MODUL SISTEM OPERASI

(CCI210)

MODUL SESI 3

THREADS

DISUSUN OLEH

ADI WIDIANTONO, SKOM, MKOM.

UNIVERSITAS ESA UNGGUL

2020

THREADS

A. Kemampuan Akhir Yang Diharapkan

Setelah mempelajari modul ini, diharapkan mahasiswa mampu :

1. Memahami konsep dasar dari sebuah thread.
2. Memahami pengelolaan dan pemrosesan thread dalam Sistem Operasi
3. Mamahami proses kerja Sistem Operasi dalam menangani thread

B. Uraian dan Contoh

1. THREAD

Dalam sistem operasi tradisional, setiap proses memiliki ruang alamat dan kendali untuk satu *process*. Dan hal itu hampir merupakan definisi dari sebuah proses. Namun, dalam banyak situasi, diinginkan untuk memiliki beberapa *process control* dalam *address space* yang sama dengan berjalan dalam quasi-parallel, seolah-olah mereka (hampir) merupakan proses terpisah (kecuali untuk ruang alamat bersama) yang dapat dikatakan sebuah proses ringan.

Dari materi tentang proses dan dari hal diatas memiliki karakteristik:

- Resource Ownership
- Scheduling/execution

Dari sudut tugas dari proses, proses terdiri dari 2 macam :

1. Proses berat (heavyweight) disebut sebagai proses tradisional
2. Proses ringan (lightweight) disebut **THREAD**

1.1. Definisi Thread

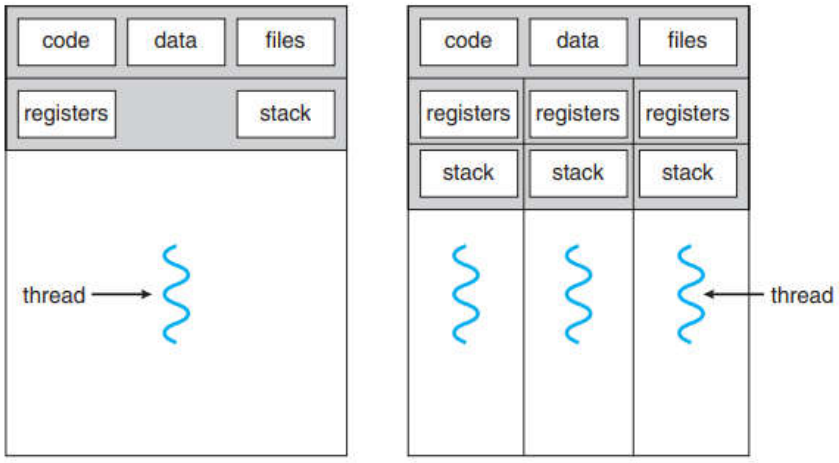
Thread dapat didefinisikan sebagai :

- Thread adalah pelaksanaan instruksi terkecil dari sebuah program yang dapat dikelola secara independen sesuai dengan urutan atau waktu yang diinginkan (dapat bersamaan).
- Sebuah thread adalah eksekusi sederhana dalam proses.

Proses dapat mengandung satu atau lebih thread. Menurut jumlah thread yang terlibat dalam proses, ada dua jenis proses yaitu:

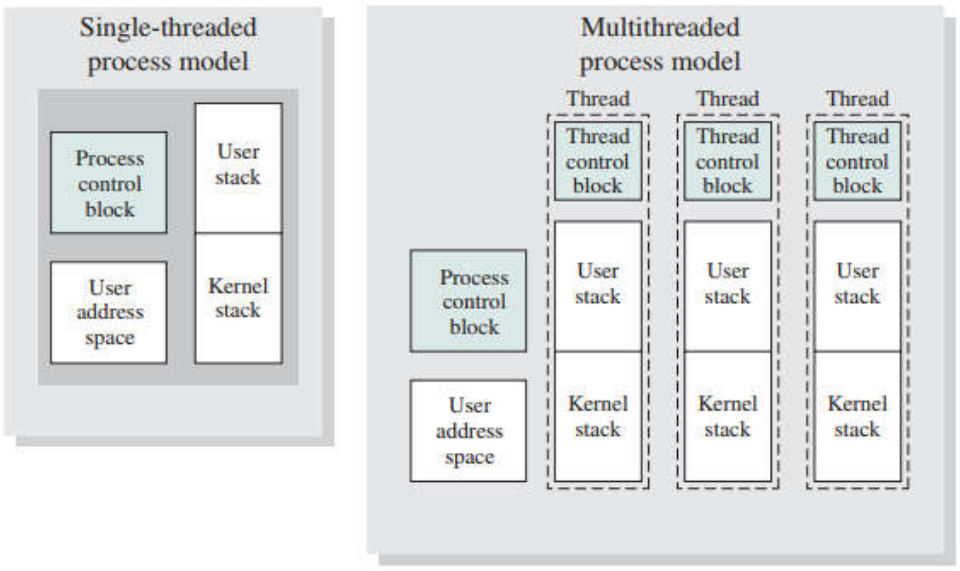
- proses single-thread - adalah proses yang hanya memiliki satu thread, jadi thread sama dengan proses itu sendiri, dan hanya ada satu aktivitas yang terjadi.
- proses multi-thread - dalam sebuah proses ada lebih dari satu thread, dan ada lebih dari satu aktivitas yang terjadi.

Proses thread dapat diilustrasikan seperti pada gambar 1.1.



Gambar 1.1. Proses single dan multi thread

Atau seperti pada gambar 1.2. dengan sudut pandangan *process management*.

