MODUL PRAKTIKUM MATAKULIAH SISTEM OPERASI
TEKNIK INFORMATI

DOSEN PENGAMPU
SUTARDI. S.KOM., MT
NATALIS RANSI. SSI., MCS
LAB COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

PRAKTIKUM 1: PENGENALAN CPU SIMULATOR

Pertemuan ke 1

Total Alokasi Waktu : 90 menit
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit

Total Skor Penilaian : 100%
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %

1.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

1. Memahami komponen arsitektur komputer tingkat bawah.
2. Menggunakan simulator untuk membuat dan mengeksekusi instruksi dasar CPU.
3. Membuat instruksi untuk memindahkan data ke register, membandingkan isi register, memasukkan data ke stack, mengambil data dari stack, melompat ke lokasi alamat, menambahkan nilai dalam register.
4. Menjelaskan fungsi-fungsi dari register khusus CPU antara lain register PC, SR, dan SP.

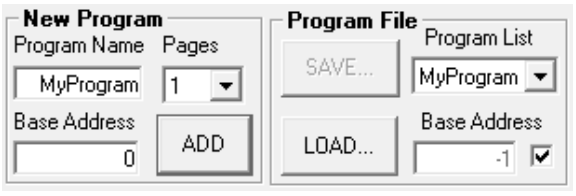
Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

1. Dapat memahami komponen arsitektur komputer tingkat bawah.
2. Dapat menggunakan simulator untuk membuat dan mengeksekusi instruksi dasar CPU.
3. Dapat membuat instruksi untuk memindahkan data ke register, membandingkan isi register, memasukkan data ke stack, mengambil data dari stack, melompat ke lokasi alamat, menambahkan nilai dalam register.
4. Dapat menjelaskan fungsi-fungsi dari register khusus CPU antara lain register PC, SR, dan SP.

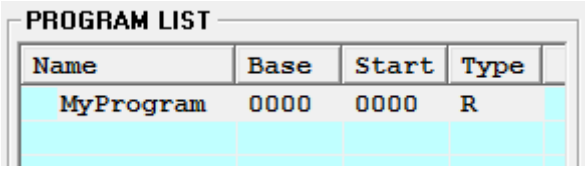
1.2. TEORI PENDUKUNG

Dibawah ini adalah deskripsi dari 4 tahap utama:

1. Membuat program CPU akan berisi instruksi-instruksi



Gambar 1 New Program Frame



Gambar 2. Program list frame

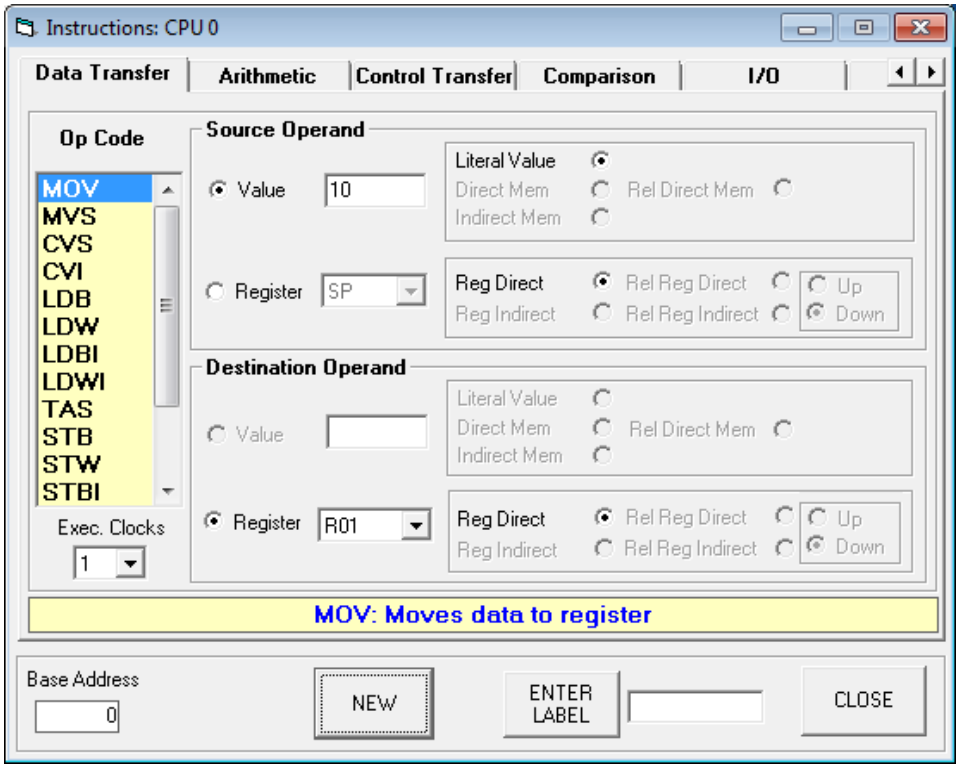
Masukkan nama program pada **Program Name** text box, contoh MyProgram. Masukkan nomor pada **Base Address** text box (saran = gunakan 0 untuk kasus ini). Kemudian klik tombol **ADD**. Nama program akan muncul pada frame LIST PROGRAM (Gambar 2).

2. Adding CPU instructions in the program

Gambar 3. Program instructions frame

Dalam frame **PROGRAM INSTRUCTION** tombol **ADD NEW** aktif. Klik tombol tersebut untuk melihat CPU instructiom kamu dapat memilih untuk program yang dibuat di atas (Gambar 4).

Memilih CPU instructions untuk memasuki program secara manual:

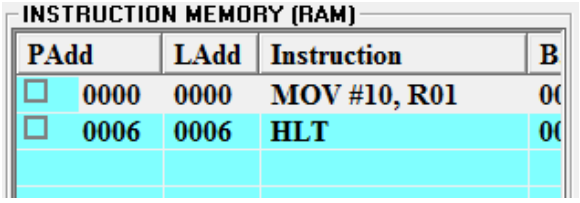


Gambar 4. Selecting CPU instructions

Pada window **Instruction** yang diperlihatkan di atas, kamu dapat memiih beberapa CPU instruction yang tersedia untuk programmu. Instructions yang terhubung dikategorikan dalam beberapa kelompok. Kamu dapat memilih sebuah kelompok dengan mengklik pada tab grup. Pilih instruction yang diinginkan dari list di bawah **OP CODE.** Jika instruction yang dipilih membutuhkan operand(s) maka tipe yang tersedia akan aktif di bawah **SOURCE OPERAND** dan frame **DESTINATION OPERAND.** Selanjutnya, klik tombol **NEW** untuk menamba instuction. Kamu dapat menambah instruction ganda tanpa menutup window ini.

Instruction akan muncul pada area memori CPU program seperti yang diperlihatkan pada Gambar 5. Simulator CPU membutuhkan membutuh instruction terakhir pada program yaitu **HLT** instruction (intructio ini memberhentikan proses simulasi. Seperti yang terdapat pada Gambar 5. Program sekarang dapat dijalankan.

CPU instructions in program:



Gambar 5. CPU program memory

Setiap entry pada view **INSTRUCTION MEMORY** terdapat informasi sebagai berikut :

Padd(Physicial Address), **Ladd**(Logical Adress) dan instruction. Informasi lainnya juga tersedia tapi tidak relevan pada tahap ini.

Editing the program instructions:

Gambar 6. Program edit functions

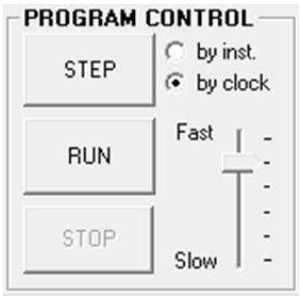
Sekali instructions program dimasukkan. Mereka dapat diedit. Untuk melakukannya, pilih instruction yang diinginkan dan gunakan satu dari fungsi edit (**EDIT, DELETE, MOVE UP, MOVE DOWN)** diperlihatkan pada Gambar 6 untuk mengedit instruction yang dipilih. Gunakan tombol **INSERT ABOVE...** dan **INSERT BELOW...** untuk memasukkan instruction baru diatas(above) atau di bawah (below) instruction yang dipilih.

Menghapus program:

Gambar 7. Program removal

Program CPU dapat dihapus dengan mengklik tombol **REMOVE PROGRAM** diperlihatkan pada Gambar 7. Sekali program dihapus, maka program tersebut tidak akan ada lagi dan hilang. Bagaimanapun kamu dapat menyimpan program tersebut sehingga kamu dapat membuka kembali program tersebut nanti (Gambar 12)

3. Running the program

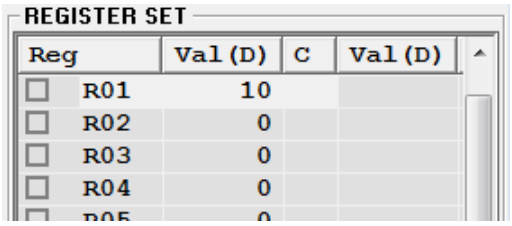


Gambar 8. Program control

Program CPU dapat dijalankan dalam 2 cara yang berbeda : 1) instruction dengan instruction, 2) otomatis dalam sekali jalan. Untuk menjalankan instruction yang dipilih dengan sendirinya, pertama pilih program seperti Gambar 5 dan double klik pada program tersebut. alternatifnya, kamu dapat mengklik tombol **STEP** yang diperlihatkan pada Gambar 8 (pastikan pilihan **by inst.** Dipilih). Gunakan tombol **STOP** untuk memberhentikan program yang berjalan. Gunakan **slider control** untuk mempercepat atau memperlambat program yang berjalan

4. Observing and controlling the simulations

Mengamati/mengubah register CPU

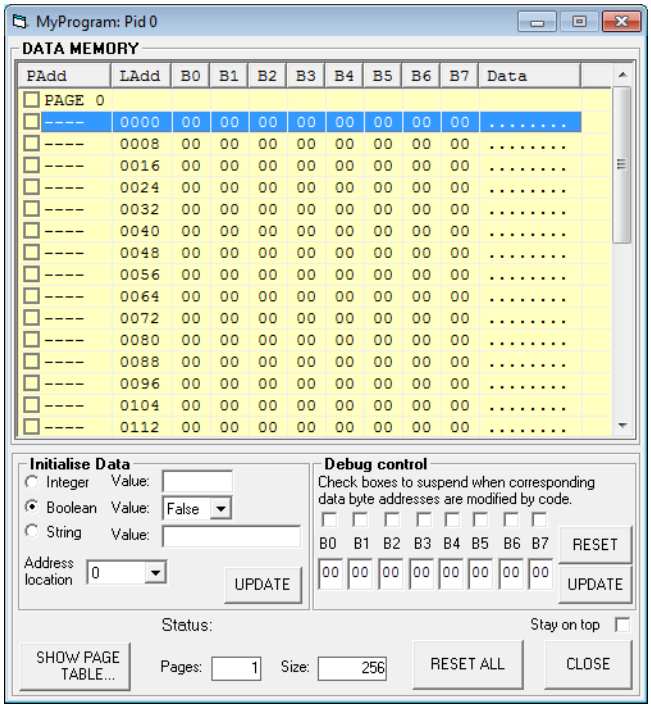


Gambar 9. Register set view

Sebuah instruction yang menulis atau membaca akses register pada frame **REGISTER SET** dan register yang diakses disorot. Frame ini menunjukkan register-register dan nilai mereka. Klik pada kolom **Val** untuk mengubah nilai dari desimal **(D)** ke format hex **(H)** dan sebaliknya.

Note: untuk mengubah nilai register secara manual, pertama pilih dan masukkan nilai pada field **Reg Value** dan klik pada tombol **CHANGE** (Gambar 9).

Observing/altering the program data:



Gambar 10. Data memory page

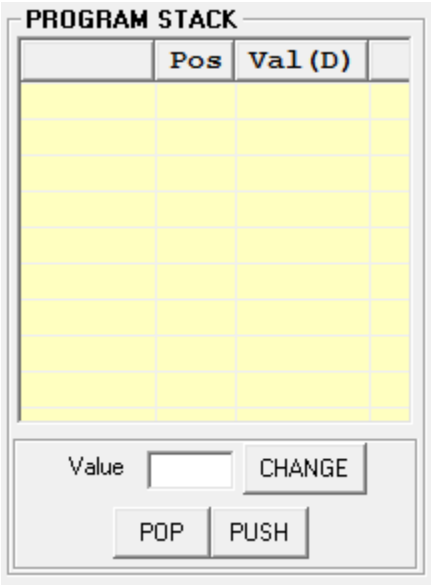
Instruction CPU yang mengakses bagian dari memori yang mengandung data dapat menulis atau membaca data yang diakses. Informasi ini tersedia pada window halaman memori pada Gambar 10. Klik tombol **SHOW PROGRAM DATA MEMORY...** yang diperilihatkan pada Gambar 7 diatas. Pada window ini juga dapat mengedit isi dari data.

Kolom **Ladd** menunjukkan starting address pada setiap garis di display. Setiap garis dari display merepresentasikan 8 byte dari informasi, jadi nomor **Ladd** terurut dari kecil hingga 8 dari setiap garis di bawah display. Kolom **B0** ke **b7** terdapat bytes 0 hingga 7. Kolom **Data** menunjukkan karakter yang dapat diperlihatkan sesuai dengan 8 bytes. Bytes tersebut berhubungan ke character yang tidak dapat diperlihatkan yang ditunjukkan sebagai dots. Data byes hanya dapat diperlihatkan dalam format hex.

Untuk mengubah nilai dari bytes manapun pertama pilih garis yang mengandung bytes. Kemudia gunakan informasi pada frame **Initialize Data** untuk memodifikasi nilai dari bytes pada garis yang dipilih sebagai format **Integer, Boolean atau String.** Kamu harus mengklik tombol **UPDATE** untuk mengubah.

