Operating System

Process

Willy Permana Putra

Konsep proses

- Sistem operasi mengeksekusi berbagai jenis program
- Pada sistem **batch program** biasanya disebut dengan **job**, sedangkan pada sistem **time sharing**, disebut dengan **program user** atau **task**.

Analogi

Tahapan eksekusi

- Proses pembuatan
- finishing

Komponen dan Ruang simpan sementara

- box
- plastik

Membutuhkan peralatan tertentu

- gunting
- lem

Analogi Proses

Tahapan eksekusi

- Program Counter
- Stack Pointer

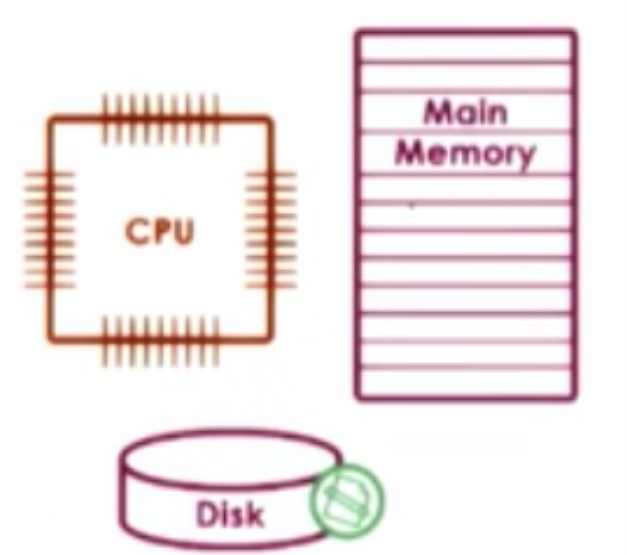
Komponen dan Ruang simpan sementara

- data
- state register

Membutuhkan peralatan tertentu

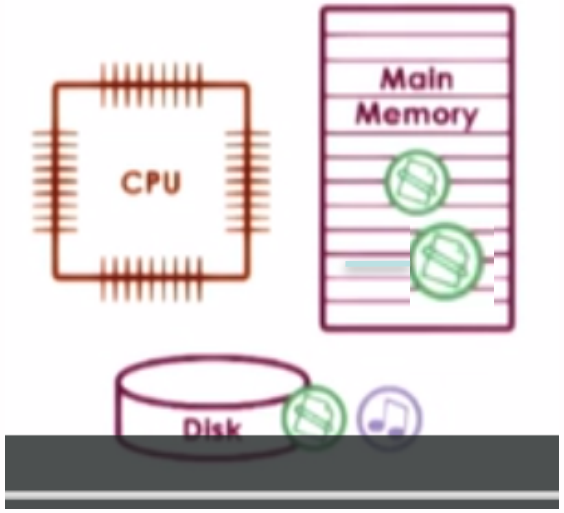
- perangkat I/O

Proses??

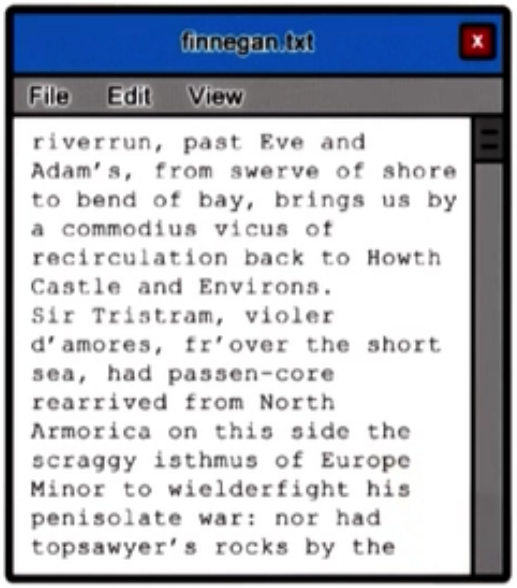


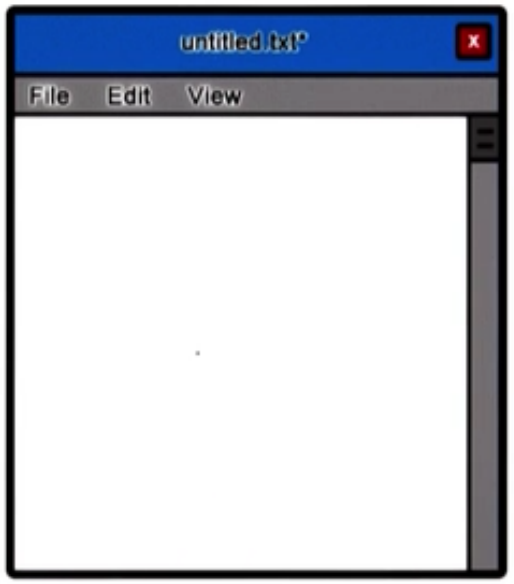
Aplikasi adalah program berada di disk, usb, flash memory, ... dsb (**static entity**)

Keadaan program saat dieksekusi
Berada di Memory
(**static entity**)



Proses??