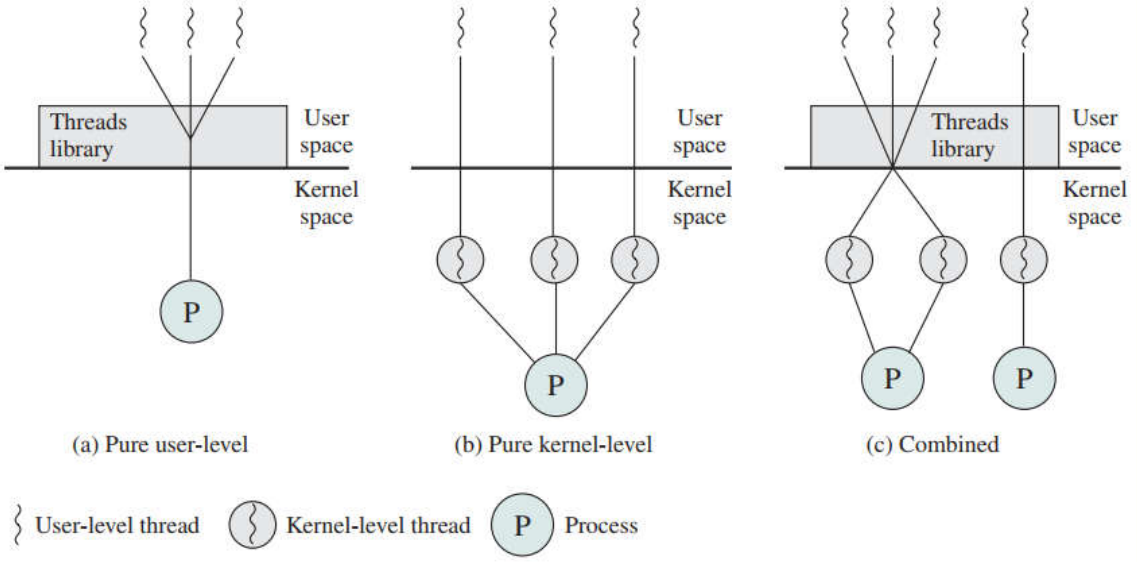


Gambar 1.2. Single dan multi-thead in process management view

1.2. Jenis Thread

Berdasarkan pengelolaannya, thread dibagi menjadi dua jenis yaitu:

- User-Level-Thread (ULT) adalah pengelolaan thread yang dilakukan oleh user level (pengguna) yaitu aplikasi, dan kernel tidak dilibatkan dalam penangannya.
- Kernel-Level-Thread (KLT) adalah pengelolaan thread yang dilakukan oleh kernel sepenuhnya sehingga user /aplikasi tidak dapat menginterupsi.



Gambar 1.3. Jenis Thread

User LeveL Thread

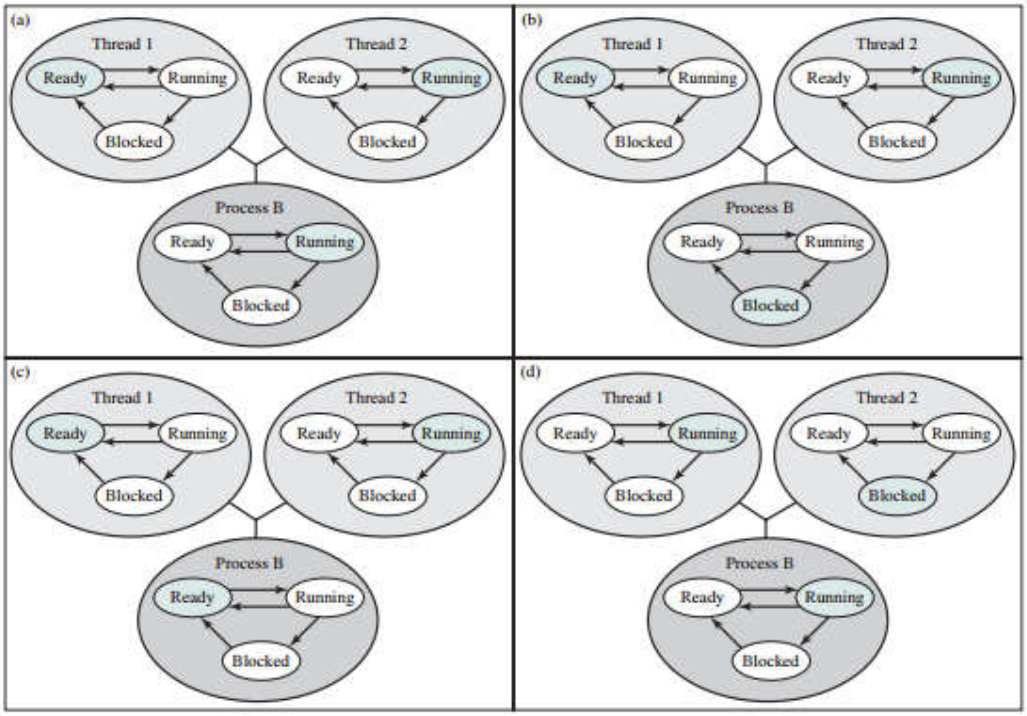
Dalam ULT, aplikasi dapat diprogram untuk menjadi multithread dengan menggunakan *thread library*, yang merupakan paket dari *routines* untuk manajemen ULT. *Thread library* berisi kode untuk membuat dan menghancurkan *thread*, untuk meneruskan pesan dan data antar *thread*, untuk penjadwalan eksekusi *thread*, dan untuk menyimpan dan memulihkan konteks *thread*.

Secara default, aplikasi dimulai dengan satu *thread*. Aplikasi ini dan *thread*-nya dialokasikan ke satu proses yang dikelola oleh kernel. Kapan pun aplikasi berjalan (proses sedang dalam status Running), aplikasi mungkin menerbitkan thread baru untuk dijalankan dalam proses yang sama. Penerbitan dilakukan dengan menjalankan fungsi penerbitan di *thread library*. Kontrol diteruskan ke utilitas itu dengan pemanggilan prosedur (*procedure call*). *Thread Libray* membuat file struktur data untuk thread baru dan kemudian meneruskan kontrol ke salah satu thread

dalam proses ini yang berada dalam status Siap (*Ready*), dengan menggunakan beberapa algoritme penjadwalan.

Ketika kontrol diteruskan ke *library*, konteks thread disimpan, dan ketika kontrol diteruskan dari *library* ke thread, konteks thread itu dipulihkan. Konteks pada dasarnya terdiri dari isi register pengguna, file penghitung program, dan penunjuk stack.

Semua aktivitas yang dijelaskan di paragraf sebelumnya terjadi di ruang pengguna dan dalam satu proses. Kernel tidak mengetahui aktivitas ini. Kernel terus menjadwalkan proses sebagai satu unit dan menetapkan status eksekusi tunggal (Siap, Berjalan, Diblokir, dll.) ke proses tsb.



Gambar 1.4. Contoh hubungan status dari proses dan status dari thread.

Kelebihan ULT :

- Peralihan thread tidak memerlukan mode kernel karena semua file struktur data manajemen benang berada dalam ruang alamat dari proses tunggal. Hal Ini menghemat overhead dari dua sakelar mode (pengguna ke kernel; kernel kembali ke pengguna).
- Penjadwalan bisa dikhususkan pada aplikasi. Satu aplikasi mungkin paling diuntungkan dari algoritma penjadwalan round-robin sederhana, sementara yang lain mungkin mendapat manfaat dari algoritma penjadwalan berbasis

prioritas. Algoritme penjadwalan bisa disesuaikan dengan aplikasi tanpa mengganggu penjadwal OS yang mendasarinya.

- ULT dapat berjalan di OS apa pun. Tidak ada perubahan yang diperlukan pada kernel yang mendasarinya untuk mendukung ULT. *Thread Library* adalah sekumpulan fungsi tingkat aplikasi dibagikan oleh semua aplikasi.