Cek check box **Stay on top** untuk memastikan window selalu berada di atas window lainnya ketika masih memperbolehkan mengakses windows di bawahnya.

(melihat/mengubah stack program):

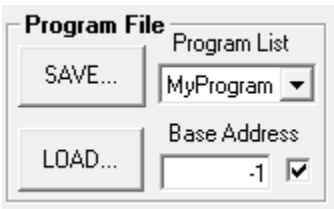


Gambar 11. Program stack frame

Programs menjadikan **PROGRAM STACK** untuk menyimpan informasi penting sementara seperti **subroutine return addressess** dan **subroutine parameters** dan informasi relevan lainnya. Terdapat instruction yang dapat mendorong **(PSH)** data di atas stack dan dapat memunculkan data (**POP)** data dari atas stack ke register.

Kamu dapat mendorong dan memunculkan data secara manual dengan mengklik tombol **PUSH dan POP.** Kamu juga dapat memodifikasi entry stack dengan memilihnya, memasukkan nilai baru pada text box nilai dan mengklik **CHANGE**

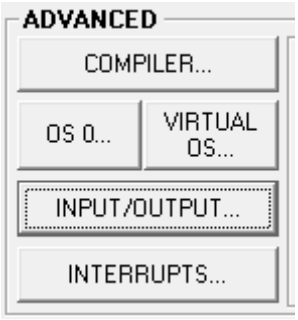
Saving and loading the CPU programs:



Gambar 12. Saving and loading programs

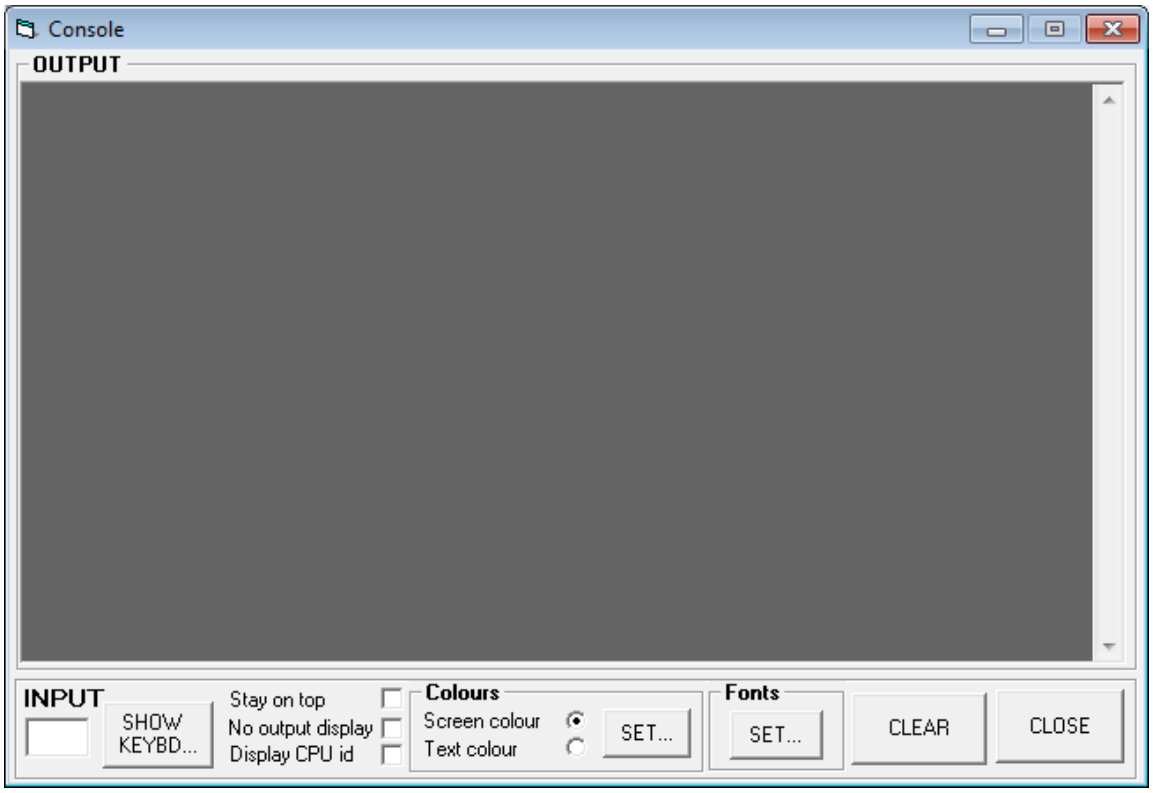
Untuk menyimpan program, pilih program dengan mencari program dari down list dan klik tombol **SAVE...** untuk memuat program yang tersimpan klik tombol **LOAD**

Observing the displayed information:



Gambar 13. Console button

Program dapat memperlihatkan informasi dan menerima data dari konsol yang disimulasi. Instruction **OUT** digunakan untuk memperlihatkan informasi dan instruction **IN** digunakan untuk menerima input dari konsol. Untuk menunjukkan konsol klik tombol **INPUT/OUTPUT...** ditunjukkan pada Gambar 13. Konsol window ditunjukkan pada Gambar 14.



Gambar 14. Console button

Cek check box **Stay on top** untuk memastikan window selallu berada di atas windows lainya ketika masih memperbolehkan akses windows di bawahnya. Klik pada **SHOW KEYBD...** untuk menunjukkan keyboard kecil virtual untuk menginput data pada Gambar 15.

Gambar 15. Virtual keyboard

Cek check box **Lower Case** untuk menginput karakter kecil. Keyboard ini adalah bagian dari keyboard standard. Kamu juga dapat menginput dengan menulis dapat text box **INPUT** yang ditunjukkan pada Gambar 14

Tabel 1. CPU Simulator Instruction Set

Instruction	Description
Data transfer instructions	
MOV	Move data to register; move register to register e.g. **MOV #2, R01** moves number 2 into register R01 **MOV R01, R03** moves contents of register R01 into register R03
LDB	Load a byte from memory to register
LDW	Load a word (2 bytes) from memory to register
STB	Store a byte from register to memory
STW	Store a word (2 bytes) from register to memory
PSH	Push data to top of hardware stack (TOS); push register to TOS e.g. **PSH #6** pushes number 6 on top of the stack **PSH R03** pushes the contents of register R03 on top of the stack
POP	Pop data from top of hardware stack to register e.g. **POP R05** pops contents of top of stack into register R05
Arithmetic instructions	
ADD	Add number to register; add register to register e.g. **ADD #3, R02** adds number 3 to contents of register R02 and stores the result in register R02. **ADD R00, R01** adds contents of register R00 to contents of register R01 and stores the result in register R01.
SUB	Subtract number from register; subtract register from register
MUL	Multiply number with register; multiply register with register
DIV	Divide number with register; divide register with register
Control transfer instructions	
JMP	Jump to instruction address unconditionally e.g. **JMP 100** unconditionally jumps to address location 100

JLT	Jump to instruction address if less than (after last comparison)
JGT	Jump to instruction address if greater than (after last comparison)
JEQ	Jump to instruction address if equal (after last comparison) e.g. **JEQ 200** jumps to address location 200 if the previous comparison instruction result indicates that the two numbers are equal.
JNE	Jump to instruction address if not equal (after last comparison)
CAL	Jump to subroutine address
RET	Return from subroutine
SWI	Software interrupt (used to request OS help)
HLT	Halt simulation
Comparison instruction	
CMP	Compare number with register; compare register with register e.g. **CMP #5, R02** compare number 5 with the contents of register R02 **CMP R01, R03** compare the contents of registers R01 and R03 Note: If R01 = R03 then the status flag Z will be set If R03 > R01 then non of the status flags will be set If R01 > R03 then the status flag N will be set
Input, output instructions	
IN	Get input data (if available) from an external IO device
OUT	Output data to an external IO device

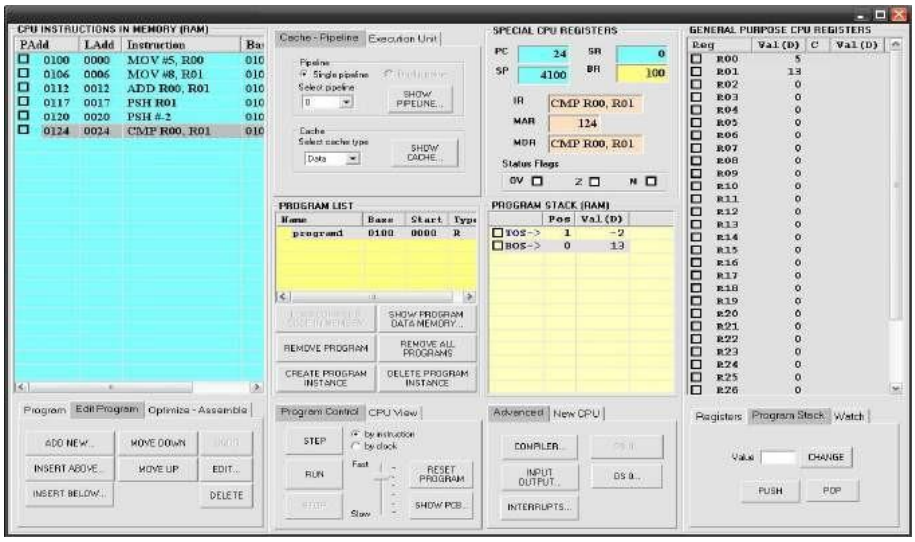
Model Pemrograman

Model pemrograman arsitektur komputer mendefinisikan komponen-komponen arsitektur pada tingkat bawah (low level architectural components) yang mencakup:

a. Set instruksi prosesor
b. Register-register
c. Jenis-jenis pengalamatan instruksi dan data
d. Interrupts dan *exceptions*

Pada model pemrograman ini terdapat interaksi antar komponen diatas. Ini merupakan model pemrograman tingkat rendah (low-level) yang memungkinkan suatu komputasi terprogram.

Perangkat Simulator

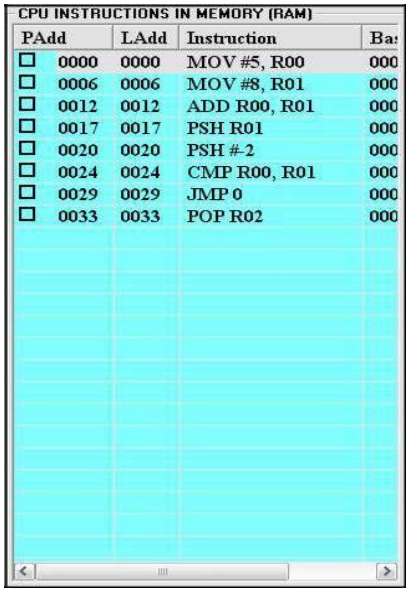


Gambar 16. Jendela utama simulator

Jendela utama terdiri dari beberapa tampilan, yang merepresentasikan beberapa bagian fungsional prosesor yang disimulasikan, yaitu:

a. Instruction memory (RAM)
b. Special registers
c. Register set
d. Program stack

Tampilan *instruction memory*

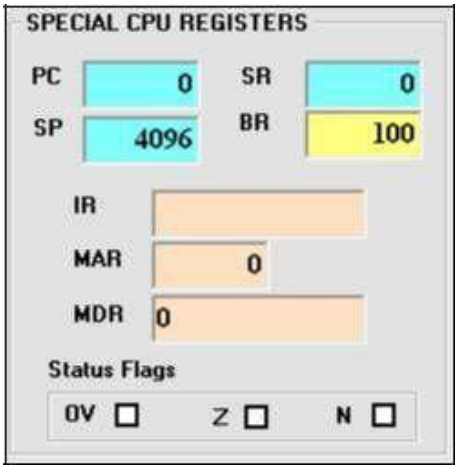


Gambar 17. Tampilan instruction view

Tampilan ini berisi instruksi-instruksi dari program. Instruksi ditampilkan dalam urutan instruksi mnemonik low-level (format assembler), tidak dalam bentuk kode biner. Hal ini bertujuan untuk memperjelas dan membuat kode lebih mudah dibaca.

Tiap instruksi mempunyai dua alamat: alamat fisik (Padd) dan alamat logika (Ladd). Tampilan ini juga menyajikan alamat dasar (Base) terhadap tiap instruksi. Urutan instruksi pada program yang sama akan mempunyai alamat dasar yang sama.

Tampilan *Special Registers*



Gambar 18. Tampilan Special Register

Tampilan ini menyajikan register-register dengan fungsi-fungsi khusus yang telah ditentukan:

PC: Program Counter, berisi alamat instruksi berikutnya yang akan dieksekusi.

IR : Instruction Register, berisi instruksi yang sedang dieksekusi saat ini.

SR : Status Register, berisi informasi yang memberikan hasil dari instruksi yang dieksekusi terakhir.

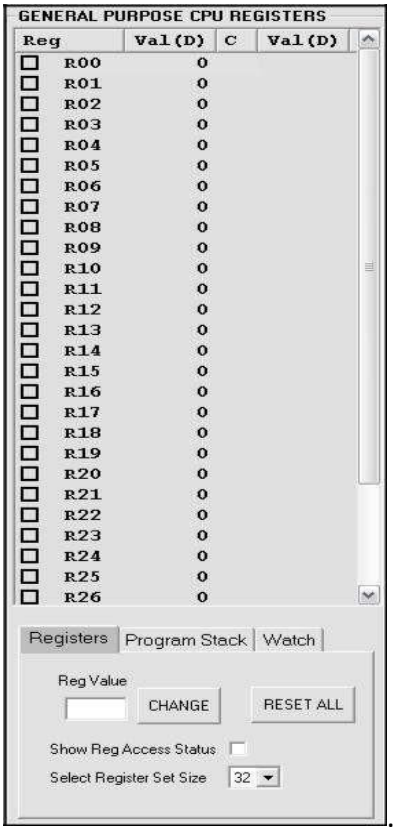
SP : Stack Pointer, register menunjuk ke nilai yang berada pada bagian atas stack.

