SISTEM OPERASI

(MANAJEMEN PROSES)

Ir. I Gede Made Karma, MT

PROSES

- Konsep proses
- Penjadwalan proses
- Operasi pada proses
- Penggabungan proses
- Komunikasi Interproses
- Komunikasi dalam sistem *Client-Server*

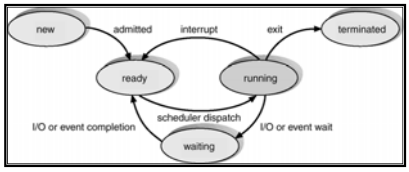
Konsep Proses

- OS mengeksekusi berbagai macam program:
  - Sistem *batch* – pekerjaan (*job*)
  - Sistem *Time-shared* – program user atau tugas (*task*)
- Digunakan istilah *job* dan proses
- Proses – program dalam eksekusi; eksekusi proses harus dilaksanakan secara berurutan
- Proses meliputi:
  - *program counter*
  - *stack*
  - *data section*

Status Proses

- Sebagaimana dalam eksekusi sebuah proses, status akan diubah:
  - **new**: proses sedang dibuat
  - **running**: instruksi sedang dieksekusi
  - **waiting**: proses menunggu terjadinya *event*
  - **ready**: proses yang telah ditetapkan sedang menunggu diproses
  - **terminated**: proses selesai dieksekusi

Diagram Status Proses

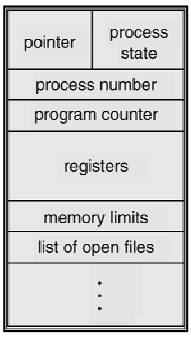


Blok Kendali Proses (Process Control Block/PCB)

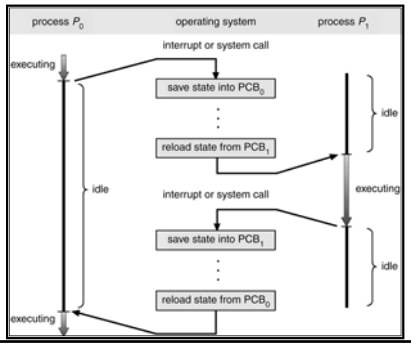
Informasi digabungkan dengan masing-masing proses:

- *Process state*
- *Program counter*
- *CPU register*
- Informasi penjadwalan CPU
- Informasi manajemen memori
- Informasi akuntansi
- Informasi status I/O

Process Control Block (PCB)



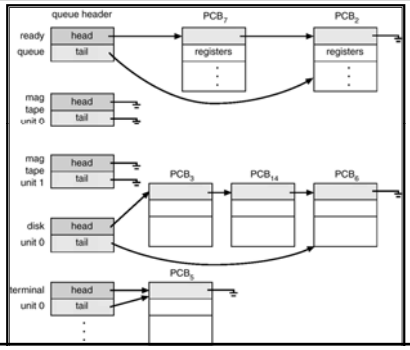
Perpindahan Proses-ke-Proses Dalam CPU



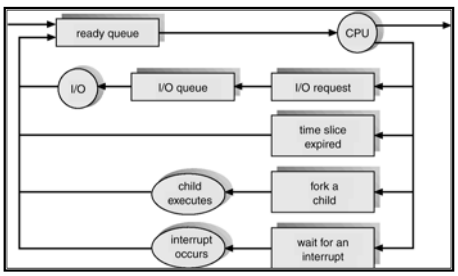
Antrian Penjadwalam Proses

- Antrian *job* – sekumpulan seluruh proses dalam sistem
- Antrian ready – sekumpulan seluruh proses di dalam memori, siap dan menunggu dieksekusi
- Antrian *device* – sekumpulan proses menunggu untuk peralatan I/O
- Perpindahan proses di antara berbagai macam antrian

