          2 Dir(s)

C:\introcs\hello> move helloworld.java temp.java
C:\introcs\hello> move temp.java HelloWorld.java
C:\introcs\hello> dir
 Volume in drive C has no label.
 Volume Serial Number is C8C7-BDCD

 Directory of C:\introcs\hello

02/10/2005  08:59 PM   DIR              .
02/10/2005  08:59 PM   DIR              ..
02/03/2005  11:53 PM              126 HelloWorld.java
01/17/2005  01:16 AM              256 readme.txt
          2 File(s)          382 bytes
          2 Dir(s)
```

Dibutuhkan dua langkah karena Windows tidak akan membiarkan Anda pindah ke nama file yang sudah ada dan di Windows, helloworld.java sama dengan HelloWorld.java.

copy: Untuk membuat salinan file, gunakan perintah copy. Perintah berikut membuat salinan cadangan dari program HelloWorld.java kami. Ini sangat berguna saat Anda memodifikasi program kerja, namun mungkin ingin kembali ke versi aslinya jika modifikasi Anda tidak berhasil.

```
C:\introcs\hello> copy HelloWorld.java HelloWorld.bak
C:\introcs\hello> dir
 Volume in drive C has no label.
 Volume Serial Number is C8C7-BDCD

 Directory of C:\introcs\hello

02/10/2005  08:59 PM   DIR              .
02/10/2005  08:59 PM   DIR              ..
02/03/2005  11:53 PM              126 HelloWorld.java
01/17/2005  01:16 AM              256 readme.txt
          2 File(s)          382 bytes
          3 Dir(s)
```

del: Selanjutnya, Anda mungkin ingin membersihkan file yang tidak berguna. Perintah del menghapus sebuah file.

```
C:\introcs\hello> del HelloWorld.bak
C:\introcs\hello> dir
 Volume in drive C has no label.
 Volume Serial Number is C8C7-BDCD

 Directory of C:\introcs
```

```
02/10/2005  08:59 PM    DIR              .
02/10/2005  08:59 PM    DIR              ..
02/03/2005  11:53 PM               126 HelloWorld.java
01/17/2005  01:16 AM               256 readme.txt
               2 File(s)          382 bytes
               3 Dir(s)
```

wildcard: Anda juga dapat menerapkan perintah copy, del, dan move ke beberapa file (atau direktori) sekaligus. Untuk membuat direktori baru yang disebut loop, dan salin semua file di direktori hello C: \ introcs \ hello \ ke dalam tipe direktori yang baru dibuat ini:

```
C:\introcs> mkdir loops
C:\introcs> copy c:\introcs\hello\* loops
```

Disini * cocokkan semua file di direktori C: \ introcs \ hello. Ini salinan mereka ke direktori loops yang baru Anda buat.

Linux Terminal Command

Pada dasarnya, shell adalah program yang menerima perintah dari pengguna dan memberikannya ke OS untuk diproses, dan ini menunjukkan hasilnya. Shell Linux adalah bagian utamanya. Distro nya datang di GUI (graphical user interface), namun pada dasarnya, Linux memiliki CLI (command line interface). Dalam tutorial ini, kita akan membahas perintah dasar yang kita gunakan di shell Linux.

Untuk membuka terminal, tekan Ctrl + Alt + T di Ubuntu, atau tekan Alt + F2, ketik gnome-terminal, dan tekan enter. Di Raspberry Pi, ketik lxterminal. Ada juga cara GUI untuk mengambilnya, tapi ini lebih baik!

Basic Commands

1. **pwd** — Saat pertama kali membuka terminal, Anda berada di direktori home pengguna Anda. Untuk mengetahui direktori mana Anda berada, Anda dapat menggunakan perintah "pwd". Ini memberi kita jalan yang mutlak, yang berarti jalan yang dimulai dari akar. Akarnya adalah dasar dari sistem berkas Linux. Hal ini dilambangkan dengan garis miring (/). Direktori pengguna biasanya seperti "/ home / username".

```
nayso@Alok-Aspire:~$ pwd
/home/nayso
```

2. **ls** — Gunakan perintah "Is" untuk mengetahui file apa yang ada di direktori tempat Anda berada. Anda dapat melihat semua file tersembunyi dengan menggunakan perintah "ls -a".

```
nayso@Alok-Aspire:~$ ls
Desktop           itsuserguide.desktop  reset-settings     VCD_Copy
Documents         Music                 School_Resources   Videos
Downloads         Pictures              Students_Works_10
examples.desktop  Public                Templates
GplatesProject    Qgis Projects         TuxPaint-Pictures
```

3. **cd** — Gunakan perintah "cd" untuk masuk ke direktori. Misalnya, jika Anda berada di folder home, dan Anda ingin masuk ke folder download, maka Anda bisa mengetikkan "cd Downloads". Ingat, perintah ini peka huruf besar, dan Anda harus mengetikkan nama folder persis seperti apa adanya. Tapi ada masalah dengan perintah ini. Bayangkan Anda memiliki folder bernama "Raspberry Pi". Dalam kasus ini, ketika Anda mengetikkan "cd Raspberry Pi", shell akan mengambil argumen kedua dari perintah sebagai perintah yang berbeda, sehingga

Anda akan mendapatkan pesan kesalahan yang mengatakan bahwa direktori tersebut tidak ada. Di sini, Anda bisa menggunakan garis miring terbalik. Artinya, Anda bisa menggunakan "cd Raspberry \ Pi" dalam hal ini. Spasi dilambangkan seperti ini: Jika Anda hanya mengetik "cd" dan tekan enter, itu akan membawa Anda ke direktori home. Untuk kembali dari folder ke folder sebelumnya, Anda bisa mengetik "cd ..". Dua titik mewakili kembali.

```
nayso@Alok-Aspire:~$ cd Downloads
nayso@Alok-Aspire:~/Downloads$ cd
nayso@Alok-Aspire:~$ cd Raspberry\ Pi
nayso@Alok-Aspire:~/Raspberry Pi$ cd ..
nayso@Alok-Aspire:~$
```

4. **mkdir & rmdir** — Gunakan perintah mkdir saat Anda ingin membuat folder atau direktori. Misalnya, jika Anda ingin membuat direktori bernama "DIY", maka Anda bisa mengetik "mkdir DIY". Ingat, seperti yang diceritakan sebelumnya, jika Anda ingin membuat direktori bernama "DIY Hacking", maka Anda bisa mengetikkan "mkdir DIY \ Hacking". Gunakan rmdir untuk menghapus sebuah direktori. Tapi rmdir hanya bisa digunakan untuk menghapus sebuah direktori kosong. Untuk menghapus direktori yang berisi file, gunakan rm.

```
nayso@Alok-Aspire:~/Desktop$ ls
nayso@Alok-Aspire:~/Desktop$ mkdir DIY
nayso@Alok-Aspire:~/Desktop$ ls
DIY
nayso@Alok-Aspire:~/Desktop$ rmdir DIY
nayso@Alok-Aspire:~/Desktop$ ls
nayso@Alok-Aspire:~/Desktop$
```

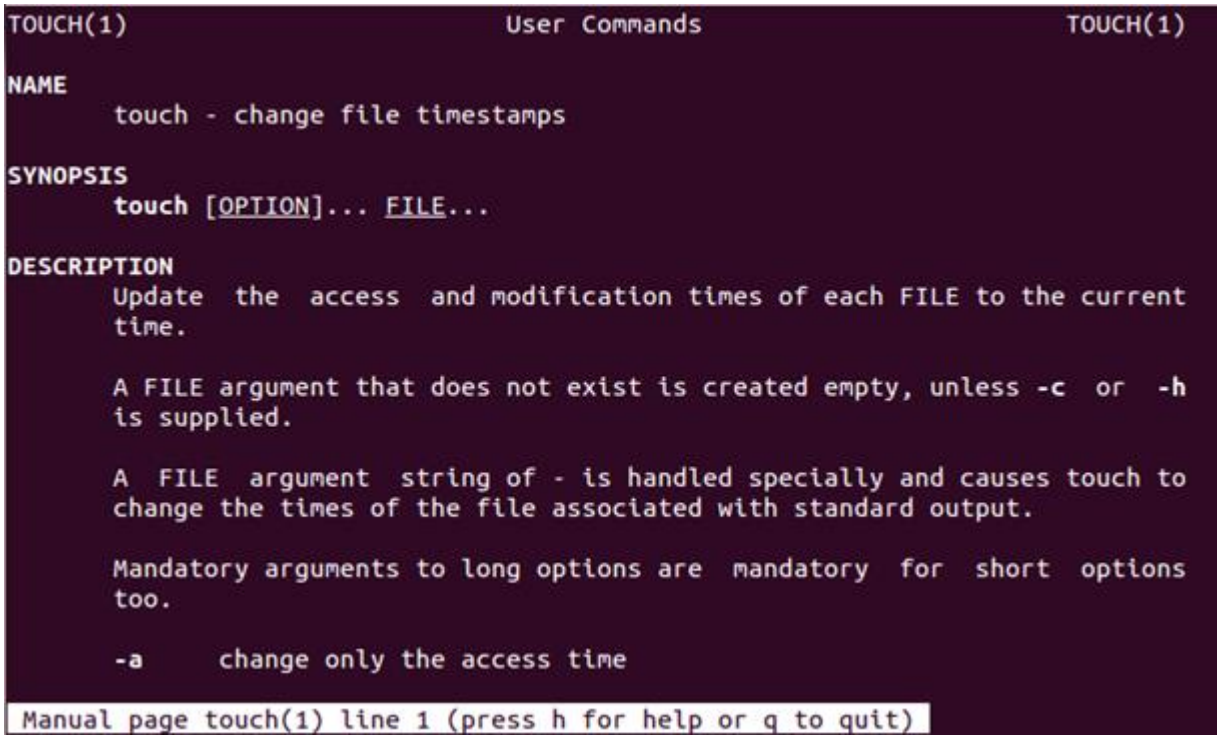
5. **rm** — Gunakan perintah rm untuk menghapus file dan direktori. Tapi rm tidak bisa begitu saja menghapus sebuah direktori. Gunakan "rm -r" untuk menghapus sebuah direktori. Dalam kasus ini, menghapus folder dan file di dalamnya.

```
nayso@Alok-Aspire:~/Desktop$ ls
newer.py  New Folder
nayso@Alok-Aspire:~/Desktop$ rm newer.py
nayso@Alok-Aspire:~/Desktop$ ls
New Folder
nayso@Alok-Aspire:~/Desktop$ rm -r New\ Folder
nayso@Alok-Aspire:~/Desktop$ ls
nayso@Alok-Aspire:~/Desktop$
```

6. **touch** — Perintah touch digunakan untuk membuat file. Bisa apa saja, dari file txt kosong ke file zip kosong. Misalnya, "touch new.txt".

```
nayso@Alok-Aspire:~/Desktop$ ls
nayso@Alok-Aspire:~/Desktop$ touch new.txt
nayso@Alok-Aspire:~/Desktop$ ls
new.txt
```

7. **man & --help** — Untuk mengetahui lebih banyak tentang sebuah perintah dan cara menggunakannya, gunakan perintah man. Ini menunjukkan halaman manual perintah. Misalnya, "man cd" menampilkan halaman manual perintah cd. Mengetik pada nama perintah dan argumen membantu menunjukkan cara perintah mana yang dapat digunakan (mis., cd -help).



8. **cp** — Gunakan perintah cp untuk menyalin file melalui command line. Dibutuhkan dua argumen: Yang pertama adalah lokasi file yang akan disalin, yang kedua adalah tempat untuk menyalin.

```
nayso@Alok-Aspire:~/Desktop$ ls /home/nayso/Music/
nayso@Alok-Aspire:~/Desktop$ cp new.txt /home/nayso/Music/
nayso@Alok-Aspire:~/Desktop$ ls /home/nayso/Music/
new.txt
```

9. **mv** — Gunakan perintah mv untuk memindahkan file melalui command line. Kita juga bisa menggunakan perintah mv untuk mengganti nama file. Misalnya, jika kita ingin mengganti nama file "text" menjadi "new", kita bisa menggunakan "mv text new". Dibutuhkan dua argumen, sama seperti perintah cp.

```
nayso@Alok-Aspire:~/Desktop$ ls
new.txt
nayso@Alok-Aspire:~/Desktop$ mv new.txt newer.txt
nayso@Alok-Aspire:~/Desktop$ ls
newer.txt
```

10. **locate** — Perintah locate digunakan untuk mencari file dalam sistem Linux, seperti perintah pencarian di Windows. Perintah ini berguna bila Anda tidak tahu di mana file disimpan atau nama file sebenarnya. Menggunakan argumen -i dengan perintah membantu mengabaikan kasus ini (tidak masalah apakah huruf besar atau huruf kecil). Jadi, jika Anda menginginkan file yang memiliki kata "halo", ini memberi daftar semua file di sistem Linux Anda yang berisi kata "halo" saat Anda mengetikkan "locate -i hello". Jika Anda mengingat dua kata, Anda dapat memisahkannya menggunakan tanda bintang (*). Misalnya, untuk mencari file yang berisi kata "halo" dan "ini", Anda bisa menggunakan perintah "locate -i * hello * this".

```
nayso@Alok-Aspire:~$ locate newer.txt
/home/nayso/Desktop/newer.txt
nayso@Alok-Aspire:~$ locate *DIY*Hacking*
/home/nayso/DIY Hacking
```

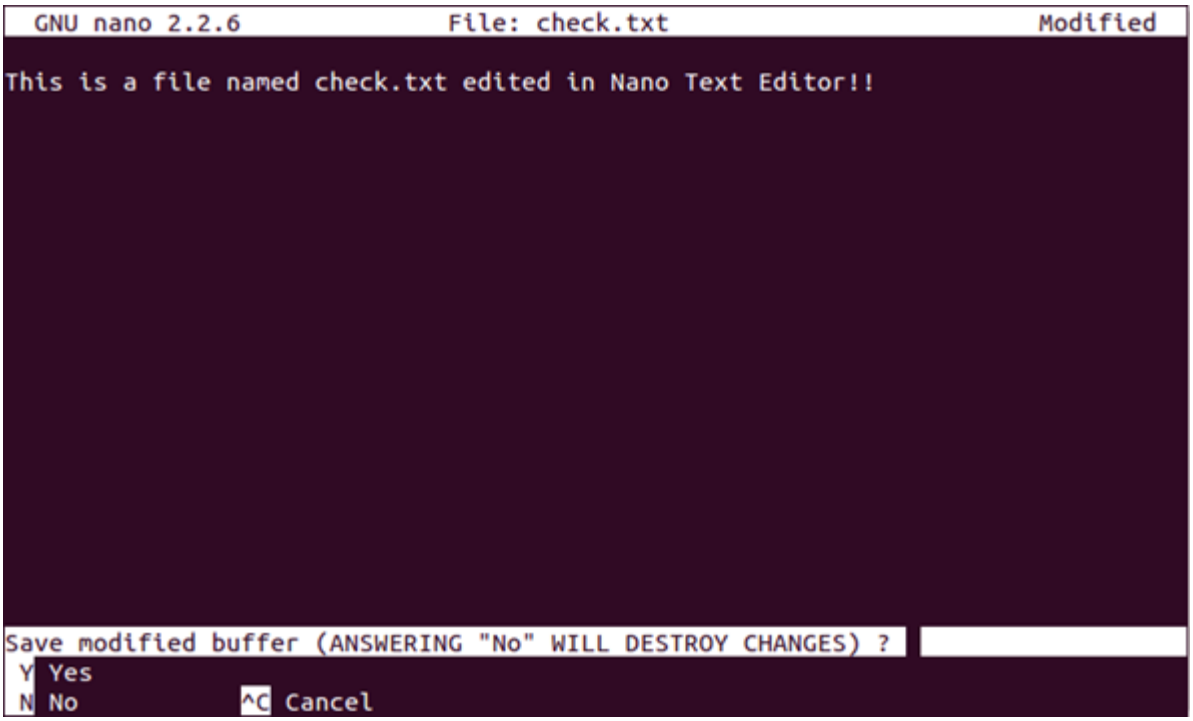
Intermediate Commands

1. **echo** — Perintah "echo" membantu kita memindahkan beberapa data, biasanya teks ke dalam sebuah file. Misalnya, jika Anda ingin membuat file teks baru atau menambahkan file teks yang sudah dibuat, Anda hanya perlu mengetikkan, "echo halo, nama saya adalah alok >> new.txt". Anda tidak perlu memisahkan ruang dengan menggunakan garis miring ke belakang, karena kita memasukkan dua kurung segitiga saat kita menyelesaikan apa yang perlu kita tulis.

2. **cat** — Gunakan perintah cat untuk menampilkan isi file. Biasanya digunakan untuk melihat program dengan mudah.

```
nayso@Alok-Aspire:~/Desktop$ echo hello, my name is alok >> new.txt
nayso@Alok-Aspire:~/Desktop$ cat new.txt
hello, my name is alok
nayso@Alok-Aspire:~/Desktop$ echo this is another line >> new.txt
nayso@Alok-Aspire:~/Desktop$ cat new.txt
hello, my name is alok
this is another line
```

3. **nano, vi, jed** — nano dan vi sudah terinstal editor teks di baris perintah Linux. Perintah nano adalah editor teks yang bagus yang menunjukkan kata kunci dengan warna dan bisa mengenali sebagian besar bahasa. Dan vi lebih sederhana dari nano. Anda bisa membuat file baru atau memodifikasi file menggunakan editor ini. Misalnya, jika Anda perlu membuat file baru bernama "check.txt", Anda bisa membuatnya dengan menggunakan perintah "nano check.txt". Anda dapat menyimpan file Anda setelah mengedit dengan menggunakan urutan Ctrl + X, lalu Y (atau N tidak). Menurut pengalaman saya, menggunakan nano untuk pengeditan HTML sepertinya tidak sebagus, karena warnanya, jadi saya merekomendasikan editor teks jed. Kami akan segera menginstall paket.



4. **sudo** — Perintah yang banyak digunakan pada command line Linux, sudo adalah singkatan dari "SuperUser Do". Jadi, jika Anda ingin perintah apapun dilakukan dengan hak akses administratif atau root, Anda bisa menggunakan perintah sudo. Misalnya, jika Anda ingin mengedit file seperti itu. alsa-base.conf, yang membutuhkan permission root, anda bisa menggunakan perintah - sudo nano alsa-base.conf. Anda dapat memasukkan baris perintah root dengan menggunakan perintah "sudo bash", lalu ketik kata sandi pengguna Anda. Anda juga bisa menggunakan perintah "su" untuk melakukan ini, tapi Anda perlu mengatur password root sebelum itu. Untuk itu, Anda bisa menggunakan perintah "sudo passwd" (tidak salah eja, itu passwd). Kemudian ketik password root yang baru.

