

Date / /

## 1). Computer Architecture .

Dalam komputer ada beberapa register .

Register adalah fast - performance memory untuk menyimpan beberapa informasi

Beberapa Register pada Computer Architecture :

- 1). PC ( Program Counter )
- 2). IR ( Instruction Register )
- 3). MBR ( Memory Buffer Register )
- 4). MAR ( Memory Address Register ) .
- 5). I/O AR ( I/O Address Register )
- 6). I/O BR ( I/O Buffer Register ) .

Didalam Computer Architecture juga terdapat Execution Unit .

Yang terjadi didalam sistem adalah :

↳ Data kita datang / masuk ke CPU sebagai instruksi .  
dan instruksi tersebut di perform dan dieksekusi oleh Execution unit .

System Bus → tempat data lewat / istilahnya kayak bus di jalan  
tapi bedanya bus di jalan → angkut orang .  
bus disini → bawa data

system bus mengakomodari data ul berpindah dari I/O-CPU maupun dari CPU ke execution unit .

IR mengandung instruksi yang akan dieksekusi .

Sekali instruksi dieksekusi , PC akan increment . / increase  
PC basically berisi instruksi yang akan dieksekusi .

Data kemudian akan di store di Memory .

↳ komputer pada dasarnya barubisa mengambil data  
jika data tersebut sudah ada di memory .  
( dari I/O data akan ke memory )

36 lines, 6 mm

## - Instruction Cycle -

1). Fetch Instruction dari Instruction Register .

2) Setelah di fetch, harus dieksekusi .

Execution Unit akan eksekusi → finished .

Semua sistem komputer baru bisa berjalan ketika diberi instruksi

## Operating systems .

1) What?

↳ Operating systems adalah program yang mengontrol eksekusi dari program aplikasi

↳ OS berperan sebagai interface / antarmuka antara aplikasi dan hardware .

(Jika ga ada interface yang bagus, kita harus kasih tau instruksi dengan bhs mesin → sangat susah)

2) Tujuan

- kenyamanan

- Efisien

- OS harus bisa berkembang .

Misal: hardware terus berkembang dari zaman ke zaman OS harus mampu ikutin perkembangannya itu .

## - Computer Hardware and software Structure -

Bottom: (abu2) → Hardware

lalu di atasnya  $\frac{\text{App}}{\text{Library}}$  → yg biasa kita bikin

$\frac{\text{OS}}$



1). fork() system call

↳ adalah sistem call untuk membuat proses baru

Ketika fork() dieksekusi, fork akan membuat 2 salinan identik

[VBL].

Yang dilakukan fork adalah membuat child process baru ketika kita membuat child process, original process masih terus berjalan, namun kita punya child process yang bisa berjalan secara concurrent bersamaan dg parent.

Setiap proses bisa melakukan fork() berkali-kali, bahkan childnya bisa fork lagi untuk membuat child proses lagi.

Bagaimana cara kita memastikan bahwa parent process dan child jalan bersamaan?

↳ Anggap kita di memory.

lalu <sup>sejoli</sup> panggil fork() → akan langsung membuat salinan dari parent dicopy ke child.

sehingga ada 2 process: 1 di parent 1 di child.

Bagaimana cara tahu kalau sebuah process itu parent/child?

- fork akan return negative number jika fork gagal
  - Return 0 jika ada di dlm child process
  - Return 1 parent.
- } pidnya yg return

↳ dari sini kita bisa tahu process mana yang mau dijalankan ataupun yang sedang berjalan.

Multi Processor vs Multi Core.

- komputer yang punya banyak CPU.
- Beberapa mesin mengkom binasikan multicore dan multi processor
- CPU yang punya banyak core: processing unit yang membaca semua instruksi untuk melakukan tindakan tertentu.
- core beroperasi sebagai prosesor terpisah didalam chip tunggal
- meningkatkan performa tanpa menaikkan clock speed processor

Unit Kegiatan Mahasiswa  
BSLC (Learning Community)

**BINUS**  
UNIVERSITY

|  |   |
|--|---|
|  <p>Unit Kegiatan Mahasiswa<br/>BSLC (Learning Community)<br/><b>BINUS</b><br/>UNIVERSITY</p> | <p>E-Learning BSLC</p> <p>Nadya Marcelinda<br/>2201744600</p> |
|--|---|

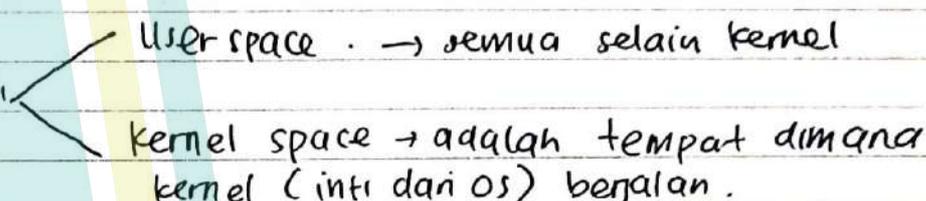
# Thread.