Antrian *Ready* Dan Antrian Berbagai Peralatan I/O



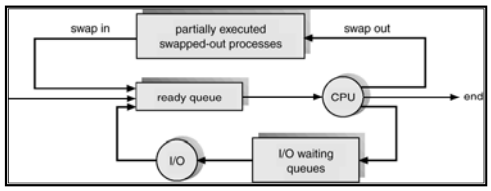
Representasi Penjadwalan Proses



Penjadwal

- *Long-term scheduler* (atau job *scheduler*) – memilih proses mana yang akan diletakkan ke dalam antrian **ready**
- *Short-term scheduler* (atau CPU *scheduler*) – memilih proses mana yang akan dieksekusi berikutnya dan mengalokasikan CPU

Tambahan Penjadwalan *Medium Term*



Penjadwal (Cont)

- *Short-term scheduler* digunakan untuk penjadwalan yang sangat sering (*millisecond*) ⇒ harus cepat
- *Long-term scheduler* digunakan untuk penjadwalan yang relatif tidak sering (*second*, *minute*) ⇒ dapat lebih lambat
- *Long-term scheduler* mengendalikan pada tingkatan *multiprogramming*
- Proses dapat dideskripsikan:
  - Proses yang dibatasi I/O – memerlukan lebih banyak waktu untuk I/O daripada melaksanakan komputasi, singkat dan sepenuhnya oleh CPU
  - Proses yang dibatasi *CPU* – memerlukan lebih banyak waktu untuk komputasi; sangat lama dan sedikit yang sepenuhnya oleh CPU

Context-Swith

- Saat CPU pindah ke proses lain, sistem harus menyimpan *state* proses lama dan memanggil *state* untuk proses baru
- Waktu *context-switch* akan bertambah; sistem tidak melaksanakan pekerjaan secara penuh saat perpindahan proses
- Waktu yang dibutuhkan tergantung pada dukungan *hardware*

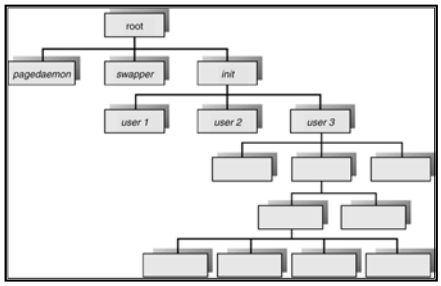
Pembuatan Process

- Proses induk akan membuat proses anak untuk proses lain, membentuk pohon proses
- *Resource sharing*
  - Induk dan anak menggunakan semua *resource*
  - Anak menggunakan bagian dari *resource* induk
  - Induk dan anak tidak menggunakan *resource* secara bersama
- Eksekusi
  - Induk dan anak mengeksekusi secara bersamaan
  - Induk menunggu hingga anak selesai (***terminate***)

Pembuatan Proses (Cont)

- *Space* alamat
  - Anak membuat duplikat alamat induk
  - Anak memiliki sebuah program yang dipanggil untuknya
- Contoh pada **UNIX**:
  - **fork:** sistem melakukan pemanggilan untuk membuat proses baru
  - **exec:** sistem melakukan pemanggilan setelah digunakan oleh **fork** untuk mengganti *space* memori dengan sebuah program baru

Pohon Proses Pada UNIX



Process Termination

- Proses mengeksekusi statemen terakhir dan menanyakan OS untuk memutuskannya (**exit**)
  - Data keluaran dari anak ke induk (via **wait**)
  - Resource yang digunakan oleh proses di-dealokasikan oleh OS
- Induk dapat menghentikan ekselusi proses anak (**abort**)
  - Anak telah dialokasikan *resource* secara lebih
  - Tugas yang diberikan ke anak tidak dibutuhkan lagi
  - Induk masih tetap ada
    - OS tidak mengijinkan anak untuk melanjutkan jika induknya telah menghentikannya
    - Penghentian bertumpukan

Proses Gabungan

- Proses *independent* tidak dapat mempengaruhi atau dipengaruhi oleh proses lain
- Proses gabungan dapat mempengaruhi atau dipengaruhi oleh eksekusi proses lain
- Kelebihan proses gabungan
  - Penggunaan bersama informasi
  - Peningkatan kecepatan komputasi
  - Modularitas
  - Kenyamanan