Kelemahan ULT:

- Dalam OS biasa, panggilan sistem bersifat memblokir. Akibatnya, saat ULT menjalankan panggilan sistem, tidak hanya thread itu diblokir, tetapi juga semua thread dalam proses diblokir.
- Aplikasi multithread tidak dapat memanfaatkan multiprocessing. Kernel menetapkan satu proses ke hanya satu prosesor pada satu waktu. Oleh karena itu, hanya satu thread dalam suatu proses yang dapat dijalankan pada satu waktu. Akibatnya, bahwa multiprogramming tingkat aplikasi dalam satu proses.

Kernel Level Thread

Dalam KLT, semua pekerjaan pengelolaan thread dilakukan oleh kernel. Tidak ada kode manajemen thread di level aplikasi, cukup sebuah antarmuka pemrograman aplikasi (API) ke kernel *thread facility*. OS Windows adalah contoh dari pendekatan ini. Kernel mempertahankan konteks informasi untuk proses secara keseluruhan dan untuk thread secara individual dalam proses. Penjadwalan oleh kernel dilakukan berbasis thread (bukan proses).

Kelebihan KLT: .

- Kernel bisa secara bersamaan menjadwalkan beberapa thread dari proses yang sama pada beberapa prosesor.
- Jika satu thread dalam proses diblokir, kernel dapat menjadwalkan thread lainnya proses yang sama.
- Rutinitas kernel sendiri bisa multithread.

Kelemahan KLT :

Transfer kontrol dari satu thread ke thread lain dalam satu proses membutuhkan mode beralih ke kernel.

Pendekatan Gabungan KLT dan ULT

Beberapa sistem operasi menyediakan ULT / Fasilitas KLT (Gambar 1.3.c). Dalam sistem gabungan, pembuatan thread yang diselesaikan sepenuhnya di ruang pengguna, seperti sebagian besar penjadwalan dan sinkronisasi thread dalam aplikasi. Beberapa ULT dari satu aplikasi adalah dipetakan (lebih kecil atau sama) ke sejumlah KLT. Programmer mungkin menyesuaikan jumlah KLT untuk dicapai oleh aplikasi dan prosesor tertentu untuk hasil keseluruhan terbaik.

Dalam pendekatan gabungan, beberapa thread dalam aplikasi yang sama bisa berjalan secara paralel pada beberapa prosesor, dan panggilan sistem pemblokiran tidak perlu diblokir seluruh proses. Jika dirancang dengan benar, pendekatan ini dapat menggabungkan keunggulan pendekatan ULT dan KLT murni sambil meminimalkan kerugian.

Solaris adalah contoh yang baik dari sebuah OS yang menggunakan pendekatan gabungan ini. Saat ini versi Solaris membatasi hubungan ULT / KLT menjadi one-to-one thread.

1.3. Karakeristik Thread

KarakteristikThread :

- Kuat/Robust, karena thread bisa melakukan proses apa saja.
- Ringan/Lightweight, membutuhkan sumberdaya yang lebih sedikit (sharing resources).
- Membaca dan menulis ke variabel dan struktur data variable yang sama
- Dapat berkomunikasi antara thread dengan mudah.

Keuntungan menggunakanThread:

- Tanggap: Multi-threading mengizinkan program untuk terus berjalan walaupun pada bagian program tersebut diblock atau sedang dalam keadaan menjalankan operasi yang lama/panjang. Contohnya multithread web browser dapat mengizinkan pengguna berinteraksi dengan suatu thread ketika suatu gambar sedang diload oleh thread yang lain.
- Pembagian sumber daya: Secara default, thread membagi memori dan sumber daya dari proses. Keuntungan dari pembagian kode adalah aplikasi

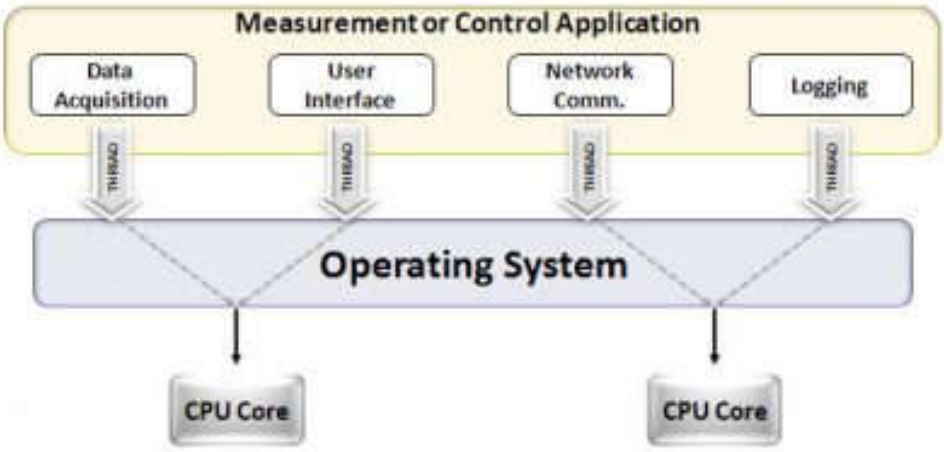
mempunyai perbedaan aktifitas thread dengan alokasi. Selain itu komunikasi antar thrad jadi mudah dan efisien.

- Ekonomis: Mengalokasikan memori dan sumber daya untuk membuat proses itu boros. Alternatifnya thread membagi sumber daya dari proses, jadi lebih ekonomis. Sehingga waktu untuk membuat, menghapus, dan beralih thread lebih sedikit / cepat dari pada proses.
- Pemberdayaan arsitektur multiprosesor: Keuntungann dari multithreading dapat ditingkatkan dengan arsitektur multiprosesor, dimana setiap thread dapat berjalan secara parallel pada prosesor yang berbeda.

2. Multi-Threading

Multi-Threading adalah membagi pekerjaan besar/ proses berat (heavyweight) menjadi beberapa bagian dan masing-masing ditugaskan untuk unit eksekusi yang disebut thread.

Saat ini multi-threading telah menjadi pendekatan umum untuk banyak masalah. Namun memerlukan kehati-hatian pemrograman karena thread berbagi struktur data yang dimodifikasi oleh thread lain pada satu waktu dan juga karena thread berbagi ruang alamat atau memori yang sama. Satu keuntungan lebih dari thread adalah bahwa thread menyediakan cara yang efisien dan efektif untuk mencapai paralelisme. Throughput system dapat ditingkatkan dengan menjalankan beberapa thread pada beberapa prosesor karena thread merupakan entitas independen yang dapat dijadwal (schedulable). Gambaran multi-thread seperti kita sedang mengetik di Ms Word, sementara ada proses cek grammar, dan lain menyimpan file di background, satu waktu ada tiga thread berjalan.