Date / /

OM OT OW OT OF OS OS

Thread memungkinkan untuk terjadinya <sup>beberapa</sup> eksekusi (instruksi) di dalam waktu bersamaan pada environment process yang sama.

Thread merupakan lightweight process. karena thread memiliki beberapa properti dari process

Thread bisa diimplementasikan   
User space . -> semua selain kernel  
kernel space -> adalah tempat dimana kernel (inti dari os) berjalan.

### Thread Implementation pada user space :

- Advantages :

- memperbolehkan setiap proses untuk memiliki algoritma penjadwalannya tersendiri
- dari segi performa -> lebih cepat.

- Disadvantages :

- Implementasi dari blocking system calls.
- Thread lain tidak bisa berjalan jika thread yang pertama tidak mengalah.

### Thread Implementation pada kernel space :

Advantages : - tidak membutuhkan non-blocking system call baru.

Disadvantages : - cost yang dibutuhkan untuk membuat dan memusnahkan thread lebih besar

- VBL -

thread adalah unit of execution didalam process sehingga thread lebih kecil dari process

Single Threading -> 1 process bisa punya 1 thread.

Multi dlm 1 process saya bisa punya > 1 thread.

Lex: sgt berguna u/ jika ingin menjalankan 2 process concurrently  
Cth : matrix multiplication kedua proses hrs berjalan independen  
- 1 thread u/ addition

sehingga overall process lebih baik

36 lines, 6 mm



Unit Kegiatan Mahasiswa  
BSLC (Learning Community)  
**BINUS**  
UNIVERSITY

E-Learning BSLC

Nadya Marcelinda  
2201744600

Kenapa kita buat thread?

- Membuat thread lebih cepat dibanding buat process.
- thread lebih cepat terminasinya
- Thread switch akan lebih cepat
- karena thread ada dalam process yang sama, sehingga thread akan berbagi memory, antar thread dapat saling berkomunikasi tanpa mengganggu kernel.
- Thread execute faster.

Implementasi thread:

1) di user space → thread dibuat di user area, namun prosesnya masih di kernel.

Advantages (dibelakang)

Disadvantages - harus membuat special system call sehingga dia tidak memblock both process dan thread karena process masih di kernel, sehingga jika kita block processnya, maka thread akan terblock juga.

2) di kernel.

- ↳ tidak perlu buat system blocking calls
- ↳ greater cost dalam membuat dan destroy thread karena lokasinya di kernel.

Dalam kernel, hanya supervisor / admin yg bs create thread. jika sbg normal user, OS harus perform user mode switch, switching dari standart user mode ke supervisor user mode.

3) Combination

- some thread akan dibuat pada user level, & beberapa di kernel.

Tujuan: untuk mendapatkan advantage dari kedua implementasi

# Operating Systems - Scheduling - MAOI.

Date 13 / 10 / 2020

OM OT OW OT OF OS OS

Kita hanya punya 1 CPU dalam satu komputer, sehingga harus dibuat jadwal agar task<sup>2</sup> tidak saling bertabrakan.

- Beberapa process punya higher priority cth: OS related process
  - Yang punya lower priority: user related process.
- Sehingga overall schedule harus dijadwalkan dengan baik.

Komputer harus memiliki kriteria tertentu u/ menjalankan scheduling.

1. Max CPU utilization
2. Throughput dimaksimalkan. Throughput: jumlah process yg dapat dijalankan dalam jumlah waktu tertentu.
3. Turnaround time harus diminimalkan.

Turnaround time = waktu yang dibutuhkan dari proses masuk ke sistem sampai process tersebut memberikan output.

4. Waiting Time / waktu harus diminimalkan process harus berjalan di CPU secara bergantian sehingga pasti ada waiting time yang terjadi.
5. Minimalisasi waktu response

Response time: dari submit process sampai mendapat first response. (tidak harus sampai kelar)

## Goals Scheduling:

All systems mendapatkan:

- fairness → memberikan setiap process pembagian CPU yang adil.
- policy enforcement: cth: priority harus dijalankan terlebih dahulu.
- Balance → harus bisa keep / <sup>sempit bagian</sup> system busy (ribuk).

## Batch systems:

- Throughput → jumlah jobs yang dapat dilakukan pada unit waktu tertentu harus dimaksimalkan
- Turn around time diminimalkan
- CPU utilization → CPU harus di tetap sibuk.

36 lines, 6 mm

Date / /

Interactive systems.