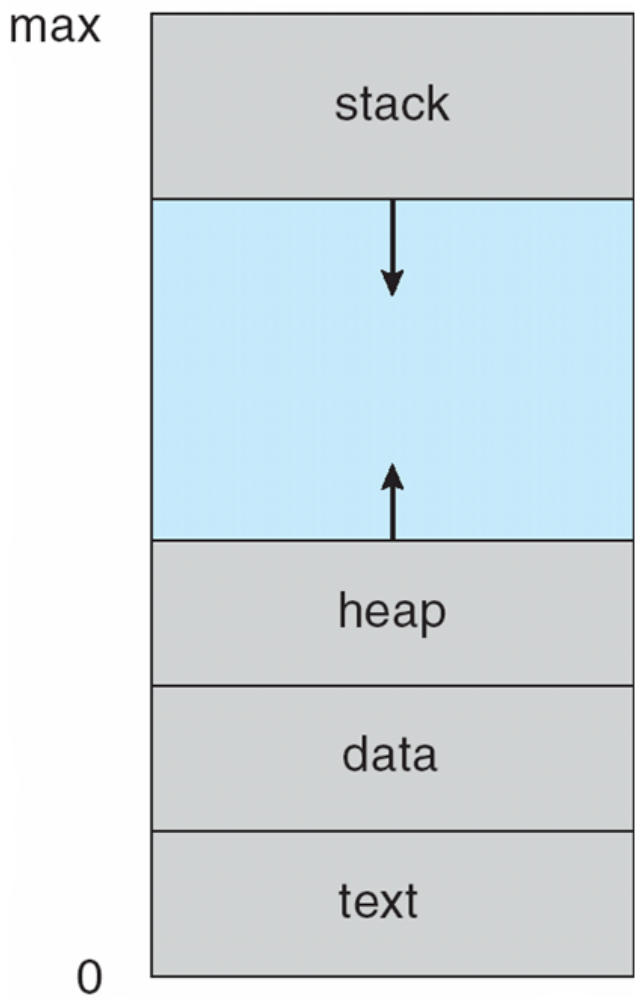


Merepresentasikan keadaan eksekusi dari sebuah aplikasi aktif

Tidak selalu running,

- Meskipun tiap-tiap proses terdiri dari suatu kesatuan yang terpisah namun adakalanya proses-proses tersebut butuh untuk saling berinteraksi.
- Satu proses bisa dibangkitkan dari output proses lainnya sebagai input.

Proses



Address Space

Stack = Data sementara seperti local var return address → bersifat dinamis LIFO

Heap = Data yang dihasilkan saat proses berjalan → bersifat dinamis

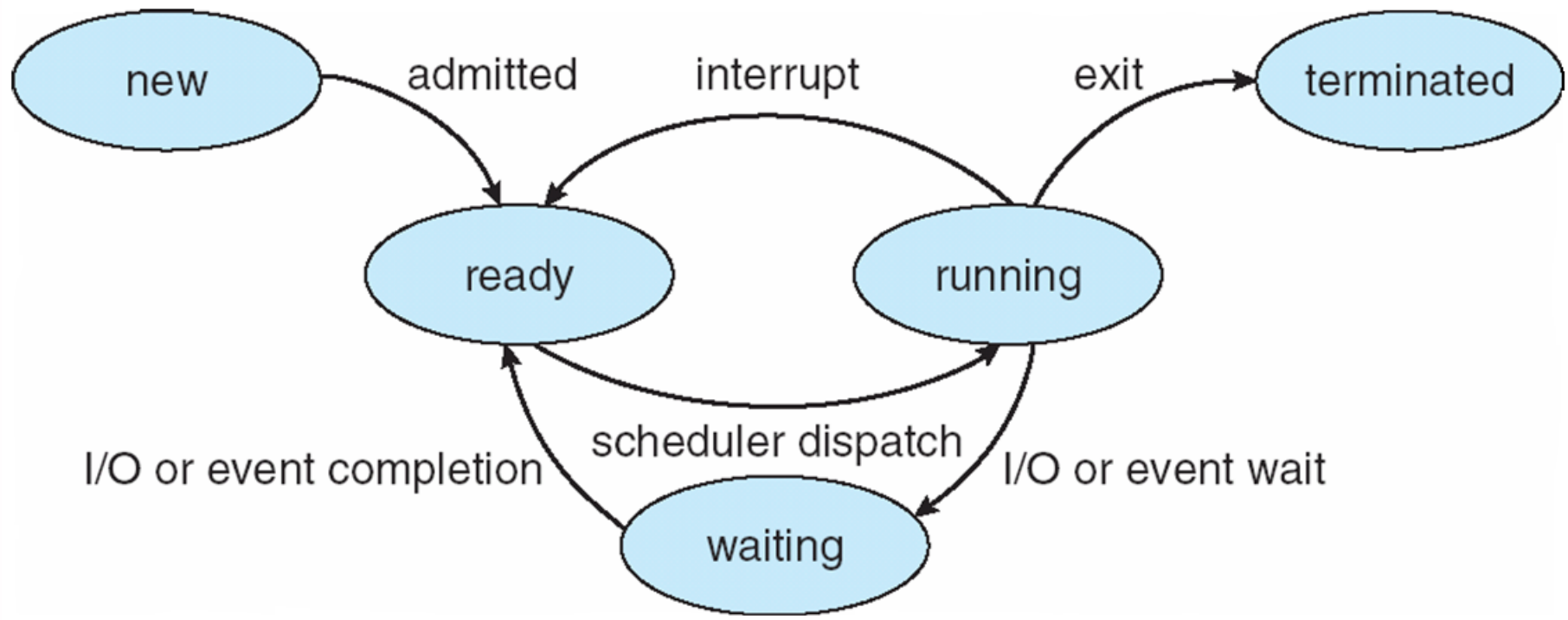
Text = Code Program
Data = Global variabel → Keadaan static ada saat pertama dieksekusi