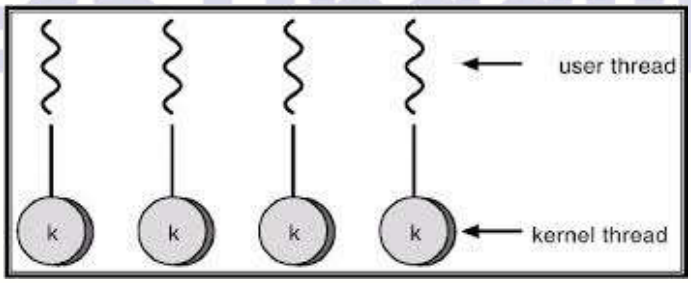
Gambar 2.1. Ilustrasi multi-threading

2.1. Model Multi-Threading

One to one: Memetakan setiap user thread ke dalam 1 kernel thread.

- Kelebihan: lebih sinkron, karena mengizinkan thread lain untuk berjalan ketika suatu thread membuat pemblokingan terhadap sistem pemanggilan, hal ini juga membuat multiplethread bisa berjalan secara parallel dalam multiprosesor .
- Kekurangan: dalam pembuatan userthread diperlukan pembuatan korespondensi user thread.

Karena dalam proses pembuatan kernelthread dapat mempengaruhi kinerja dari aplikasi, maka kebanyakan dari implementasi model ini membatasi jumlah thread yang didukung oleh sistem.Contoh : Windows 95/98/NT/2000



Gambar 2.2. One to One thread

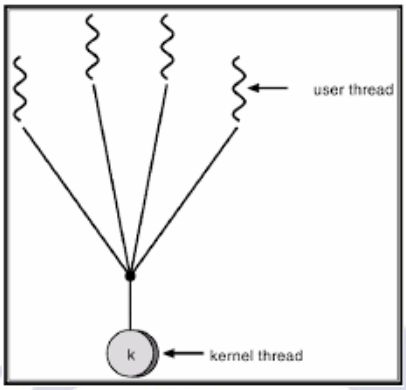
Many to One: memetakan beberapa thread user-level ke dalam satu kernel thread.

- Kelebihan: Managemen proses thread dilakukan oleh (di ruang) pennguna/sistem operasi, sehingga menjadi lebih efisien.

- Kekurangan: multithread tidak dapat berjalan atau bekerja secara paralel di dalam multiprosesor karena hanya satu thread saja yang bisa mengakses kernel dalam suatu waktu.

Penggunaannya pada system tidak memerlukan dukungan kernel thread.

Contoh :Solaris Green dan GNU Portable.

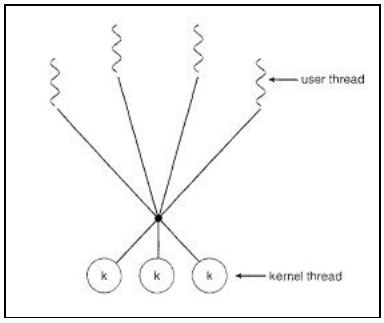


Gambar 2.3. Many to One thread

Many to Many : Membolehkan setiap tingkatan user thread dipetakan ke banyak kernel thread.

- Kelebihan:
  - Developer dapat membuat user thread sebanyak yang diperlukan dan kernel thread yang bersangkutan dapat berjalan secara parallel pada multiprocessor.
  - Dan ketika suatu thread menjalankan blocking systemcall maka kernel dapat menjadwalkan thread lain untuk melakukan eksekusi.
- Kekurangan:Developer dapat membuat userthread sebanyak mungkin, tetapi konkurensi tidak dapat diperoleh karena hanya satu thread yang dapat dijadwalkan oleh kernel pada suatu waktu.

Contoh :Solaris, IRIX, dan Digital UNIX.

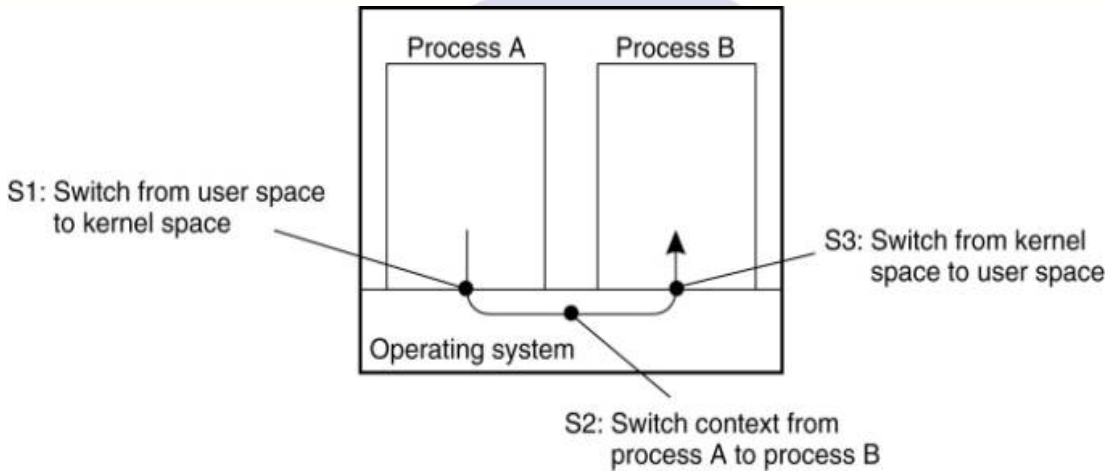


Gambar 2.4. Many to many thread

1.2. Pemanfatan Multi Threading

Thread on Single Processor

Pemanfaatan multi-thread yang utama adalah dengan menjalankan thread secara bersamaan sehingga tercipta proses secara parallel. Penerapan pada arsitektur prosesor tunggal, CPU biasanya berpindah-pindah antara setiap thread dengan cepat, sehingga terdapat ilusi proses paralel, tetapi pada kenyataannya hanya satu thread yang berjalan di setiap waktu. Dibutuhkan mekanisme komunikasi antar proses yang disebut Interproccess Communication (IPC) untuk melakukan proses berpindah (Switching). IPC ini yang menciptakan proses perpindahan (switching) antar proses dengan sangat cepat.