BR : Base Register, berisi alamat dasar saat ini.

MAR : Memory Address Register, berisi alamat memori yang sedang diakses.

Status Bits : OV: Overflow; **Z**: Zero; **N**: Negative

Tampilan *Register Set*



Gambar 19. Tampilan Register Set

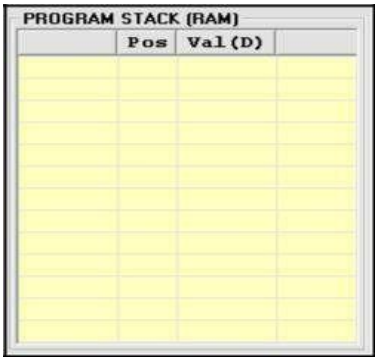
Tampilan register set menunjukkan isi dari semua register-register tujuan umum (general-purpose), yang digunakan untuk menampung nilai sementara ketika instruksi program dieksekusi.

Arsitektur ini mendukung 8 hingga 64 register. Register-register ini sering digunakan untuk menyimpan nilai variabel program pada bahasa tingkat tinggi.

Tidak semua arsitektur memiliki register sebanyak ini. Beberapa lebih banyak (misalnya 128 register) dan beberapa lainnya lebih sedikit (misalnya 8 register). Pada semua kasus, register-register ini bekerja untuk tujuan yang sama.

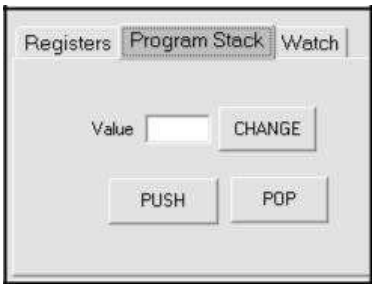
Bagian ini menampilkan nama tiap register (Reg), nilai (Val), dan beberapa informasi lainnya untuk keperluan debug program. Kita juga dapat melakukan reset (RESET) atau mengisikan nilai register (CHANGE) secara manual.

Tampilan *Program Stack*



Program Stack menampung nilai sementara ketika instruksi dieksekusi. Stack menggunakan struktur data LIFO (last-In-First-Out). Stack sering digunakan untuk efisiensi *interrupt handling* dan *sub-routine call*.

Gambar 20. Tampilan Program Stack



Instruksi PUSH dan POP digunakan untuk menyimpan dan mengambil nilai pada bagian teratas stack.

1.3. ALAT DAN BAHAN

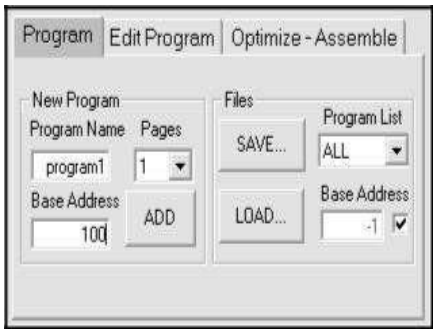
Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. CPU Simulator.

1.4. LANGKAH PRAKTIKUM

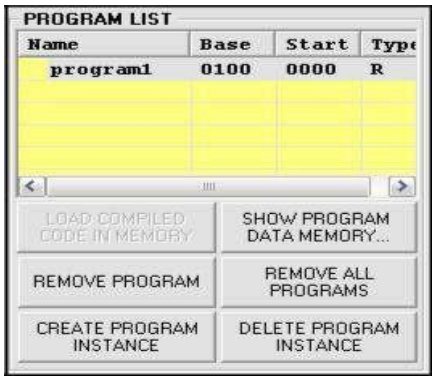
A. Persiapan Praktikum

Pertama-tama kita perlu menempatkan sejumlah instruksi pada tampilan instruction memory (menggambarkan RAM pada mesin yang nyata) sebelum melakukan eksekusi instruksi. Berikut adalah cara menempatkan instruksi ke instruction memory :



Pada tampilan program instruction, masukkan nama program (misal: program1), dan juga alamat dasar (untuk praktikum ini beri alamat dasar nilai 100). Klik tombol ADD. Maka program baru dengan namanya akan dimasukkan ke tampilan Program List seperti yang disajikan gambar 22. Gunakan tombol SAVE.../ LOAD... untuk menyimpan dan mengambil instruksi dari file.

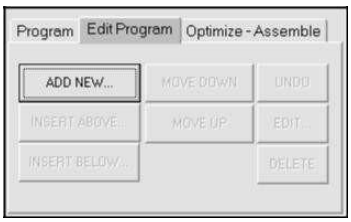
Gambar 21. Tampilan Program Instructions



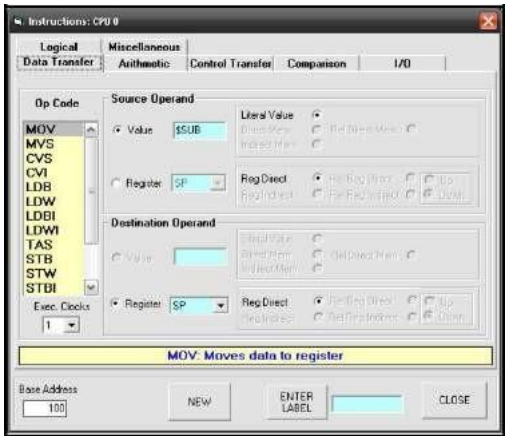
Gunakan tombol REMOVE PROGRAM untuk menyingkirkan program yang disorot. Gunakan tombol REMOVE ALL PROGRAMS untuk menyingkirkan semua program yang ada di daftar. Sebagai catatan, ketika program disingkirkan maka instruksi-instruksi yang terkait dengannya juga akan dihapus dari tampilan instruction memory.

Gambar 22. Tampilan Program List

B. Langkah Praktikum



Untuk memasukkan instruksi ke CPU, klik tombol ADD NEW... yang akan menampilkan jendela Instructions: CPU0.



Gunakan jendela ini untuk memasukkan instruksi. Pada **Appendix** tersedia beberapa instruksi yang dikenali oleh simulator ini dan terdapat pula contoh penggunaannya.

a. Transfer data

1. Buatlah instruksi yang memindahkan (move) angka 5 ke register R00.
2. Eksekusi instruksi diatas (dengan klik dua kali pada tampilan instruction memory).
3. Buatlah instruksi yang memindahkan angka 8 ke register R01.
4. Eksekusilah.
5. Amati isi R00 dan R01 pada tampilan Register Set.

b. Aritmatika

6. Buatlah suatu instruksi yang menambahkan (add) isi R00 dan R01.
7. Eksekusilah.
8. Amati dimana hasil penjumlahan tersebut disimpan.

c. Stack Pointer (SP)

9. Buatlah instruksi yang menaruh hasil diatas pada program stack, kemudian eksekusilah.

10. Buatlah instruksi untuk menaruh (push) angka -2 pada stack teratas dan eksekusilah.
11. Amati nilai register SP.

d. Pembanding

12. Buatlah instruksi untuk membandingkan nilai register R00 dan R01.
13. Eksekusilah.
14. Amati nilai register SR.
15. Amati bit status OV/Z/N pada status register.
16. Analisa status register tersebut.

e. Stack Pointer (SP)

17. Buatlah instruksi untuk mengambil (pop) nilai teratas dari program stack ke register R02.
18. Eksekusilah.
19. Amati nilai pada register SP.
20. Buatlah suatu instruksi untuk mengambil nilai teratas dari program stack ke register R03.
21. Eksekusilah.
22. Amati nilai pada register SP.
23. Eksekusi lagi instruksi yang terakhir. Apa yang terjadi? Jelaskan.

1.5. TUGAS

Buatlah program CPU sesuai tahap-tahap yang telah dijelaskan. Jalankan serta buat kesimpulan.

PRAKTIKUM 2: THREADS

Pertemuan ke 2

Total Alokasi Waktu : 90 menit (Alokasi waktu disesuaikan dengan RPS)
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit
- dst

Total Skor Penilaian : 100% (Bobot skor disesuaikan dengan RPS)
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %
- dst

2.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

1. Menulis kode sumber yang membuat thread
2. Menampilkan daftar proses/ thread dan pohon proses yang menggambarkan hubungan antar proses induk/anak.
3. Memodifikasi kode sumber untuk membuat versi tanpa thread dan membandingkannya dengan versi yang menggunakan thread
4. Memahami bagaimana thread-thread berbagi sumber daya dari proses induk

Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

1. Dapat menulis kode sumber yang membuat thread
2. Dapat menampilkan daftar proses/ thread dan pohon proses yang menggambarkan hubungan antar proses induk/anak.
3. Dapat memodifikasi kode sumber untuk membuat versi tanpa thread dan membandingkannya dengan versi yang menggunakan thread.
4. Dapat memahami bagaimana thread-thread berbagi sumber daya dari proses induk.

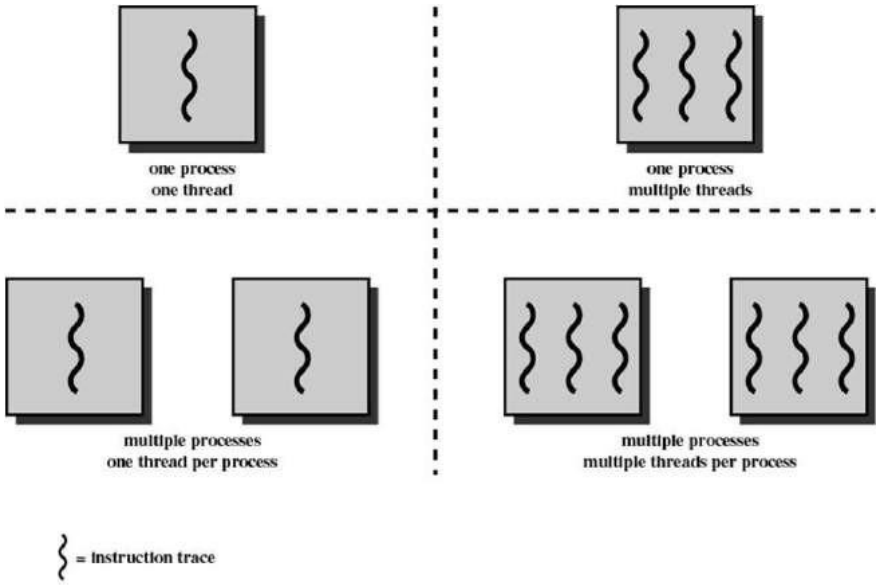
2.2. TEORI PENDUKUNG

Konsep proses mengandung dua karakteristik:

1. Unit kepemilikan sumber daya
2. Unit pengiriman (dispatching)

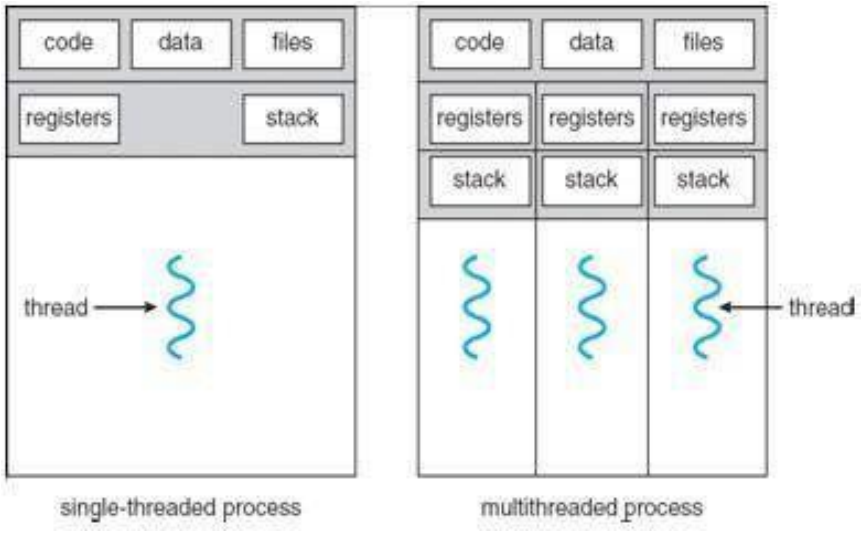
Untuk membedakan kedua karakteristik itu, unit dispatching biasanya disebut sebagai thread, atau lightweight process, sedangkan unit kepemilikan sumber daya biasanya masih disebut sebagai proses atau task (Stallings, 2003).

Dengan begitu pada tiap thread terdapat ID thread, program counter, register, dan stack (terkait unit dispatching). Namun saling berbagi dengan thread yang lain (dalam satu proses) terhadap kepemilikan sumber daya yang sama.



Gambar 23. Thread dan Proses

.



Gambar 24. Thread dan sumber daya yang dimiliki

2.3. ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. CPU Simulator

2.4. LANGKAH PRAKTIKUM

Menampilkan daftar proses/ thread dan pohon proses yang menggambarkan hubungan antar proses induk/anak

1. Jalankan Simulator. Tuliskan kode berikut ini ke compiler.

```
program ThreadTest1
     sub thread1 as thread
          writeln("In thread1")
          while true
          wend
     end sub

     sub thread2 as thread
          call thread1
          writeln("In thread2")
          while true
          wend
     end sub

     call thread2
     writeln("In main")

     do
     loop
end
```

2. Compile dan Run kode menggunakan CPU-OS Simulator
   a. Compile dengan menekan tombol COMPILE pada compiler..
   b. Melalui CPU Simulator, tampilkan jendela konsol dengan klik tombol INPUT/OUTPUT... aktifkan STAY ON TOP.
   c. Beralihlah ke jendela OS Simulator, buat satu proses dengan CREATE NEW PROCESS.
   d. Pastikan jenis penjadwalan adalah ROUND ROBIN, 10 ticks dengan kecepatan simulasi maksimum.
3. Tekan tombol START dan pada waktu yang bersamaan amati tampilan pada jendela console.
   a. Berapakah jumlah proses yang telah dibuat?
   b. Sebutkan mana yang disebut proses dan mana yang disebut thread!
4. Klik tombol VIEW PROCESS LIST. Kemudian klik PROCESS TREE.. Identifikasi proses induk dan proses turunan!
5. Gunakan tombol KILL untuk menghentikan proses!