```
nayso@Alok-Aspire:~/Desktop$ sudo passwd
[sudo] password for nayso:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
nayso@Alok-Aspire:~/Desktop$ su
Password:
root@Alok-Aspire:/home/nayso/Desktop#
```

5. **df** — Gunakan perintah df untuk melihat ruang disk yang tersedia di masing-masing partisi di sistem Anda. Anda bisa mengetikkan di df di baris perintah dan Anda dapat melihat setiap partisi yang terpasang dan ruang yang digunakan / tersedia di% dan di KB. Jika Anda menginginkannya ditampilkan dalam megabyte, Anda bisa menggunakan perintah "df -m".

```
root@Alok-Aspire:/home/nayso/Desktop# df -m
Filesystem     1M-blocks  Used Available Use% Mounted on
udev                 940     1       940   1% /dev
tmpfs                191     2       189   1% /run
/dev/sda5          96398 23466     68013  26% /
none                   1     0         1   0% /sys/fs/cgroup
none                   5     0         5   0% /run/lock
none                 951     1       950   1% /run/shm
none                 100     1       100   1% /run/user
```

6. **du** — Gunakan du untuk mengetahui penggunaan disk dari file di sistem anda. Jika Anda ingin mengetahui penggunaan disk untuk folder atau file tertentu di Linux, Anda dapat mengetikkan perintah df dan nama folder atau file. Misalnya, jika Anda ingin mengetahui ruang disk yang digunakan oleh folder dokumen di Linux, Anda bisa menggunakan perintah "du Documents". Anda juga bisa menggunakan perintah "ls -lah" untuk melihat ukuran file semua file dalam sebuah folder.

```
nayso@Alok-Aspire:~$ du Documents
516     Documents/DIYHacking
548     Documents
```

7. **tar** — Gunakan tar untuk bekerja dengan tarball (atau file yang dikompres dalam arsip tarball) di baris perintah Linux. Ini memiliki daftar panjang kegunaan. Hal ini dapat digunakan untuk kompres dan uncompress berbagai jenis arsip tar seperti .tar, .tar.gz, .tar.bz2, dll. Ini bekerja berdasarkan argumen yang diberikan kepadanya. Misalnya, "tar -cvf" untuk membuat arsip .tar, -vv untuk untar arsip tar, -tvf untuk mencantumkan isi arsip, dan lain-lain.

8. **zip, unzip** — Gunakan zip untuk memampatkan file ke dalam arsip zip, dan unzip untuk mengekstrak file dari arsip zip.

9. **uname** — Gunakan uname untuk menunjukkan informasi tentang sistem distro Linux anda yang sedang berjalan. Menggunakan perintah "uname -a" mencetak sebagian besar informasi tentang sistem. Ini mencetak tanggal rilis kernel, versi, jenis prosesor, dll.

```
nayso@Alok-Aspire:~$ uname -a
Linux Alok-Aspire 4.4.0-22-generic #40~14.04.1-Ubuntu SMP Fri May 13 17:27:18 UT
C 2016 i686 i686 i686 GNU/Linux
```

10. **apt-get** — Gunakan apt untuk bekerja dengan paket di baris perintah Linux. Gunakan apt-get untuk menginstal paket. Hal ini membutuhkan hak akses root, jadi gunakan sudocommand dengan itu. Sebagai contoh, jika Anda ingin menginstal jendel teks jed (seperti yang saya sebutkan sebelumnya), kita bisa mengetikkan perintah "sudo apt-get install jed". Demikian pula, setiap paket bisa dipasang seperti ini. Sebaiknya perbarui repositori Anda setiap kali Anda mencoba memasang paket baru. Anda bisa melakukannya dengan mengetikkan "sudo apt-get update". Anda bisa mengupgrade sistem dengan mengetikkan "sudo apt-get upgrade". Kita juga bisa mengupgrade distro dengan mengetikkan "sudo apt-get dist-upgrade". Perintah "apt-cache search" digunakan untuk mencari sebuah paket. Jika Anda ingin mencarinya, Anda bisa mengetikkan "pencarian jt apt-cache" (ini tidak memerlukan root).

```
nayso@Alok-Aspire:~$ sudo apt-get install jed
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  jed-common libslang2-modules slsh
Suggested packages:
  gpm
The following NEW packages will be installed:
  jed jed-common libslang2-modules slsh
0 upgraded, 4 newly installed, 0 to remove and 419 not upgraded.
Need to get 810 kB of archives.
After this operation, 2,992 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

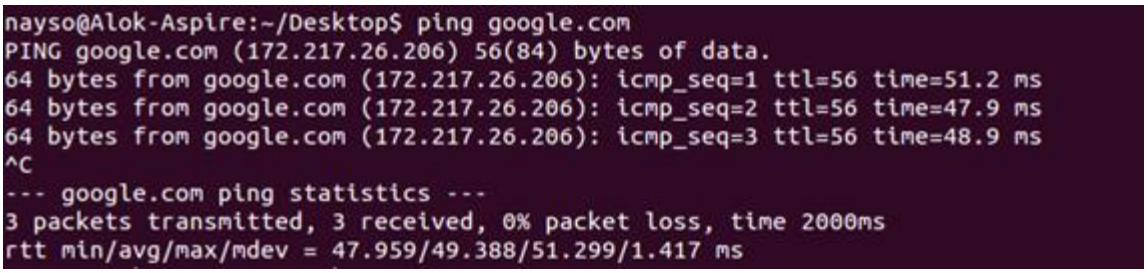
11. **chmod** — Gunakan chmod untuk membuat file executable dan mengubah hak akses yang diberikan padanya di Linux. Bayangkan Anda memiliki kode python bernama numbers.py di komputer Anda. Anda harus menjalankan "python numbers.py" setiap kali Anda perlu menjalankannya. Alih-alih itu, saat Anda membuatnya bisa dijalankan, Anda hanya perlu menjalankan "numbers.py" di terminal untuk menjalankan file. Untuk membuat file executable, Anda bisa menggunakan perintah "chmod + x numbers.py" dalam kasus ini. Anda bisa menggunakan "chmod 755 numbers.py" untuk memberikan izin root atau "sudo chmod + x numbers.py" untuk root executable. Berikut adalah beberapa informasi lebih lanjut tentang perintah chmod.

```
nayso@Alok-Aspire:~/Desktop$ ls
numbers.py
nayso@Alok-Aspire:~/Desktop$ chmod +x numbers.py
nayso@Alok-Aspire:~/Desktop$ ls
numbers.py
```

12. **hostname** — Gunakan nama host untuk mengetahui nama Anda di host atau jaringan Anda. Pada dasarnya, ini menampilkan nama host dan alamat IP Anda. Ketik saja "hostname" berikan hasilnya. Mengetik pada "hostname -I" memberi Anda alamat IP Anda di jaringan Anda.

```
nayso@Alok-Aspire:~/Desktop$ hostname
Alok-Aspire
nayso@Alok-Aspire:~/Desktop$ hostname -I
192.168.1.36
```

13. **ping** — Gunakan ping untuk memeriksa koneksi Anda ke server. Wikipedia mengatakan, "Pingis adalah utilitas perangkat lunak administrasi jaringan komputer yang digunakan untuk menguji jangkauan host pada jaringan Protokol Internet (IP)". Cukup, saat Anda mengetik, misalnya, "ping google.com", cek apakah bisa terhubung ke server dan kembali lagi. Ini mengukur waktu pulang-pergi ini dan memberi Anda rincian tentang hal itu. Penggunaan perintah ini untuk pengguna sederhana seperti kita adalah dengan mengecek koneksi internet anda. Jika mengaitkan server Google (dalam kasus ini), Anda dapat mengonfirmasi bahwa koneksi internet Anda aktif!



Tip dan Trik Menggunakan Linux Command Line

1. Anda dapat menggunakan perintah yang jelas untuk menghapus terminal jika terisi dengan terlalu banyak perintah.
2. TAB bisa digunakan untuk mengisi terminal. Sebagai contoh, Anda hanya perlu mengetikkan "cd Doc" dan kemudian TAB dan terminal mengisi sisanya dan membuatnya menjadi "cd Documents".
3. Ctrl + C dapat digunakan untuk menghentikan perintah apapun di terminal dengan aman. Jika tidak berhenti dengan itu, maka Ctrl + Z bisa digunakan untuk memaksa menghentikannya.
4. Anda bisa keluar dari terminal dengan menggunakan perintah exit.
5. Anda bisa mematikan atau reboot komputer dengan menggunakan perintah sudo haltand sudo reboot.

C. Tugas

1. Sebutkan 5 command lain yang ada dalam command prompt windows, jelaskan, serta berilah contoh penggunaannya.
2. Sebutkan 5 command lain yang ada dalam linux terminal, jelaskan, serta berilah contoh penggunaannya

PERTEMUAN 3
PENGENALAN OS SIM: MANAJEMEN PROSES

A. Tujuan Praktikum

- Praktikan dapat memahami proses thread pada sistem operasi
- Praktikan dapat memahami proses yang terjadi pada sistem operasi

B. Materi dan Aktivitas Praktikum

OS-SIM Introduction

OS Sim (Operating System Concept Simulator) adalah aplikasi tujuan pendidikan untuk mensimulasikan secara grafis konsep Sistem Operasi dan mendukung proses belajar siswa sains.

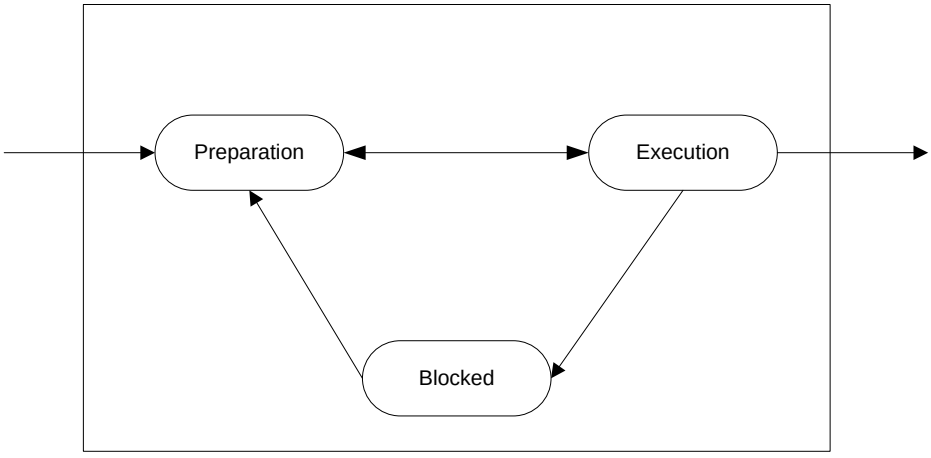
Secara khusus, aplikasi dibagi menjadi empat blok atau simulasi independen: penjadwalan proses, manajemen memori, manajemen sistem file dan kebijakan penjadwalan disk.

Di luar mencoba untuk menggambarkan sistem real yang kompleks, ia berfokus pada konsep Sistem Operasi dasar karena dijelaskan kepada siswa ilmu komputer, termasuk algoritma utama dan parameter yang paling relevan.

Ini juga memiliki sejumlah contoh dan latihan dengan simulasi yang sesuai untuk dapat mengeksplorasi semua potensi aplikasi.

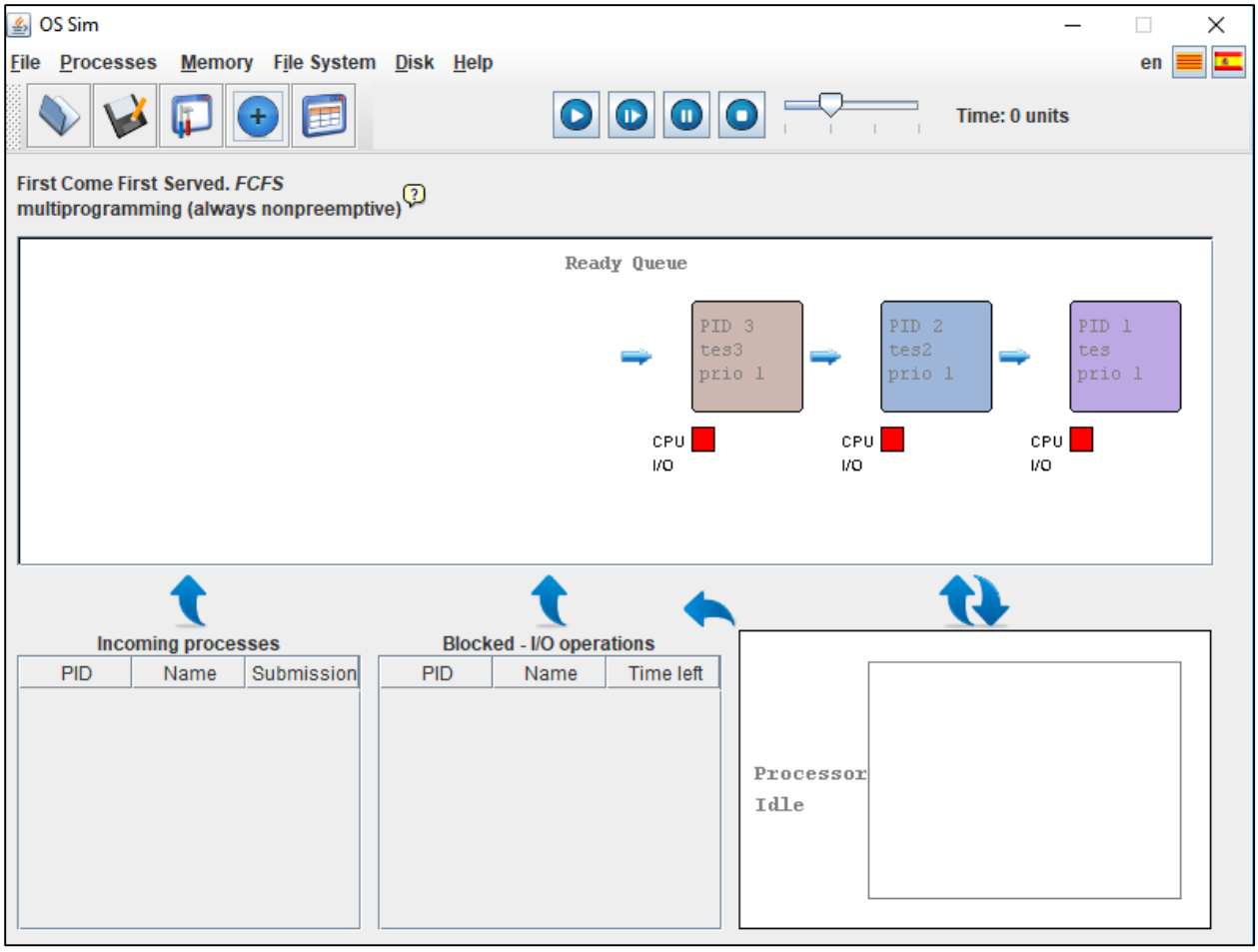
Process Scheduling

Simulasi ini berfokus pada penjadwalan proses jangka pendek dengan satu prosesor, tujuannya adalah memilih salah satu proses yang tersedia untuk dijalankan pada prosesor. Eksekusi proses melibatkan siklus bolak-balik CPU dan I / O burst (CPU dan I / O Burst Cycle). Proses hanya tersedia saat burst CPU. Simulasi didasarkan pada model tiga negara, yang mendefinisikan tiga status proses dan transisi yang mungkin: Running, Waiting (atau Ready) dan Blocked.



Setiap saat hanya satu proses yang berjalan, proses lain menunggu dalam antrian siap sampai penjadwal memilihnya, atau diblokir melakukan operasi I / O.

Screen



Processor (CPU)

Bisa idle (gratis) atau sibuk, dalam hal ini ia menunjukkan proses running, informasi yang relevan (PID, name, priority) dan complete life cycle (CPU dan I / O bursts). Setiap kotak dari siklus burst sesuai dengan unit waktu simulasi, dalam warna merah itu menunjukkan di mana dalam siklus adalah prosesnya. Proses yang berjalan meninggalkan prosesor saat selesai, kembali ke antrian siap atau memulai burst I / O.

Ready queue

Berisi proses yang tersedia untuk dieksekusi (waiting). Antrian selalu dipesan tergantung pada algoritma penjadwalan (first right, last one left):

FCFS (First Come First Served): Pemesanan kedatangan ke antrian.

SJF (Shortest Job First): Kurang waktu prosesor tersisa (proses CPU burst saat ini).

Prioritas: Dipesan oleh prioritas proses, nilai prioritas yang lebih tinggi menunjukkan prioritas

yang lebih tinggi, proses prioritas yang sama ditempatkan sesuai urutan kedatangan.
Round Robin: Urutan kedatangan ke antrian.
Untuk setiap proses, informasi tersebut menunjukkan informasi yang paling relevan (PID, name, priority) dan siklus hidup lengkap (ledakan CPU dan I / O). Setiap kotak dari siklus burst sesuai dengan satuan waktu simulasi, dalam warna merah itu menunjukkan di mana dalam siklus adalah prosesnya. Sementara simulasi dihentikan, klik kanan pada proses apapun untuk menampilkan menu popup yang memungkinkan Anda memodifikasi dan menghapus prosesnya. Proses hanya meninggalkan antrian siap untuk masuk ke prosesor.
Kriteria masuk urutan proses dalam antrian setelah memulai simulasi: proses baru (masuk) -> proses yang menyelesaikan proses I / O -> yang meninggalkan CPU.

Incoming processes
Proses yang dimulai nanti (time start> 0) diletakkan di antrian ini. Seiring berjalannya simulasi dan saatnya tiba, mereka memasuki antrian siap. Antrian ini memungkinkan pengguna merencanakan keseluruhan simulasi dan kemudian menjalankannya sepenuhnya tanpa intervensi lebih lanjut. Proses diurutkan berdasarkan waktu yang tersisa untuk memulai, (atau sisa waktu untuk masuk ke antrian siap).
Sementara simulasi dihentikan, memilih proses apapun dari tabel menunjukkan menu popup yang memungkinkan Anda memodifikasi dan menghapus prosesnya.