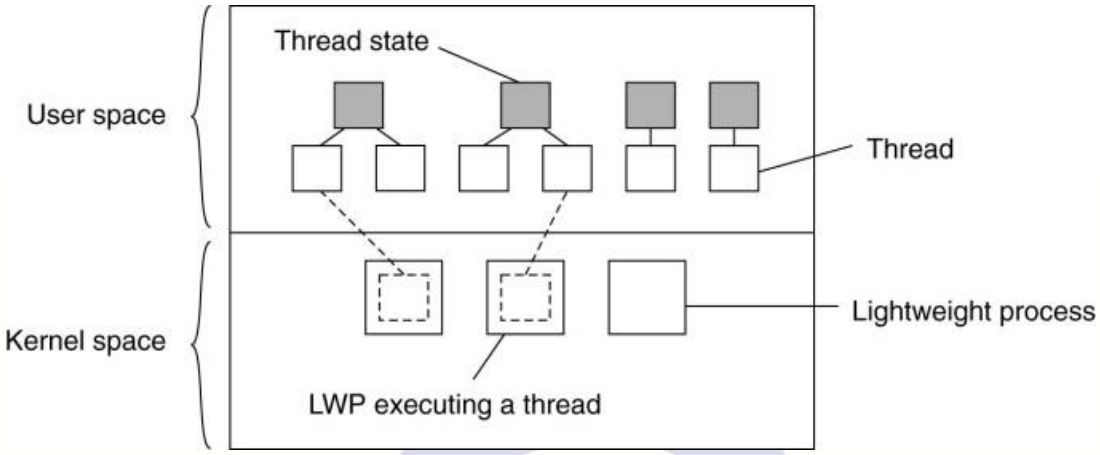


Gambar 2.4. perpindahan proses dengan IPC

Gabungan User dan Kernel Level -Thread.

Terdapat beberapa teknik pemrogramman untuk menciptakan multi-threading dengan menggunakan library sebagai penggunaan sumberdaya bersama, dan memecah task menjadi fungsi kecil-kecil yang dapat digunakan berulang dan bersama. Hal ini akan menciptakan multi-thread pada User-Level atau sebagai User-Thread. Namun pada kenyataanya, proses dari thread tetap saja melakukan pencegahan terhadap proses thread lain jika terjadi bloking (seperti menunggu entry/masukan dari keyboard), walaupun dalam arsitektur multi prosesor sekalipun. Agar supaya tetap berjalan maka pada saat proses, system operasi secara tidak telihat dari sisi user akan menciptakan thread dalam bentuk Lightweight Proccess (LWP) pada Kervel-Level. LWP ini proses yang ringan yang dapat berjalan independent. LWP akan mencari user-thread yang mungkin dijalankan, dan

memprosesnya, sehingga tercipta proses parallel. Hal ini disebut kombinasi dari User-Level dan Kernel-Level Thread.



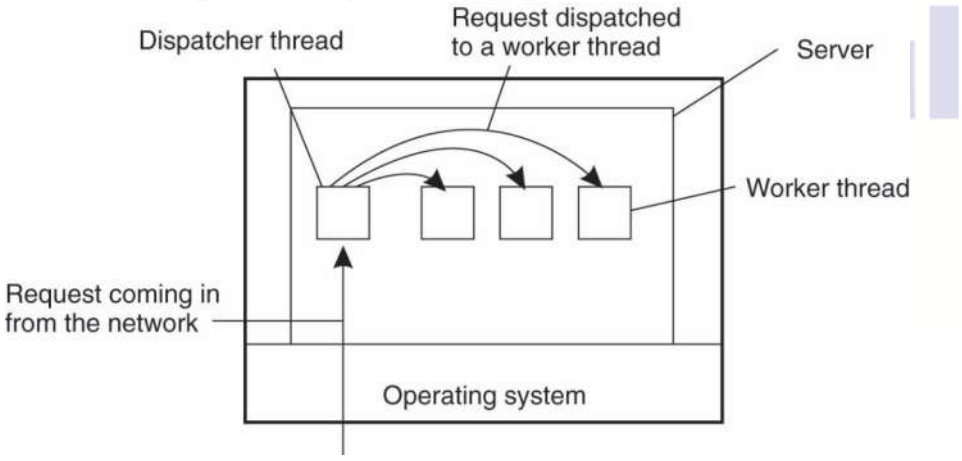
Gambar 2.5. Kombinasi User-Kernel Thread

Multi-Threaded Server

Konsepnya adalah membagi tugas dalam bentuk thread ke multi prosessor dalam sebuah server (multi-proccessor architecture server), tidak seperti single prosesor beberapa tugas dilakukan bergantian.

Beberapa tugas terdiri dari :

- Dispatcher thread – satu prosesor bertugas sebagai pembagi dan pengumpul tugas, dan
- Worker thread - prosesor lain sebagai pekerja thread.



Gambar 2.6. A multithreaded server organized in a dispatcher/worker model.

Ada kelemahan, jika salah satu thread ada tugas membaca file, maka terjadi idle di prosesor tersebut dan bisa juga prosesor pembagi juga idle jika menunggu proses lengkap (blocking). Untuk mengatasi tersebut dapat dilalukan pemilihan thread dan pembagian spesialisasi dari prosesor.

Terdapat 3 cara membangun sebuah server multi-prosesor dengan berdasarkan dari model thread itu sendiri yaitu:

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls

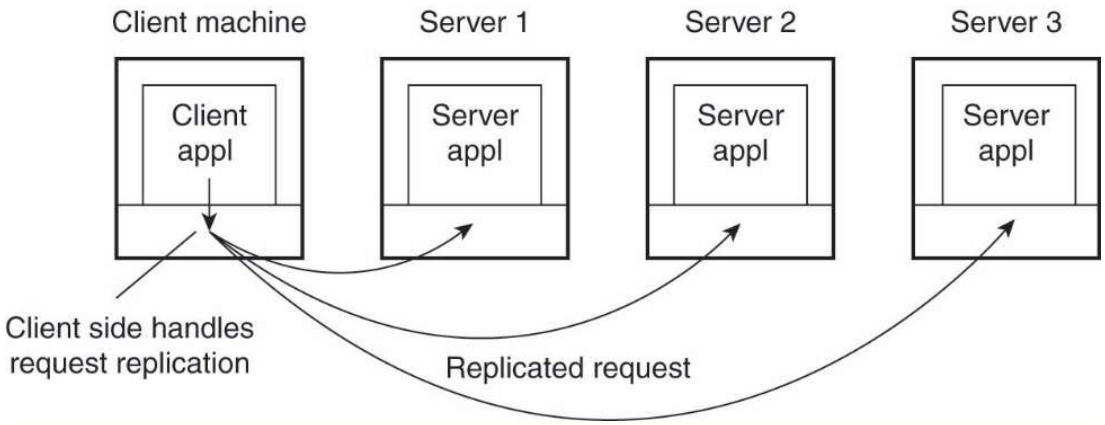
Dengan model di atas, sistem yang idle dapat di minimalkan, dan mendapatkan peningkatan kemampuan yang maksimal.

Multi-Threaded Client

Multi-threaded client ini secara konsep adalah menjalankan multi-thread di sisi client yang disebar kebeberapa server.

Contoh proses ini adalah web browser. Dalam banyak kasus, sebuah dokumen web terdiri dari sebuah file HTML yang berisiteks biasa bersama dengan kumpulan gambar, ikon, dll. Untuk mengambil setiap elemen dari dokumen web, browser harus mengatur sambungan TCP/IP, membaca data masuk, dan menyebarkannya ke komponen tampilan. Mengatur koneksi serta membaca data yang masuk,memblokir operasi secara inheren.