Memodifikasi kode sumber untuk membuat versi tanpa thread dan membandingkannya dengan versi yang menggunakan thread.

1. Modifikasi source code ThreadTest1 dengan menghapus instance “as thread” pada deklarasi subroutine. Ganti nama program menjadi ThreadTest3.
2. Compile code hasil modifikasi tersebut.
3. Buat proses baru lagi, kemudian jalankan di OS simulator.
4. Amati tampilan pada jendela console, apa perbedaan proses tanpa thread dan proses menggunakan thread?
5. Gunakan tombol KILL untuk menghentikan proses!

Memahami bagaimana thread-thread berbagi sumber daya dari proses induk.

1. Buatlah program baru dengan memodifikasi source code ThreadTest1. Penambahan code ditandai dengan cetak tebal dan underline.

```
program ThreadTest2
      var s1 string(6)
      var s2 string(6)

      sub thread1 as thread
           s1 = "hello1"
           writeln("In thread1")
           while true
           wend
      end sub

      sub thread2 as thread
           call thread1
           s2 = "hello2"
           writeln("In thread2")
           while true
           wend
      end sub

      call thread2
      writeln("In main")
      wait

      writeln(s1)
      writeln(s2)
end
```

2. Compile program ThreadTest2. Klik tombol SYMBOL TABLE pada jendela compiler. Amati informasi tentang variabel s1 dan s2.

3. Melalui CPU Simulator, tampilkan jendela konsol dengan klik tombol INPUT/OUTPUT... aktifkan STAY ON TOP.
4. Beralihlah ke jendela OS Simulator, buat satu proses dengan CREATE NEW PROCESS.
5. Pastikan jenis penjadwalan adalah ROUND ROBIN, kecepatan simulasi maksimum.
6. Tekan tombol START dan pada waktu yang bersamaan amati tampilan pada jendela console. Gunakan tombol SUSPEND untuk menghentikan proses sementara.
7. Tekan tombol SHOW MEMORY…, amati alamat memory yang digunakan oleh variable s1 dan s2. Simpulkan apa yang terjadi.

2.5. TUGAS

Buatlah program baru dengan memodifikasi source code ThreadTest2 (pada contoh program sebelumnya) dengan tahap-tahap yang sesuai dan buat kesimpulan

PRAKTIKUM 3: ALGORITMA PENJADWALAN

Pertemuan ke 3

Total Alokasi Waktu : 90 menit (Alokasi waktu disesuaikan dengan RPS)
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit
- dst

Total Skor Penilaian : 100% (Bobot skor disesuaikan dengan RPS)
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %
- dst

3.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

Mensimulasikan algoritma penjadwalan CPU non-preemptive untuk menemukan waktu penyelesaian dan waktu tunggu

Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

Dapat mensimulasikan algoritma penjadwalan CPU non-preemptive untuk menemukan waktu penyelesaian dan waktu

3.2. TEORI PENDUKUNG

A. ALGORITMA PENJADWALAN FCFS CPU

Untuk algoritma penjadwalan FCFS, baca jumlah proses / pekerjaan dalam sistem, waktu burst CPU-nya. Penjadwalan dilakukan berdasarkan waktu kedatangan dari proses terlepas dari parameter lainnya. Setiap proses akan dieksekusi sesuai dengan waktu kedatangannya. Hitung waktu tunggu dan waktu penyelesaian masing-masing proses sesuai.

B. ALGORITMA PENJADWALAN SJF CPU

Untuk algoritma penjadwalan SJF, baca jumlah proses / pekerjaan dalam sistem, waktu burst CPU-nya. Atur semua pekerjaan sesuai dengan waktu ledakan mereka. Mungkin ada dua pekerjaan dalam antrian dengan waktu eksekusi yang sama, dan kemudian pendekatan FCFS harus dilakukan. Setiap proses akan dieksekusi sesuai dengan lamanya waktu meledaknya. Kemudian hitung waktu tunggu dan waktu penyelesaian masing-masing proses sesuai.

C. ALGORITMA PENJADWALAN ROUND ROBIN CPU

Untuk algoritma penjadwalan round robin, baca jumlah proses / pekerjaan dalam sistem, waktu burst CPU-nya, dan ukuran slice waktu. Irisan waktu ditugaskan untuk setiap proses dalam porsi yang sama dan dalam urutan melingkar, menangani semua proses eksekusi. Ini memungkinkan setiap proses untuk mendapatkan kesempatan yang sama. Hitung waktu tunggu dan waktu penyelesaian masing-masing proses yang sesuai.

D. ALGORITMA PENJADWALAN CPU PRIORITY

Untuk algoritma penjadwalan prioritas, baca jumlah proses / pekerjaan dalam sistem, waktu burst CPU-nya, dan prioritas. Atur semua pekerjaan agar sesuai dengan prioritas mereka. Mungkin ada dua pekerjaan dalam antrian dengan prioritas yang sama, dan kemudian pendekatan FCFS harus dilakukan. Setiap proses akan dieksekusi sesuai dengan prioritasnya. Hitung waktu tunggu dan waktu penyelesaian masing-masing proses yang sesuai

3.3. ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

3. Komputer.
4. Pemrograman C

3.4. LANGKAH PRAKTIKUM

A. ALGORITMA PENJADWALAN FCFS CPU

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

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

INPUT

Masukkan jumlah proses -- 3

Masukkan Burst Time untuk Proses 0 -- 24

Masukkan Burst Time untuk Proses 1 -- 3

Masukkan Burst Time untuk Proses 2 -- 3

OUTPUT

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

Rata-rata Waiting Time-- 17.000000

Rata-rata Turnaround Time ----- 27.000000

B. ALGORITMA PENJADWALAN SJF CPU

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

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

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

        wt[0] = wtavg = 0;

        tat[0] = tatavg = bt[0];

        for(i=1;i<n;i++)

        {

                wt[i] = wt[i-1] +bt[i-1];

                tat[i] = tat[i-1] +bt[i];

                wtavg = wtavg + wt[i];

                tatavg = tatavg + tat[i];

                }

        printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

                printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);

        printf("\nAverage Waiting Time -- %f", wtavg/n);

        printf("\nAverage Turnaround Time -- %f", tatavg/n);

        getch();

}
```

INPUT

```
Masukkan jumlah proses --  4
Masukkan Burst Time untuk Proses 0 -- 6
Masukkan Burst Time untuk Proses 1 -- 8
Masukkan Burst Time untuk Proses 2 -- 7
Masukkan Burst Time untuk Proses  3 --  3
```

OUTPUT

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

Rata-rata Waiting Time-- 7.000000

Rata-rata Turnaround Time ----- 13.000000

C. ALGORITMA PENJADWALAN ROUND ROBIN CPU

```
#include<stdio.h>
main()
{
        int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
        float awt=0,att=0,temp=0;
        clrscr();
        printf("Enter the no of processes -- ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter Burst Time for process %d -- ", i+1);
                scanf("%d",&bu[i]);
                ct[i]=bu[i];
        }
        printf("\nEnter the size of time slice -- ");
        scanf("%d",&t);
        max=bu[0];
        for(i=1;i<n;i++)
                if(max<bu[i])
                        max=bu[i];
        for(j=0;j<(max/t)+1;j++)
                for(i=0;i<n;i++)
                        if(bu[i]!=0)
                                if(bu[i]<=t)
                                {
                                        tat[i]=temp+bu[i];
                                        temp=temp+bu[i];
                                        bu[i]=0;
                                }
                                else
                                {
                                        bu[i]=bu[i]-t;
                                        temp=temp+t;
                                }
for(i=0;i<n;i++)
{
```

```
            wa[i]=tat[i]-ct[i];
            att+=tat[i];
            awt+=wa[i];
        }
        printf("\nThe Average Turnaround time is -- %f",att/n);

        printf("\nThe Average Waiting time is -- %f ",awt/n);

        printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
        for(i=0;i<n;i++)

                printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);

        getch();

}
```

INPUT

Masukkan jumlah proses -- 3
Masukkan Burst Time untuk Proses 1 -- 24
Masukkan Burst Time untuk Proses 2 -- 3
Masukkan Burst Time untuk Proses 3 -- 3

Masukkan ukuran potongan waktu (time slice) -- 3

OUTPUT

Rata-rata Waiting Time-- 15.666667

Rata-rata Turnaround Time ------- 5.666667

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

D. ALGORITMA PENJADWALAN PRIORITY CPU

```
#include<stdio.h>
main()
{
        int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
        float wtavg, tatavg;
        clrscr();
        printf("Masukkan jumlah proses --- ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                p[i] = i;
                printf("Masukkan Burst Time & Prioritas Proses %d --- ",i);
                scanf("%d %d",&bt[i], &pri[i]);
        }
        for(i=0;i<n;i++)
```

```
                for(k=i+1;k<n;k++)
                        if(pri[i] > pri[k])
                        {
                                temp=p[i];
                                p[i]=p[k];
                                p[k]=temp;

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

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

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

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

INPUT

```
Masukkan jumlah proses --  5
Masukkan Burst Time untuk Proses  0 --  10        3
Masukkan Burst Time untuk Proses  1 --  1         1
Masukkan Burst Time untuk Proses  2 --  2         4
Masukkan Burst Time untuk Proses  3 --  1         5
Masukkan Burst Time untuk Proses  4 --  5         2
```

OUTPUT

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

3	5	1	18	19

Rata-rata Waiting Time-- 8.200000
Rata-rata Turnaround Time ----- 12.000000

1. Tulis program C untuk mengimplementasikan algoritma penjadwalan round robin CPU untuk skenario yang diberikan berikut. Semua proses dalam sistem dibagi menjadi dua kategori - proses sistem dan proses pengguna. Proses sistem harus diberi prioritas lebih tinggi daripada proses pengguna. Pertimbangkan ukuran kuantum waktu untuk proses sistem dan proses pengguna masing-masing 5 msec dan 2 msec.
2. Tulis program C untuk mensimulasikan algoritma penjadwalan SJF CPU pre-emptive

3.5. TUGAS

1. Apa keuntungan dari algoritma penjadwalan round robin CPU?
2. Algoritma penjadwalan CPU manakah untuk sistem operasi real-time?
3. Secara umum, algoritma penjadwalan CPU mana yang bekerja dengan waktu tunggu tertinggi?
4. Apakah mungkin untuk menggunakan algoritma penjadwalan CPU yang optimal dalam praktik?
5. Apa kesulitan sebenarnya dengan algoritma penjadwalan CPU SJF?

PRAKTIKUM 4: ALGORITMA PENJADWALAN

Pertemuan ke 4

Total Alokasi Waktu : 90 menit (Alokasi waktu disesuaikan dengan RPS)
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit
- dst

Total Skor Penilaian : 100% (Bobot skor disesuaikan dengan RPS)
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %
- dst

4.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

Mensimulasikan algoritma penjadwalan CPU non-preemptive untuk menemukan waktu penyelesaian dan waktu tunggu

Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

Dapat mensimulasikan algoritma penjadwalan CPU non-preemptive untuk menemukan waktu penyelesaian dan waktu

4.2. TEORI PENDUKUNG

E. ALGORITMA PENJADWALAN FCFS CPU

Untuk algoritma penjadwalan FCFS, baca jumlah proses / pekerjaan dalam sistem, waktu burst CPU-nya. Penjadwalan dilakukan berdasarkan waktu kedatangan dari proses terlepas dari parameter lainnya. Setiap proses akan dieksekusi sesuai dengan waktu kedatangannya. Hitung waktu tunggu dan waktu penyelesaian masing-masing proses sesuai.

F. ALGORITMA PENJADWALAN SJF CPU

Untuk algoritma penjadwalan SJF, baca jumlah proses / pekerjaan dalam sistem, waktu burst CPU-nya. Atur semua pekerjaan sesuai dengan waktu ledakan mereka. Mungkin ada dua pekerjaan dalam antrian dengan waktu eksekusi yang sama, dan kemudian pendekatan FCFS harus dilakukan. Setiap proses akan dieksekusi sesuai dengan lamanya waktu meledaknya. Kemudian hitung waktu tunggu dan waktu penyelesaian masing-masing proses sesuai.

G. ALGORITMA PENJADWALAN ROUND ROBIN CPU

Untuk algoritma penjadwalan round robin, baca jumlah proses / pekerjaan dalam sistem, waktu burst CPU-nya, dan ukuran slice waktu. Irisan waktu ditugaskan untuk setiap proses dalam porsi yang sama dan dalam urutan melingkar, menangani semua proses eksekusi. Ini memungkinkan setiap proses untuk mendapatkan kesempatan yang sama. Hitung waktu tunggu dan waktu penyelesaian masing-masing proses yang sesuai.