Keadaan Proses

Sebagai proses yang diekseikusi, terdapat **keadaan** dari proses :

- **new**: Proses pembuatan
- **running**: Instruksi dieksekusi
- **waiting**: Proses menunggu beberapa event terjadi (I/O, sinyal)
- **ready**: Proses menunggu untuk dijalankan oleh sebuah processor
- **terminated**: Proses telah selesai eksekusi

Diagram State Proses



OS mengetahui apa yang dilakukan oleh proses

Program Counter

CPU Register

Stack Pointer

.....

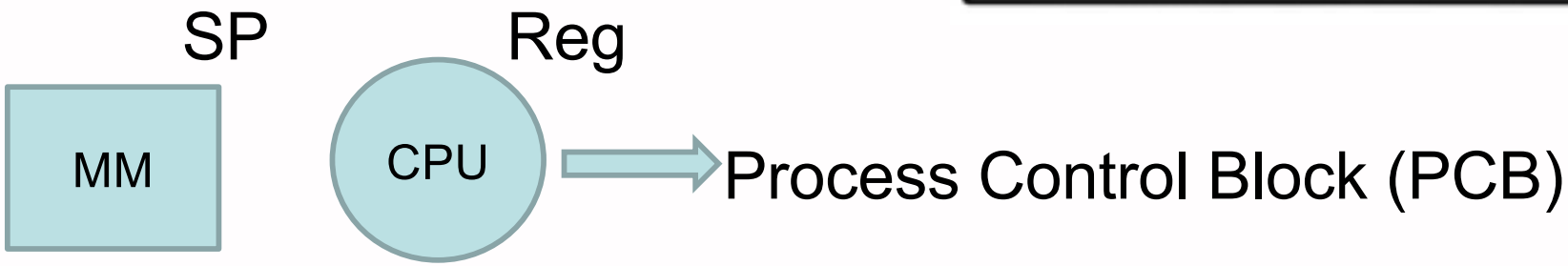
```
sum = 0;
for (int i = 0; i < 10; ++i) {
  sum += i;
}
```

(a) Simple Loop Code

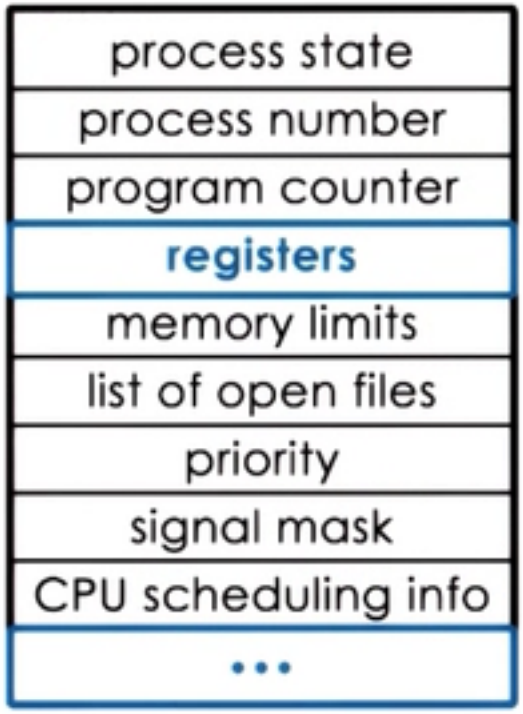
```
4004b8:     movl    $0x0,-0x8(%rbp)
4004bf:     movl    $0x0,-0x4(%rbp)
4004c6:     movl    $0x0,-0x4(%rbp)
4004cd:     jmp     4004d9 <main+0x25>
4004cf:     mov     -0x4(%rbp),%cax
4004d2:     add     %cax,-0x8(%rbp)
4004d5:     addl    $0x1,-0x4(%rbp)
4004d9:     cmpl    $0x9,-0x4(%rbp)
4004dd:     jle     4004cf <main+0x1b>
```