- Response times
- proportionality : → harus sesuai dengan expectation

Real-time systems : → tidak boleh lose any data.

- Meeting deadlines →
- harus predictable supaya tidak bertemu degradation

Scheduling Algorithm

1) First Come First serve.

↳ jika proses datang pertama kali, maka dia akan dieksekusi terlebih dahulu.  
 Kalau ada process lain datang → queue.

Burst time : waktu yang dibutuhkan buat finish

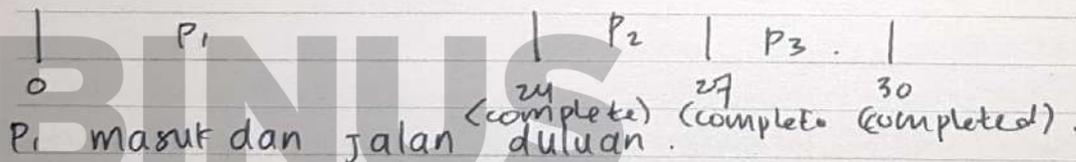
Contoh:

| Process        | Burst Time |
|----------------|------------|
| P <sub>1</sub> | 24         |
| P <sub>2</sub> | 3          |
| P <sub>3</sub> | 3          |

arrive in order ↓

|                |    |
|----------------|----|
| P <sub>1</sub> | 24 |
| P <sub>2</sub> | 3  |
| P <sub>3</sub> | 3  |

Gantt chart (chart untuk time) ul/ schedule tsb :



Turn around Time : P<sub>1</sub> = 24

P<sub>2</sub> = 27

P<sub>3</sub> = 30

Waiting time : P<sub>1</sub> = 0 → tidak wait sama sekali

P<sub>2</sub> = 24 → nunggu P<sub>1</sub> keluar.

P<sub>3</sub> = 27 → nunggu P<sub>2</sub> keluar

Avg waiting time :  $(0 + 24 + 27) / 3 = 17$

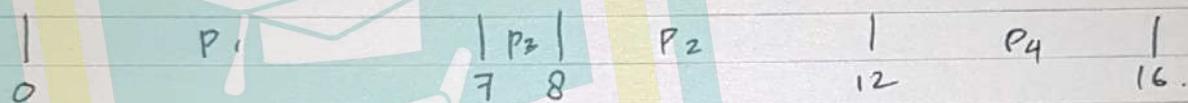
|   |                                |
|---|--------------------------------|
|  | E-Learning BSLC                |
|  | Nadya Marcelinda<br>2201744600 |

2) Shortest job first - <sup>Non</sup>Preemptive → can't be interrupted.

| Process        | Arrival Time | Burst |
|----------------|--------------|-------|
| P <sub>1</sub> | 0.0          | 7     |
| P <sub>2</sub> | 2.0          | 4     |
| P <sub>3</sub> | 4.0          | 1     |
| P <sub>4</sub> | 5.0          | 4     |

↓  
sekali masuk CPU gabs diganggu.

→ P<sub>1</sub> masuk duluan nih → waktu dia 7 detik, otomatis dari dia masuk - kelar si P<sub>2</sub> P<sub>3</sub> P<sub>4</sub> udah queue (nungguin) ok. P<sub>1</sub> mulai di CPU



Abis itu dari P<sub>2</sub> P<sub>3</sub> P<sub>4</sub> dipilih yang paling cepet dulu → P<sub>3</sub> P<sub>2</sub> dan P<sub>4</sub> burst time sama oleh karena itu P<sub>2</sub> di jalanin duluan.

Turn around Time P<sub>1</sub> : 7  
P<sub>2</sub> : 12  
P<sub>3</sub> : 1  
P<sub>4</sub> : 16.

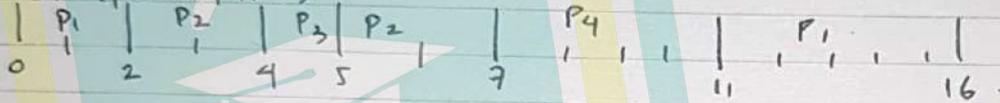
Waiting Time : P<sub>1</sub> = 0  
(~~Turn around~~ Time start - Arrival Time) P<sub>3</sub> = 3 (7 - 4).  
P<sub>2</sub> = 6 (8 - 2).  
P<sub>4</sub> = 7 (12 - 5).