H. ALGORITMA PENJADWALAN CPU PRIORITY

Untuk algoritma penjadwalan prioritas, baca jumlah proses / pekerjaan dalam sistem, waktu burst CPU-nya, dan prioritas. Atur semua pekerjaan agar sesuai dengan prioritas mereka. Mungkin ada dua pekerjaan dalam antrian dengan prioritas yang sama, dan kemudian pendekatan FCFS harus dilakukan. Setiap proses akan dieksekusi sesuai dengan prioritasnya. Hitung waktu tunggu dan waktu penyelesaian masing-masing proses yang sesuai

4.3. ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

5. Komputer.
6. Pemrograman C

4.4. LANGKAH PRAKTIKUM

Asumsikan semua proses tiba pada waktu yang bersamaan

A. ALGORITMA PENJADWALAN FCFS CPU

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

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

INPUT

Masukkan jumlah proses -- 3

Masukkan Burst Time untuk Proses 0 -- 24

Masukkan Burst Time untuk Proses 1 -- 3

Masukkan Burst Time untuk Proses 2 -- 3

OUTPUT

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

Rata-rata Waiting Time-- 17.000000

Rata-rata Turnaround Time ----- 27.000000

B. ALGORITMA PENJADWALAN SJF CPU

```
#include<stdio.h>
#include<conio.h>
main()
{
        int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
        float wtavg, tatavg;
        clrscr();
```

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

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

        wt[0] = wtavg = 0;

        tat[0] = tatavg = bt[0];

        for(i=1;i<n;i++)

        {

                wt[i] = wt[i-1] +bt[i-1];

                tat[i] = tat[i-1] +bt[i];

                wtavg = wtavg + wt[i];

                tatavg = tatavg + tat[i];

                }

        printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

                printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);

        printf("\nAverage Waiting Time -- %f", wtavg/n);

        printf("\nAverage Turnaround Time -- %f", tatavg/n);

        getch();

}
```

INPUT

```
Masukkan jumlah proses --  4
Masukkan Burst Time untuk Proses 0 -- 6
Masukkan Burst Time untuk Proses 1 -- 8
Masukkan Burst Time untuk Proses  2 --  7
```

Masukkan Burst Time untuk Proses 3 -- 3

OUTPUT

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

Rata-rata Waiting Time-- 7.000000

Rata-rata Turnaround Time ----- 13.000000

C. ALGORITMA PENJADWALAN ROUND ROBIN CPU

```
#include<stdio.h>
main()
{
        int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
        float awt=0,att=0,temp=0;
        clrscr();
        printf("Enter the no of processes -- ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter Burst Time for process %d -- ", i+1);
                scanf("%d",&bu[i]);
                ct[i]=bu[i];
        }
        printf("\nEnter the size of time slice -- ");
        scanf("%d",&t);
        max=bu[0];
        for(i=1;i<n;i++)
                if(max<bu[i])
                        max=bu[i];
        for(j=0;j<(max/t)+1;j++)
                for(i=0;i<n;i++)
                        if(bu[i]!=0)
                                if(bu[i]<=t)
                                {
                                        tat[i]=temp+bu[i];
                                        temp=temp+bu[i];
                                        bu[i]=0;
                                }
                                else
                                {
                                        bu[i]=bu[i]-t;
                                        temp=temp+t;
                                }
```

```
for(i=0;i<n;i++)
{
        wa[i]=tat[i]-ct[i];
        att+=tat[i];
        awt+=wa[i];
}
printf("\nThe Average Turnaround time is -- %f",att/n);

printf("\nThe Average Waiting time is -- %f ",awt/n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
for(i=0;i<n;i++)

        printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);

getch();

}
```

INPUT

Masukkan jumlah proses -- 3
Masukkan Burst Time untuk Proses 1 -- 24
Masukkan Burst Time untuk Proses 2 -- 3
Masukkan Burst Time untuk Proses 3 -- 3

Masukkan ukuran potongan waktu (time slice) -- 3

OUTPUT

Rata-rata Waiting Time-- 15.666667

Rata-rata Turnaround Time ------- 5.666667

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

D. ALGORITMA PENJADWALAN PRIORITY CPU

```
#include<stdio.h>
main()
{
        int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
        float wtavg, tatavg;
        clrscr();
        printf("Masukkan jumlah proses --- ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                p[i] = i;
                printf("Masukkan Burst Time & Prioritas Proses %d --- ",i);
                scanf("%d %d",&bt[i], &pri[i]);
```

```
        }
        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(pri[i] > pri[k])
                        {
                                temp=p[i];
                                p[i]=p[k];
                                p[k]=temp;

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

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

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

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

INPUT

```
Masukkan jumlah proses --  5
Masukkan Burst Time untuk Proses  0 --  10        3
Masukkan Burst Time untuk Proses  1 --  1         1
Masukkan Burst Time untuk Proses  2 --  2         4
Masukkan Burst Time untuk Proses  3 --  1         5
Masukkan Burst Time untuk Proses  4 --  5         2
```

OUTPUT

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

2	4	2	16	18
3	5	1	18	19

Rata-rata Waiting Time-- 8.200000
Rata-rata Turnaround Time ----- 12.000000

3. Tulis program C untuk mengimplementasikan algoritma penjadwalan round robin CPU untuk skenario yang diberikan berikut. Semua proses dalam sistem dibagi menjadi dua kategori - proses sistem dan proses pengguna. Proses sistem harus diberi prioritas lebih tinggi daripada proses pengguna. Pertimbangkan ukuran kuantum waktu untuk proses sistem dan proses pengguna masing-masing 5 msec dan 2 msec.
4. Tulis program C untuk mensimulasikan algoritma penjadwalan SJF CPU pre-emptive

4.5. TUGAS

1. Algoritma penjadwalan CPU manakah untuk sistem operasi real-time?
2. Secara umum, algoritma penjadwalan CPU mana yang bekerja dengan waktu tunggu tertinggi?
3. Apakah mungkin untuk menggunakan algoritma penjadwalan CPU yang optimal dalam praktik?
4. Apa kesulitan sebenarnya dengan algoritma penjadwalan CPU SJF?

PRAKTIKUM 5: SINKRONISASI PROSES

Pertemuan ke 5

Total Alokasi Waktu : 90 menit (Alokasi waktu disesuaikan dengan RPS)
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit
- dst

Total Skor Penilaian : 100% (Bobot skor disesuaikan dengan RPS)
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %
- dst

5.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

1. Mengkompilasi kode sumber dengan thread yang berbagi akses ke area global yang tidak diproteksi.
2. Menunjukkan bahwa menulis ke area global yang tidak diproteksi dapat menimbulkan efek samping yang tidak diinginkan ketika diakses oleh thread.
3. Mengkompilasi kode sumber dengan thread yang tersinkronisasi.
4. Menunjukkan bahwa hardware menyediakan bantuan untuk membuat critical region.
5. Mengkompilasi kode sumber dengan mutex untuk membuat wilayah exclusion.
6. Menunjukkan bahwa OS menyediakan bantuan untuk membuat critical region.

Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

7. Dapat mengkompilasi kode sumber dengan thread yang berbagi akses ke area global yang tidak diproteksi.
8. Dapat menunjukkan bahwa menulis ke area global yang tidak diproteksi dapat menimbulkan efek samping yang tidak diinginkan ketika diakses oleh thread.
9. Dapat mengkompilasi kode sumber dengan thread yang tersinkronisasi.
10. Dapat menunjukkan bahwa hardware menyediakan bantuan untuk membuat critical region.
11. Dapat mengkompilasi kode sumber dengan mutex untuk membuat wilayah exclusion.
12. Dapat menunjukkan bahwa OS menyediakan bantuan untuk membuat critical region

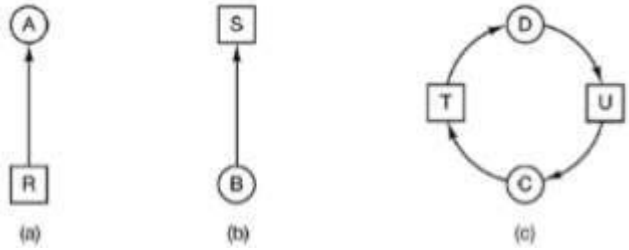
5.2. TEORI PENDUKUNG

Proses konkuren mengakses sumber daya global pada saat yang bersamaan dapat menghasilkan efek samping yang tidak diinginkan. Perangkat keras atau sistem operasi

komputer dapat menyediakan mutual exclusion ketika sumber daya tersebut digunakan bersama.

DEADLOCK

Deadlock yang terjadi ketika proses-proses membutuhkan akses ke lebih dari satu sumber daya (yang tidak dapat dipakai bersama) sering terjadi pada sistem operasi modern.



Gambar 25. Skema Deadlock

Keterangan:

a. Proses A sedang memakai (allocate) sumber daya R
b. Proses B meminta (request) sumber daya S
c. Proses C meminta sumber daya T yang sedang dipakai oleh D, namun D tidak (akan) melepas D karena sedang meminta sumber daya U yang dipakai oleh C.

OS dapat menerapkan berbagai teknik unuk menghindari, mencegah, atau mengatasi deadlock yang akan menyebabkan sistem berjalan se-efisien mungkin.

5.3. ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

7. Komputer.
8. CPU simulator

5.4. LANGKAH PRAKTIKUM

Asumsikan semua proses tiba pada waktu yang bersamaan

Jalankan Simulator.

1. Tuliskan kode berikut ini pada compiler.

```
program CriticalRegion
        var g integer

        sub thread1 as thread
             writeln("In thread1")
             g = 0
             for n = 1 to 20
                  g = g + 1
             next
             writeln("thread1 g = ", g)
             writeln("Exiting thread1")
        end sub

        sub thread2 as thread
             writeln("In thread2")
             g = 0
             for n = 1 to 12
                  g = g + 1
             next
             writeln("thread2 g = ", g)
             writeln("Exiting thread2")
        end sub

        writeln("In main")

        call thread1
        call thread2

        wait

        writeln("Exiting main")
end
```

Kode diatas akan membuat dua thread yaitu thread1 dan thread2. Masing-masing thread menaikkan nilai variabel global g dalam tiap loop. Ketika dua loop selesai nilai apa yang anda perkirakan dalam variabel g untuk dua kasus tersebut?

Dapatkah anda menduga fungsi statement **wait**?

a. Sekarang compile kode tersebut, muatkan dalam memori.
b. Tampilkan jendela konsol, aktifkan STAY ON TOP.
c. Beralihlah ke OS simulator, akan tampak CriticalRegion pada PROGRAM LIST.
d. Buatlah proses dengan ketentuan : ROUND ROBIN, 10 ticks, kecepatan maksimum.
e. Tekan START.
f. Ketika program berhenti, catatlah dua nilai g yang ditampilkan. Apakah nilai ini seperti yang anda perkirakan tadi? Bagaimana pendapat anda.

2. Sekarang kita memodifikasi program seperti dibawah ini. Bagian yang dimodifikasi ditandai dengan tebal (bold).

```
program CriticalRegion
        var g integer

        sub thread1 as thread synchronise
             writeln("In thread1")
             g = 0
             for n = 1 to 20
                  g = g + 1
             next
             writeln("thread1 g = ", g)
             writeln("Exiting thread1")
        end sub

        sub thread2 as thread synchronise
             writeln("In thread2")
             g = 0
             for n = 1 to 12
                  g = g + 1
             next
             writeln("thread2 g = ", g)
             writeln("Exiting thread2")
        end sub

        writeln("In main")

        call thread1
        call thread2

        wait

        writeln("Exiting main")
end
```

a. Hapus kode sebelumnya dengan cara: pada jendela CPU Simulator, klik REMOVE ALL PROGRAMS.
b. Compile kode diatas dan muatkan ke memori. Pada jendela compiler, carilah baris pada *program code (output)* yang menerjemahkan **synchronise**, catatlah kode asembler beserta keterangannya.
c. Pada jendela OS simulator buatlah proses baru, kemudian jalankan.
d. Tunggu hingga program selesai, sembari mengamati yang terjadi pada sub jendela waiting processes,
e. Catat dua nilai variabel g yang ditampilkan pada layar consol.

3. Kita modifikasi lagi kode sebelumnya. Bagian yang ditambahkan ditandai dengan tebal.

```
program CriticalRegion
        var g integer

        sub thread1 as thread
             writeln("In thread1")
             enter
             g = 0
             for n = 1 to 20
                  g = g + 1
             next
             writeln("thread1 g = ", g)
             leave
             writeln("Exiting thread1")
        end sub

        sub thread2 as thread
             writeln("In thread2")
             enter
             g = 0
             for n = 1 to 12
                  g = g + 1
             next
             writeln("thread2 g = ", g)
             leave
             writeln("Exiting thread2")
        end sub

        writeln("In main")

        call thread1
        call thread2

        wait
        writeln("Exiting main")
end
```

a. Hapus program sebelumnya dari memori.
b. Compile kode diatas, muatkan ke memori. Amati hasil kompilasi perintah enter dan leave pada jendela compiler output
c. Beralihlah ke OS simulator, buat proses baru, kemudian jalankan.
d. Catat 2 nilai variabel g.

4. Jadi apa kesimpulannya? Untuk memahami percobaan diatas, jawablah pertanyaanpertanyaan berikut ini:
   a. Jelaskan tujuan utama praktimum ini menurut anda.
   b. Mengapa kita menggunakan variabel global (g) yang sama pada dua thread?
   c. Sudahkan kita menggunakan variabel lokal, dan apakah hasilnya akan berbeda? Lakukan eksperimen kecil dengan sedikit modifikasi pada kode yang ada dan jalankanlah program tersebut.

d. Pada modifikasi yang pertama ditambahkan kata kunci synchronise. Jelaskan tujuan modifikasi ini. Beri contoh istilah untuk synchronise dalam bahasa pemrograman nyata.
e. Pada modifikasi yang kedua digunakan kata kunci enter dan leave. Jelaskan fungsi modifikasi ini, dan apa bedanya dengan (d)?
f. Critical regions seringkali diimplementasikan menggunakan semaphore dan mutex. Jelaskan pengertiannya dan apa perbedaannya.
g. Berikan contoh nyata untuk suatu critical region (atau mutex region). Beri contoh nyata suatu mutex.
h. Beberapa arsitektur komputer memiliki instruksi “test-and-set” untuk menerapkan critical region. Jelaskan bagaimana teknik ini bekerja dan mengapa penerapannya pada hardware.
i. Jika hardware maupun OS tidak menyediakan bantuan, bagaimana anda dapat memproteksi critical region dalam kode anda? Berikan saran anda dan jelaskan dimana perbedaannya dengan metode-metode yang telah diuji coba kan diatas.