Settings.

Menerapkan simulation behavior

- *Multiprogramming or monoprogramming:* Penjadwal saat proses berjalan memulai burst I / O, dalam multiprogramming proses baru menggunakan CPU, jika tidak, prosesor akan menganggur sampai I / O burst selesai.
- Algoritma:
    - *FCFS (First Come First Served)*
    - *SJF (Shortest Job First)*
    - *Priority*
    - *Round Robin*

Menambahkan Proses

Proses hanya dapat ditambahkan saat simulasi dihentikan, karena setiap proses baru memasukkan informasi berikut:

- *PID* Pengenal proses dihitung secara otomatis dan bersifat bertahap.
- *Name (required)* Teks apapun
- *Priority* nilai yang lebih tinggi menunjukkan prioritas yang lebih tinggi (Range: 1 - 10).
- *Submission,* proses waktu kedatangan ke antrian siap, jika 0 itu ditambahkan ke antrian siap, sedangkan jika lebih besar dari 0 proses ditambahkan ke tabel proses masuk sampai simulasi mencapai waktu pengiriman (Range: 1 - 100).
- *Color,* warna yang akan menarik elemen grafis yang terkait dengan prosesnya.

- *Not complete,* menunjukkan bahwa proses tidak berakhir dan siklus burst berulang tanpa henti.
- *Burst cycle,* tabel ini memungkinkan Anda menentukan durasi *burst* CPU dan I / O, setiap baris adalah unit waktu, semua proses harus dimulai dengan burst CPU selain jika selesai, burst terakhir juga harus berupa CPU. Durasi proses maksimum adalah 10 unit waktu.

Pengaturan hanya tersedia saat simulasi dihentikan

Data dan statistik

Lihat informasi penjadwalan yang dihasilkan kapan saja dari simulasi.

Data yang terhimpun adalah:

- *Efficiency,* persentase waktu prosesor sedang sibuk.
- *Throughput,* # proses selesai per satuan waktu.
- *Average duration,* durasi rata-rata proses selesai (duration = completion - submission).
- *Average waiting time,* Proses rata-rata waktu tunggu (jumlah waktu dalam antrian siap).
- *Average response time,* Proses rata-rata waktu tunggu (jumlah waktu dalam antrian siap).

Untuk setiap proses ada tabel dengan informasi berikut

- *PID,* ID proses (Warna latar belakang adalah warna proses).
- *Name,* dari proses.
- *Priority* prioritas proses.
- *Submission,* waktu kedatangan ke antrian siap.
- *Periodic* prosesnya tidak lengkap.
- *CPU* Jumlah waktu proses yang telah berjalan sejauh ini.
- *Response* waktu dari pengiriman sampai masukan pertama ke prosesor.
- *Waiting,* Jumlah waktu proses sudah dalam antrian siap sejauh ini.
- *Duration,* jumlah waktu dari pengajuan sampai proses selesai.
- *% CPU,* persentase CPU pada waktu tunggu.
- *% I/O,* waktu I / O pada durasi proses (Semua burst).

C. Tugas

- Implementasikan simulasi proses diatas, buatlah laporan pengamatan yang berisi screenshot implementasi beserta data-data yang dapat dihimpun dari OS-SIM.
- Impelementasikan program yang memanfaatkan thread dalam Java.

PERTEMUAN 4
PENJADWALAN PROSES (OS-SIM)

A. Tujuan Praktikum

- Praktikan dapat memahami proses penjadwalan pada suatu sistem operasi
- Praktikan dapat memahami proses monoprogramming dan multiprogramming
- Praktikan dapat memahami penggunaan preemptive
- Praktikan dapat menguasai dan mengenal macam-macam jenis proses penjadwalan.

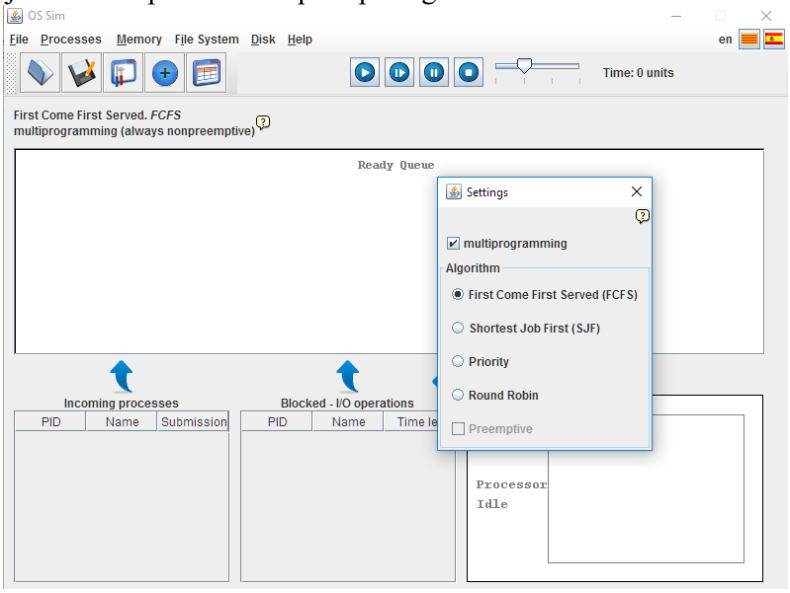
B. Materi

Penjadwalan Proses

Proses penjadwalan atau Process Scheduling adalah cara menjadwalkan berbagai proses yang akan di kerjakan oleh CPU berdasarkan algoritma penjadwalan tertentu. Ada 4 algoritma penjadwalan yang akan dibahas yaitu:

- First-Come, First-Served (FCFS)
- Shortest-Job-First (SJF)
- Priority
- Round Robin(RR)

Algoritma penjadwalan sendiri ada yang bersifat preemptive dan non-preemptive. Algoritma non-preemptive dibuat agar jika ada proses yang masuk ke running state, maka proses itu tidak bisa diganggu sampai menyelesaikan prosesnya, sedangkan penjadwalan preemptive menggunakan prioritas dimana proses dengan prioritas tinggi dapat menyela proses dengan prioritas rendah untuk dikerjakan CPU terlebih dahulu. OS-SIM digunakan sebagai simulator untuk memahami cara kerja dari Proses Penjadwalan. Tampilan OS-SIM untuk proses penjadwalan dapat dilihat seperti pada gambar 4.1.



Gambar 4.1. OS-SIM untuk proses penjadwalan

Pada algoritma penjadwalan terdapat beberapa hal penting yang perlu di perhatikan diantaranya:

- Efisiensi, persentase waktu saat Processor sibuk
- Throughput, Jumlah Proses yang selesai per satuan waktu
- Average Duration, rerata durasi penyelesaian proses (duration = completion – submission)

- Average waiting time, rerata waktu tunggu proses (Service Time - Arrival time)
- Average response time, Rerata waktu respon proses (response = respon proses pertama CPU – submission)

C. Aktivitas Praktikum

Praktikum 1. First Come First Serve (FCFS)

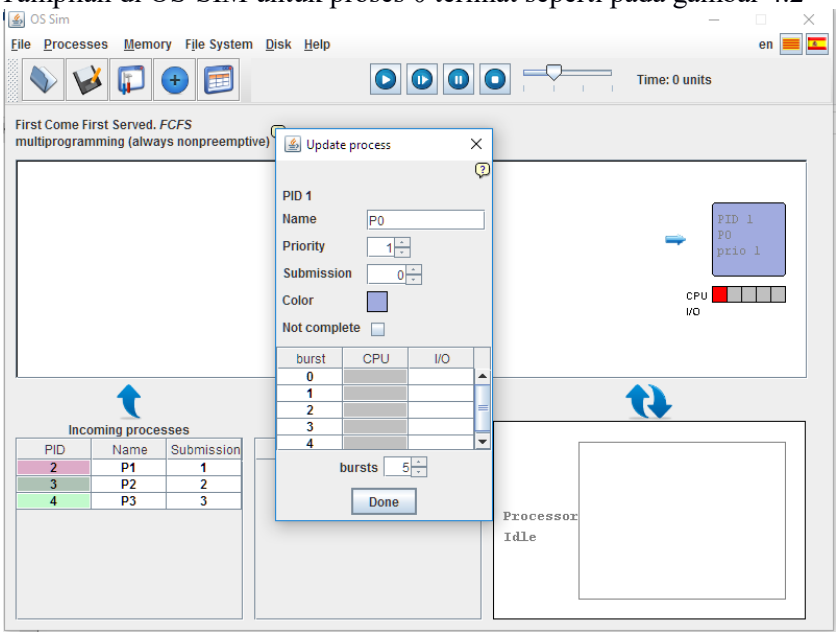
Algoritma FCFS dalam proses penjadwalan memiliki sifat sebagai berikut:

- Proses dieksekusi berdasarkan yang pertama kali datang
- Merupakan algoritma penjadwalan non-preemptive
- Mudah untuk dipahami dan digunakan
- Implementasinya berdasarkan antrian FIFO (First In, First Out)

Tabel 4.1 Data FCFS

Process	Arrival Time	Burst Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

Gunakan data pada tabel 4.1 untuk proses penjadwalan dengan algoritma FCFS
Tampilan di OS-SIM untuk proses 0 terlihat seperti pada gambar 4.2



Gambar 4.2. Penambahan proses pada OS_SIM

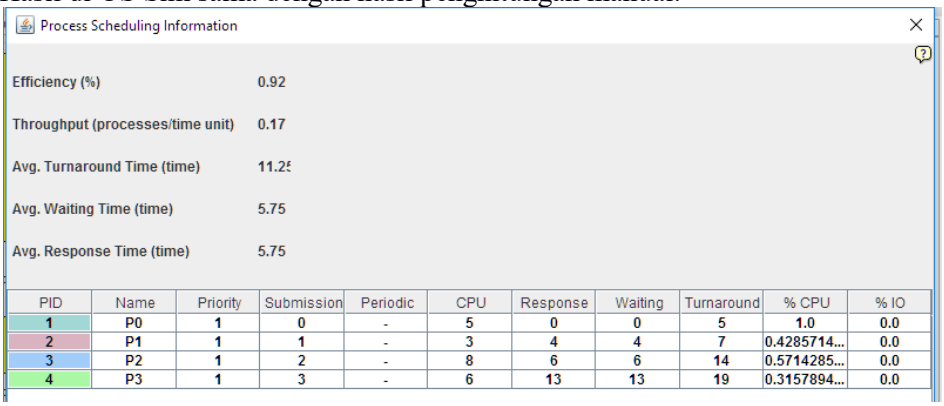
Wait time setiap proses :

Process	Wait time : Service Time – Arrival Time
P0	0-0=0
P1	5-1=4

P2	8-2=6
P3	16-3=13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Hasil di OS Sim sama dengan hasil penghitungan manual.



Praktikum 2. Shortest Job First (SJF)

- Pendekatan terbaik untuk mengurangi waiting time
- Mudah diimplementasikan di Batch systems dimana kebutuhan CPU sudah diketahui
- Tidak mungkin diimplementasikan pada sistem yang interaktif dimana kebutuhan CPU tidak diketahui.
- Processor harus sudah tahu berapa lama waktu yang akan dipakai tiap prosesnya.

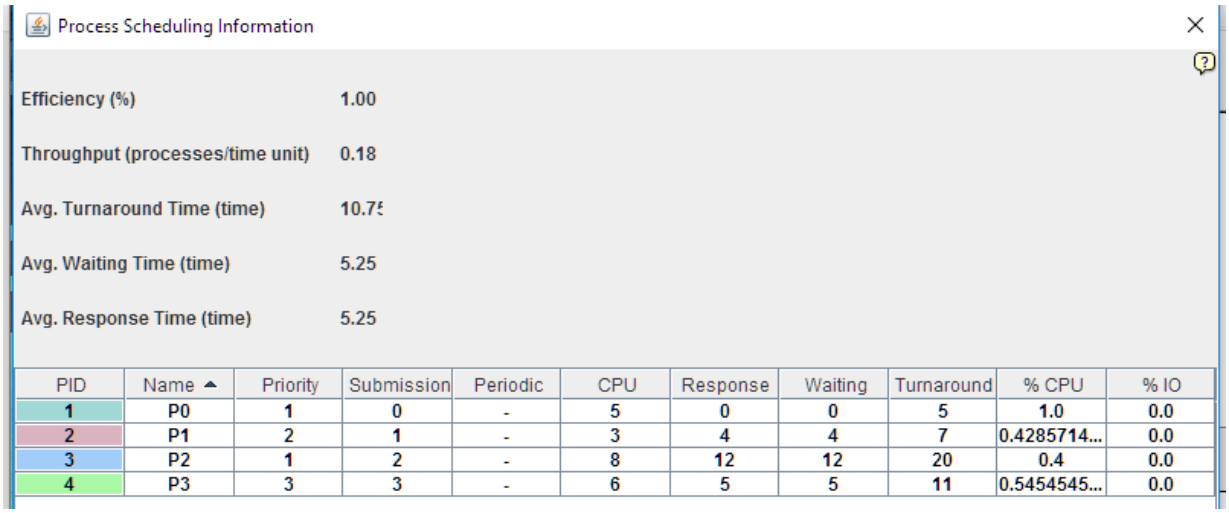
Non – preemptive

Process	Arrival Time	Burst Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	14
P3	3	6	8

Wait time setiap proses :

Process	Wait time : Service Time – Arrival Time
P0	0 – 0 = 0
P1	5 – 1 = 4
P2	14 – 2 = 12
P3	8 – 3 = 5

Average Wait Time: (0+4+12+5) / 4 = 5.25

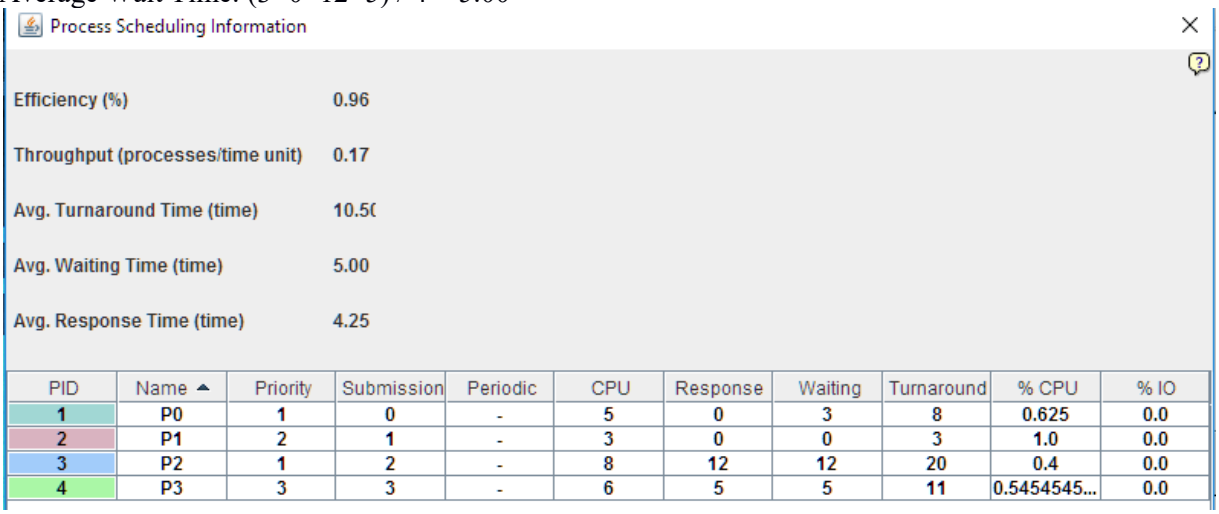


Preemptive

Wait time setiap proses :

Process	Wait time : Service Time – Arrival Time
P0	(0 – 0) + (4 – 1) = 3
P1	1 – 1 = 0
P2	14 – 2 = 12
P3	8 – 3 = 5

Average Wait Time: (3+0+12+5) / 4 = 5.00



Perbedaan antara penjadwalan preemptive dan non-preemptive.

Basis	Preemptive	Non Preemptive
Dasar	Resources dialokasikan pada proses dalam waktu tertentu	Sewaktu resources dialokasikan kepada proses. Proses akan memegang resources sampai selesai burst time atau berganti menjadi waiting state

Interrupt	Process dapat di-interrupt	Process tidak dapat di-interrupt hingga selesai atau berganti menjadi waiting state
Starvation	Jika proses berprioritas tinggi sering datang dalam antrian, maka proses berprioritas rendah dapat starve	Jika proses dengan burst time yang lama sedang berjalan, maka proses dengan burst time kecil dapat starve
Overhead	Ya	Tidak
Fleksibilitas	Lebih Fleksibel	Kaku
Cost	Cost associated	Tidak cost associated

Praktikum 3. Priority

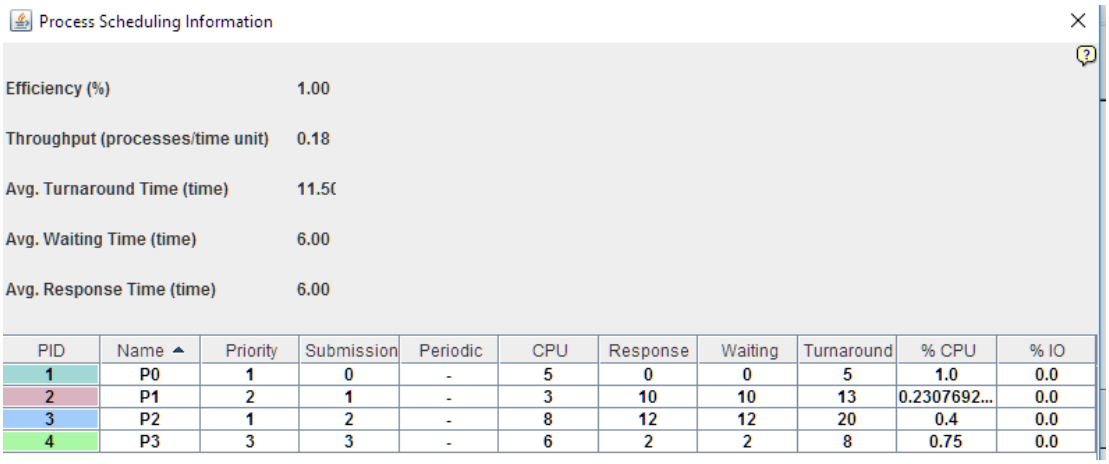
- Penjadwalan Priority merupakan algoritma non-preemptive dan salah satu algoritma penjadwalan yang paling sering dipakai di batch system.
- Setiap proses diberi prioritas. Proses dengan prioritas tertinggi akan dikerjakan pertama kali.
- Proses dengan prioritas yang sama akan dijalankan dengan prinsip FIFO.
- Prioritas dapat ditentukan berdasarkan kebutuhan memori, kebutuhan waktu, atau kebutuhan resources yang lain.