(b) Assembly Code

PC



Process Control Block



PCB dibuat saat Proses dibuat

Field berubah saat terjadi perubahan process state

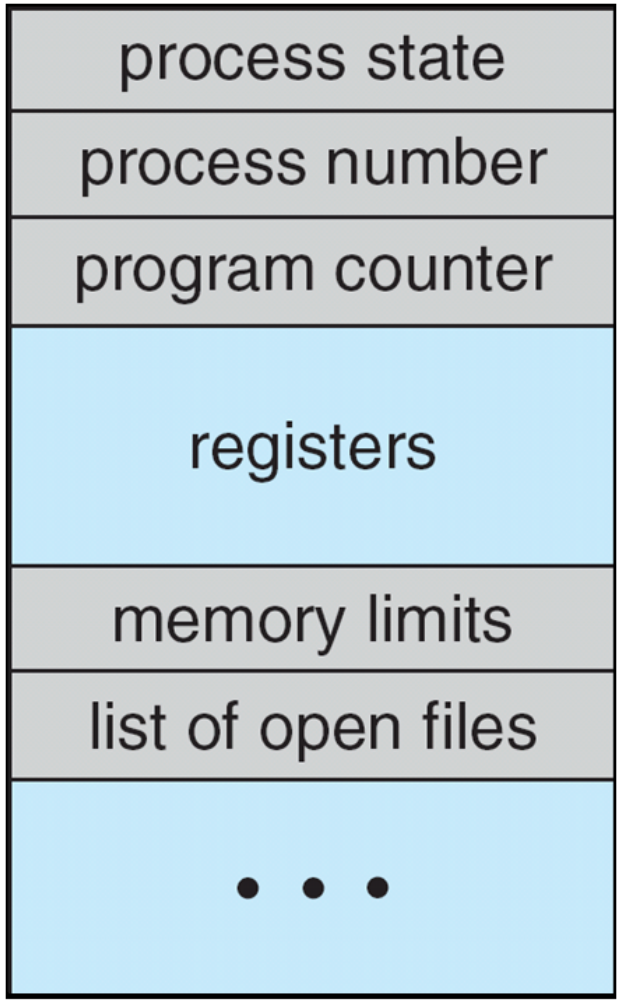
Field berubah pada waktu-waktu tertentu

Data struktrur yang dimaintain oleh SO untuk setiap proses yang dijalankan

PCB

Informasi yang terasosiasi terhadap pada setiap **proses** (**task control block**):

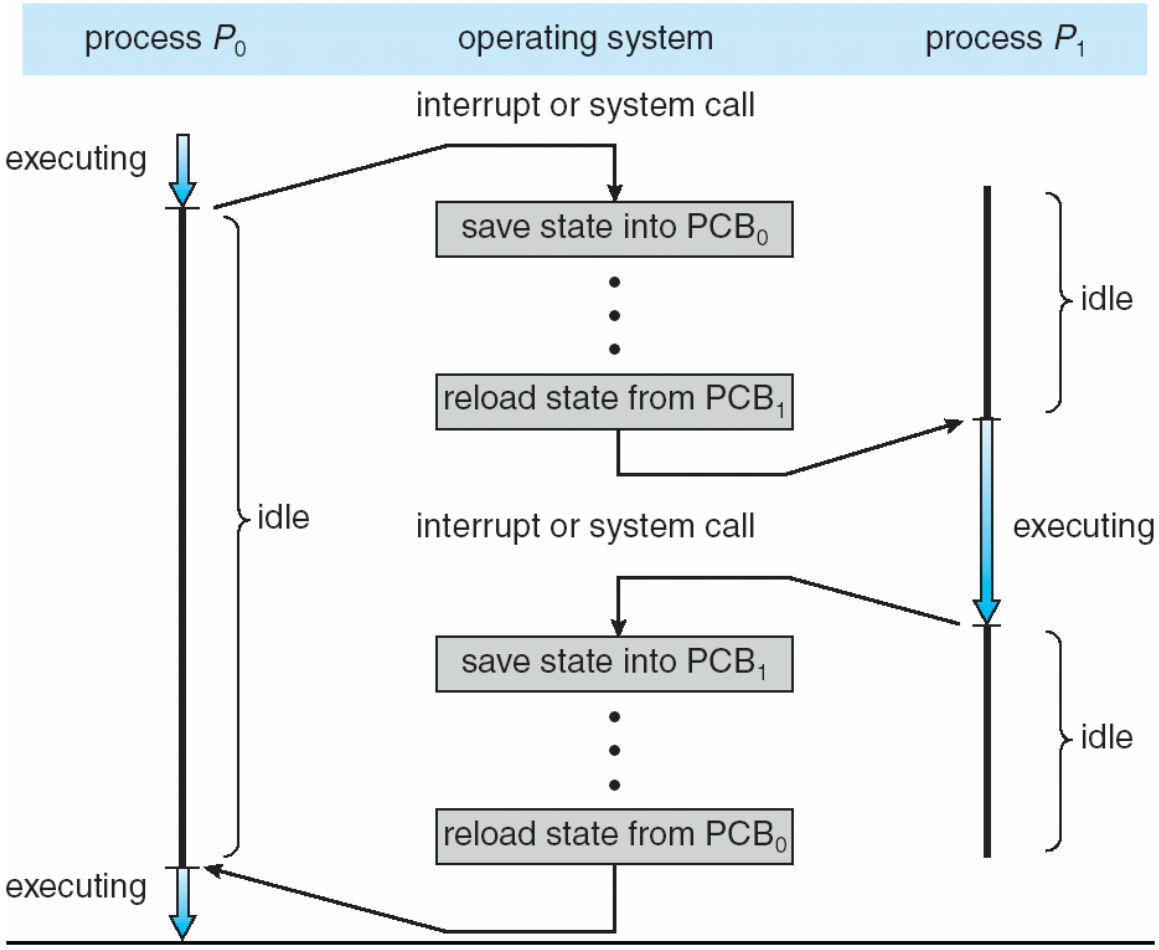
- Process state – running, waiting dsb
- Program counter – Lokasi instruksi yang selanjutnya akan dieksekusi
- CPU registers – Isi dari seluruh proses register (accumulator, stack pointer,...)
- CPU scheduling information- prioritas, scheduling queue pointers
- Memory-management information – memory yang dialokasikan untuk proses
- Accounting information – Penggunaan CPU, waktu, time limits
- I/O status information – I/O yang dialokasikan proses, list file yang di open