Langkah Praktikum Deadlock

1. Pada kertas, buatlah suatu siklus *resource allocation graph* untuk 4 proses dengan 4 sumber daya (bisa sumber daya apa saja). Hanya empat sumber daya dengan tiap proses menggunakan satu sumber daya dan meminta sumber daya yang lainnya. Kondisi ini akan menyebabkan deadlock.
   a. Gambarlah graf untuk skenario ini dan jelaskan bagaimana deadlock dapat terjadi.
   b. Berapa proses yang mengalami deadlock untuk kasus ini?

   Jalankan Simulator.

2. Tuliskan kode berikut ini pada compiler.

```
Program Deadlocks
     while true
          n = 1
     wend
end
```

   a. Kode diatas akan menjalankan loop selamanya.
   b. Compile program tersebut, muatkan ke memori.
   c. Beralihlah ke OS simulator.
   d. Sorot program Deadlocks pada PROGRAM LIST, klik tombol CREATE NEW PROCESS.
   e. Pastikan menggunakan ROUND ROBIN dengan 15 ticks.
   f. Maksimalkan kecepatan simulator. (**jangan klik START dulu**)
   g. Kemudian ikuti langkah-langkah berikutnya.

3. Simulator dapat mengalokasikan dan men-dealokasikan beberapa sumber daya secara imajiner untuk proses sehingga kita dapat mempelajari permasalahan terkait pengalokasian sumber daya dan deadlock.

- Buatlah satu proses lagi sehingga akan ada dua proses pada antrian ready.
- Biarkan berada pada antrian ready (**jangan klik START dulu**)
- Klik tombol VIEW RESOURCES... pada jendela OS simulator, akan muncul jendela System Resources. Ada 4 sumber daya yang teridentifikasi yaitu R0, R1,R2, dan R3.
- Gambar yang mewakili sumber daya berwarna hijau yang artinya sumber daya tersebut belum dialokasikan ke suatu proses.
- Klik STAY ON TOP pada jendela resource.
- Kemudian pilih proses pada antrian ready dan klik tombol *allocate*.
- Akan tampak nomor ID proses pada kotak USED BY dan warna sumber daya akan berubah menjadi kuning.
- Kemudian klik (ID) proses yang lain dan lakukan hal yang sama, maka nomor ID akan tampak dalam kotak daftar *requested by*, dan warna menjadi merah. Jelaskan yang terjadi.
- Berlatihlah membuat kondisi deadlock untuk 2 sumber daya.
- Sekarang, klik tombol RELEASE ALL untuk men-dealokasikan sumber daya. Sumber daya seharusnya menjadi hijau kembali dan nomor ID proses hilang.
- Hapus proses pada antrian ready. Untuk melakukannya, klik tombol CLEAR pada jendela READY PROCESSES.

4. Setelah memahami mekanisme pengalokasian sumber daya menggunakan simulator, berikutnya buatlah 4 proses yang berkaitan dengan graf alokasi sumber daya yang tadi anda buat. Kemudian alokasikan sumber daya tersebut untuk menciptakan kondisi deadlock. Bagaimana hasilnya? Jika berhasil, apa yang anda lihat?

5. Sekali deadlock terbentuk, cobalah menjalankan proses dengan klik pada tombol START. Jelaskan yang terjadi. Proses pada antrian ready belum mengalami deadlock, kejadian tersebut terjadi ketika proses telah berjalan.

6. Dalam kondisi proses yang deadlock, apa yang dapat anda lakukan untuk mengatasinya? Berikan dua cara untuk mengatasinya, kemudian praktikkan menggunakan simulator.

5.5. TUGAS

Beri penjelasan dan kesimpulan terhadap hasil simulasi deadlock yang telah dilakukan diatas!

PRAKTIKUM 6: BANKIR ALGORITHM FOR DEADLOCK

Pertemuan ke 6

Total Alokasi Waktu : 90 menit (Alokasi waktu disesuaikan dengan RPS)
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit
- dst

Total Skor Penilaian : 100% (Bobot skor disesuaikan dengan RPS)
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %
- dst

6.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

Mensimulasikan algoritma Bankir untuk tujuan penghindaran kebuntuan

Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

Dapat mensimulasikan algoritma Bankir untuk tujuan penghindaran kebuntuan

6.2. TEORI PENDUKUNG

Dalam lingkungan multi-pemrograman, beberapa proses dapat bersaing untuk sejumlah sumber daya yang terbatas. Suatu proses meminta sumber daya; jika sumber daya tidak tersedia pada waktu itu, proses memasuki keadaan menunggu. Kadang-kadang, proses menunggu tidak pernah lagi dapat mengubah keadaan, karena sumber daya yang diminta dimiliki oleh proses menunggu lainnya. Situasi ini disebut jalan buntu. Menghindari kebuntuan adalah salah satu teknik untuk menangani kebuntuan. Pendekatan ini mensyaratkan bahwa sistem operasi terlebih dahulu diberikan informasi tambahan mengenai sumber daya mana yang akan diminta dan digunakan oleh suatu proses selama masa pakainya. Dengan pengetahuan tambahan ini, dapat memutuskan untuk setiap permintaan apakah proses harus menunggu atau tidak. Untuk memutuskan apakah permintaan saat ini dapat dipenuhi atau harus ditunda, sistem harus mempertimbangkan sumber daya yang tersedia saat ini, sumber daya yang saat ini dialokasikan untuk setiap proses, dan permintaan di masa mendatang dan pelepasan setiap proses. Algoritme Banker adalah algoritme penghindaran kebuntuan yang berlaku untuk sistem dengan banyak instance dari setiap jenis sumber daya.

6.3. ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

9. Komputer.
10. Pemrograman C.

6.4. LANGKAH PRAKTIKUM

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

                printf("\nEnter New Request Details -- ");

                printf("\nEnter pid \t -- \t");

                scanf("%d",&id);

                printf("Enter Request for Resources \t -- \t");

                for(i=0;i<r;i++)  {

                        scanf("%d",&newr);

                        f[id].all[i] += newr;


                        avail[i]=avail[i] - newr;

                }

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

                         for(j=0;j<r;j++)  {
```

```
f[i].need[j]=f[i].max[j]-f[i].all[j];
if(f[i].need[j]<0)
        f[i].need[j]=0;
        }
}
cnt=0;
fl=0;
while(cnt!=n)
 {
        g=0;
        for(j=0;j<n;j++) {
                if(f[j].flag==0)  {
                        b=0;
                        for(p=0;p<r;p++)
                        {
                                if(avail[p]>=f[j].need[p])
                                b=b+1;
                else
                         b=b-1;
                }
                if(b==r)
                {
                        printf("\nP%d is visited",j);
                        seq[fl++]=j;
                        f[j].flag=1;
                        for(k=0;k<r;k++)
                                avail[k]=avail[k]+f[j].all[k];
                        cnt=cnt+1;
                        printf("(");
                        for(k=0;k<r;k++)
                                printf("%3d",avail[k]);
                        printf(")");
                        g=1;
```

```
                                        }
                                    }
                                }
                    if(g==0)
                    {
                    printf("\n REQUEST NOT GRANTED -- DEADLOCK
                OCCURRED");
                    printf("\n SYSTEM IS IN UNSAFE STATE");
                    goto y;
                     }
                }
                printf("\nSYSTEM IS IN SAFE STATE");
                printf("\nThe Safe Sequence is -- (");
                for(i=0;i<fl;i++)
                printf("P%d  ",seq[i]);
                printf(")");
                y: printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");
                for(i=0;i<n;i++)  {
                    printf("P%d\t",i);
                    for(j=0;j<r;j++)
                        printf("%6d",f[i].all[j]);
                    for(j=0;j<r;j++)
                        printf("%6d",f[i].max[j]);
                    for(j=0;j<r;j++)
                        printf("%6d",f[i].need[j]);
                    printf("\n");
                    }
            getch();
        }
```

INPUT

Masukkan jumlah proses - 5

Masukkan jumlah sumber daya - 3

Masukkan detail untuk P0

Masukkan alokasi - 0 1 0

Masukkan Maks - 7 5 3

Masukkan detail untuk P1

Masukkan alokasi - 2 0 0

Masukkan Maks - 3 2 2

Masukkan detail untuk P2

Masukkan alokasi - 3 0 2

Masukkan Maks - 9 0 2

Masukkan detail untuk P3

Masukkan alokasi - 2 1 1

Masukkan Maks - 2 2 2

Masukkan detail untuk P4

Masukkan alokasi - 0 0 2

Masukkan Maks - 4 3 3

Masukkan Sumber Daya yang Tersedia - 3 3 2

Masukkan Detail Permintaan Baru –

Masukkan pid - 1

Masukkan Permintaan untuk Sumber Daya - 1 0 2

OUTPUT

P1 dikunjungi (5 3 2)

P3 dikunjungi (7 4 3)

P4 dikunjungi (7 4 5)

P0 dikunjungi (7 5 5)

P2 dikunjungi (10 5 7)

SISTEM DALAM KEADAAN AMAN

Safe Sequence adalah - (P1 P3 P4 P0 P2)

Process	Allocation	Max	Need
P0	0 1 0	7 5 3	7 4 3
P1	3 0 2	3 2 2	0 2 0
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1

6.5. TUGAS

1. Bagaimana grafik alokasi sumber daya dapat digunakan untuk mengidentifikasi situasi jalan buntu?
2. Bagaimana kelebihan penggunaan algoritma Bankir disbanding teknik grafik alokasi sumber daya?
3. Jelaskan perbedaan deadlock avoidance dengan deadlock prevention?
4. Tulis program C untuk menerapkan teknik deteksi kebuntuan untuk skenario berikut?
    a. Satu contoh dari setiap jenis sumber daya
    b. Banyak contoh dari setiap jenis sumber daya

PRAKTIKUM 7: PENJADWALAN PROSES DAN MANAJEMEN MEMORI

Pertemuan ke 7

Total Alokasi Waktu : 90 menit (Alokasi waktu disesuaikan dengan RPS)
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit
- dst

Total Skor Penilaian : 100% (Bobot skor disesuaikan dengan RPS)
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %
- dst

7.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

1. Mendeskripsikan keadaan proses dan transisi akibat timeout.
2. Menjelaskan perbedaan antara penjadwalan pre-emptive dan non-pre-emptive.
3. Menelusuri nilai register PC dalam PCB suatu proses ketika berada dalam antrian Ready.
4. Menjelaskan pemanfaatan nilai Register PC pada penjadwalan Round Robin.
   Menjelaskan page swapping ketika proses berpindah dari keadaan ready ke running

Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

1. Dapat mendeskripsikan keadaan proses dan transisi akibat timeout.
2. Dapat menjelaskan perbedaan antara penjadwalan pre-emptive dan non-pre-emptive.
3. Dapat menelusuri nilai register PC dalam PCB suatu proses ketika berada dalam antrian Ready.
4. Dapat menjelaskan pemanfaatan nilai Register PC pada penjadwalan Round Robin.
5. Dapat menjelaskan page swapping ketika proses berpindah dari keadaan ready ke running

7.2. TEORI PENDUKUNG

Pada materi kuliah dibahas mengenai berbagai jenis penjadwalan proses dan manajemen memori. Praktikum ini berdasarkan materi tersebut.

7.3. ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

11. Komputer.
12. CPU Simulator.

7.4. LANGKAH PRAKTIKUM

A. Penjadwalan

1. Masukkan kode ini pada compiler :

```
program LoopForeverTest
     While true
          N = N + 1
     wend
end
```

   a. Compile kode diatas, kemudian muatkan ke memori.
   b. Beralihlah ke jendela OS Simulator.
   c. Pilih program LoopForeverTest, selanjutnya kita akan membuat 3 proses, namun sebelum tiap proses dibuat aturlah nilai LIFETIME dan pilih SECS untuk tiap proses: **10 seconds, 32 seconds, 6 seconds**.
   d. Berikutnya memilih nilai timeslot melalui menu pull-down pada sub jendela SCHEDULER/POLICIES/RR Time Slice/Secs, pilih 4.
   e. Tekan START dan tunggu hingga semua proses selesai.
   f. Bukalah jendela OS ACTIVITY LOG dengan klik pada tombol VIEW LOG... Amati data log yang relevan kemudian perhatikan urutan pada proses yang berjalan (running processes) dan catat waktu yang dihabiskan oleh tiap proses selama dalam keadaan running.

2. Sekarang, beralihlah ke compiler dan masukkan kode berikut ini. Klik NEW untuk membuat tab editor baru.