Secara sederhana, thread sudah dibagi dalam di tingkat client (web browser) dan menyebarkan tugas (thread) ke server – server dalam jaringan untuk memenuhi tugasnya, sehingga memungkinkan dalam waktu singkat memenuhi seluruh kebutuhan dari tampilan web tersebut. Sementara itu web browser juga menyembunyikan tampilan yang belum terselesaikan, sehingga secara umum web tetap bekerja (ditampilkan) sambil menunggu proses lainnya selesai. Dari gambaran tersebut, web browser melaksanakan beberapa tugas secara bersamaan(multi-tasking), dengan demikian perogramman pembuatan web browser memiliki konsep multi-threaded client.



Gambar 2.7. Transparent replication of a server using a client-side solution.

Perkembangan dari gabungan server dan client multi-thread, mendorong kearah teknologi virtualisasi.

3. Pemanfaatan Thread dalam Sistem Operasi

3.1. Sistem Operasi Windows

Karakteristik

Karakteristik proses dalam OS Windows adalah sebagai berikut:

- Proses Windows diimplementasikan sebagai objek.
- Sebuah proses dapat dibuat sebagai proses baru, atau sebagai salinan dari proses yang sudah ada.
- Proses yang dapat dieksekusi mungkin berisi satu atau lebih thread.
- Kedua objek proses dan thread memiliki kemampuan sinkronisasi bawaan.

Multithreading

Windows mendukung konkurensi antar proses karena thread berbeda proses dapat dijalankan secara bersamaan (tampaknya berjalan pada waktu yang sama). Selain itu, beberapa thread dalam proses yang sama dapat dialokasikan ke prosesor terpisah dan dieksekusi secara bersamaan (sebenarnya dijalankan pada waktu yang sama). Proses multithread mencapai konkurensi tanpa overhead menggunakan banyak proses. Thread dalam proses yang sama dapat bertukar informasi melalui alamat ruang umum mereka dan memiliki akses ke sumber daya bersama dari proses tersebut. Thread berbeda proses dapat bertukar informasi melalui memori bersama yang telah diatur antara dua proses tersebut.

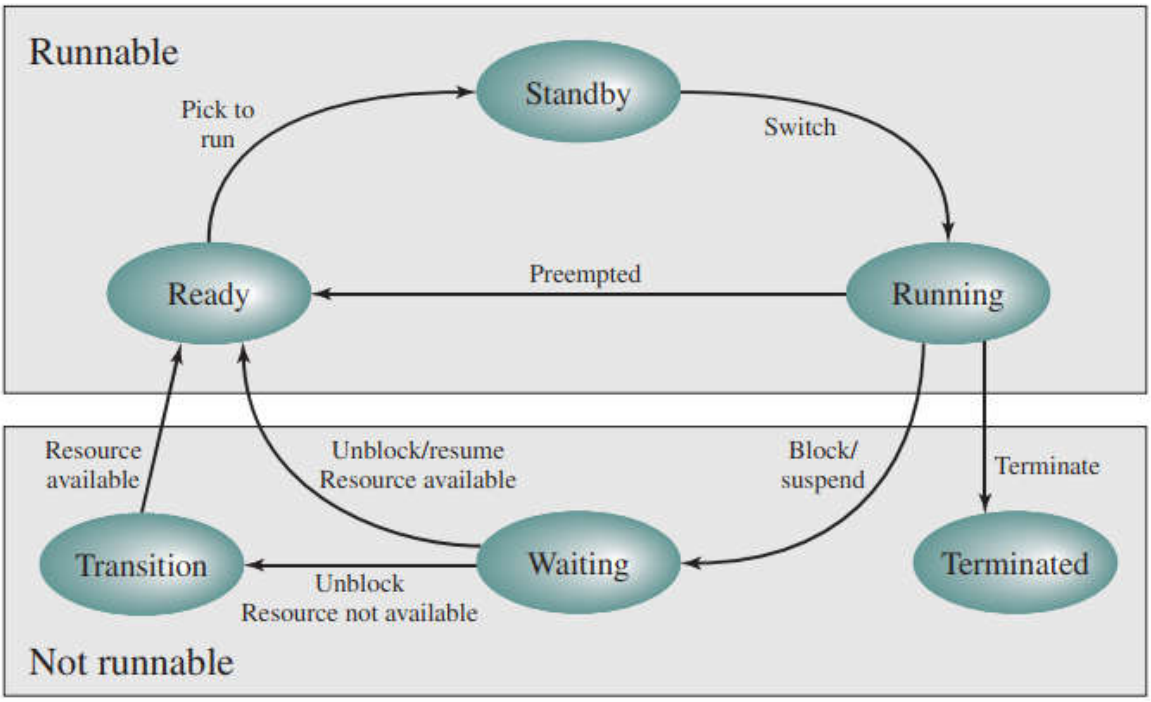
Proses multithread berorientasi objek adalah cara implementasi yang efisien aplikasi server. Misalnya, satu proses server dapat melayani sejumlah klien secara bersamaan.

Status Thread

Thread Windows yang ada berada dalam salah satu dari enam status:

- Siap/Ready : Thread siap dijadwalkan untuk dieksekusi. Operator Kernel melacak semua thread yang siap dan menjadwalkannya dalam urutan prioritas.
- Siaga/Standby: Thread siaga telah dipilih untuk dijalankan berikutnya pada prosesor tertentu. Thread menunggu dalam keadaan ini sampai prosesor tersebut tersedia. Jika prioritas thread standby cukup tinggi, thread yang berjalan di atasnya dapat diutamakan untuk mendukung thread siaga. Jika tidak, file thread standby menunggu sampai thread yang berjalan memblokir atau menghabiskan sisa waktunya.
- Berjalan/Running : Setelah operator Kernel melakukan perpindahan thread, thread standby memasuki status Running dan memulai eksekusi dan melanjutkan eksekusi hingga didahului oleh thread dengan prioritas lebih tinggi, menghabiskan sisa waktunya, blok, atau berakhir. Dalam dua kasus pertama, dikembalikan ke status Siap.
- Menunggu/Wait: Sebuah thread memasuki status Menunggu ketika (1) itu diblokir di sebuah kejadian (misalnya, I / O), (2) menunggu untuk tujuan sinkronisasi, atau (3) sebuah subsistem lingkungan mengarahkan thread untuk menangguhkan dirinya sendiri. Saat menunggu, jika kondisi terpenuhi, thread berpindah ke status Siap jika semua sumber dayanya tersedia.
- Transition: Sebuah thread memasuki status ini setelah menunggu jika siap untuk dijalankan tetapi sumber daya tidak tersedia. Misalnya, kehabisan memori. Saat sumber daya tersedia, thread masuk ke status siap.
- Dihentikan/Terminated: Sebuah thread dapat diakhiri dengan sendirinya, oleh thread lain, atau ketika proses induknya berakhir. Setelah pekerjaan selesai, file thread dihapus dari system.