Process	Arrival Time	Burst Time	Priority	Service Time
P0	0	5	1	0
P1	1	3	2	11
P2	2	8	1	14
P3	3	6	3	5

Wait time setiap proses :

Process	Wait time : Service Time – Arrival Time
P0	0 – 0 = 0
P1	11 – 1 = 10
P2	14 – 2 = 12
P3	5 – 3 = 2

Average Wait Time: (0+10+12+2) / 4 = 6.0



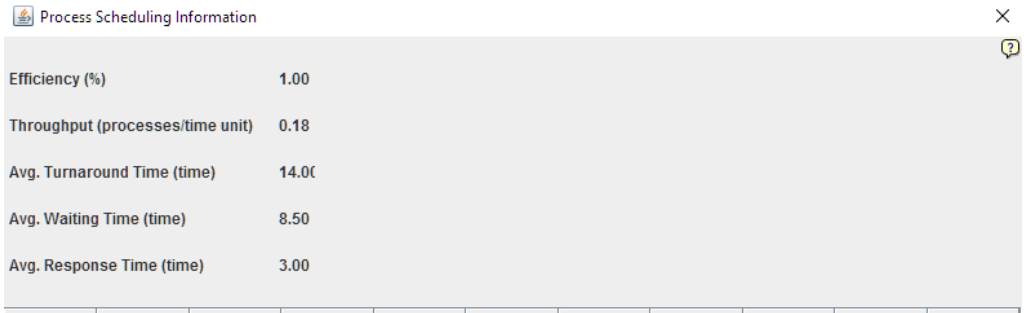
PID	Name	Priority	Submission	Periodic	CPU	Response	Waiting	Turnaround	% CPU	% IO
1	P0	1	0	-	5	0	0	5	1.0	0.0
2	P1	2	1	-	3	10	10	13	0.2307692...	0.0
3	P2	1	2	-	8	12	12	20	0.4	0.0
4	P3	3	3	-	6	2	2	8	0.75	0.0

Praktikum 4. Round Robin

- Round Robin adalah algoritma penjadwalan preemptive.
- Setiap proses dilakukan dalam interval waktu yang sama yang disebut quantum.
- Ketika proses sedang dikerjakan dalam jangka waktu yang ditentukan, maka proses akan di-interrupt dan proses lain akan dikerjakan dalam jangka waktu yang sama.
- Context switching dipakai untuk menyimpan state dari proses yang di-interrupt.
- Wait time setiap proses dengan nilai quantum = 3.

Process	Wait time : Service Time – Arrival Time
P0	(0 – 0) + (12 – 3) = 9
P1	3 – 1 = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5



PID	Name	Priority	Submission	Periodic	CPU	Response	Waiting	Turnaround	% CPU	% IO
1	P0	1	0	-	5	0	9	14	0.3571428...	0.0
2	P1	2	1	-	3	2	2	5	0.6	0.0
3	P2	1	2	-	8	4	12	20	0.4	0.0
4	P3	3	3	-	6	6	11	17	0.3529411...	0.0

D. Tugas

Jawab pertanyaan berikut:

1. Apa beda monoprogramming dan multiprograming pada FCFS? Lengkapi jawaban dengan bukti data.
2. Apa pengaruh penggunaan Preemptive pada SJF dan Priority? Lengkapi jawaban dengan bukti data
3. Apa pengaruh nilai quantum pada roundrobin? Lengkapi jawaban dengan bukti data.

PRAKTIKUM 5
PEMROGRAMAN MULTI-THREAD

A. Tujuan Praktikum

- Praktikan dapat memahami pemrograman multithread
- Praktikan dapat menguasai cara pembuatan program multithread

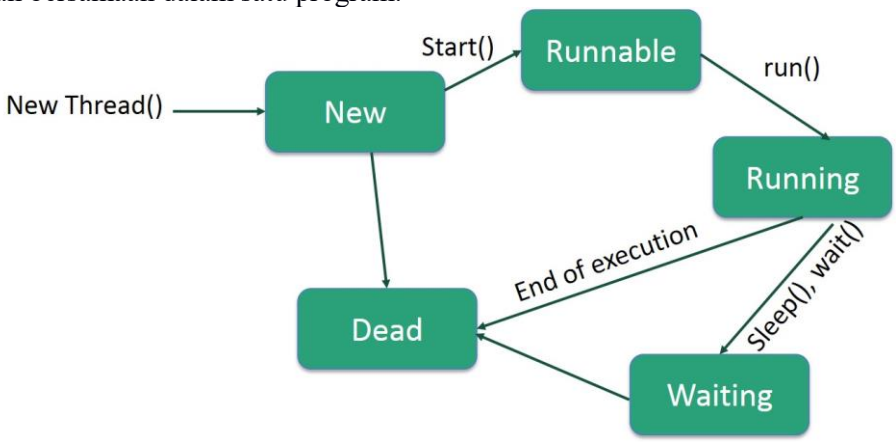
B. Materi

Multi Thread

Program multi-thread berisi 2 atau lebih proses yang dapat di run secara bersamaan dan setiap proses dapat meng-handle task yang berbeada dalam waktu yang sama agar penggunaan resource yang ada lebih optimal jika computer kita memiliki beberapa CPU.

Multi-threading memperluas pemikiran dari multitasking pada aplikasi dimana kita dapat membagi beberapa operasi yang spesifik dalam satu aplikasi menjadi thread. Dimana setiap thread berjalan secara parallel. Sistem Operasi membagi processing time tidak hanya aplikasi yang berbeda, tetapi juga setiap thread dalam satu aplikasi.

Dengan multi-threading, kita menulis program dengan banyak aktivitas yang harus berjalan bersamaan dalam satu program.



- New – Thread baru muncul dalam new state. Thread akan tetap seperti ini sampai ada program yang memulau thread tersebut. Biasa disebut born thread.
- Runnable – Setelah thread baru dimulai, thread menjadi runnable. Thread yang berada dalam kondisi ini berarti sedang menjalankan suatu task.
- Waiting – Terkadang, thread ber-transisi ke waiting state ketika thread menunggu thread lain menyelesaikan task-nya. Thread dapat kembali ke runnable state ketika thread lain memberi sinyal agar waiting state dapat kembali dijalankan.
- Timed Waiting – Runnable thread dapat menjadi timed waiting state pada interval waktu tertentu.
- Terminated (Dead) – Runnable thread dapat menjadi terminated state ketika sudah menyelesaikan tasknya.

Kita harus meng-override method run( ) di class Thread. Method ini memberi titik masuk untuk thread dan kita menulis kode program di method ini. Ini contoh syntax sederhana method run().

```
public void run() {
   /**
     * Your Multithread codes goes here
    */
}
```

Ketika Thread sudah terbentuk, kita dapat memulainya dengan menggunakan method start(), dimana akan menjalankan method run(). Ini contoh syntax sederhana method start().

```
   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

Lalu, untuk membuat thread.

```
class RunnableDemo implements Runnable {
   private Thread t;
   private String threadName;
   int a = 0;

   RunnableDemo( String name) {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }
}

public class JavaApplication1 {

   public static void main(String args[]) {
      RunnableDemo R1 = new RunnableDemo( "Thread-1");
      R1.start();

      RunnableDemo R2 = new RunnableDemo( "Thread-2");
      R2.start();
   }
}
```

C. Aktivitas Praktikum

Praktikum 1. Pemrograman Multithread

Berikut adalah syntax sederhana yang mamakai multi threading dan output-nya :

```
package javaapplication1;

/**
 *
 * @author Winstein
 */
class RunnableDemo implements Runnable {
   private Thread t;
   private String threadName;
   int a = 0;

   RunnableDemo( String name) {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }

   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 0; i < 2; i++) {
            a++;
            System.out.println("Nilai : " + a + " , " + threadName);
            Thread.sleep(50);
         }
      }catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
      }
      System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}

public class JavaApplication1 {

   public static void main(String args[]) {
      RunnableDemo R1 = new RunnableDemo( "Thread-1");
      R1.start();

      RunnableDemo R2 = new RunnableDemo( "Thread-2");
      R2.start();
   }
}
```

Output :

```
run:
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Nilai : 1 , Thread-1
Running Thread-2
Nilai : 1 , Thread-2
Nilai : 2 , Thread-1
Nilai : 2 , Thread-2
Thread Thread-2 exiting.
Thread Thread-1 exiting.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Seperti yang bisa kita lihat, int a dipakai oleh 2 proses (Thread-1 dan Thread-2) masing – masing 2 kali. Tetapi nilai a bukan 4, melainkan 2. Hal ini karena 2 proses tadi berjalan bersamaan.

Thread-1	Thread-2	Value of a by Thread -1	Value of a by Thread -2
CREATE		NULL	NULL
START		0	NULL
	CREATE	0	0
	START	0	0
RUN : a++		1	0
	RUN : a++	1	1
a++		2	1
	a++	2	2
	EXIT	2	2
EXIT		2	2

Thread Methods

Ini beberapa method penting dalam class Thread.

No.	Method & Description
1	public void start() Memulai thread, lalu memicu method run() dalam Thread.
2	public void run() Jika thread dimulai menggunakan runnable target yang berbeda, maka method run() akan memulai runnable object.
3	public final void setName(String name) Mengubah nama Thread. Terdapat method getName() untuk mengambil nama Thread.

4	public final void setPriority(int priority) Mengatur prioritas Thread. Nilainya dapat diatur dari 1 sampai 10.
5	public final void setDaemon(boolean on) Parameter true untuk thread yang Daemon
6	public final void join(long millisec) Thread sekarang memulai thread kedua dan selanjutnya, menimbulkan thread sekarang untuk mengblok sampai thread kedua selesai atau setelah beberapa milisekon.
7	public void interrupt() Interrupt thread ini. Membuat thread berjalan jika diblok.
8	public final boolean isAlive() Return true jika thread masih berjalan, yaitu kapanpun semenjak thread dimulai tetapi sebelum selesai.

Method – method diatas dipakai pada Thread – thread tertentu. Method dibawah ini bersifat statis. Menggunakan satu method statis memulai operasi pada thread yang sedang berjalan.

No.	Method & Description
1	public static void yield() Membuat thread yang sedang berjalan untuk memberikan waktunya untuk thread lain dengan prioritas yang sama yang sedang waiting state.
2	public static void sleep(long millisec) Membuat thread yang sedang berjalan untuk berhenti selama beberapa milisekon.
3	public static boolean holdsLock(Object x) Return true jika thread sekarnag memakai lock dari objek x
4	public static Thread currentThread() Return suatu reference pada thread yang sedang berjalan, yaitu thread yang memulai method ini.
5	public static void dumpStack() Mencetak jejak stack yang dipakai thread yang sedang berjalan, berguna ketika debugging aplikasi multithread.

D. Tugas

Buat sebuah program dengan 4 thread untuk menampilkan sebuah pesan

PERTEMUAN 6
SEMAPHORE PROGRAMMING

A. Tujuan Praktikum

- Praktikan dapat memahami pemrograman semaphore
- Praktikan dapat menguasai cara penyelesaian permasalahan produser-konsumer

B. Materi

Semaphore

Semaphore adalah sebuah struktur data komputer yang digunakan untuk sinkronisasi proses, yaitu untuk memecahkan masalah di mana lebih dari satu proses atau thread dijalankan secara bersamaan dan harus diatur urutan kerjanya. Semaphore dicetuskan oleh Edsger Dijkstra dan pertama digunakan dalam sistem operasi THE.

Karena semaphore dapat memiliki variabel penghitung, ia dapat digunakan ketika beberapa threads ingin mencapai suatu tujuan secara bekerja sama.

Misalnya:

Ada thread A yang ingin informasi dari 2 basis data sebelum ia dapat maju. Akses ke kedua basis data tersebut dikendalikan oleh dua thread B dan C. Keduanya memiliki message-processing loop; semua yang ingin menggunakannya menulis pesan dalam antrian pesan. Thread A menginisialisasi semaphore S dengan init(S,-1). A kemudian menulis permohonan data, termasuk pointer ke semaphore S, kepada baik B maupun C. Kemudian A memanggil P(S), yang memblokir. Kedua thread lain mengambil informasi; setelah mereka selesai, mereka memanggil V(S) dalam semaphore. Hanya setelah keduanya selesai mengambil data, thread A dapat maju. Semaphore seperti ini disebut "counting semaphore".

Selain counting semaphore, terdapat juga "blocking semaphore", yaitu semaphore yang diinisialisasi dengan nilai 0. Artinya setiap thread yang melakukan P(S) akan diblokir sampai thread lain memanggil V(S). Jenis ini sangat berguna ketika urutan eksekusi thread harus diperhatikan.

Bentuk semaphore paling sederhana adalah "semaphore biner", digunakan untuk mengendalikan akses kepada satu resource, atau disebut juga mutual exclusion. Sebuah semaphore biner selalu diinisialisasi dengan nilai 1. Ketika resource sedang digunakan, thread yang menggunakan memanggil P(S) untuk mengurangi nilai ini menjadi 0, dan setelah selesai menginkrementasi kembali nilainya menjadi 1 untuk menandakan resource yang sekarang bebas.

Permasalahan Produsen-Konsumen

Dalam ilmu komputer, produsen-konsumen masalah (juga dikenal sebagai masalah bounded-buffer) adalah contoh klasik dari masalah sinkronisasi multi-proses. Masalahnya menggambarkan dua proses, produsen dan konsumen, yang berbagi bersama, tetap ukuran buffer. Tugas produsen adalah untuk menghasilkan sepotong data, memasukkannya ke buffer dan mulai lagi. Pada saat yang sama konsumen adalah mengkonsumsi data (yaitu mengeluarkannya dari buffer) salah satu bagian pada suatu waktu. Masalahnya adalah untuk memastikan bahwa produsen tidak akan mencoba untuk menambahkan data ke dalam buffer jika itu penuh dan bahwa konsumen tidak akan mencoba untuk menghapus data dari buffer kosong.

Solusi untuk produsen adalah baik pergi untuk tidur atau membuang data jika buffer

penuh. Kali berikutnya konsumen menghapus item dari buffer, ini akan memberitahu produsen yang mulai mengisi buffer lagi. Dengan cara yang sama, konsumen bisa tidur jika menemukan buffer yang akan kosong. Kali berikutnya produsen menempatkan data ke dalam buffer, maka konsumen bangun tidur. Solusi ini dapat dicapai melalui komunikasi antar-proses, biasanya menggunakan semaphores. Sebuah solusi tidak memadai bisa mengakibatkan kebuntuan dimana kedua proses sedang menunggu untuk dibangunkan.

C. Aktivitas Praktikum

Praktikum 1. Bounded-buffer

Listing Program:

```
#include<stdio.h>
void main(){
    int buffer[10], bufsize, in, out, produce, consume, choice=0;
    in = 0;
    out = 0;
    bufsize = 10;
    while(choice !=3){
        printf("\n1. Produce \t 2. Consume \t3. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
        case 1: if((in+1)%bufsize==out)
                    printf("\nBuffer is Full");
                else{
                    printf("\nEnter the value: ");
                    scanf("%d", &produce);
                    buffer[in] = produce;
                    in = (in+1)%bufsize;
                }
                Break;
        case 2: if(in == out)
                    printf("\nBuffer is Empty");
                else{
                    consume = buffer[out];
                    printf("\nThe consumed value is %d", consume);
                    out = (out+1)%bufsize;
                }
                break;
        }
    }
}
```

OUTPUT:

1. Produce
2. Consume
3. Exit

Enter your choice: 2
Buffer is Empty
1. Produce
2. Consume
3. Exit
Enter your choice: 1
Enter the value: 1
00
1. Produce
2. Consume

3. Exit
Enter your choice: 2
The consumed value is 100
1. Produce
2. Consume
3. Exit
Enter your choice: 3

D. Tugas

Jawab pertanyaan berikut:

1. Jelaskan cara kerja listing dari program 1 dengan menggunakan flowchart.
2. Apa pengaruh buffer terbatas dan buffer tak terbatas untuk penyelesaian masalah produser konsumer?
3. Bisakah produser dan konsumer mengakses shared memory secara bersamaan? Jika tidak teknik apa yang bisa dilakukan untuk melakukan hal tersebut?

PERTEMUAN 7
PERMASALAHAN READERS/WRITERS

A. Tujuan Praktikum

- Praktikan dapat memahami readers/writers problem

B. Materi

Keadaan readers/writers problem merupakan bagian dari banyak permasalahan dalam sinkronisasi yang sering terjadi. Ketika ada 2 jenis proses readers dan writers yang bisa saling shared data dan mengakses database secara paralel. Proses reader bertugas untuk membaca data, sedangkan proses writer bertugas untuk (menulis data baru) mengupdate nilai dalam data. Solusi permasalahan ini adalah menjadikan proses tersebut dijalankan dalam keadaan terpisah/terisolasi dari proses lain. Salah satu solusi dari permasalahan ini adalah dengan menggunakan Semaphore.