```
program LoopForeverTest2
     while true
          n = n + 1
          i = i + n
          p = n + i + 5
     wend
end
```

   a. Compile kemudian muatkan ke memori kode diatas
   b. Beralihlah ke OS Simulator.
   c. Pada program list akan tampak 2 program: LoopForeverTest, dan LoopForeverTest2.
   d. Klik **LoopForeverTest** dan buatlah satu proses.

e. Klik **LoopForeverTest2** dan buat satu proses.
f. Sekarang pada antrian ready seharusnya ada 2 proses.
g. Pilih penjadwalan ROUND ROBIN, tipe prioritas NON-PREEMPTIVE dan RR Time Slice adalah **10 tick**.
h. Atur pada setengah kecepatan simulasi.

Lakukan langkah-langkah percobaan berikut ini:

1. Sorot proses kemudian klik tombol PCB... amati nilai pada PC REGISTERS (pada PCB Info) pada masing-masing proses, dan catatlah.
2. Klik **START** dan siapkan mouse anda pada tombol SUSPEND (jangan klik).
3. Sewaktu running process kembali ke ready queue, segera klik tombol SUSPEND.
4. Sorot proses pada antrian ready, dan klik tombol PCB... pada antrian ready. Catatlah nilai PC REGISTER nya.
5. Sekarang, beri check pada SUSPEND ON RUN dalam tampilan RUNNING PROCESS. Kurangi kecepatan OS simulation. Klik tombol RESUME dan amati ketika proses yang antri kembali dalam keadaan running, yang menyebabkan simulasi **berhenti (suspend) secara otomatis**.
6. Perhatikan baik-baik pada kotak PC REGISTER dalam **jendela CPU Simulator**. Sekarang klik RESUME pada OS Simulator. Catatlah nilai pada PC REGISTER ketika nilainya berubah. Berikan pendapat anda terhadap nilai tersebut dan jelaskan apa yang terjadi.

B. Manajemen Memori

1. Masih menggunakan ROUND-ROBIN (RR) dan NON-PREEMPTIVE pada OS Simulator.
2. Masukkan kode berikut ini pada compiler :

```
program ForeverLoop
     While true
          I = 1
     wend
end
```

   compile kode diatas, kemudian muatkan ke memori. Beralihlah ke jendela OS Simulator.

3. Klik tombol VIEW MEMORY.... aktifkan STAY ON TOP. Matikan PAGING ENABLED (dengan menghilangkan tanda check). Pilih program ForeverLoop, buatlah proses dengan ketentuan berikut :

Proses	Page
P1	3
P2	5

P3	4
P4	2
P5	6

Perhatikan 4 proses yang terdapat pada jendela READY PROCESS.
Amati data pada kolom SWAP. Beberapa tertulis "Yes", jelaskan artinya.
Bandingkanlah dengan tampilan *page* yang tampil pada jendela MAIN MEMORY.

4. Klik VIEW UTILIZATION... informasi apa yang anda lihat? Klik STAY ON TOP. Sekarang, pada jendela OS Simulator, atur kecepatan pada kisaran 90, tekan START.
   Amati hal-hal berikut ini dan buatlah catatan:
   1. Jelaskan kejadian yang tampak pada jendela MAIN MEMORY (RAM).
   2. Apa yang tampak pada kolom SWAP di jendela RUNNING PROCESSES? Mengapa selalu sama? Apakah ada kalanya berbeda dengan yang ditampilkan pada kolom SWAP di jendela READY PROCESSES? Jelaskan.
   3. Amati bagaimana proses-proses yang ready berganti (swap) keadaan dari waktu ke waktu. Beri penjelasannya.
   4. Informasi apa yang anda peroleh dari jendela RESOURCE UTILISATION terkait dengan manajemen memori?

7.5. TUGAS

Buatlah penjadwalan proses dengan membuat 3 proses, namun sebelum tiap proses dibuat aturlah nilai LIFETIME dan pilih SECS untuk tiap proses: **15 seconds, 37 seconds, 11 seconds**. Dengan langkah-langkah sesuai dengan contoh diatas!

PRAKTIKUM 8: TEKNIK ALOKASI MEMORY

Pertemuan ke 8

Total Alokasi Waktu : 90 menit
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit

Total Skor Penilaian : 100%
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %

8.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

Mensimulasikan teknik alokasi memori yang berdekatan berikut ini

a) Worst-fit b) Best-fit c) First-fit

Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

Dapat mensimulasikan teknik alokasi memori yang berdekatan berikut ini

a) Worst-fit b) Best-fit c) First-fit

8.2. TEORI PENDUKUNG

Salah satu metode paling sederhana untuk alokasi memori adalah membagi memori menjadi beberapa partisi berukuran tetap. Setiap partisi dapat berisi tepat satu proses. Dalam metode multi-partisi ini, ketika sebuah partisi bebas, suatu proses dipilih dari antrian input dan dimuat ke dalam partisi bebas. Ketika proses berakhir, partisi menjadi tersedia untuk proses lain. Sistem operasi menyimpan tabel yang menunjukkan bagian memori mana yang tersedia dan mana yang ditempati. Akhirnya, ketika suatu proses tiba dan membutuhkan memori, bagian memori yang cukup besar untuk proses ini disediakan. Ketika tiba saatnya untuk memuat atau menukar proses ke dalam memori utama, dan jika ada lebih dari satu blok memori bebas dengan ukuran yang cukup, maka sistem operasi harus memutuskan blok bebas mana yang akan dialokasikan. Strategi paling cocok memilih blok yang paling dekat dengan permintaan. First-fit memilih blok pertama yang tersedia yang cukup besar. Worst-fit memilih blok terbesar yang tersedia

8.3. ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

13. Komputer.
14. Program C.

8.4. LANGKAH PRAKTIKUM

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

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

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

```
        getch();
}
```

INPUT

Masukkan jumlah blok : 3
Masukkan jumlah file : 2

Masukkan ukuran blok : -
Blok 1 : 5
Blok 2 : 2
Blok 3 : 7

Masukkan ukuran file :-
File 1 : 1
File 2 : 4

OUTPUT

Jumlah File	Ukuran File	Jumlah Blok	Ukuran Blok	Fragment
1	1	1	5	4
2	4	3	7	3

A. BEST-FIT

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

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

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

printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)  {
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++) {
```

```
            for(j=1;j<=nb;j++)  {
                    if(bf[j]!=1) {
                            temp=b[j]-f[i];
                            if(temp>=0)
                            if(lowest>temp)  {
                                    ff[i]=j;
                                    lowest=temp;
                             }
                     }
            }
            frag[i]=lowest;
            bf[ff[i]]=1;
            lowest=10000;
    }
    printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
    for(i=1;i<=nf && ff[i]!=0;i++)
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
    getch();
}
```

INPUT

Masukkan jumlah blok : 3
Masukkan jumlah file : 2

Masukkan ukuran blok : -
Blok 1 : 5
Blok 2 : 2
Blok 3 : 7

Masukkan ukuran file :-
File 1 : 1
File 2 : 4

OUTPUT

Jumlah File	Ukuran File	Jumlah Blok	Ukuran Blok	Fragment
1	1	2	2	1
2	4	1	5	1

B. FIRST-FIT

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

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

```
 printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)  {
                printf("Block %d:",i);
                scanf("%d",&b[i]);
        }
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)  {
        printf("File %d:",i);
        scanf("%d",&f[i]);
        }
        for(i=1;i<=nf;i++) {
                for(j=1;j<=nb;j++)   {
                        if(bf[j]!=1)   //if bf[j] is not allocated
                        {
                                temp=b[j]-f[i];
                                if(temp>=0)
                                        if(highest<temp)
                                        {
                                        ff[i]=j; highest=temp;
                                }
                        }
                }
                frag[i]=highest;
                bf[ff[i]]=1;
                highest=0;
                }
                printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
                for(i=1;i<=nf;i++)
                        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
                getch();
        }
```

INPUT

Masukkan jumlah blok : 3
Masukkan jumlah file : 2

Masukkan ukuran blok : -
Blok 1 : 5
Blok 2 : 2
Blok 3 : 7

Masukkan ukuran file :-
File 1 : 1

File 2 : 4

OUTPUT

Jumlah File	Ukuran File	Jumlah Blok	Ukuran Blok	Fragment
1	1	3	7	6
2	4	1	5	1

8.5. TUGAS

1. Manakah dari strategi alokasi memori dinamis yang berdekatan yang mengalami fragmentasi eksternal?
2. Apa solusi yang mungkin untuk masalah fragmentasi eksternal?
3. Apa itu aturan 50 persen?
4. Apa itu pemadatan?
5. Manakah dari teknik alokasi memori yang paling cocok, paling cocok, paling buruk yang efisien? Mengapa?
6. Tulis program C untuk menerapkan teknik pemadatan!

PRAKTIKUM 9: SIMULASI PAGING

Pertemuan ke 9

Total Alokasi Waktu : 90 menit
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit

Total Skor Penilaian : 100%
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %

9.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

Mensimulasikan teknik paging manajemen memori

Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

Dapat mensimulasikan teknik paging manajemen memori

9.2. TEORI PENDUKUNG

Dalam sistem operasi komputer, paging adalah salah satu skema manajemen memori di mana komputer menyimpan dan mengambil data dari penyimpanan sekunder untuk digunakan dalam memori utama. Dalam skema manajemen memori paging, sistem operasi mengambil data dari penyimpanan sekunder dalam blok ukuran yang sama yang disebut halaman. Paging adalah skema manajemen memori yang memungkinkan ruang alamat fisik suatu proses menjadi tidak bersebelahan. Metode dasar untuk menerapkan paging melibatkan memecah memori fisik menjadi blok berukuran tetap yang disebut bingkai dan memecah memori logis menjadi blok dengan ukuran yang sama yang disebut halaman. Ketika suatu proses akan dieksekusi, halaman-halamannya dimuat ke dalam setiap frame memori yang tersedia dari sumbernya.

9.3. ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

15. Komputer.
16. Program C.

9.4. LANGKAH PRAKTIKUM

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

```
main() {
int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
int s[10], fno[10][20];

clrscr();

printf("\nEnter the memory size -- ");
scanf("%d",&ms);

printf("\nEnter the page size -- ");
scanf("%d",&ps);

 nop = ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);

printf("\nEnter number of processes -- ");
scanf("%d",&np);

rempages = nop;

for(i=1;i<=np;i++)  {

        printf("\nEnter no. of pages required for p[%d]-- ",i);
        scanf("%d",&s[i]);

        if(s[i] >rempages)  {
                printf("\nMemory is Full");
                break;
        }
        rempages = rempages - s[i];

        printf("\nEnter pagetable for p[%d] --- ",i);
        for(j=0;j<s[i];j++)
        scanf("%d",&fno[i][j]);
}
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset --  ");

scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
        printf("\nInvalid Process or Page Number or offset");
else  {
        pa=fno[x][y]*ps+offset;
        printf("\nThe Physical Address is -- %d",pa);
}
getch();
}
```

INPUT
Masukkan ukuran memori - 1000
Masukkan ukuran halaman - 100
Jumlah halaman yang tersedia di memori adalah - 10
Masukkan jumlah proses - 3
Masukkan jumlah halaman yang diperlukan untuk p [1] - 4
Masukkan pagetable untuk p [1] --- 8 6 9 5

Masukkan jumlah halaman yang diperlukan untuk p [2] - 5
Masukkan tabel halaman untuk p [2] --- 1 4 5 7 3

Masukkan jumlah halaman yang dibutuhkan untuk hal [3] - 5

OUTPUT
Memori Penuh

Masukkan Alamat Logis untuk menemukan
Alamat Fisik Masukkan jumlah proses dan jumlah tenaga dan offset - 2 3 60
Alamat Fisik adalah - 760

9.5. TUGAS

1. Membedakan antara teknik alokasi memori paging dan segmentasi?
2. Apa tujuan dari tabel halaman/page table?
3. Apakah teknik manajemen memori paging menderita masalah fragmentasi internal atau eksternal. Mengapa?
4. Apa efek dari paging pada keseluruhan waktu pengalihan konteks!
5. Tulis program C untuk mensimulasikan teknik paging dua tingkat!
6. Tulis program C untuk mensimulasikan teknik manajemen memori segmentasi!

PRAKTIKUM 10: FILE MANAGEMENT

Pertemuan ke 10

Total Alokasi Waktu : 90 menit
- Pre-Test : 10 menit
- Praktikum : 70 menit
- Post-Test : 10 menit

Total Skor Penilaian : 100%
- Pre-Test : 25 %
- Praktikum : 40 %
- Post-Test : 35 %

10.1. TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan: (sesuaikan dengan RPS) PLO (Prodi) dan CLO (masing2 Kuliah)

Mensimulasikan teknik-teknik pengorganisasian file berikut ini

a) Single level directory b) Two level directory c) Hierarchical

Indikator ketercapaian diukur dengan: (sesuaikan dengan RPS)

Dapat mensimulasikan teknik-teknik pengorganisasian file berikut ini

a) Single level directory b) Two level directory c) Hierarchical

10.2. TEORI PENDUKUNG

Struktur direktori adalah pengorganisasian file ke dalam hierarki folder. Dalam sistem direktori satu tingkat, semua file ditempatkan dalam satu direktori. Ada direktori root yang memiliki semua file. Ini memiliki arsitektur yang sederhana dan tidak ada sub direktori. Keuntungan dari sistem direktori tingkat tunggal adalah mudahnya menemukan file dalam direktori. Dalam sistem direktori dua tingkat, setiap pengguna memiliki direktori file pengguna sendiri (UFD). Sistem mempertahankan blok master yang memiliki satu entri untuk setiap pengguna. Blok master ini berisi alamat direktori pengguna. Ketika pekerjaan pengguna dimulai atau pengguna masuk, direktori file master sistem (MFD) dicari. Ketika seorang pengguna merujuk ke file tertentu, hanya UFD-nya sendiri yang dicari. Ini secara efektif menyelesaikan masalah tabrakan nama dan mengisolasi pengguna dari satu sama lain. Struktur direktori hirarkis memungkinkan pengguna untuk membuat subdirektori sendiri dan mengatur file mereka sesuai. Pohon adalah struktur direktori yang paling umum. Pohon memiliki direktori root, dan setiap file dalam sistem memiliki nama jalur yang unik. Direktori (atau subdirektori) berisi sekumpulan file atau subdirektori.