.



Gambar 3.1. Status Thread OS Windows

Beberapa hal yang dapat didukung OS Window adalah:

- Mendukung Subsystem OS
- Mendukung Symetric Multiprocessing (SMP)
    - Mendukung konfigurasi hardware SMP

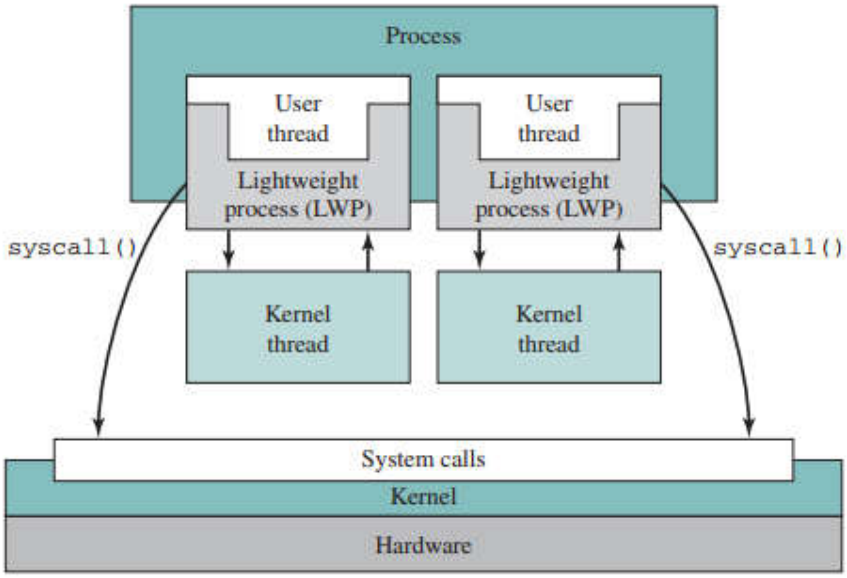
3.2. Sistem Operasi Solaris

Arsitektur Multithreaded

Solaris menggunakan empat konsep terkait thread:

- Proses: Ini adalah proses UNIX normal dan menyertakan alamat ruang, tumpukan, dan blok kontrol proses.
- Thread tingkat pengguna: Diimplementasikan melalui pustaka thread di alamat ruang proses, ini tidak terlihat oleh OS. Thread tingkat pengguna (ULT) adalah unit eksekusi yang dibuat pengguna dalam suatu proses.
- Proses ringan: Proses ringan (LWP) dapat dilihat sebagai pemetaan antara ULT dan thread kernel. Setiap LWP mendukung ULT dan peta ke satu thread kernel. LWP dijadwalkan oleh kernel secara independen dan dapat dijalankan secara paralel pada multiprosesor.

- Thread Kernel: Ini adalah entitas fundamental yang dapat dijadwalkan dan dikirim untuk dijalankan di salah satu prosesor sistem.



Gambar. 3.2. Proses dan Thread di Solaris

Eksekusi Thread

Gambar 3.3 menunjukkan tampilan yang disederhanakan dari kedua status eksekusi thread. Status ini mencerminkan status eksekusi dari thread kernel dan LWP yang terikat padanya. Beberapa thread kernel tidak terkait dengan LWP; diagram eksekusi yang sama berlaku.

Status tersebut adalah sebagai berikut:

• RUN: Thread dapat dijalankan; artinya, thread siap untuk dieksekusi.

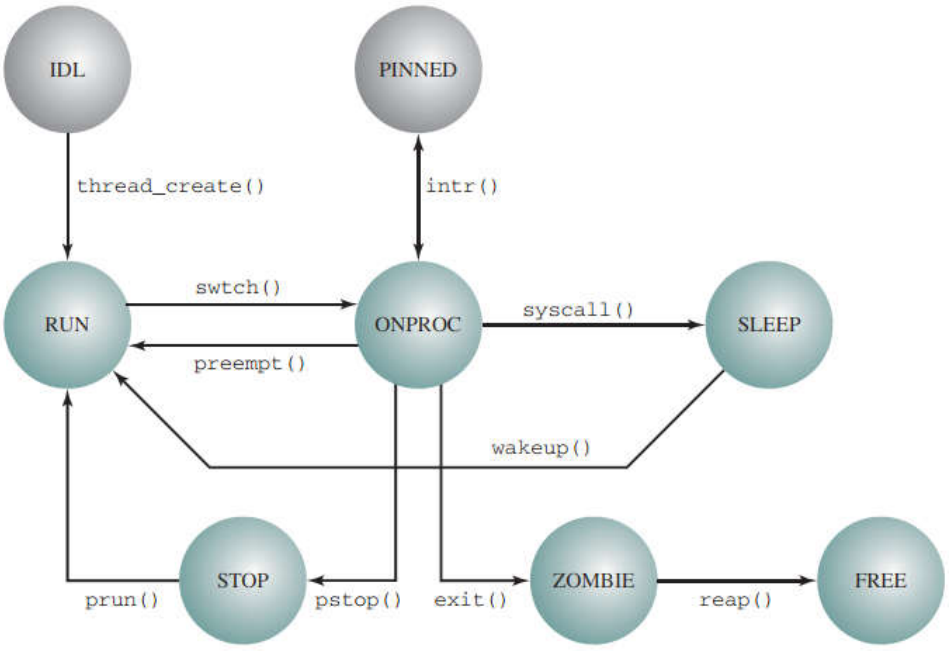
• ONPROC: Thread sedang dijalankan pada prosesor.

• SLEEP: Thread diblokir.

• STOP: Thread dihentikan.

• ZOMBIE: Thread telah dihentikan.

• FREE: Sumber daya thread telah dirilis dan thread sedang menunggu penghapusan dari struktur data thread OS.



Gambar 3.3. Status thread dalam Solaris.

3.3. Sistem Operasi Linux

Tugas Linux

Sebuah proses, atau tugas, di Linux diwakili oleh struktur data *task_struct*. Merupakan struktur data *task_struct* berisi informasi dalam beberapa kategori:

• Status: Status eksekusi proses (mengeksekusi, siap, ditangguhkan, berhenti, zombie).

• Informasi penjadwalan: Informasi yang dibutuhkan oleh Linux untuk menjadwalkan proses. Suatu proses bisa normal atau real time dan memiliki prioritas. Proses waktu nyata dijadwalkan sebelum proses normal, dan dalam setiap kategori, prioritas relatif dapat digunakan. Penghitung melacak jumlah waktu suatu proses diizinkan untuk mengeksekusi.

