Sistem Operasi

Catatan Kuliah #4

Alauddin Maulana Hirzan, M. Kom
0607069401

Manajemen Proses - *Thread dan Symmetric MultiProcessing*

Thread

Thread

Apa itu *Thread*? #1

Jika proses adalah **kode program yang berjalan**, maka *thread* / utas adalah bagian dari proses itu sendiri. Sehingga sebuah proses yang berjalan setidaknya memiliki **1 thread/utas**.

Normalnya, instruksi yang ada di dalam proses hanya bisa dieksekusi oleh sistem operasi satu per satu. Namun dengan menggunakan **thread/utas**, eksekusi paralel menjadi memungkinkan. Hal ini dikarenakan **thread/utas** sama-sama mengakses area kode, memori, dan data yang sama.

Utas dapat digunakan untuk melakukan tugas kompleks seperti penanganan I/O, pembaruan antarmuka pengguna grafis, atau komunikasi jaringan, tanpa memblokir program utama atau menyebabkan penundaan.

Thread

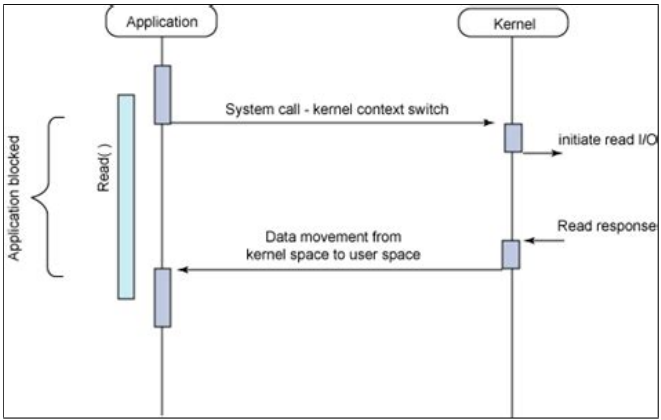
Apa itu *Thread*? #2

Thread sudah muncul di tahun 1993, namun dalam ***Single Thread*** karena di jaman itu belum ada / muncul ide untuk menggunakan **Pemrograman Async**. Sehingga banyak aplikasi pasaran di jaman itu yang menggunakan ***Single-thread*** dan ***Pemrograman Sync***.

Informasi
Paradigma **Pemrograman Async** berbeda dengan ***Pemrograman Sync*** dan baru diterapkan semenjak era **World Wide Web** ada.

Thread

Apa itu *Thread*? #3 - *Synchronous Programming* #1



Thread

Apa itu *Thread*? #4 - *Synchronous Programming* #2

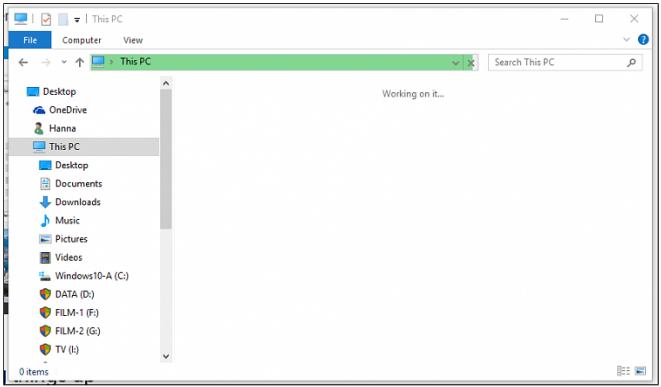
Aplikasi yang berjalan dan akan melakukan akses/komunikasi akan mengalami bloking hingga akses tersebut selesai atau komunikasinya mendapatkan respon. Sehingga permasalahan bisa muncul apabila komunikasi maupun sumber daya yang diakses proses mengalami *error*

Informasi

Contoh sederhana dalam kasus ini seperti **File Explorer** di suatu sistem operasi ketika mencoba mengakses penyimpanan eksternal seperti **Flash Disk**. Aplikasi ini akan melakukan **blocking** apabila **Flash Disk** mengalami **read error** ketika di akses