10.3. ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

17. Komputer.
18. Program C.

10.4. LANGKAH PRAKTIKUM

A. ORGANISASI DIREKTORI LEVEL SATU

```
#include<stdio.h>

struct
{
    char dname[10],fname[10][10];
    int fcnt;
}dir;


    void main() {

    int i,ch;

    char f[30];

    clrscr();

    dir.fcnt = 0;

    printf("\nEnter name of directory -- ");

    scanf("%s", dir.dname);

    while(1)  {

            printf("\n\n1. Create File\t2. Delete File\t3. Search File \n
                                    4. Display Files\t5. Exit\nEnter your choice -- ");

            scanf("%d",&ch);

            switch(ch)  {

                case 1 : printf("\nMasukkan nama file -- ");

                        scanf("%s",dir.fname[dir.fcnt]);

                        dir.fcnt++;

                        break;

                case 2 : printf("\nMasukkan nama file-- ");

                        scanf("%s",f);

                        for(i=0;i<dir.fcnt;i++)

                        {

                        if(strcmp(f, dir.fname[i])==0)

                        {
```

```
                printf("File %s terhapus",f);
                strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
                break;
                }
        }
        if(i==dir.fcnt)
                printf("File %s tidak ditemukan",f);
        else
                dir.fcnt--;
        break;

case 3 : printf("\nMasukkan nama file-- ");
        scanf("%s",f);
        for(i=0;i<dir.fcnt;i++)
        {
                if(strcmp(f, dir.fname[i])==0)
                {
                printf("File %s ditemukan ", f);
                break;
                }
        }
        if(i==dir.fcnt)
        printf("File %s tidak ditemukan",f);
        break; case 4: if(dir.fcnt==0)
        printf("\nDirektori kosong");
        else
        {
                printf("\nThe Files are -- ");
                for(i=0;i<dir.fcnt;i++)
                        printf("\t%s",dir.fname[i]);
                }
                break;
```

```
                default: exit(0);
                }
        }
        getch();
}
```

OUTPUT:

Masukkan nama direktori - CSE

1. Buat File 2. Hapus File 3. Cari File
4. Tampilkan File 5. Keluar Masukkan pilihan Anda - 1

Masukkan nama file - A

1. Buat File 2. Hapus File 3. Cari File
4. Tampilkan File 5. Keluar Masukkan pilihan Anda - 1

Masukkan nama file - B

1. Buat File 2. Hapus File 3. Cari File
4. Tampilkan File 5. Keluar Masukkan pilihan Anda - 1

Masukkan nama file - C

1. Buat File 2. Hapus File 3. Cari File
4. Tampilkan File 5. Keluar Masukkan pilihan Anda - 4

File adalah - A B C

1. Buat File 2. Hapus File 3. Cari File
4. Tampilkan File 5. Keluar Masukkan pilihan Anda - 3

Masukkan nama file - File ABC
Tidak ditemukan ABC

1. Buat File 2. Hapus File 3. Cari File
4. Tampilkan File 5. Keluar Masukkan pilihan Anda - 2

Masukkan nama file – B

File B terhapus

1. Buat File 2. Hapus File 3. Cari File
4. Tampilkan File 5. Keluar Masukkan pilihan Anda - 5

B. ORGANISASI DIREKTORI LEVEL DUA

```
#include<stdio.h>
struct
 {
     char dname[10],fname[10][10];
     int fcnt; }dir[10];

void main() {
     int i,ch,dcnt,k;
     char f[30], d[30];
     clrscr();  dcnt=0;

     while(1)  {
               printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
               printf("\n4. Search File\t\t5. Display\t6. Exit\t Enter your choice -- ");
               scanf("%d",&ch);
     switch(ch)  {
               case 1 : printf("\nEnter name of directory -- ");
                        scanf("%s", dir[dcnt].dname);
                        dir[dcnt].fcnt=0;     dcnt++;
                        printf("Directory created");
                        break;
               case 2: printf("\nEnter name of the directory -- ");
                        scanf("%s",d);     for(i=0;i<dcnt;i++)
                        if(strcmp(d,dir[i].dname)==0)
                        {
                                 printf("Enter name of the file -- ");
                                 scanf("%s",dir[i].fname[dir[i].fcnt]);
                                 dir[i].fcnt++;
                                 printf("File created");
                                 break;
                         }
                        if(i==dcnt)
                        printf("Directory %s not found",d);
                         break;
                        case 3: printf("\nEnter name of the directory -- ");
                                 scanf("%s",d);
                                 for(i=0;i<dcnt;i++)
                                  {
                                  if(strcmp(d,dir[i].dname)==0)
                                 {
                                          printf("Enter name of the file -- ");
                                          scanf("%s",f);
                                 for(k=0;k<dir[i].fcnt;k++)
                                 {
                                 if(strcmp(f, dir[i].fname[k])==0)
                                 {
                                          printf("File %s is deleted ",f);
```

```
                dir[i].fcnt--;
                strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
                goto jmp;
                }
        }
                printf("File %s not found",f);
                goto jmp;
                }
        }
        printf("Directory %s not found",d);
        jmp  : break;

case 4: printf("\nEnter name of the directory -- ");
        scanf("%s",d);
        for(i=0;i<dcnt;i++)
        {
         if(strcmp(d,dir[i].dname)==0)
        {
                printf("Enter the name of the file -- ");
                scanf("%s",f);
                for(k=0;k<dir[i].fcnt;k++)
                {
                if(strcmp(f, dir[i].fname[k])==0)
                {
                        printf("File %s is found ",f);
                        goto jmp1;
                        }
                }
                printf("File %s not found",f);
                goto jmp1;
                }
                }
                printf("Directory %s not found",d);
                jmp1:
                break;
case 5: if(dcnt==0)
         printf("\nNo Directory's ");
        else    {
        printf("\nDirectory\tFiles");
        for(i=0;i<dcnt;i++)
        {
        printf("\n%s\t\t",dir[i].dname);
        for(k=0;k<dir[i].fcnt;k++)
        printf("\t%s",dir[i].fname[k]);
        }
        }
        break;
default:exit(0);
 }
```

```
                }
        getch();
}
```

OUTPUT:

```
1. Buat Direktori       2. Buat File        3. Hapus File
4. Cari File        5. Tampilan         6. Keluar           Masukkan pilihan Anda - 1

Masukkan nama direktori - DIR1
Direktori dibuat


1. Buat Direktori       2. Buat File        3. Hapus File
4. Cari File        5. Tampilan         6. Keluar           Masukkan pilihan Anda - 1

Masukkan nama direktori – DIR2
Direktori dibuat


1. Buat Direktori       2. Buat File        3. Hapus File
4. Cari File        5. Tampilan         6. Keluar           Masukkan pilihan Anda - 2

Masukkan nama direktori -- DIR1
Masukkan nama file -- A1
Direktori dibuat


1. Buat Direktori       2. Buat File        3. Hapus File
4. Cari File        5. Tampilan         6. Keluar           Masukkan pilihan Anda - 2

Masukkan nama direktori -- DIR1
Masukkan nama file -- A2
Direktori dibuat


1. Buat Direktori       2. Buat File        3. Hapus File
4. Cari File        5. Tampilan         6. Keluar           Masukkan pilihan Anda - 2

Masukkan nama direktori – DIR2
Masukkan nama file -- B1
Direktori dibuat


1. Buat Direktori       2. Buat File        3. Hapus File
4. Cari File        5. Tampilan         6. Keluar           Masukkan pilihan Anda - 5

Direktori       File

DIR1            A1      A2

DIR2            B1
```

```
1. Buat Direktori      2. Buat File      3. Hapus File
4. Cari File      5. Tampilan      6. Keluar      Masukkan pilihan Anda - 4

Masukkan nama direktori – DIR
Direktori tidak ditemukan


1. Buat Direktori      2. Buat File      3. Hapus File
4. Cari File      5. Tampilan      6. Keluar      Masukkan pilihan Anda - 3

Masukkan nama direktori – DIR1
Masukkan nama file --  A2
File A2 terhapus


1. Buat Direktori      2. Buat File      3. Hapus File
4. Cari File      5. Tampilan      6. Keluar      Masukkan pilihan Anda – 6
```

C. ORGANISASI DIREKTORI HIERARKIS

```
#include<stdio.h>
#include<graphics.h>

struct tree_element {
    char name[20];
    int x, y, ftype, lx, rx, nc, level; struct tree_element *link[5];
    };
    typedef struct tree_element node;

void main()  {

    int gd=DETECT,gm;

    node *root;

    root=NULL;

    clrscr();

    create(&root,0,"root",0,639,320);

    clrscr();

    initgraph(&gd,&gm,"c:\tc\BGI");

    display(root);

    getch();

    closegraph();

}

create(node **root,int lev,char *dname,int lx,int rx,int x)  {
```

```
int i, gap;
if(*root==NULL)  {
        (*root)=(node *)malloc(sizeof(node));
        printf("Enter name of dir/file(under %s) : ",dname);
        fflush(stdin);
        gets((*root)->name);
        printf("enter 1 for Dir/2 for file :");
        scanf("%d",&(*root)->ftype);
        (*root)->level=lev;
        (*root)->y=50+lev*50;
        (*root)->x=x;
        (*root)->lx=lx;
        (*root)->rx=rx;
        for(i=0;i<5;i++)
        (*root)->link[i]=NULL;
        if((*root)->ftype==1)
        {
                printf("No of sub directories/files(for %s):",(*root)->name);
                scanf("%d",&(*root)>nc);
                if((*root)->nc==0)
                        gap=rx-lx;
                else
                        gap=(rx-lx)/(*root)->nc;
                for(i=0;i<(*root)->nc;i++)
                        create(&((*root)>link[i]),lev+1,(*root)>name,lx+gap*i,lx+gap
                *i+gap,       lx+gap*i+gap/2); }
                else
                        (*root)->nc=0;
                        }
                 }
                display(node *root)
                {
```

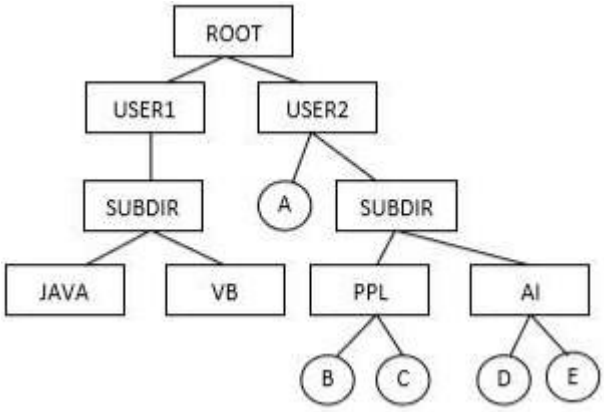
```
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if(root !=NULL)  {
        for(i=0;i<root->nc;i++)
        line(root->x,root->y,root->link[i]->x,root->link[i]->y);
        if(root->ftype==1)
        bar3d(root->x-20,root->y-10,root-
>x+20,root>y+10,0,0);
        else
        fillellipse(root->x,root->y,20,20);
        outtextxy(root->x,root->y,root->name);
        for(i=0;i<root->nc;i++)
        display(root->link[i]);
        }
}
```

INPUT

```
Masukkan Nama dir / file (di bawah root): ROOT
Masukkan 1 untuk Dir / 2 untuk File: 1
Jumlah subdirektori / file (untuk ROOT): 2
Masukkan Nama dir / file (di bawah ROOT): USER1
Masukkan 1 untuk Dir / 2 untuk File: 1
Jumlah subdirektori / file (untuk USER1): 1
Masukkan Nama dir / file (di bawah USER1): SUBDIR1
Masukkan 1 untuk Dir / 2 untuk File: 1
Jumlah subdirektori / file (untuk SUBDIR1): 2
Masukkan Nama dir / file (di bawah USER1): JAVA
Masukkan 1 untuk Dir / 2 untuk File: 1
Jumlah subdirektori / file (untuk JAVA): 0
Masukkan Nama dir / file (di bawah SUBDIR1): VB
Masukkan 1 untuk Dir / 2 untuk File: 1
Jumlah subdirektori / file (untuk VB): 0
Masukkan Nama dir / file (di bawah ROOT): USER2
Masukkan 1 untuk Dir / 2 untuk File: 1
Jumlah subdirektori / file (untuk USER2): 2
Masukkan Nama dir / file (di bawah ROOT): A
```

Masukkan 1 untuk Dir / 2 untuk File: 2
Masukkan Nama dir / file (di bawah USER2): SUBDIR2
Masukkan 1 untuk Dir / 2 untuk File: 1
Jumlah subdirektori / file (untuk SUBDIR2): 2
Masukkan Nama dir / file (di bawah SUBDIR2): PPL
Masukkan 1 untuk Dir / 2 untuk File: 1
Tidak ada subdirektori / file (untuk PPL): 2
Masukkan Nama dir / file (di bawah PPL): B
Masukkan 1 untuk Dir / 2 untuk File: 2
Masukkan Nama dir / file (di bawah PPL): C
Masukkan 1 untuk Dir / 2 untuk File: 2
Masukkan Nama dir / file (di bawah SUBDIR) : AI
Masukkan 1 untuk Dir / 2 untuk File: 1
Jumlah subdirektori / file (untuk AI): 2
Masukkan Nama dir / file (di bawah AI): D
Masukkan 1 untuk Dir / 2 untuk File: 2
Masukkan Nama dir / file (di bawah AI): E
Masukkan 1 untuk Dir / 2 untuk File: 2

OUTPUT



10.5. TUGAS

1. Manakah dari struktur direktori yang efisien? Mengapa?
2. Struktur direktori mana yang tidak memberikan isolasi dan perlindungan tingkat pengguna?
3. Apa keuntungan dari struktur direktori hierarkis?
4. Tulis C untuk mensimulasikan struktur direktori grafik asiklik!
5. Tulis C untuk mensimulasikan struktur direktori grafik umum!

LEMBAR JAWABAN PRE-TEST / POST-TEST / EVALUASI PRAKTIKUM

Nama : **NIM :**	**Asisten:** **Paraf Asisten:**	**Tanggal:** **Nilai:**

DAFTAR PUSTAKA

Silberschatz, A., Galvin, P., Gagne, G., 2013, *Operating System Concept*, Ed.9, John Wiley and Sons (Asia), Inc.

Tanenbaum, A.S., 2015, *Modern Operating System*, 4rd Ed., Pearson Education International, Prentice Hall.

Bambang, H., 2003, Sistem Operasi, Rev,4, Informatika


LABORATORIUM
TEKNIK INFORMATIKA
UNIVERSITAS AHMAD DAHLAN


UNIVERSITAS AHMAD DAHLAN