Permasalahan *Producer-Consumer*

- Paradigma untuk proses gabungan, proses *producer* process menghasilkan informasi yang digunakan oleh proses c*onsumer*
  - *unbounded-buffer* menempatkan batas yang tidak praktis pada ukuran *buffer*
  - *bounded-buffer* mengasumsikan tidak ada ukuran *buffer* yang tetap

Solusi Bounded-Buffer – Shared-Memory

- Data yang di-*share*

```
#define BUFFER_SIZE 10
Typedef struct {
  . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Solusi benar, tetapi hanya dapat menggunakan sebanyak BUFFER_SIZE-1 elemen

Proses Bounded-Buffer – Producer

```
item nextProduced;

while (1) {
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
```


Proses Bounded-Buffer – Consumer

```
item nextConsumed;

while (1) {
    while (in == out)
        ; /* do nothing */
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
}
```

Komunikasi Antar Proses (Interprocess Communication/IPC)

- Mekanisme untuk mengkomunikasikan proses dan sinkronisasi
- Sistem *message* –dikomunikasikan dengan masing-masing proses tanpa mengurutkan kembali ke variabel-variabel yang dgunakan bersama
- Fasilitas IPC memberikan 2 operasi:
  - **send**(*message*) – ukuran *message* tetap atau variabel
  - **receive**(*message*)
- Jika *P* and *Q* mengharapkan komunikasi, maka perlu:
  - Membentuk penghubung komunikasi di antara keduanya
  - Menukarkan *message* melalui **send**/**receive**
- Implementasi penghubung komunikasi
  - fisik (misal, memori yang digunakan bersama, *bus hardware*)
  - logikal (misal properti logik)

Pertanyaan-pertanyaan Tentang Implementasi

- Bagaimana membentuk *link* komunikasi?
- Dapatkah sebuah *link* digabungkan dengan lebih dari 2 proses?
- Berapa *link* yang dapat dibuat di antara setiap kabel pada proses komunikasi?
- Berapa kapasitas sebuah *link*?
- Apakah ukuran *message* yang diakomodasi tetap atau variabel?
- Apakah sebuah *link* dapat digunakan untuk 1 arah atau 2 arah?

Komunikasi Langsung

- Proses harus memberi nama satu dengan yang lainnya secara tegas:
  - **send** (*P, message*) – mengirimkan sebuah *message* ke proses P
  - **receive**(*Q, message*) – menerima sebuah *message* dari proses Q
- Properti link komunikasi
  - Bayak *link* yang dibuat secara otomatis
  - Sebuah *link* diasosiasikan dengan tepat sebuah kabel pada proses komunikasi
  - Masing-masing kabel yang ada digunakan untuk sebuah *link*
  - *Link* mungkin 1 arah, tetapi biasanya 2 arah

Komunikasi Tidak Langsung

- Pesan dikirimkan dan diterima dari *mailbox* (juga di-refer sebagai port)
  - Masing-masing *mailbox* memiliki **id** yang unik
  - proses dapat berkomunikasi hanya jika men-*share* *mailbox*
- Properti pada *link* komunikasi
  - *Link* hanya disediakan jika proses men-*share* *mailbox*