Thread

- Proses hanya memiliki satu thread eksekusi.
- Kebanyakan SO modern memiliki konsep tambahan untuk melakukan lebih dari satu tugas dalam satu waktu
  - PCB menambahkan beberapa informasi untuk melakukan kontrol -> **threads**

CPU Berpindah dari Proses ke Proses



Representasi Proses di linux

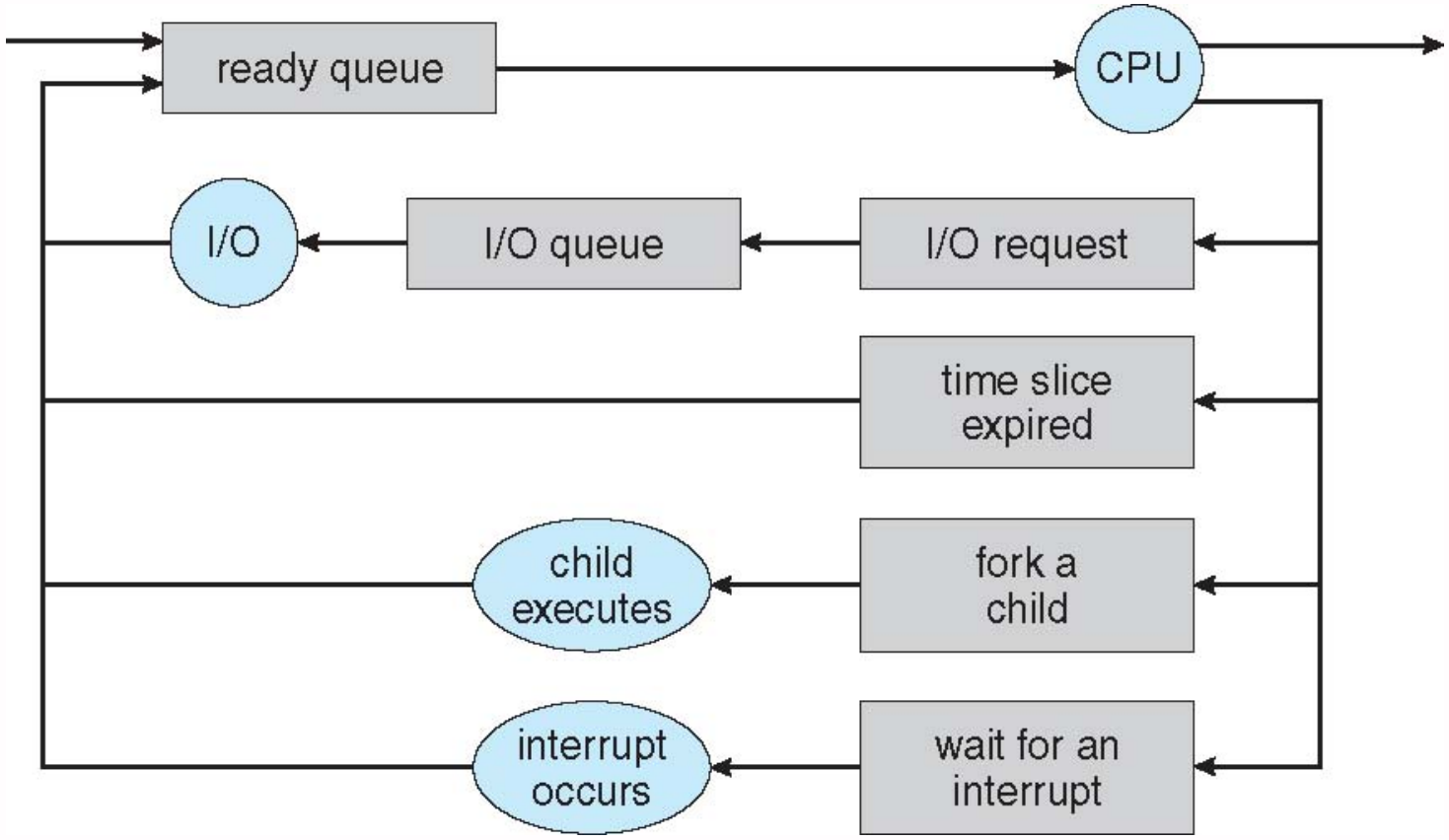
Direprensentasikan dengan structur C yang disebut `task_struct` dan ditemukan pada <linux/sched.h>

```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```

Penjadwalan Proses

- **Multiprogramming** bertujuan untuk memaksimalkan utilitas CPU
- **Time sharing** bertujuan agar lebih dari satu user dapat berinteraksi dengan program yang sedang berjalan
- **Process scheduler** memilih diantara proses yang tersedia untuk dieksekusi oleh CPU
- Pemeliharaan **scheduling queues** proses:
  - **Job queue** – Berisi seluruh proses didalam sistem
  - **Ready queue** – Berisi proses yang berada didalam main memory, baik dalam keadaan *ready* atau *waiting* untuk dieksekusi
  - **Device queues** – Berisi proses dalam keadaan *waiting* untuk perangkat I/O
  - Proses perpindahan antar queues

Representasi Proses Penjadwalan

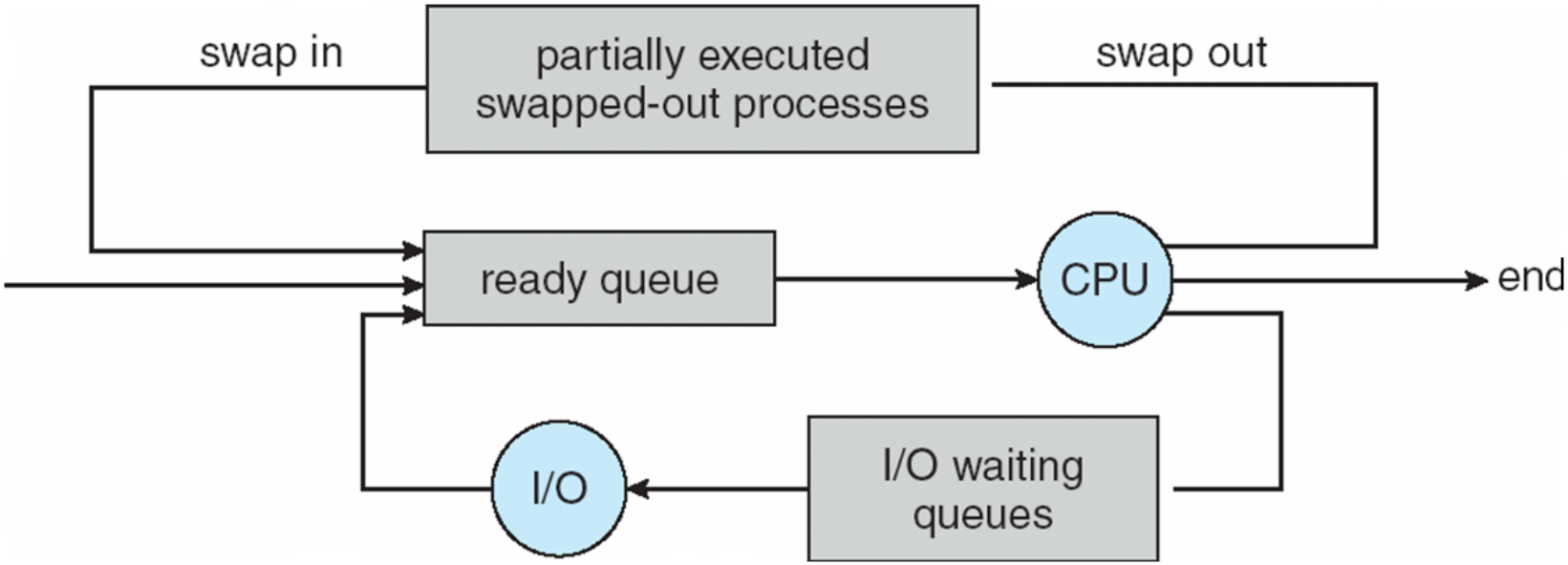


Penjadwalan

- **Short-term scheduler** (or **CPU scheduler**) – memilih proses mana yang selanjutnya akan dieksekusi dan di alokasikan ke CPU
  - Short-term scheduller berlangsung dalam orde millisecond
- **Long-term scheduler** (or **job scheduler**) – memilih proses mana yang akan diletakan pada *ready queue*:
  - Long-term scheduler berlangsung dalam orde detik atau menit
  - Long term scheduler mengkontrol jumlah multiprogramming yang dapat dilakukan
- Processes dapat dideskripsikan:
  - **I/O-bound process** – menghabiskan waktu yang lebih lama pada I/O dibandingkan dengan proses komputasi
  - **CPU-bound process** – Menghabikan waktu yang lebih lama pada proses komputasi

Penjadwalan tambahan

- **Medium-term scheduler** dapat ditambahkan jika multiple programming perlu dikurangi
  - Menghapus process dari memory, menyimpan pada disk, dan mengembalikan dari disk untuk melanjutkan ekskusi: **swapping**



Context Switch

- Ketika CPU berpindah dari satu proses, sistem harus menyimpan keadaan **(saved stated)** dari proses yang lama dan mengembalikan saved state tersebut untuk menjadi proses baru **(stated restore)** melalui **context switch**
- **Context** dari proses direpresentasikan di dalam PCB
- Sistem tidak berkerja ketika sedang melakukan proses *switching*
  - Semakin kompleks SO dan PCB ➔ semakin panjang dan kompleks context switch
- Waktu switching tergantung dengan hardware
  - Beberapa hardware menyediakan beberapa set register untuk setiap CPU ➔ dapat melakukan beberapa context switch dlam satu waktu

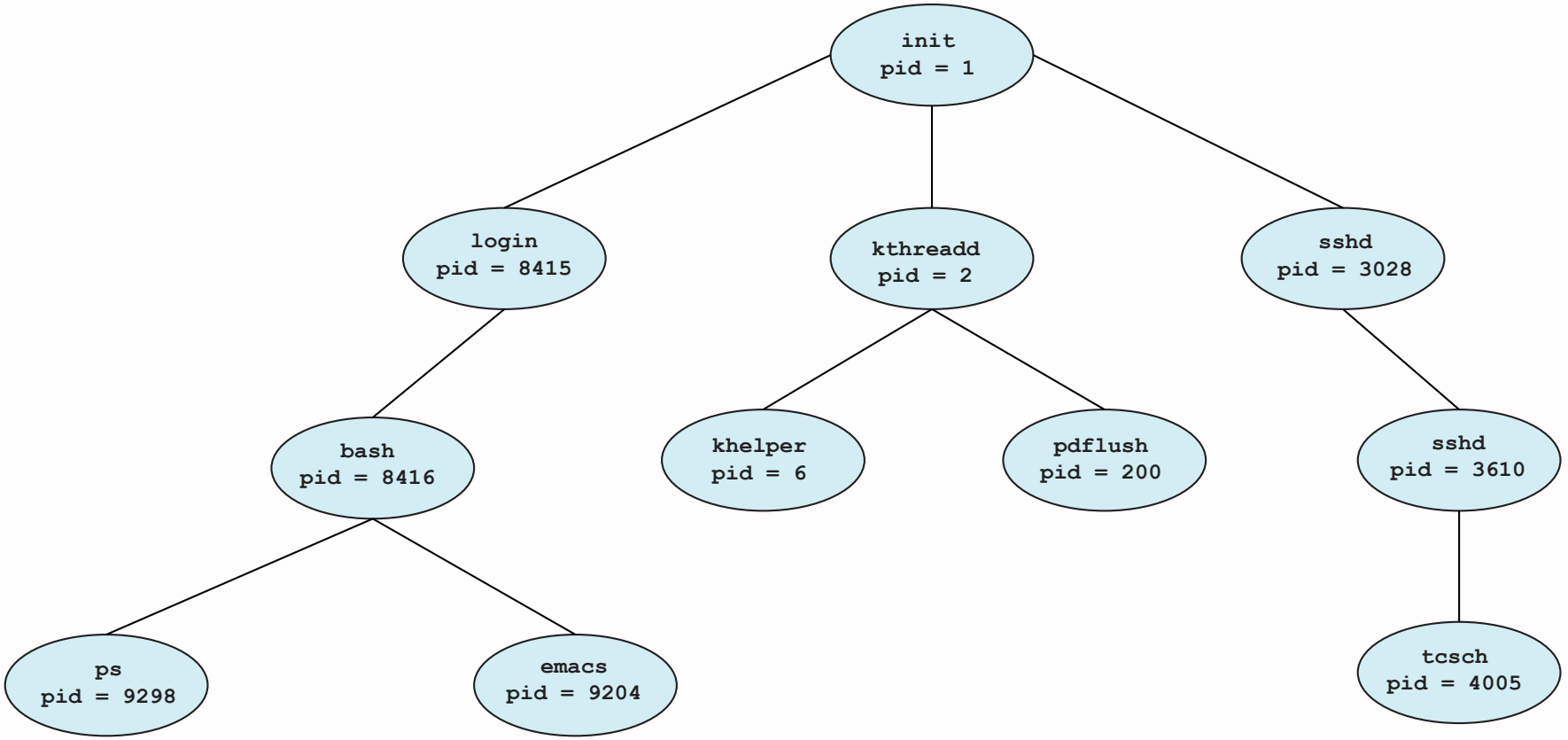
Operasi Dalam Proses

- Sistem Operasi hari menyediakan mekanime untuk :
  - Pembuatan proses,
  - Penghentian proses,
  - dsb

Pembuatan Proses

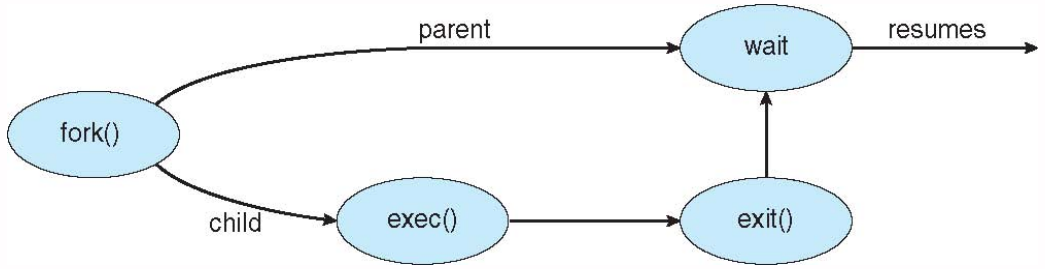