Thread

Apa itu *Thread*? #5 - *Synchronous Programming* #3



Thread

Apa itu *Thread*? #6 - *Asynchronous Programming* #1

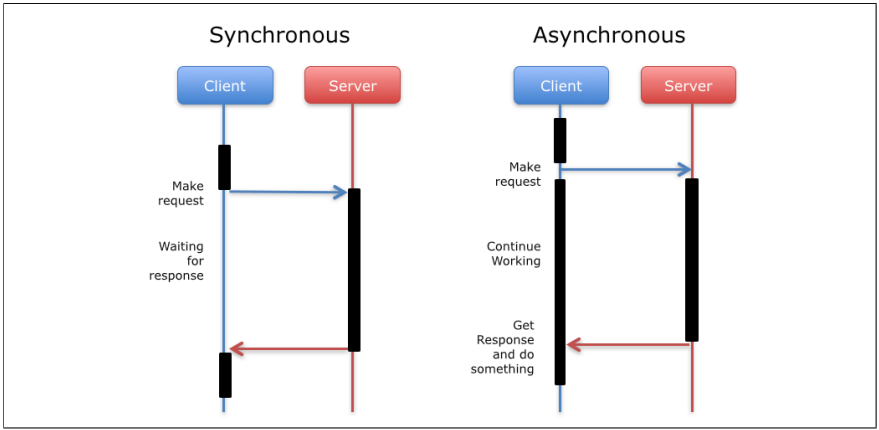
Pemrograman Async memanfaatkan **thread** untuk melakukan pemrosesan lain tanpa harus menunggu I/O menyelesaikan prosesnya atau komunikasi mendapatkan respons.

Di era WWW ini, Server memanfaatkan **thread** untuk menerima dan melayani permintaan dari pengguna lalu mati.

Informasi
Jaman dahulu, **thread** bersifat ***long-lived*** satu **thread** yang berumur panjang. Namun kini **thread** bersifat **short-lived** alias berumur pendek.

Thread

Apa itu *Thread*? #7 - *Asynchronous Programming* #2



Thread

Komponen *Thread* #1

Sama halnya seperti **proses**, **thread** terdiri dari beberapa komponen

1. **Thread ID**
2. **Program Counter**
3. **Stack**
4. **Registers**
5. **Thread-Specific Data**
6. **Scheduling Parameters**
7. **Execution State**

Thread

Komponen *Thread* #2 - **Thread ID**

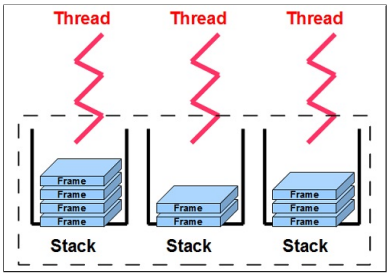
Pengidentifikasi unik yang membedakan utas dari utas lainnya dalam proses yang sama. ID diberikan oleh sistem operasi dan digunakan untuk mengelola status dan sumber daya utas.

Informasi
Thread ID normalnya tidak bisa diakses secara langsung seperti **Process ID**. Hal ini disebabkan oleh jumlah **thread** yang dapat dimunculkan oleh proses

Thread

Komponen *Thread* #3 - **Stack**

Struktur data yang menyimpan variabel sementara, parameter fungsi, dan mengembalikan alamat untuk utas. **Stack** dalam **thread** dapat tumbuh dan menyusut secara dinamis saat utas menjalankan fungsi dan kembali darinya.