  - *Link* dapat dihubungkan dengan banyak proses
  - Masing-masing media proses dapat menggunakan beberapa *link*
  - *Link* dapat bersifat searah atau dua arah

Komunikasi Tidak Langsung

- Operasi:
  - Membuat *mailbox* baru
  - Mengirim dan menerima pesan melalui *mailbox*
  - Menghapus *mailbox*
- Secara sederhana didefinisikan sebagai:

  send(***A***, ***message***) – mengirim sebuah pesan ke *mailbox* A

  receive(***A***, ***message***) – menerima pesan dari *mailbox* A

Komunikasi Tidak Langsung

- *Sharing mailbox*
  - P_1, P_2, dan P_3 men-*share* mailbox A
  - P_1 mengirimkan; P_2 *dan* P_3 menerima
  - Siapa yang menerima pesan…?
- Solusi:
  - Menyediakan sebuah *link* yang dihubungkan dengan 2 proses
  - Hanya menyediakan satu proses pada suatu saat untuk mengeksekusi sebuah operasi yang diterima
  - Menyediakan sistem untuk memilih sembarang penerima. Pengirim memberitahukan penerima mana yang dituju

Sinkronisasi

- Pesan yang dilewatkan bisa dalam bentuk *blocking* atau *non-blocking*
- **Blocking** dipertimbangkan sebagai **synchronous**
- **Non-blocking** dipertimbangkan sebagai **asynchronous**
- **send** dan **receive** sederhana bisa jadi *blocking* atau *non-blocking*

Buffering

- Antrian pesan disertakan pada *link*; diimplementasikan dalam satu di antara 3 berikut:
  1. *Zero capacity* – 0 pesan
     Pengirim harus menunggu dari penerima
  2. *Bounded capacity* – panjang terbatas pada ***n*** pesan
     Pengirim harus menunggu jika *link* penuh
  3. *Unbounded capacity* – panjang tak terbatas
     Pengirim tidak pernah menunggu

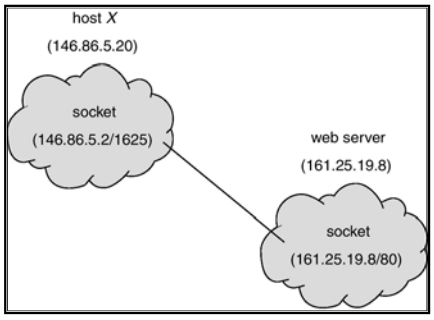
Komunikasi Client-Server

- *Socket*
- *Remote Procedure Call*
- *Remote Method Invocation* (Java)

Socket

- *Socket* didefinisikan sebagai sebuah titik akhir untuk komunikasi
- Gabungan alamat **IP** dan ***port***
- *Socket* **161.25.19.8:1625** menunjukkan *port* **1625** pada *host* **161.25.19.8**
- Komunikasi meliputi sebuah jalur pada so *socket*

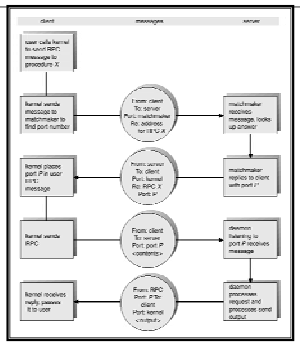
Komunikasi Socket



Remote Procedure Call (RPC)

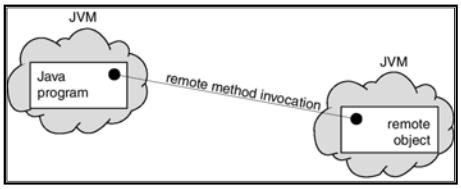
- RPC merupakan prosedur abstrak yang dipanggil di antara proses pada sistem jaringan
- **Stubs** – proxy *client-side* untuk prosedur aktual pada *server*
- *Client-side stub* menempatkan *server* dan menyusunnya sebagai parameter
- *Server-side stub* menerima pesan tersebut, mengurai parameter yang disusun, dan melaksanakan prosedur pada *server*

Eksekusi Pada RPC

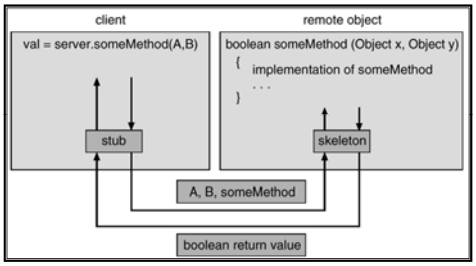


Remote Method Invocation (RMI)

- RMI adalah mekanisme Java yang mirip dengan RPC
- RMI mengijinkan sebuah program Java pada mesin untuk memanggil sebuah metode pada obyek jarak jauh



Penyusunan Parameter



Operating System Concepts