Terdapat 3 solusi ini, yaitu :

1. Solusi Dengan Pembaca Diprioritaskan
2. Solusi Dengan Penulis Diprioritaskan
3. Solusi Prioritas Bergantian

C. Aktivitas Praktikum

Langkah percobaan:

1. Ketiklah program Java untuk solusi dengan prioritas bergantian di bawah ini, kemudian jalankan program tersebut!

```
01. public class ReaderWriterServer {
02. public static void main(String args[]) { Database server = new Database();
03. Reader[] readerArray = new Reader[NUM_OF_READERS]; Writer[] writerArray = new Writer[NUM_OF_WRITERS]; for (int i = 0; i < NUM_OF_READERS; i++) {
04. readerArray[i] = new Reader(i, server); readerArray[i].start();
05. }
06. for (int i = 0; i < NUM_OF_WRITERS; i++) { writerArray[i] = new Writer(i, server); writerArray[i].start();
07. }
08. }
09. private static final int NUM_OF_READERS = 3; private static final int NUM_OF_WRITERS = 2;
10. }
11. class Reader extends Thread {
12. public Reader(int r, Database db) { readerNum = r;
13. server = db;
14. }
15. public void run() { int c;
16. while (true) { Database.tunggu();
17. System.out.println("reader " + readerNum + " wants to read.");
18. c = server.mulaiBaca(); System.out.println("reader " + readerNum + " is reading. Reader Count = " + c); Database.tunggu();
19. System.out.print("reader " + readerNum + " is done reading. ");
20. c = server.selesaiBaca();
21. }
22. }
23. private Database server; private int readerNum;
24. }
25. class Writer extends Thread {
26. public Writer(int w, Database db) { writerNum = w;
27. server = db;
28. }
29. public void run() { while (true) {
30. System.out.println("writer " + writerNum + " is sleeping.");
31. Database.tunggu();
32. System.out.println("writer " + writerNum + " wants to write.");
33. server.mulaiTulis();
34. System.out.println("writer " + writerNum + " is writing.");
35. Database.tunggu();
36. System.out.println("writer " + writerNum + " is done writing.");
37. server.selesaiTulis();
38. }
39. private Database server; private int writerNum;
```

```
39. private Database server; private int writerNum;
40. }
41. final class Semaphore { public Semaphore() {
42. value = 0;
43. }
44. public Semaphore(int v) { value = v;
45. }
46. public synchronized void tutup() { while (value <= 0) {
47. try {wait(); }
48. catch (InterruptedException e) {}
49. }
50. value--;
51. }
52. public synchronized void buka() {
53. ++value; notify();
54. }
55. private int value;
56. }
57. class Database {
58. public Database() { banyakReader = 0;
59. mutex = new Semaphore(1); db = new Semaphore(1);
60. }
61. public static void tunggu(){
62. int sleepTime = (int) (NAP_TIME * Math.random() ); try {Thread.sleep(sleepTime*1000); } catch(InterruptedException e) {}
63. }
64. public int mulaiBaca() { mutex.tutup();
65. ++banyakReader;
66. if (banyakReader == 1) { db.tutup();
67. }
68. mutex.buka();
69. return banyakReader;
70. }
71. public int selesaiBaca() { mutex.tutup();
72. --banyakReader;
73. if (banyakReader == 0) { db.buka();
74. mutex.buka();
75. System.out.println("Reader count = " + banyakReader); return banyakReader;
76. }
77. public void mulaiTulis() { db.tutup();
78. }
79. public void selesaiTulis() { db.buka();
80. }
81. private int banyakReader; Semaphore mutex; Semaphore db;
82. private static final int NAP_TIME = 15;
83. }
```

2. Dari program yang dijalankan rule apa yang dapat disimpulkan?

D. Tugas

1. Modifikasi program yang ada jika writers dipriotitaskan? Jelaskan rule nya
2. Bagaimana program yang ada jika readers yang di prioritaskan? Jelaskan rule nya

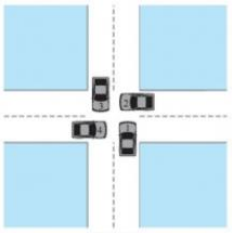
PERTEMUAN 8
IMPLEMENTASI PROGRAM UNTUK DEADLOCK

A. Tujuan Praktikum

- Praktikan dapat memahami kondisi deadlock
- Praktikan dapat memahami Dining-Philosophers problem dan Banker Algorithm

B. Materi dan Aktivitas Praktikum

Deadlock terjadi pada lingkungan multi-programming. Pada beberapa proses yang bekerja pada sejumlah sumber. Sebuah proses me-request recources, jika recources tidak ada maka proses dalam status "menunggu", kemudian ada proses lain juga yang dalam status "menunggu" nah kondisi disebut deadlock (semua proses dalam kondisi menunggu). Ilusrasi ditunjukkan seperti pada Gambar di bawah ini.



Ketiklah program di bawah ini dan jalankan!

```
#include <iostream>
#include <thread>
#include <exception>
using namespace std;

bool doNot = 1; // this bool will prevent func1 and func2 from doing their tasks
thread* fn1; // pointer to t1 thread (defined in main function)
thread* fn2; // pointer to t2 thread (defined in main function)
void func1(bool x)
{
    while(doNot) {;} // waiting for changing doNot to false (doing it in main function)
    ::fn2->join(); // trying to cause deadlock
}

void func2(bool x)
{
    while(doNot) {;} // waiting for changing doNot to false (doing it in main function)
    ::fn1->join(); // trying to cause deadlock
}


int main()
{
    // making threads for func1 and func2. I don't know why I've added arguments to func1 & func2
    thread t1(func1, 1);
    thread t2(func2, 1);
    // assigning values to pointers
    ::fn1 = &t1;
    ::fn2 = &t2;
    ::doNot = 0; // STARTING!
    t1.join();
    t2.join();
    cout << "Hello!"; // This code will never be done...
}
```

Apa yang terjadi? Jelaskan!

Ilustrasi Dining Philosophers Problem

Pada Dining Philosophers Problem, diketahui bahawa terdapat sejumlah (N) Philosophers yang hanya memiliki tiga status, berpikir, lapar, dan makan. Semua Philosopher berada di sebuah meja makan bundar yang ditata sehingga di depan setiap Philosopher ada sebuah piring berisi mie ayam. Diantara dua piring yang bersebelahan terdapat sebuah sumpit. Kemudian terdapat kejadian seperti ini:

Semua Philosopher akan berpikir selama waktu yang tidak tentu. Setelah berpikir lama, Philosopher akan merasa lapar. Pada saat lapar, ia berusaha untuk mengambil 2 buah sumpit yang ada di kanan dan di kirinya untuk makan. Dia mengambil sumpitnya satu per satu. Begitu ia mendapat sebuah sumpit, ia tidak akan melepaskannya.

Saat dia hanya berhasil mengambil kurang dari 2 sumpit, maka dia akan menunggu hingga diperoleh 2 sumpit dan langsung memakan mie ayamnya dan meletakkan kedua sumpitnya.

Kedua sumpit ini kemudian dapat digunakan oleh Philosophers yang lainnya.

Ketiklah program di bawah ini dan jalankan!
Kemudian masukkan input-an seperti di bawah ini!

```
------------------------------------------------------------
INPUT
Enter the total no. of philosophers: 5
How many are hungry : 3  Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5
------------------------------------------------------------
OUTPUT
1. One can eat at a time 2.Two can eat at a time
Enter your choice: 1 or 2
------------------------------------------------------------
```

Jelaskan apa yang terjadi?

```c
#include <stdio.h>
#include <conio.h>
#include <math.h>

int tph, philname[20], status[20], howhung, hu[20], cho;


int one()
{
    int pos=0, x, i;
    printf("\nAllow one philosopher to eat at any time\n");
    for(i=0;i<howhung; i++, pos++)
    {
        printf("\nP %d is granted to eat", philname[hu[pos]]);
        for(x=pos;x<howhung;x++)
        printf("\nP %d is waiting", philname[hu[x]]);
    }
}
int two()
{
    int i, j, s=0, t, r, x;
    printf("\n Allow two philosophers to eat at same time\n");
    for(i=0;i<howhung;i++)
    {
        for(j=i+1;j<howhung;j++)
        {
            if(abs(hu[i]-hu[j])>=1&&    abs(hu[i]-hu[j])!=4)
            {
                printf("\n\ncombination %d \n", (s+1));
                t=hu[i];
                r=hu[j];
                s++;
                printf("\nP %d and P %d are granted to eat", philname[hu[i]], philname[hu[j]]);
                for(x=0;x<howhung;x++)
                {
                    if((hu[x]!=t)&&(hu[x]!=r))
                    printf("\nP %d is waiting", philname[hu[x]]);
```

```
                }}}}}

int main()
{
    int i;
    printf("\n\nDINING PHILOSOPHER PROBLEM");
    printf("\nEnter the total no. of philosophers: ");
    scanf("%d",&tph);
    for(i=0;i<tph;i++)
    {
        philname[i] = (i+1);
        status[i]=1;
    }
    printf("How many are hungry : ");
    scanf("%d", &howhung);
    if(howhung==tph)
    {
        printf("\nAll are hungry..\nDead lock stage will occur");
        printf("\nExiting..");
    }
    else
    {
        printf("\nAll are hungry..\nDead lock stage will occur"); printf("\nExiting..");
        for(i=0;i<howhung;i++)
        {
            printf("Enter philosopher %d position: ",(i+1));
            scanf("%d", &hu[i]);
            status[hu[i]]=2;
        }
        do
        {
            printf("1.One can eat at a time\t2.Two can eat at a time\nEnter your choice:");
            scanf("%d", &cho); switch(cho)
            {
                case 1: one();
                break;
                case 2: two();
                break;
                default: printf("\nInvalid option..");
            }}while(1);}}
```

Algoritma Banker

Algoritma Banker berfungsi untuk mencegah terjadinya deadlock. Ilustrasi dari algoritma ini adalah menggunakan komponen banker, kredit, dan customer. Banker akan memutuskan apakah menyetujui atau menunda permohonan kredit oleh customer. Saat customer memohon kredit, maka permohonan kredit tersebut diperiksa oleh banker. Setiap customer memiliki batas kredit dan apabila seorang customer telah mencapai batas maksimum kredit, maka diasumsikan customer tersebut telah menyelesaikan semua permasalahan bisnisnya dan dapat mengembalikan semua pinjamannya kepada bank. Namun, customer juga dapat mengembalikan semua pinjamannya kepada bank sebelum mencapai batas kredit maksimum. Algoritma Banker ini terdiri atas algoritma Safety dan algoritma Resource Request.

C. Tugas

1. Buatlah program untuk mengimplementasikan algoritma Banker
2. Jelaskan proses yang terjadi saat program dijalankan

PERTEMUAN 9
MEMORY MANAGEMENT

A. Tujuan Praktikum

- Praktikan dapat memahami proses manajemen memori.
- Praktikan dapat men-setting parameter dalam manajemen memori.

B. Materi

Simulasi

Simulasi ini menunjukkan proses pada memori utama sesuai dengan sistem manajemen memori. Pada simulasi dapat menunjukkan versi proses sederhana dari **swap** dan perilaku **virtual memory**. Algoritma manajemen memori dibagi menjadi dua yaitu:

➢ *Contiguous memory management*, yaitu seluruh proses dialokasikan ke memori. Terdapat dua algoritma manajemen memori:
  ➔ Partisi tetap (f*ixed-sized paritions*), memori awalnya dibagi menjadi beberapa partisi dengan ukuran tetap (*user* yang membuat). Pada umumnya proses yang dialokasikan kedalam sebuah partisi memiliki ukuran yang lebih kecil daripada ukuran partisinya, dan oleh karena itu ada sebagian ukuran partisi yang tidak terpakai, yang disebut sebagai *internal fragmentation*.
  ➔ Partisi tidak tetap (*variable-sized partitions*), memori awalnya merupakan sebuah partisi besar. Lalu untuk setiap proses akan dialokasikan ukuran partisi yang sesuai dengan ukuran proses, dan setelah selesai maka partisi tersebut dapat digunakan ulang. Partisi yang telah selesai digunakan tersebut dapat disebut sebagai *external fragmentation*.
  Konsep lain yang menjadi perhatian adalah aturan alokasi memori, yaitu bagaimana sistem operasi memilih satu dari beberapa partisi yang tersedia. Beberapa diantaranya adalah:
    o *First fit*, memilih partisi pertama kali yang dapat memuat proses.
    o *Best fit,* memilih partisi yang dapat memuat proses secara optimal
    o *Worst fit*, memilih partisi yang dapat memuat proses dengan pemilihan paling buruk.

➢ *Non-contiguous memory management*, yaitu seluruh proses dibagi menjadi bagian-bagian yang dapat dialokasikan secara terpisah di memori, antara lain:
  ➔ *Pagination,* memori dan proses dibagi menjadi bagian-bagian yang berukuran sama (frame dan page), dimana page dari proses akan dialokasikan pada frame dari memori.
  ➔ *Segmentation,* proses dibagi menjadi bagian-bagian secara logical (misalnya code, data, atau stack) dan masing-masing akan dialokasikan ke partisi memori secara independen.

Main Memory

Proses-proses nantinya akan dialokasikan di sini, baik itu secara keseluruhan maupun per bagian page atau segmen. Secara definisi, pada setiap sistem manajemen memori, akan ada selalu terdapat satu proses pertama yang akan dialokasikan ke memori yang mendapatkan alamat dasar (*base addresses*), yaitu Operating System, yang ukurannya bervariasi dapat diantara 1, 2, atau 4 unit.

Alamat, proses, partisi yang tersedia maupun partisi yang sedang digunakan akan selalu terlihat, dan juga fragmentasi mengikuti kode warna yang memungkinkan pemahaman mahasiswa lebih cepat.

Warna	Keterangan
Abu-abu	Operating System
Putih	Memori belum digunakan
Titik biru	Fragmentasi internal
Titik merah muda	Fragmentasi eksternal
Warna lain	Proses (masing-masing)

Ukuran memori dari simulasi ini bervariasi antara 64 sampai dengan 256 unit.

*Klik kanan pada memori untuk menampilkan **pop up menu** yang memungkinkan untuk melakukan beberapa pilihan:*

- Saat simulasi berhenti (hanya untuk partisi tetap) >> untuk update dan delete partisi
- Saat waktu simulasi untuk setiap proses yang dialokasikan ke memori >> delete proses, swap proses, melihat address translation, melihat informasi detail (hanya untuk pagination dan segmentation) dan untuk melakukan defragment memori (hanya untuk partisi tidak tetap dan segmentation).

Process queue

Memuat proses yang tersedia untuk dialokasikan ke memori diurutkan berdasarkan waktu kedatangan proses. Setiap proses menunjukkan informasi yang relevan (PID, nama, dan durasi), disamping dari ukuran dan distribusinya, dimana setiap kotak bersesuaian ke satu unit space, yang dapat digunakan oleh satu alamat memori.
Pada pagination dan segmentation, proses-proses dibedakan menjadi komponen-komponennya, dan di-*highlight* oleh warna abu-abu untuk komponen yang belum dialokasikan ke memori tetapi dialokasikan ke swap area.

Saat simulasi berhenti, klik kanan pada sebarang proses untuk melihat pop up menu yang memungkinkan untuk meng-update dan delete.

Settings

➔ *Memory Size* (Ukuran memori, bervariasi antara 64 s/d 256 unit).
➔ *Operating System Size* (Ukuran memori untuk OS, bervariasi dari 1, 2 atau 4 unit).
Algorithm:
- *Fixed-sized partitions* (contiguous memory management) >> required untuk mempartisi seluruh memori.
- *Variable-sized partitions* (contiguous memory management).
- *Pagination* (non-contiguous memory management) >> harus menentukan ukuran page sistem (antara 1, 2 atau 4).

- *Segmentation* (non-contiguous memory management).
- Allocation policy, memilih algoritma pemilihan partisi, yaitu: *First fit, Best fit, Worst fit*

➔ Perubahan pada algoritma, ukuran memori, atau ukuran OS membutuhkan untuk me-restart simulasi dan menghapus seluruh proses yang ada.

Menambah proses

Proses hanya dapat ditambahkan saat simulasi berhenti, untuk setiap proses baru dapat ditambahkan dengan cara berikut:

➔ *PID >> proses identifier, bersifat automatically calculated dan incremental.*
➔ *Name (required) >> nama proses.*
➔ *Size >> ukuran proses (nilai: 1 s/d 64 unit).*
➔ *Duration >> proses dapat mempunyai durasi tak terhingga (diisi -1) atau terhingga (1-100), tapi tidak boleh 0.*
➔ *Color >> warna yang akan digunakan untuk menandai proses yang dimaksud.*

Tambahan untuk pagination:

- Page table >> jumlah page yang tergantung pada ukuran proses, untuk setiap page mahasiswa harus menentukan apakah sudah dialokasikan ke memori secara langsung (intially allocated) atau tidak (swapped).

Tambahan untuk segmentasi

- Segment table >> setiap proses mempunyai tiga segmen: code, data, dan stack. Untuk setiap segmen, mahasiswa harus menentukan ukuran dan apakah sudah dialokasikan ke memori (initially allocated) atau tidak (swapped). Ukuran dari jumlahan segmen harus sama dengan ukuran dari proses.

Data dan statistik

Menunjukkan informasi alokasi memori pada setiap waktu simulasi. Informasi ini beragam antara satu algoritma dengan algoritma yang lain.

Contiguous memory management, tabel dengan informasi berikut:

- *Direction,* alamat partisi mula-mula.
- *Size,* ukuran dari partisi.

(jika pada partisi terdapat sebuah proses)

- *PID,* proses ID (warna background menunjukkan warna proses).
- *Name,* nama proses.
- *Size,* ukuran proses.
- *Duration,* durasi proses.

Pada pagination, tabel dengan informasi berikut:

- *Address,* alamat frame mula-mula.
- *Frame,* nomor frame.

(jika partisi sedang digunakan)

- *PID,* proses ID (warna background menunjukkan warna proses).
- *Page,* proses yang menggunakan frame yang bersesuaian.
- *Name,* nama proses.
- *Dimensions,* total proses.
- *Duration,* durasi proses.

Pada segmentation, tabel dengan informasi berikut:
- *Address,* alamat frame mula-mula.
- *Size,* ukuran partisi.

(jika pada partisi terdapat sebuah proses)
- *PID,* proses ID (warna background menunjukkan warna proses).
- *Segment,* segmen proses, yaitu: code, data atau stack.
- *Name,* nama proses.
- *Size,* ukuran proses.
- *Duration,* durasi proses.

Tabel page dan tabel segment

Menunjukkan informasi alokasi memori dari sebuah proses pada non-contiguous memory management.

Pada pagination, konten tabel adalah sebagai berikut:
- *Page,* proses yang menggunakan frame yang bersesuaian.
- *Frame,* nomor frame.
- *Valid,* bit valid dari page (v valid, i invalid). Sebuah page dikatakan tidak valid ketika page tersebut tidak dialokasikan ke memori.

Pada segmentation, konten tabel adalah sebagai berikut:
- *Segment,* segmen dari proses: code, data, atau stack.
- *Size,* ukuran dari segmen.
- *Address,* alamat frame mula-mula.
- *Valid,* bit valid dari segmen (v valid, i invalid). Sebuah segmen dikatakan tidak valid ketika segmen tersebut tidak dialokasikan ke memori.

Membuat partisi (untuk partisi berukuran tetap)

Menentukan alamat mula-mula dan ukuran partisi.
Partisi tidak bisa saling overlap, dan tidak bisa melebihi batas akhir memori.