Avg waiting time :  
(0 + 6 + 3 + 7) / 4  
= 4 //

## 3) Shortest Job First (Preemptive)

| Process        | Arrival Time | Burst Time |
|----------------|--------------|------------|
| P <sub>1</sub> | 0.0          | 7          |
| P <sub>2</sub> | 2.0          | 4          |
| P <sub>3</sub> | 4.0          | 1          |
| P <sub>4</sub> | 5.0          | 4          |

SJF (preemptive)



P<sub>1</sub> masuk → dijalankan. Jalan 2 detik datang yg baru diqueue (P<sub>2</sub>)  
 BT P<sub>2</sub> < BT P<sub>1</sub> (yang belum berjalan /  $7 - 2 = 5$ ), maka P<sub>2</sub>  
 masuk jalan. P<sub>1</sub> wait.  
 P<sub>2</sub> jalan 2 detik, datang yg baru P<sub>3</sub>.  
 BT P<sub>2</sub>' > BT P<sub>3</sub> (4 - 2 = 2), oleh karena itu P<sub>3</sub> jalan. (1 detik → keluar)

Masuk si P<sub>4</sub>, bandingin BT P<sub>1</sub> vs P<sub>2</sub> vs P<sub>4</sub> → P<sub>2</sub> jalan  
 5 vs 2 vs 4

sampe abis (abis di detik ke 7, lalu jalan lah P<sub>4</sub> (4 detik)  
 baru P<sub>1</sub> lanjutin (5 detik) → done

Turn around time: (end time - Arrival time)

$$P_1 = 16 - 0 = 16$$

$$P_2 = 7 - 2 = 5$$

$$P_3 = 5 - 4 = 1$$

$$P_4 = 11 - 5 = 6$$

Waiting Time:

$$P_1 = \text{wait dr detik ke 2 sampai 11} \rightarrow \text{wait dr detik ke } 2 - 11 = 9$$

$$P_2 = \text{wait dr 4 ke 5} = 1 \text{ s.}$$

$$P_3 = 0$$

$$P_4 = 2$$

$$\text{Avg} = 9 + 1 + 0 + 2 / 4 = 3$$



Date / /

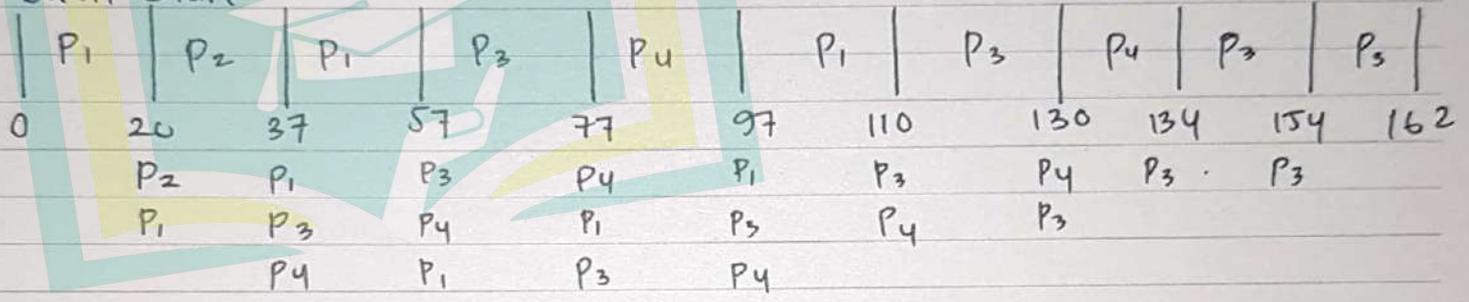
Berjalan berurutan sesuai siapa yang datang duluan, tapi punya time limit

OM OT OW OT OF OS OS (time slice)

5) Round Robin Scheduling → punya time limit jadi kalau lg running → time limit lewat harus stop terus masuk ke queue yang plg belakang.

| Process        | Burst Time | Arrival Time: |
|----------------|------------|---------------|
| P <sub>1</sub> | 53         | 0             |
| P <sub>2</sub> | 17 ✓       | 20            |
| P <sub>3</sub> | 68         | 25            |
| P <sub>4</sub> | 24         | 30            |

Gantt chart:



Burst time bank:

$P_1: 53 - 20 = 33 - 20 = 13 - 13 = 0$   
 $P_3: 68 - 20 = 48 - 20 = 28 - 20 = 8$   
 $P_4: 24 - 20 = 4 - 4 = 0$

Turn around Time: (waktu selesai - waktu arrive).

$P_1 = 110 - 0 = 110$   
 $P_2 = 37 - 20 = 17$   
 $P_3 = 162 - 25 = 137$   
 $P_4 = 134 - 30 = 104$

$$\text{Avg} = \frac{368}{4} = 92$$

Waiting Time:

$P_1 = 0 + 17 + 40 = 57$   
 $P_2 = 0$   
 $P_3 = 32 + 20 + 13 + 4 = 69$   
 $P_4 = 7 + 40 + 13 + 20 = 80$

$$\text{Avg} = \frac{206}{4} = 51.5 \text{ unit of time!}$$