Thread

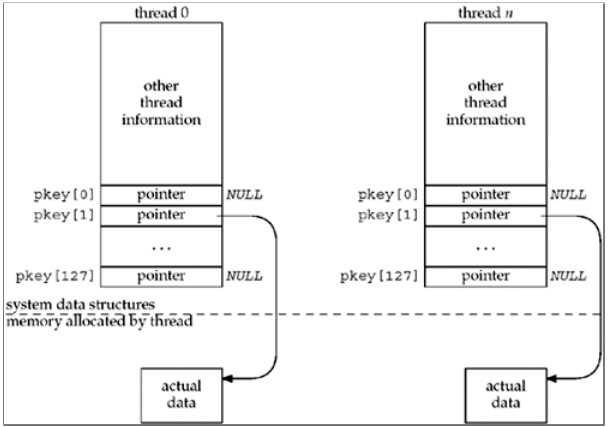
Komponen *Thread* #4 - **Thread-specific Data** #1

Sekumpulan variabel dan struktur data yang bersifat privat untuk thread dan hanya dapat diakses oleh thread itu sendiri. Data khusus thread dapat digunakan untuk mempertahankan informasi status, berbagi data antar fungsi, atau meneruskan informasi antar thread.

Informasi
Data yang ada di dalam **thread** hanyalah dalam bentuk **key** saja, sehingga **thread** hanya cukup menunjuk data yang ada di dalam memori saja.

Thread

Komponen *Thread* #5 - **Thread-specific Data** #2



Thread

Komponen *Thread* #6 - **Scheduling Parameter**

Sekumpulan parameter yang menentukan prioritas thread, kebijakan penjadwalan, dan kuantum waktu.

Parameter daripada Thread:

- **Scope** : Lingkup eksekusi sebuah thread ditentukan oleh model thread yang digunakan dalam pustaka thread.
- **Policy** : Kebijakan penjadwalan sebuah thread mendefinisikan bagaimana penjadwal memperlakukan thread setelah mendapatkan kendali atas CPU.
- **Priority** : Prioritas penjadwalan sebuah thread mendefinisikan tingkat kepentingan relatif dari pekerjaan yang sedang dilakukan oleh setiap thread.

Thread

Komponen *Thread* #7 - **Execution State**

Status thread saat ini, yang dapat berupa berjalan, diblokir, atau menunggu. Status eksekusi menentukan apakah thread secara aktif mengeksekusi instruksi, menunggu peristiwa I/O atau sinkronisasi, atau ditangguhkan oleh sistem operasi.

Informasi
Sama halnya dengan proses, **thread** dapat diatur dengan menggunakan status eksekusi oleh sistem operasi.

Thread

Jenis *Thread* #1

Thread hanya memiliki tiga (3) jenis yang dikategorikan berdasarkan level eksekusinya:

- **User-level Thread (ULT)**
- **Kernel-level Thread (KLT)**
  - **Kernel-supported Thread**
  - **Lightweight Thread**
- ***Hybrid Thread***
  - **LinuxThreads**
  - **Native POSIX Thread Library (NTPL)**

Thread

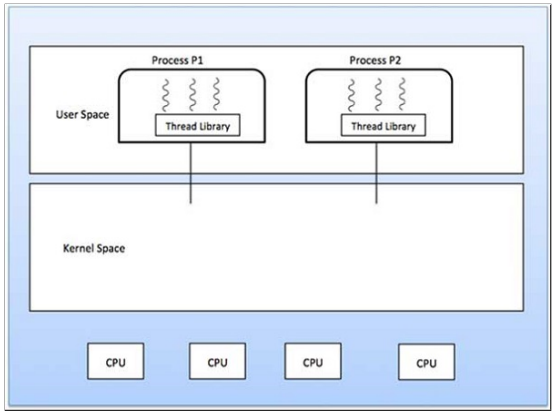
Jenis *Thread* #2 - **User-level Thread (ULT)**

User-level Thread adalah thread yang dikelola oleh aplikasi itu sendiri, bukan oleh sistem operasi. Dalam model ini, pembuatan, penjadwalan, dan sinkronisasi thread dilakukan oleh pustaka tingkat pengguna atau runtime, tanpa melibatkan kernel. Thread diimplementasikan menggunakan struktur data tingkat pengguna dan tidak memiliki pemetaan langsung ke thread atau proses kernel.