Defragment memori (untuk partisi berukuran tidak tetap dan segmentation)

Paritisi berukuran tidak tetap dan segmentation mempunyai kesamaan pada partisi yang dibuat secara dinamis dan diukur secara otomatis untuk mengakomodasi proses atau segmen secara tepat. Operating System memilih sebuah partisi yang sama atau sedikit lebih besar dan dibagi menjadi dua bagian, satu untuk mengalokasikan proses atau segmen, satu untuk yang masih tersedia.

Setelah selesai digunakan (proses sudah selesai atau swapped out), partisi tersebut menjadi kosong dan berubah menjadi lebih kecil (memorinya ter-degradasi).

Proses defragmentasi dapat menanggulangi untuk me-reorganize memori, mengatur semua proses menjadi menggunakan memori dengan alamat rendah (lower memory addresses) dan menggabungkan seluruh partisi kosong menjadi satu partisi kosong yang besar.

Ini adalah tanggung jawab dari OS, dan ini diserahkan kepada mahasiswa untuk simulasi (pop up menu dari memori).

Swap

Melakukan swap dapat memungkinkan kita untuk menjalankan proses yang lebih banyak dari kapasitas memori utama, dan biasanya diimplementasikan pada memori sekunder.

Hal ini muncul atas dasar untuk mempunyai space ekstra untuk memindah proses atau bagian darinya (page atau segmen) yang sedang kurang aktif dan mengalokasikan ulang ke memori saat sedang dibutuhkan.

Simulasi digunakan untuk menyederhanakan proses ini, mensimulasikan swap in dan swap out antar memori dan swap dilakukan oleh mahasiswa secara manual.

Swap out, dari pop up menu memori (klik kanan pada memori). Melakukan swap out sebuah proses atau bagian darinya.

Swap in, dari pop up menu memori (klik kanan pada memori). Mencoba untuk melakukan swap in pada sebuah proses atau bagian darinya kepada memori, jika memori penuh akan menghasilkan sebuah error dan tidak akan bisa swap in.

Algoritma pagination dan segmentation memungkinkan untuk melakukan swap out page atau segmen pada awal (initial). Area ini tidak dibatasi pada jumlah atau ukuran dari proses yang dimuat.

Error pada simulasi

Memory full, karena sedang memproses antrian proses pada simulasi, memori mengalokasikan untuk proses dan selesai mengalokasikannya. Saat proses tidak bisa dialokasikan karena kurang space-nya, maka menyebabkan tidak dapat dilanjutkan ke proses selanjutnya. Memori dapat dilepaskan (released) secara otomatis (karena proses sudah selesai menggunakannya) atau oleh campur tangan user (delete atau swap out proses).

Memory not fully partitioned, algoritma paritisi berukuran tetap (fixed-size partition) membutuhkan semua memori untuk dipartisi (alamat akhir satu partisi adalah alamat awal partisi yang lain).

C. Aktivitas Praktikum

Contiguous memory management dengan menggunakan partisi berukuran tetap (fixed-size partition) dan aturan first fit

Memori dibagi menjadi partisi-partisi berukuran tetap, lalu seluruh proses akan dialokasikan pada satu dari partisi yang tersedia. Pada aturan first fit, partisi pertama yang dapat memuat proses akan dipilih.

Lakukan Setting berikut pada OS SIM :

Partisi		Proses	
Alamat awal	Unit ukuran	Proses	Unit ukuran

@4	10	P1	5
@14	5	P2	10
@19	4	P3	3
@23	15		
@38	10		
@48	14		

Amati prosesnya saat simulasi dijankan!

Proses akan dialokasikan sesuai dengan urutan pada tabel Keterangan: Aturan first fit dapat menyelesaikan masalah secara cepat, tapi tidak efisien, partisi dapat menjadi lebih besar dari proses dan akan menimbulkan fragmentasi internal.

Contiguous memory management dengan menggunakan partisi berukuran tetap (fixed-size partition) dan aturan best fit

Memori dibagi menjadi partisi-partisi berukuran tetap, lalu seluruh proses akan dialokasikan pada satu dari partisi yang tersedia. Pada aturan best fit, partisi terbaik (yang ukurannya sama atau hampir sama) yang dapat memuat proses akan dipilih.

Lakukan Setting berikut pada OS SIM :

Partisi		Proses	
Alamat awal	Unit ukuran	Proses	Unit ukuran
@4	10	P1	5
@14	5	P2	10
@19	4	P3	3

@23	15	
@38	10	
@48	14	

Amati prosesnya saat simulasi dijankan!

Proses akan dialokasikan sesuai dengan urutan pada tabel Keterangan: Aturan best fit meminimalkan fragmentasi internal.

Contiguous memory management dengan menggunakan partisi berukuran tidak tetap (variable-size partition) >> defragmentasi

Memori pertama tidak dipartisi, lalu partisi akan dibuat saat proses dialokasikan dan mempunyai ukuran yang sama dengan ukuran proses. Selanjutnya partisi akan dibebaskan (saat proses sudah selesai menggunakan memori) dan digunakan oleh proses lain. Jika ukuran proses lebih kecil dari ukuran partisi, maka partisi akan dibagi menjadi dua bagian, bagian pertama sama besarnya dengan proses dan bagian kedua adalah sisanya.

Lakukan Setting berikut pada OS SIM :

Proses	Unit ukuran	Durasi
P1	5	5
P2	8	4
P3	3	3
P4	15	∞
P5	2	1
P6	20	∞
P7	3	∞
P8	10	∞

Amati prosesnya saat simulasi dijankan!

Proses akan dialokasikan sesuai dengan urutan pada tabel. Keterangan:

- *Proses P7 dialokasikan pada memori setelah partisi dibebaskan oleh sebuah proses yang sudah selesai.*
- *Proses terakhir tidak dapat dialokasikan ke memori karena tidak ada space yang mencukupi pada memori untuk mengakomodasinya. Padahal pada keseluruhan partisi yang kosong cukup untuk mengakomodasi proses terakhir.*
- *Jika memori sudah di-defragment (klik kanan pada memori, pop up menu), semua space yang kosong akan digabungkan menjadi satu partisi yang besar dan proses terakhir dapat dimuat ke memori.*

Contiguous memory management dengan menggunakan partisi berukuran tidak tetap (variable-size partition) >> swap

Memori pertama tidak dipartisi, lalu partisi akan dibuat saat proses dialokasikan dan mempunyai ukuran yang sama dengan ukuran proses. Selanjutnya partisi akan dibebaskan (saat proses sudah selesai menggunakan memori) dan digunakan oleh proses lain. Jika ukuran proses lebih kecil dari ukuran partisi, maka partisi akan dibagi menjadi dua bagian, bagian pertama sama besarnya dengan proses dan bagian kedua adalah sisanya.
Memori mempunyai ukuran yang berhingga, hanya beberapa proses saja yang bisa dimuat di memori pada saat yang sama, tetapi space swap memungkinkan kita untuk menyimpan beberapa proses yang sedang kurang aktif, sehingga space pada memori dapat dialokasikan untuk proses yang lain yang lebih aktif. Proses yang telah di swap out akan dialokasikan kembali ke memori saat dibutuhkan.

Lakukan Setting berikut pada OS SIM :

Proses	Unit ukuran
P1	5
P2	8
P3	3
P4	***15***
P5	2
P6	20
P7	3

P8	10

Amati prosesnya saat simulasi dijankan!

Proses akan dialokasikan sesuai dengan urutan pada tabel. Keterangan:

- *Tidak ada fragmentasi internal, partisinya selalu sesuai dengan ukuran proses.*
- *Dua proses terakhir (P7 dan P8) tidak dapat dimuat ke memori karena memori penuh.*
- *Jika beberapa proses di-swap out, akan membebaskan beberapa space pada memori untuk dialokasikan untuk proses yang lain. Contohnya P2 dan P4 (klik kanan pada proses, pop up menu).*
- *Untuk mengalokasikan kembali proses yang sudah di-swap out, maka pada memori harus ada partisi yang dapat memuat proses tsb, contohnya adalah swap P6. Lalu P2 dan P4 dapat di- swap in (klik kanan pada memori, pop up menu).*

Pagination (ukuran page 2 unit)

Proses akan dibagi menjadi beberapa page dengan ukuran tetap (contohnya adalah 2 unit), memori lalu dibagi menjadi beberapa frame berukuran sama. Page dari proses lalu dialokasikan pada frame memori yang kosong.

Lakukan Setting berikut pada OS SIM :

Proses	Ukuran	Durasi
P1	5 unit (3 page)	4
P2	4 unit (2 page)	∞
P3	11 unit (6 page)	2
P4	5 unit (3 page)	∞
P5	7 unit (4 page)	∞
P6	3 unit (2 page)	∞

Amati prosesnya saat simulasi dijankan!

Proses akan dialokasikan sesuai dengan urutan pada tabel. Keterangan:

- *Page terkahir dari sebuah proses tidak selalu harus penuh memenuhi page, ini dapat membuat fragmentasi internal.*
- *Pada pagination, aturan-aturan pemilihan frame untuk page tidak berlaku, karena page dan frame*

mempunyai ukuran yang sama.

Segmentation (alokasi parsial)

Proses dibagi menjadi beberapa segmen, tiap segmen bisa mempunyai ukuran yang berbeda. Segmen ini akan dialokasikan ke memori secara independen, lalu partisi akan dibuat untuk menampung segmen tersebut dengan ukuran yang sama. Pembagian ini adalah secara fungsional dan berdasarkan pada logical structure dari proses tersebut (data, stack, code, dll).

Tidak semua segmen harus dimuat pada memori agar proses dapat berjalan, selama sebuah segmen tidak dipakai maka dapat di-swap out.

Lakukan Setting berikut pada OS SIM :

Proses	Ukuran total	Segmen	Ukuran (unit)	Dialokasikan diawal
P1	20	Code	2	Yes
		Data	10	Yes
		Stack	8	No
P2	20	Code	5	Yes
		Data	14	No
		Stack	1	Yes
P3	40	Code	10	Yes
		Data	20	Yes
		Stack	10	Yes

Amati prosesnya saat simulasi dijankan!

Proses akan dialokasikan sesuai dengan urutan pada tabel. Ukuran memori adalah 64 unit, OS mendapatkan 4 unit.

Keterangan:

- *Tidak ada fragmentasi internal, partisi selalu mempunyai ukuran yang sama dengan ukuran segmen.*
- *Ukuran total adalah 80 unit dan total ukuran memori melebihi space pada memori (60 unit), tetapi semua proses dapat dialokasikan baik secara total maupun parsial, oleh karena itu, semua proses dapat bekerja dengan optimal.*

D. Tugas

1. Jawab latihan OS-SIM fixed sized partition
2. Jawab latihan OS-SIM pagination

PERTEMUAN 10
DISK MANAGEMENT

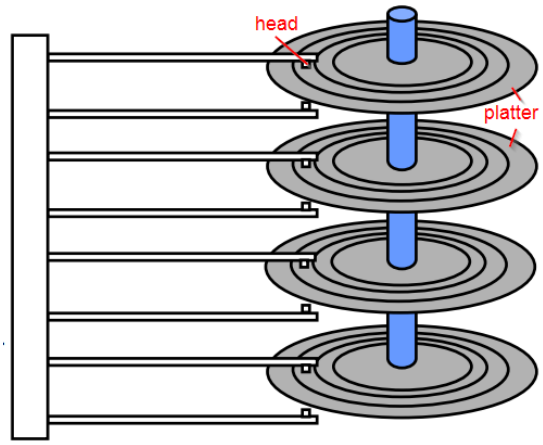
A. Tujuan Praktikum

- Praktikan dapat memahami manajemen disk
- Praktikan dapat memahami proses penjadwalan disk

B. Materi

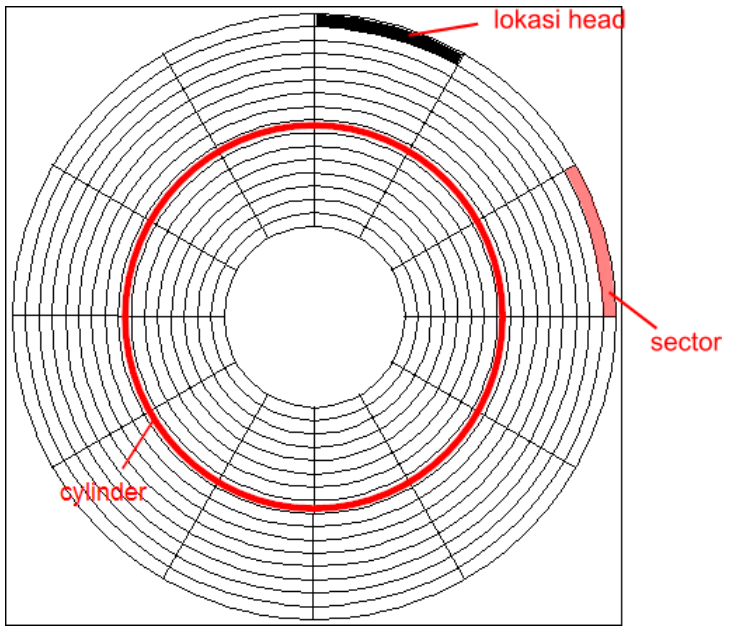
Struktur Disk

Hard Disk terbuat dari perangkat magnetik yang digunakan untuk menyimpan data. Disk dapat tersusun atas beberapa *platter* dengan masing-masing *head* disk yang berfungsi untuk membaca data di platter tersebut. Susunan disk tersebut diperlihatkan pada gambar 10.1.



Gambar 10.1 *platter* dan *head* pada hard disk

Pada permukaan platters, disk dibagi dalam beberapa *sector* yang dikelompokkan dalam *tracks*. Ketika sistem operasi memberikan perintah membaca atau menulis ke hard disk, maka *platter* akan diputar dan dibaca oleh *head* disk. Head disk dapat bergerak tegak lurus terhadap *tracks*. Kumpulan *tracks* yang berada pada *head* disk yang sama disebut sebagai *cylinder*. Gambar 10.2 menunjukkan struktur disk di satu *platter*. Pada gambar tersebut, ditunjukkan posisi head berada pada *sector* pertama. Penomoran *sector* dimulai dari *cylinder* terluar dalam satu *platter*, lalu berputar hingga melingkupi satu *cylinder*, kemudian dilanjutkan ke dalam *cylinder* selanjutnya. Gambar 10.2 menunjukkan contoh struktur disk yang ada di simulator OS SIM. Pada OS SIM, disk direpresentasikan dengan 16 *cylinder* dengan masing-masing 12 *sector* di setiap *cylinder*-nya. Posisi *head* disk ditunjukkan oleh warna hitam di *sector* pertama berada di *cylinder* terluar pada gambar 10.2.



Gambar 10.2 struktur disk

Simulasi dengan OS SIM

Pada saat komputer berjalan, sistem pengoperasian akan banyak melakukan operasi baca dan tulis ke disk. Untuk memastikan operasi berjalan dengan efisien, diperlukan algoritma untuk menjadwalkan urutan pengambilan data. Algoritma ini bertujuan untuk mengefisienkan pergerakan *head* dan perputaran *platter*, sehingga didapatkan waktu respon yang lebih cepat.

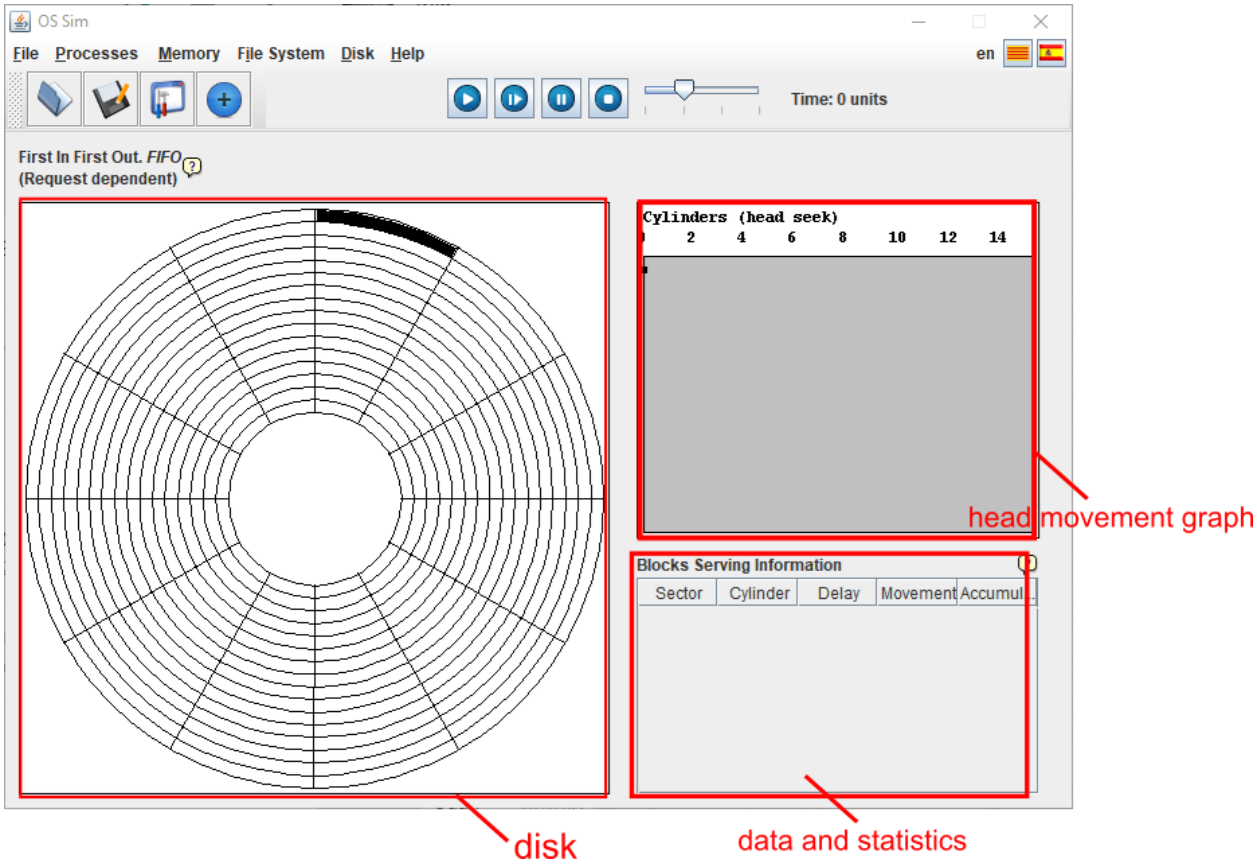
Algoritma Penjadwalan

Dalam simulasi OS SIM, tersedia beberapa algoritma penjadwalan sebagai berikut:

- *FIFO* (*Fist In First Out*). Penjadwalan ini cukup sederhana, melayani permintaan sesuai dengan urutan kedatangan. *Head* selalu bergerak mencari permintaan berikutnya berdasarkan urutan kedatangan di mana pun itu. Pada algoritma ini, prioritas pengambilan data tidak didukung.
- *LIFO* (*Last In First Out*). Tidak seperti FIFO, algoritma ini melayani permintaan dalam urutan yang berkebalikan dengan kedatangan, permintaan yang terakhir datang yang akan dilayani pertama kali. Hal ini menyebabkan kemungkinan permintaan pertama tidak akan pernah dilayani jika ada permintaan baru yang tiba secara terus-menerus.
- *STF* (*Shortest (Seek) Time First*). Algoritma ini melayani permintaan terdekat dari posisi *head*. Penjadwalan seperti ini dapat meminimalkan gerakan *head*, namun juga dapat menyebabkan kemungkinan tidak dilayaninya permintaan yang posisinya jauh dari *head*.
- *SCAN.* Algoritma ini sering juga disebut sebagai "*elevator*". *Head* bergerak dari *cylinder* terluar menuju ke dalam. Ketika sudah mencapai *cylinder* terdalam, *head* berubah arah menuju *cylinder* terluar. Dalam perjalanannya, *head* mengambil data sesuai dengan daftar permintaan yang ada.
- *C-SCAN. Circular* SCAN memiliki kemiripan dengan algoritma SCAN. Perbedaannya adalah ketika *head* mencapai *cylinder* terdalam dari disk, head langsung dimulai lagi dari head terluar. Dengan kata lain, *head* pada algoritma ini hanya bergerak ke satu arah saja. Seperti algoritma SCAN, saat berjalan, *head* mengambil data sesuai dengan daftar permintaan yang ada.
- LOOK. Algoritma ini adalah variasi untuk meningkatkan performa SCAN. Perbedaannya adalah bahwa *head* hanya bergerak hingga ke permintaan terakhir, namun tidak sampai *cylinder* terakhir.