• Pengidentifikasi: Setiap proses memiliki pengidentifikasi proses yang unik dan juga memiliki pengguna dan pengenal grup. Pengidentifikasi grup digunakan untuk menetapkan hak akses sumber daya ke grup proses.

• Komunikasi antarproses: Linux mendukung mekanisme IPC yang ditemukan di UNIX SVR4.

• Tautan: Setiap proses menyertakan tautan ke proses induknya, tautan ke saudara kandungnya (proses dengan induk yang sama), dan tautan ke semua turunannya.

• Waktu dan pengatur waktu: Termasuk waktu pembuatan proses dan jumlah waktu prosesor yang digunakan selama ini oleh proses. Suatu proses mungkin juga terkait satu atau lebih pengatur waktu interval. Sebuah proses mendefinisikan pengatur waktu interval dengan menggunakan sebuah panggilan sistem; sebagai hasilnya, sinyal dikirim ke proses ketika pengatur waktu berakhir. Sebuah timer mungkin sekali pakai atau berkala.

• Sistem file: Termasuk petunjuk ke file apa pun yang dibuka oleh proses ini, juga menunjuk ke direktori saat ini dan root untuk proses ini.

• Ruang alamat: Mendefinisikan ruang alamat virtual yang ditetapkan untuk proses ini.

• Konteks khusus prosesor: Register dan tumpukan informasi yang menyusun konteks proses ini.

Status eksekusi

Status eksekusi dalam LINUX adalah sebagai berikut:

• Running: Nilai status ini sesuai dengan dua status. Proses sedang dieksekusi atau siap untuk dieksekusi.

• Interruptible: Ini adalah status diblokir, di mana proses sedang menunggu peristiwa, seperti akhir operasi I / O, ketersediaan sumber daya, atau sebuah sinyal dari proses lain.

• Uninterruptible: Ini adalah keadaan diblokir lainnya. Perbedaan antara ini dan status Interruptible adalah bahwa dalam status Uninterruptible, sebuah proses menunggu secara langsung pada kondisi perangkat keras dan oleh karena itu tidak akan menangani sinyal apa pun.

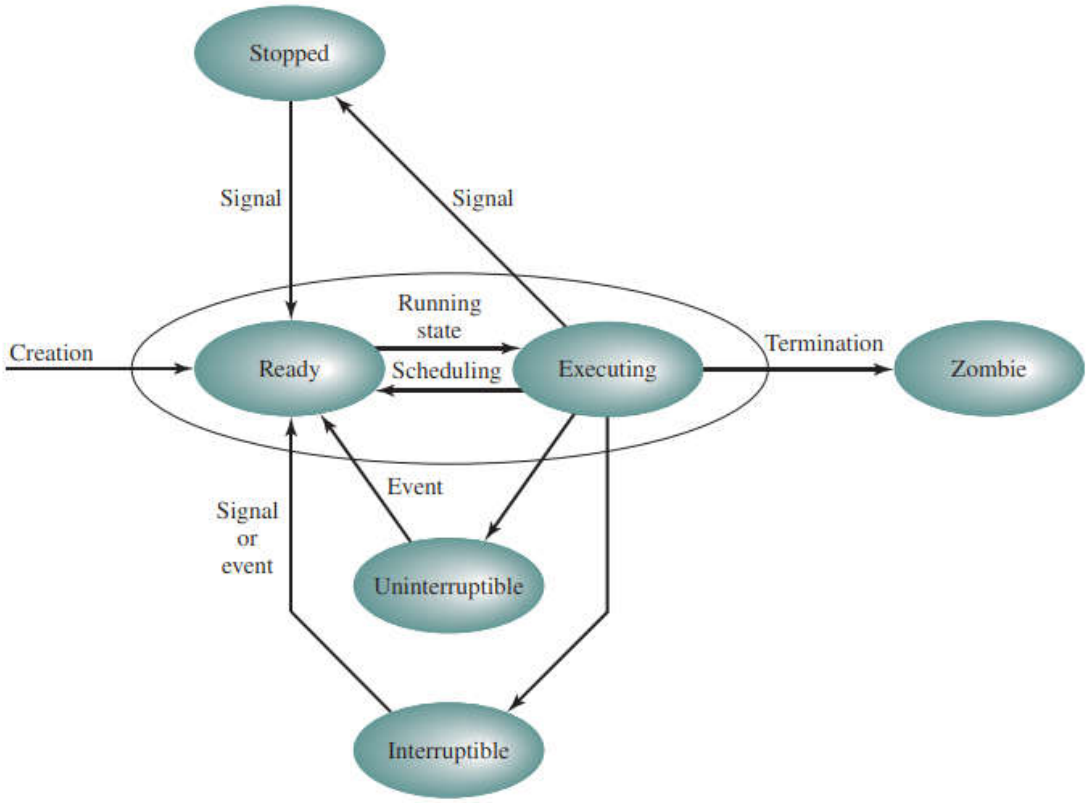
• Berhenti/Stopped: Proses telah dihentikan dan hanya dapat dilanjutkan dengan tindakan positif dari proses lain. Misalnya, proses yang sedang di-debug bisa dimasukkan ke dalam status Berhenti.

• Zombie: Prosesnya sudah dihentikan tapi, entah kenapa, masih memiliki struktur tugasnya di tabel proses.

Thread Linux

Sistem UNIX tradisional mendukung satu rangkaian eksekusi per proses, sementara sistem UNIX modern biasanya menyediakan dukungan untuk beberapa thread tingkat kernel per proses. Seperti sistem UNIX tradisional, kernel Linux versi lama tidak menawarkan dukungan untuk multithreading. Sebaliknya, aplikasi perlu

ditulis dengan sekumpulan fungsi pustaka tingkat pengguna, yang paling populer adalah dikenal sebagai pustaka pthread (thread POSIX), dengan semua thread dipetakan ke dalamnya satu proses tingkat kernel. Versi modern dari UNIX menawarkan thread tingkat kernel. Linux menyediakan solusi unik karena tidak mengenali perbedaan antara thread dan proses. Menggunakan mekanisme yang mirip dengan proses ringan Solaris, thread tingkat pengguna dipetakan ke dalam tingkat kernel proses. Beberapa thread tingkat pengguna yang merupakan proses tingkat pengguna tunggal dipetakan ke dalam proses tingkat kernel Linux yang memiliki ID grup yang sama. Hal ini memungkinkan proses ini untuk berbagi sumber daya seperti file dan memori dan untuk menghindarinya kebutuhan untuk sakelar konteks ketika penjadwal beralih di antara proses di kelompok yang sama.



Gambar 3.4. Status dalam LINUX.

C. Daftar Pustaka

1. Operating System, Internals and design Principles, William Stallings 7th Ed. 2012
2. Modern Operating System 4th Ed. Andrew S Tanembaun 2009