- Murah untuk dibuat karena tidak perlu membuat ruang alamat sendiri. Mereka adalah bit memori virtual yang dialokasikan dari ruang alamat pada saat dijalankan.
- Cepat disinkronkan karena sinkronisasi dilakukan pada tingkat aplikasi, bukan pada tingkat kernel.

Thread

Jenis *Thread* #3 - **User-level Thread (ULT)**



Thread

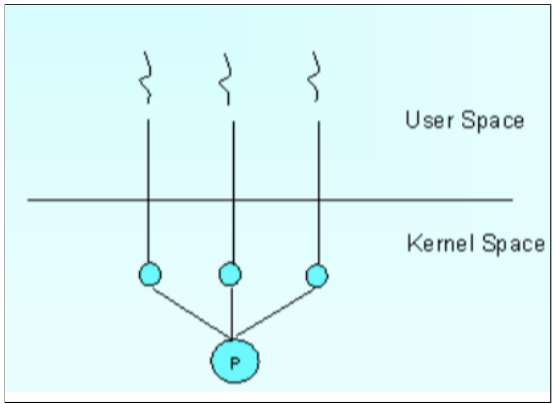
Jenis *Thread* #4 - **Kernel-level Thread (KLT)**

Thread tingkat kernel ditangani oleh sistem operasi secara langsung dan manajemen thread dilakukan oleh kernel. Informasi konteks untuk proses serta thread proses semuanya dikelola oleh kernel. Oleh karena itu, thread tingkat kernel lebih lambat daripada thread tingkat pengguna.

- Beberapa thread dari proses yang sama dapat dijadwalkan pada prosesor yang berbeda dalam thread tingkat kernel.
- Rutinitas kernel juga dapat di-multithread.
- Jika thread tingkat kernel diblokir, thread lain dari proses yang sama dapat dijadwalkan oleh kernel.

Thread

Jenis *Thread* #5 - **Kernel-level Thread (KLT)**



Symmetric MultiProcessing (SMP)

Symmetric MultiProcessing (SMP)

Symmetric MultiProcessing (SMP)

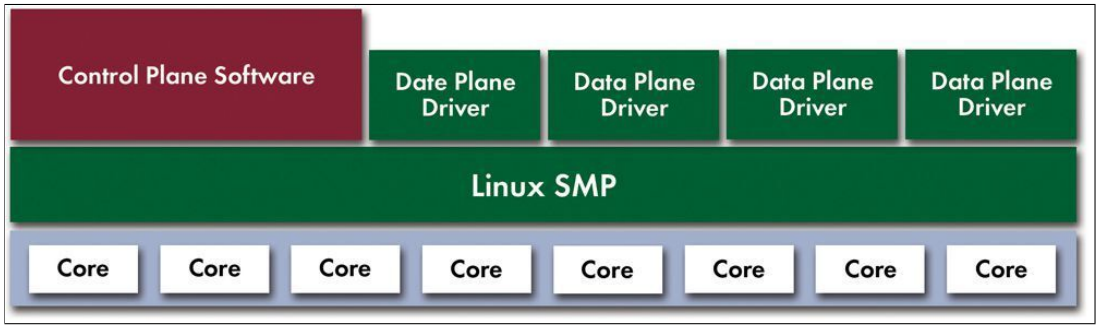
Symmetric MultiProcessing / **SMP** mengacu pada arsitektur komputer di mana beberapa prosesor identik saling terhubung ke satu memori utama bersama, dengan akses penuh ke semua perangkat I/O, tidak seperti MP asimetris. Dengan kata lain, semua prosesor memiliki memori bersama (umum) dan jalur data atau bus I/O yang sama seperti yang ditunjukkan pada gambar.

Karakteristik:

- Identik
- Komunikasi
- Kompleksitas
- Biaya Mahal
- Pembagian Tugas Merata

Symmetric MultiProcessing (SMP)

Symmetric MultiProcessing (SMP)




THANK YOU