Algoritma mendeteksi jika sudah tidak ada lagi permintaan di depan arah yang dituju, maka head akan digerakkan berubah arah.

- *C-LOOK* (*Circular* LOOK). Algoritma ini memiliki kemiripan dengan algoritma LOOK. Perbedaannya, ketika head bergerak ke posisi permintaan terakhir di satu arah, algoritma ini akan menggerakkan head ke posisi awal dan bergerak ke satu arah yang sama lagi.



Gambar 10.3 Tampilan antarmuka OS SIM

Gambar 10.3 menunjukkan tampilan antarmuka OS SIM untuk penjadwalan disk. Berikut adalah penjelasan untuk masing-masing bagian dari antarmuka OS SIM tersebut:

- Disk
  Tampilan ini menunjukkan pembagian *cylinder* dan *sector* dari disk. Sector 0 (sebagai titik awal) berada pada cylinder terluar, sisi kanan atas dari garis vertikal. Posisi *head* ditunjukkan pada tanda hitam pada *sector* 0 tersebut. Permintaan (*request*) data akan ditunjukkan dengan warna yang berbeda, sesuai dengan permintaannya. Ketika simulasi dalam posisi berhenti, gunakan klik kanan pada *request* mana pun untuk membuka menu *pop up* untuk memodifikasi atau menghapus data.
- *Head movement graph*
  Graf menunjukkan pergerakan head ke *cylinder* sesuai dengan urutan melayani permintaan. Sumbu x menunjukkan nomor cylinder, sedangkan sumbu y menunjukkan waktu. Setiap *request* yang sudah dilayani ditandai dengan tanda titik (dengan warna yang sesuai) menunjukkan lokasi *cylinder* di *sector* mana dan kapan *request* dilayani. Titik-titik tersebut dihubungkan dengan garis

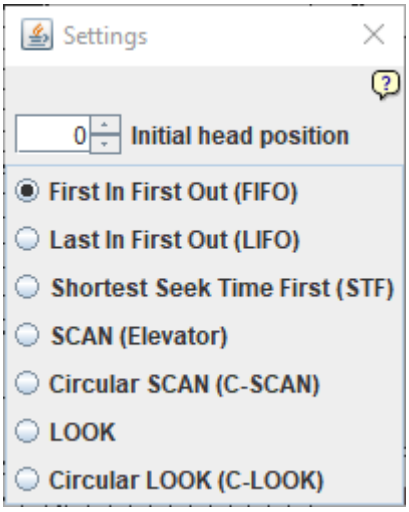
yang merepresentasikan pergerakan *head* pada *cylinder*.

- Data dan statistics

  Bagian ini menunjukkan semua request, dengan penjelasan kolom sebagai berikut:
- *Sector*, nomor *sector* yang diminta
- *Cylinder*, lokasi *sector* yang diminta
- *Delay*, waktu yang diperlukan oleh *request* untuk tiba.
- *Movement*, hanya untuk *request* yang sudah dilayani, menunjukkan pergerakan *head* sejak *request* terakhir.
- *Accumulated*, jumlah pergerakan *cylinder* sejak simulasi dimulai.

  Request dapat berada di salah satu status sebagai berikut:
- *Served*, *request* telah dilayani
- *Waiting, request* berada dalam antrian, menunggu untuk dilayani
- *Delayed, request* baru akan datang kemudian

  Request yang sudah dilayani akan dimunculkan terlebih dahulu, diikuti dengan *request* yang sedang dalam status *waiting*, kemudian baru *request* yang dalam status *delayed*.



Gambar 10.4 Tampilan antarmuka konfigurasi simulasi

Gambar 10.4 menunjukkan tampilan antarmuka konfigurasi simulasi. Berikut adalah penjelasan dari masing-masing bagian:

- *Initial head position*, memungkinkan untuk menentukan lokasi awal dari *head*, berada pada kisaran *sector* 0 hingga 192.
- Pilihan algoritma, yaitu
  - First in First Out (FIFO)
  - Last in First Out (LIFO)
  - Shortest Seek Time First (STF)
  - SCAN (Elevator)
  - Circular SCAN (C-SCAN)
  - LOOK
  - Circular LOOK (C-LOOK)

Menambahkan Permintaan (Request)

Permintaan hanya dapat ditambahkan ketika simulasi berada dalam posisi berhenti. Dalam pembuatan permintaan baru informasi berikut harus dimasukkan:

- *Sector*, nomor *sector* di mana permintaan dihasilkan (Nilai: 1 - 192).
- *Delay*, waktu jeda permintaan untuk memasuki antrian. Jika diisi dengan 0, maka permintaan langsung dimasukkan ke antrian, selain itu maka permintaan akan ditambahkan kemudian sesuai dengan jeda waktunya. (Kisaran: 1-100).
- Color,warna untuk menunjukkan permintaan.

Selain itu, sistem menghitung dan menunjukkan *cylinder* yang sesuai dengan *sector* yang diminta.

Operasi Lain

Simulasi yang dilakukan dapat disimpan dan dibuka kembali saat simulasi berada pada posisi berhenti. Selain itu, kecepatan jalannya simulasi dapat dikendalikan, apakah dengan berjalan secara normal, langkah per langkah (satu satuan waktu untuk setiap langkah), menghentikan simulasi, atau mengulang kembali simulasi dari awal. Selain itu kecepatan jalannya simulasi dapat diatur dari tingkat kecepatan 0,5 detik hingga 2 detik per satuan waktu.

FIFO

Permintaan untuk *sector* disk disesuaikan dengan urutan kedatangan.

Informasi: geometri hard disk adalah salah satu piring, 16 silinder, 12 sektor per silinder, 192 sektor total.

Sector	Cylinder
100	8
3	0
150	12
23	1
120	10

Head awalnya diposisikan di sektor 0 (Cylinder 0). Semua permintaan tiba di urutan tabel pada saat 0.

Komentar:

- Head selalu bergerak mencari permintaan berikutnya
- Head melewati *cylinder* yang berisi permintaan, tetapi tidak dilayani karena mereka bukan yang akan dicari

STF scheduling, contoh Starvation

Permintaan paling dekat dengan head dilayani pertama, hal ini dapat menyebabkan bahwa

sementara permintaan tiba dekat dengan head, yang lebih jauh tidak dilayani (*starvation*)
Informasi: geometri harddisk adalah salah satu piring, 16 silinder, 12 sektor per silinder, 192 sektor total.

Sector	Cylinder	Delay
170	14	0
150	12	0
120	10	0
40	3	1
20	1	2
35	2	5
2	0	6
10	0	8
50	4	15
0	0	15

Head awalnya diposisikan di sektor 50 (Cylinder 4). Semua permintaan tiba di urutan tabel, menurut delay yang ditunjukkan.
Komentar:

- Permintaan pertama berada di silinder bagian dalam, dan tidak melayani sebagai permintaan baru yang datang dekat dengan head
- Situasi ini bisa dilanjutkan tanpa batas

Elevator (SCAN)

Gerakan head tidak tergantung pada permintaan, scan permintaan di kedua arah, permintaan dilayani seperti yang ditemukan.
Informasi: geometri harddisk adalah salah satu piring, 16 silinder, 12 sektor per silinder, 192 sektor total.

Sector	Cylinder
100	8
3	0
150	12

23	1
120	10

Head awalnya diposisikan di sektor 90 (Cylinder 7), bergerak maju silinder terdalam. Semua permintaan tiba di urutan meja pada saat 0.
Komentar:

- Permintaan terdalam di silinder 12, tapi head terus lanjut sampai silinder terakhir
- Pada saat mencapai silinder terakhir, seperti yang baru saja dilayani terdalam, dan mereka yang tetap berada di silinder terluar, tetapi head berubah arah dan bergerak dari dalam ke luar.

Circular SCAN

Gerakan head tidak tergantung pada permintaan, scan dalam satu arah, dari luar ke silinder bagian dalam, permintaan dilayani seperti yang ditemukan.
Informasi: geometri harddisk adalah salah satu piring, 16 silinder, 12 sektor per silinder, 192 sektor total.

Sector	Cylinder
100	8
3	0
150	12
23	1
120	10

Head awalnya diposisikan di sektor 90 (Cylinder 7), bergerak maju silinder terdalam. Semua permintaan tiba di urutan meja pada saat 0.
Komentar:

- Permintaan terdalam dalam silinder 12, tapi head terus sampai silinder terakhir.
- (Peningkatan SCAN) Pada saat mencapai silinder terakhir, seperti yang baru saja menjabat permintaan terdalam, dan orang-orang yang tetap berada di silinder terluar, sehingga kepala bergerak langsung ke silinder pertama dan mulai lagi

Circular LOOK

Gerakan head tidak tergantung pada permintaan, dan scan dalam satu arah, dari luar ke silinder bagian dalam, dan hanya mencapai permintaan terdalam, permintaan dilayani seperti yang ditemukan.
Informasi: geometri harddisk adalah salah satu piring, 16 silinder, 12 sektor per silinder, 192 sektor total.

Sect	Cylind

or	er
100	8
3	0
150	12
23	1
120	10

Head awalnya diposisikan di sektor 90 (Cylinder 7), bergerak maju silinder terdalam. Semua permintaan tiba di urutan meja pada saat 0.

Komentar:

- (Peningkatan SCAN) Permintaan terdalam di silinder 12, ketika disajikan, head tidak berlanjut sampai silinder terakhir
- (Peningkatan SCAN) Dalam saat mencapai permintaan terakhir, seperti yang baru saja dilayani terdalam, dan orang-orang yang tetap berada di silinder terluar, sehingga head bergerak langsung ke permintaan terluar dan dimulai lagi

C. Aktivitas Praktikum

Simulasi dengan Program

FIFO

Berikut adalah contoh program dengan menggunakan bahasa C++:

```
#include<iostream>
using namespace std;

int main()
{
    int queue[100],n,head,i,j,k,seek=0,diff;
    float avg;
    // clrscr();
    cout << "*** FCFS Disk Scheduling Algorithm ***\n";
    cout << "Enter the size of Queue\t";
    cin >> n;
    cout << "Enter the Queue\t";
    for(i=1;i<=n;i++)
    {
            cin >> queue[i];
    }
    cout << "Enter the initial head position\t";
    cin >> head;
    queue[0]=head;
    cout << "\n";
    for(j=0;j<=n-1;j++)
    {
            diff=abs(queue[j+1]-queue[j]);
            seek+=diff;
```

```
            cout << "Move from " << queue[j] <<" to " << queue[j+1] << " with Seek "<< diff << "\n";
        }
        cout << "\nTotal Seek Time is "<< seek << "\t";
        avg=seek/(float)n;
        cout << "\nAverage Seek Time is "<< avg << "\t";
        return 0;
    }
```

Program di atas menggunakan array untuk menampung lokasi permintaan data. Selanjutnya, *head* digerakkan sesuai dengan urutan kedatangan. Jumlah jarak dan rata-rata pergerakan *head* dihitung untuk ditampilkan.

D. Tugas

1) Buatlah implementasi program untuk simulasi algoritma penjadwalan disk SCAN
2) Buatlah implementasi program untuk simulasi algoritma penjadwalan disk C-LOOK

PERTEMUAN 11
MANAJEMEN FILE SYSTEM

A. Tujuan Praktikum

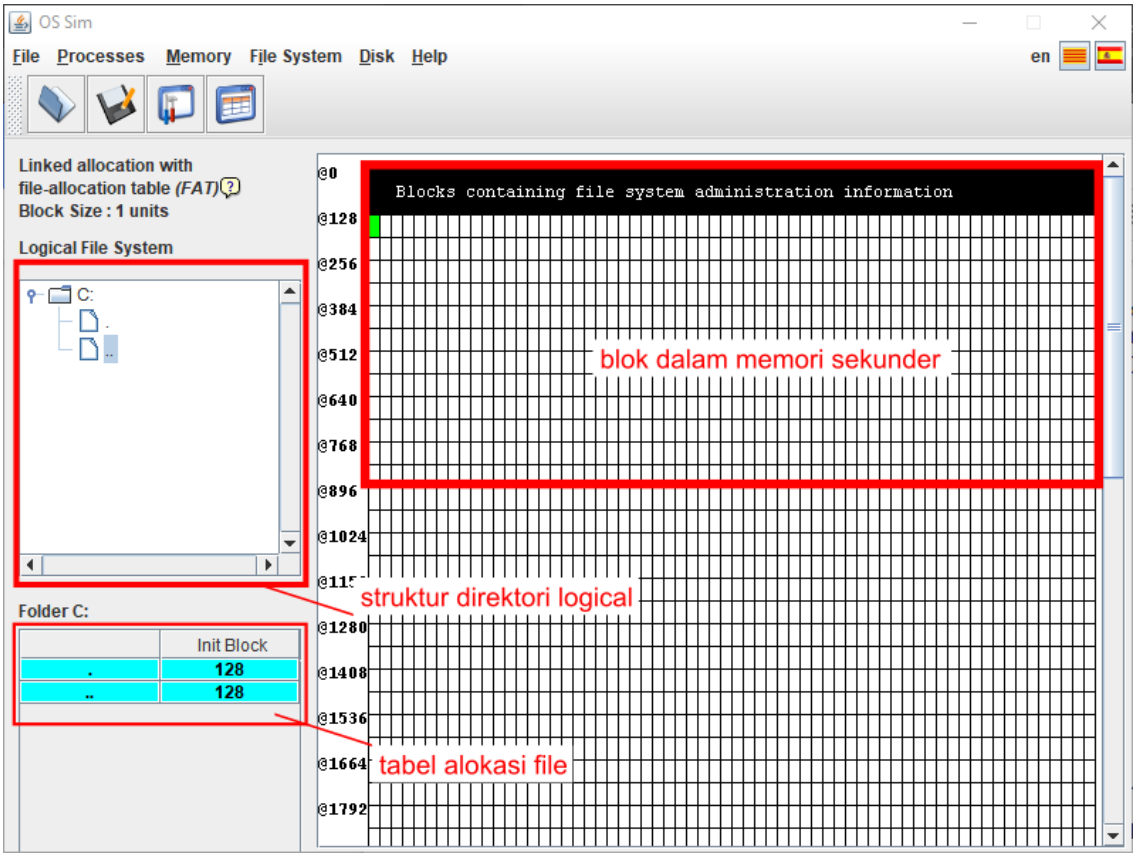
- Praktikan dapat memahami manajemen file system

B. Materi

File System

File system mengatur hubungan antara *logical space* yang dilihat pengguna seperti *file*, direktori, atau tautan dengan struktur sesuai lokasi fisik nyata dari objek tersebut di memori sekunder. Unit terkecil dari *file system* adalah blok. Dalam penyimpanannya, sebuah file atau direktori dibagi menjadi blok dengan ukuran yang sama. Manajemen *file system* memiliki fungsi untuk mengalokasikan blok-blok objek ke dalam blok yang tersedia pada memori sekunder.

Antarmuka OS SIM untuk simulasi manajemen file system diperlihatkan pada Gambar 11.1. Struktur direktori *logical* menunjukkan tampilan struktur direktori dari sisi pengguna. Pada struktur direktori tersebut, dapat dilakukan operasi terkait *file* yang biasa ditemui. Pada tabel alokasi file ditunjukkan daftar *file* dan direktori beserta lokasi blok awal atau i-node untuk *file* dan direktori tersebut. Pada sisi kanan, ditampilkan disk yang direpresentasikan dalam blok. 128 blok di awal disk digunakan untuk menyimpan informasi tentang administrasi *file system*, ditunjukkan dengan warna hitam.