- **Parent** process membuat **children** processes, dengan kata lain membuat proses lainnya, sehingga membetuk **tree** of processes.
- Secara umum, proses diidentifikasi dan diatur melalui **process identifier** (**pid**)
- Eksekusi yang terjadi setelah child proses terbentu
  - Parent dan children dieksekuis secara bersama
  - Parent menunggu sampai children proses terminate

Tree Proses didalam Linux



Pembuatan Proses (lanj...)

- Pengalamatan
  - Child menduplikasi parent
  - Child memiliki program yang di-*load*
- Abstaksi pada UNIX
  - **`fork()`** system call proses baru
  - **`exec()`** system call digunakan setelah **`fork()`** untuk menggantikan proses di dalam memory dengan program baru



C Program Forking

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

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

Membuat Proses via Windows API

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

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
      "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
      NULL, /* don't inherit process handle */
      NULL, /* don't inherit thread handle */
      FALSE, /* disable handle inheritance */
      0, /* no creation flags */
      NULL, /* use parent's environment block */
      NULL, /* use parent's existing directory */
      &si,
      &pi))
    {
       fprintf(stderr, "Create Process Failed");
       return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Pengakhiran Proses

- Beberapa SO tidak mengizinkan child proses untuk ada jika parent prosesnya sudah dimatikan. Ketika ada proses dimatikan, makan seluruh childdren prosesnya juga harus dimatikan.
  - **cascading termination** -> seluruh anak-anak proses dimatikan
  - *Termination* diinsiasi oleh sistem operasi.

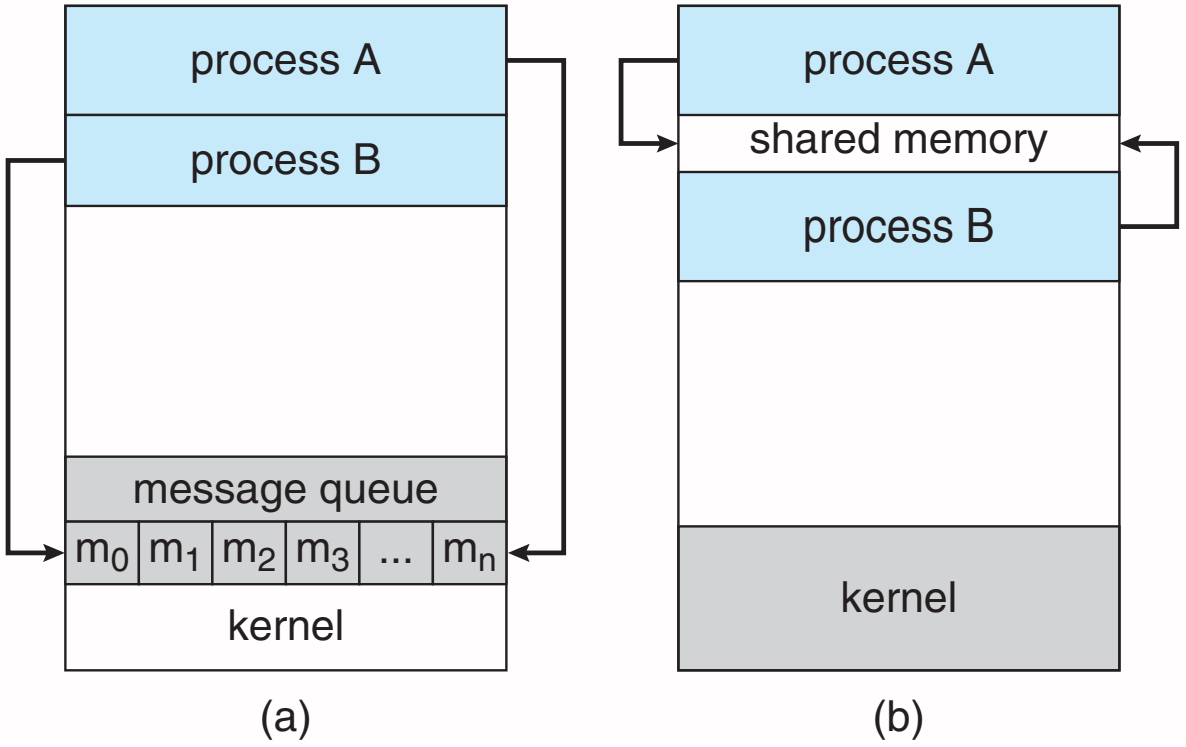
- Parent proses menunggu proses termination dari child dengan menggunakan system call **wait()** system call**.**
- Jika tidak ada parent proses yang menunggu (tidak meminta invoke **wait()**) maka proses ini disebut **zombie**
- Jika parent proses diakhiri tanpa meminta **wait()** , proses ini disebut **orphan**

Komunikasi Antar Proses

- Proses dalam sistem dapat ***independent*** ataupun ***berkerjasama***
- Kerjasama antar proses dapat mempengaruhi atau dipengarudi oleh proses lainnya, termasuk berbagi data antar proses (*sharing data*)
- Alasan proses untuk berbagi:
  - Berbagi informasi
  - Mempercepat proses komputasi
  - Modularitas
  - Kenyamanan
- Komunikasi antar proses membutuhkan **interprocess communication** (**IPC**)
- Ada dua model IPC
  - **Shared memory**
  - **Message passing**

Model IPC

(a) Message passing. (b) shared memory.



Contoh IPC (Pipe)

- Pipes (*pipes)* Berfungsi sebagai pipa penyalur yang mengizinkan dua proses berkomunkasi
- Pipa tidak dapat diakss dari luar proses yang membuatnya. Biasanya parent proses membuat sebuah pipa untuk berkomunikasi dengan *child proses*-nya.
- *Named pipes* – dapat diakses tanpa hubungan *parent-child* proses