Gambar 11.1 Antarmuka simulasi manajemen *file system*

Logical File System

Pada bagian *logical file system* atau struktur direktori logical pada Gambar 11.1, ditampilkan struktur

bentuk pohon yang dilihat oleh pengguna sistem pengoperasian. *Root* pada *file system* ini dapat berbeda sesuai metode alokasi file yang diimplementasikan. Untuk FAT/MS-DOS, rootnya adalah "C:\", sedangkan untuk UNIX, rootnya adalah "/". Pada bagian ini, dapat dilakukan klik kanan untuk melakukan operasi-operasi yang umum dilakukan oleh pengguna terkait pengelolaan file, yaitu:

- Menambahkan file pada direktori yang dipilih
- Menambahkan sub direktori pada direktori yang dipilih
- Menambahkan tautan pada direktori yang dipilih
- Mengubah objek
- Menghapus objek

File System Physical (Device)

Pada bagian sisi kanan Gambar 11.1, ditunjukkan bentuk fisik dari *file system* yang dibagi menjadi beberapa blok beserta alamatnya. Blok-blok tersebut bisa memiliki berbagai macam warna sesuai dengan statusnya, sebagai berikut:

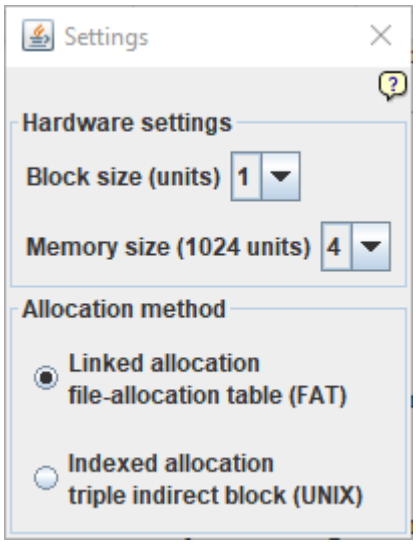
Colors:	**Block use**
black	Administration (File System)
white	Available
Green	Folder
Blue	Soft link
[Other colors]	Occupied by a file *UNIX "I" means it is an indirect block*

Allocation Table

Bagian sisi kiri dari Gambar 11.1 menunjukkan informasi bagaimana sebuah *file* atau direktori disimpan/diindeks. Setiap objek di sini diberikan warna sesuai dengan jenis objeknya. Setiap objek diberikan referensi ke sistem *file* fisik. Untuk metode alokasi FAT, referensi yang digunakan adalah lokasi blok awal. Pada metode alokasi UNIX, referensi yang digunakan adalah i-node dari objek tersebut.

Pengaturan

Gambar 11.2 memperlihatkan pengaturan yang dapat dilakukan pada OS SIM. Manajemen *file system* dapat diubah ukuran bloknya (bernilai 1, 2, atau 4). Ukuran dari memori sekunder/*disk* juga dapat diubah (bernilai 4, 6, atau 8 dalam satuan 1024 unit/Mega unit). Metode alokasi yang disediakan oleh OS SIM juga dapat dipilih, apakah FAT atau UNIX. Setiap adanya perubahan metode alokasi yang digunakan, diperlukan restart simulasi dan menghapus semua objek yang sudah ada.



Gambar 11.2 Pengaturan OS SIM untuk manajemen *file system*

Menambah Sebuah Direktori

Direktori ditambahkan ke folder yang sedang dipilih, di mana kita bisa memberikan informasi nama direktori yang akan dibuat.

Menambah Sebuah Tautan

Tautan merupakan objek yang menunjuk ke direktori yang sudah ada. Untuk menambahkan sebuah tautan, diperlukan informasi sebagai berikut:

- Nama tautan dengan format teks
- Tipe tautan, apakah *soft link* atau *hard link*. Untuk MS-DOS, yang diperbolehkan hanya soft link
- Target tautan, menunjuk ke direktori yang sudah ada sebelumnya

Informasi Rinci dari Sebuah Objek

Pada bagian *Allocation Table*/Tabel Alokasi (sisi kiri bawah), diberikan fasilitas untuk melihat informasi rinci dari sebuah objek sesuai dengan metode alokasi yang dipilih.
Dari tampilan folder, pengguna dapat membuka informasi rinci dari objek apapun dalam folder, informasi ini tergantung pada sistem file.
Untuk UNIX, ditunjukkan struktur i-node sebagai berikut:

- Informasi, berupa jumlah tautan yang menunjuk ke i-node
- 12 blok dari data, diberikan nomor blok untuk menunjukkan lokasi data, atau diberikan "nil" jika tidak digunakan
- *Indirection* (pengalihan), pertama, kedua, dan ketiga, diberikan nomor blok untuk menunjukkan lokasi dari blok pengalih, atau "nil" jika tidak digunakan
  Untuk setiap pengalihan, bisa didapatkan juga informasi berikut:
  - ❏ Index pointer
  - ❏ Pointed block number, atau "nil" jika tidak digunakan
  - ❏ Tipe, jika data bernilai "data" atau bernilai "indirection" jika merupakan pengalihan

Untuk MS-DOS ditampilkan entri tabel FAT untuk objek sebagai berikut:

- Nomor blok (FAT table entry)
- Nomor blok objek selanjutnya jika ada, jika tidak akan muncul "nil"

Data dan Statistik

Menu ini dapat dibuka berada pada deretan tombol di atas sisi sebelah kanan. Menu ini menampilkan informasi perangkat sesuai dengan metode alokasi yang digunakan.

Untuk UNIX, ditunjukkan tabel dengan semua blok perangkat data. Untuk setiap bloknya diperlihatkan:

- Nomor blok, dimulai dari blok pertama, @128
- Tipe, apakah blok data ("data block") atau blok pengalihan ("indirect block")

Untuk MS-DOS, ditampilkan tabel FAT yang merupakan snapshot dari semua blok perangkat. Untuk setiap entri tabel ditampilkan informasi sebagai berikut:

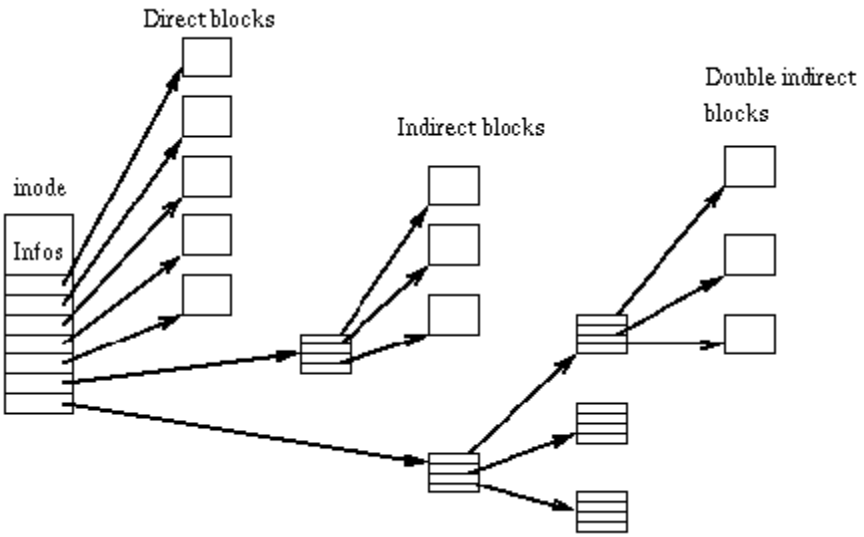
- Nomor blok, dimulai dari blok pertama, @128
- Next, berupa nomor blok objek berikutnya ketika blok berisikan objek data, atau "nil" jika blok merupakan blok terakhir dari suatu objek, atau bisa juga bernilai 0 jika blok tersebut masih tersedia
- Kondisi dari blok, apakah sudah digunakan ("used") atau masih kosong ("free")

Metode Alokasi *File*

Metode alokasi file pada OS SIM dijelaskan sebagai berikut:

- *Linked allocation with file allocation table* (alokasi bertaut dengan tabel alokasi *file*)
  Pada metode ini, sistem memiliki sebuah tabel dengan jumlah isian sebanyak jumlah blok yang dimiliki oleh sistem (*File Allocation Table*, FAT). FAT digunakan untuk menyimpan referensi dari blok pertama dari setiap objek. Referensi ini berada pada direktori yang mengandung objek tersebut. Pada tabel tersebut, setiap blok memiliki referensi ke blok selanjutnya, sehingga setiap blok *file* dapat ditelusuri dari blok awal hingga blog terakhir. Ukuran tabel FAT sama dengan ukuran dari disk dibagi ukuran dari blok. Jumlah blok yang dibutuhkan untuk setiap *file* sama dengan ukuran *file* dibagi dengan ukuran blok. Pada OS SIM, diasumsikan bahwa direktori hanya menempati satu blok, sedangkan informasi tautan disimpan di direktori yang memiliki tautan tersebut.
- *Indexed allocation with three levels of indirection* (alokasi berindeks dengan tiga level pengalihan)
  Pada metode ini, alokasi berindeks didasarkan pada struktur yang disebut dengan nama i-node. Setiap objek diasosiasikan dengan sebuah i-node. Struktur dari sebuah i-node adalah sebagai berikut:
  - Kolom informasi (*information fields*)
  - 12 pointer ke blok data
  - 1 pointer ke blok pengalihan (*indirect block*) *single*
  - 1 pointer ke blok pengalihan (*indirect block*) *double*
  - 1 pointer ke blok pengalihan (*indirect block*) *triple*

  Blok pengalihan *single* menunjuk ke blok data, blok pengalihan *double* menunjuk ke blok yang kemudian menunjuk ke blok data, sedangkan blok pengalihan *triple* menunjuk ke blok yang menunjuk ke blok yang menunjuk ke blok data.
  Untuk membuat sebuah *file*, blok data i-node digunakan terlebih dahulu, baru kemudian pengalihan blok *single, double,* dan *triple* untuk mengisi informasi yang dibutuhkan kaitannya dengan ukuran dari *file*. Pada OS SIM, digunakan asumsi bahwa direktori menempati sebuah blok data, sedangkan tautan tidak menggunakan blok. Gambar 11.3 menunjukkan ilustrasi metode *indexed allocation with three levels of indirection*.



Gambar 11.3 Ilustrasi *Indexed allocation with three levels of indirection*

Dalam OS SIM, jumlah maksimum i-node yang dapat dibuat adalah 128 dan blok pengalihan memiliki 20 pointer. Tabel berikut menunjukkan cara menghitung blok yang dibutuhkan untuk membuat *file*.

	Available Data blocks	Administration Blocks	Total data Blocks	Total Adm. Blocks	Total Blocks
One i-node	12 blocks	Head	12	0	12
First indirection	Past more ... 20 blocks	1 indirect block	32	1	33
Second indirection	Past more ... 20 * 20 blocks	1 indirect block 1 +20 Indirect b.	432	22	454
Third indirection	Past more ... 20 * 20 * 20 blocks	1 indirect block 1 +20 Indirect b. 1 +20 +400 Indirect b.	8.432	433	8.875

Simulasi dengan OS SIM
Alokasi Linked dengan FAT. Sistem File Objek

Dalam contoh ini direktori *tree* dibuat dengan berbagai jenis objek *file system*: direktori, *file,* dan tautan.
Informasi 1: Warna direktori adalah hijau dan tautan berwarna biru. Warna dari *file* didefinisikan ketika file tersebut dibuat.
Informasi 2: Dalam simulasi ukuran direktori adalah 1 blok, sedangkan tautan tidak menempati ruang

disk, namun hanya merupakan isian data dari direktorinya. Ukuran blok adalah 1 unit.

Tree				Type	Size Units
C:				Folder	1
	folder1			Folder	1
		file1.txt		File	101
		file2.txt		File	200
	folder2			Folder	1
		file3.txt		File	35
		Folder3		Folder	1
			file4.txt	File	120
		link1		Soft link (to file1.txt)	0

Komentar:

- File sistem root adalah "C:" dan penghubung *path* adalah "\"
- Dalam setiap folder baru, dua link secara otomatis dibuat "." dan ".." menunjuk ke direktori itu sendiri dan direktori induk masing-masing.
- Blok pertama dari objek yang berbeda sesuai dengan entri pertama dalam tabel FAT
- Jika file yang direferensikan oleh tautan dihapus (link1 → file1.txt), tautan tetap ada, namun menjadi tidak dapat digunakan
- Jika direktori dihapus, semua objek di dalamnya juga akan terhapus

Alokasi Linked dengan FAT

Dalam contoh ini direktori *tree* dibuat dengan berbagai jenis objek *file system*: direktori, *file*, dan tautan.
Informasi 1: Direktori berwarna hijau dan tautan berwarna biru. Warna *file* didefinisikan ketika *file* tersebut dibuat.
Informasi 2: Dalam simulasi ukuran direktori adalah 1 blok, sedangkan tautan tidak menempati ruang disk, namun hanya merupakan isian data dari direktorinya. Ukuran blok adalah 4 unit.

Tree				Type	Size Units	Blocks
C:				Folder	1	1
	folder1			Folder	1	1
		file1.txt		File	101	26
		file2.txt		File	200	50

	folder2			Folder	1	1
		file3.txt		File	35	9.
		Folder3		Folder	1	1
			file4.txt	File	120	30
		link1		Soft link (to file1.txt)	0	0

Komentar:

- Blok terakhir dari file "file1.txt" dan "file3.txt" tidak penuh, namun ruang tambahan tidak dapat dimanfaatkan, karena seluruh blok digunakan
- (Dibandingkan dengan Contoh 1) Direktori berisi informasi yang sama namun memakan lebih banyak ruang, dalam contoh ini 4 unit

Alokasi Indexed (UNIX)

Dalam contoh ini direktori *tree* dibuat dengan berbagai jenis objek *file system*: direktori, *file,* dan tautan.
Pada UNIX, setiap objek terkait dengan struktur yang disebut sebagai i-node. Referensi ini berada di entri direktori di mana objek tersebut berada.
Tautan merujuk ke objek lain dan berkaitan dengan i-node yang sama dari objek yang dirujuk. Struktur i-node mencatat semua objek lain yang terhubung dengan objek tersebut.
Informasi 1: Direktori berwarna hijau dan tautan berwarna biru. Warna *file* didefinisikan ketika *file* tersebut dibuat.
Informasi 2: Dalam simulasi ukuran direktori adalah 1 blok, sedangkan tautan tidak menempati ruang disk, namun hanya merupakan isian data dari direktorinya. Ukuran blok adalah 1 unit.
Informasi 3: Struktur dari i-node

- Kolom informasi (*information fields*)
- 12 pointer ke blok data
- 1 pointer ke blok pengalihan (*indirect block*) *single*
- 1 pointer ke blok pengalihan (*indirect block*) *double*
- 1 pointer ke blok pengalihan (*indirect block*) *triple*

Tree				Type	Size Units
/				Folder	1
	folder1			Folder	1
		file1.txt		File	101
		file2.txt		File	200
		link1		Hard link (to file1.txt)	1
	folder2			Folder	1

		file3.txt		File	460
		Folder3		Folder	1
			file4.txt	File	120
		link2		Hard link (to link1)	0

Komentar

- Objek-objek yang terkait, link1 → file1.txt, merujuk ke i-node yang sama.
- Dengan membuka informasi pada i-node (klik kanan pada tampilan folder, pop up menu) terlihat tautan berjumlah 3, yaitu: file1.txt, link1, dan link2

Alokasi Indexed (UNIX). Indirections

Dalam contoh ini direktori *tree* dibuat dengan berbagai jenis objek *file system*: direktori, *file,*dan tautan.
Pada UNIX, setiap objek terkait dengan struktur yang disebut sebagai i-node. Referensi ini berada di entri direktori di mana objek tersebut berada.
Tautan merujuk ke objek lain dan berkaitan dengan i-node yang sama dari objek yang dirujuk. Struktur i-node mencatat semua objek lain yang terhubung dengan objek tersebut.
Informasi 1: Direktori berwarna hijau dan tautan berwarna biru. Warna *file* didefinisikan ketika *file* tersebut dibuat.
Informasi 2: Dalam simulasi ukuran direktori adalah 1 blok, sedangkan tautan tidak menempati ruang disk, namun hanya merupakan isian data dari direktorinya. Ukuran blok adalah 1 unit.
Informasi 3: Struktur dari i-node

- Kolom informasi (*information fields*)
- 12 pointer ke blok data
- 1 pointer ke blok pengalihan (*indirect block*) *single*
- 1 pointer ke blok pengalihan (*indirect block*) *double*
- 1 pointer ke blok pengalihan (*indirect block*) *triple*

Untuk setiap *file* blok data baru diisi dengan urutan i-node pointer ke blok data secara langsung, kemudian blok pengalihan *single, double,* dan *triple*.

Tree				Type	Size Units
/				Folder	1
	folder1			Folder	1
		file1.txt		File	101
		file2.txt		File	200
		link1		Hard link (to file1.txt)	1
	folder2			Folder	1
		file3.txt		File	460

		Folder3		Folder	1
			file4.txt	File	120
		link2		Hard link (to link1)	0

Komentar:

- Blok pengalihan ditampilkan dengan karakter "I". Blok ini tidak berisi data, namun hanya penunjuk ke blok data atau blok pengalihan lainnya
- Blok pengalihan *double* ("I", "I") berarti bahwa blok pengalihan *double* mulai diisi, begitu juga dengan blok pengalihan *triple* ("I", "I", "I")
- Informasi tentang pengalihan dan blok data dapat dilakukan dengan membuka informasi yang sesuai i-node (klik kanan pada tampilan folder, pop up menu, dan kemudian klik kanan lagi pada blok tidak langsung).

C. Aktivitas Praktikum

Simulasi dengan Program

FAT

Berikut adalah contoh sederhana implementasi program menggunakan FAT dengan metode alokasi *linked*.

```
#include<iostream>
using namespace std;

struct file
{
  char fname[10];
  int start,size,block[10];
}f[10];

int main()
{
  int i,j,n;
  cout << "Enter number of files:";
  cin >> n;
  for(i=0;i<n;i++)
  {
            cout << "Enter file name:";
            cin >> f[i].fname;
            cout << "Enter starting block:";
            cin >> f[i].start;
            f[i].block[0]=f[i].start;
            cout << "Enter number of blocks:";
            cin >> f[i].size;
            cout << "Enter block numbers:";
            for(j=1;j<=f[i].size;j++)
            {
            cin >> f[i].block[j];
            }
```

```
 }
 cout << "File\tstart\tsize\tblock\n";
 for(i=0;i<n;i++)
 {
        cout << f[i].fname << "\t" << f[i].start << "\t" << f[i].size << "\t";
        for(j=0;j<f[i].size;j++)
        cout << f[i].block[j] << "--->";
        cout << f[i].block[j];
        cout << "\n";
 }
 return 0;
}
```

D. Tugas

Modifikasi kode program simulasi FAT dengan metode alokasi linked di atas:

- Tambahkan pengecekan sehingga satu nomor blok hanya dapat ditempati oleh satu *file*
- Tambahkan fungsi untuk dapat menambah atau mengurangi ukuran *file*