MANAJEMEN PROSES

1. DEFINISI PROSES

Terdapat beberapa definisi mengenai proses, antara lain :

a. Merupakan konsep pokok dalam sistem operasi, sehingga masalah manajemen proses adalah masalah utama dalam perancangan sistem operasi.
b. Proses adalah program yang sedang dieksekusi.
c. Proses adalah unit kerja terkecil yang secara individu memiliki sumber daya dan dijadwalkan oleh sistem operasi.

2. CONTENT PROSES

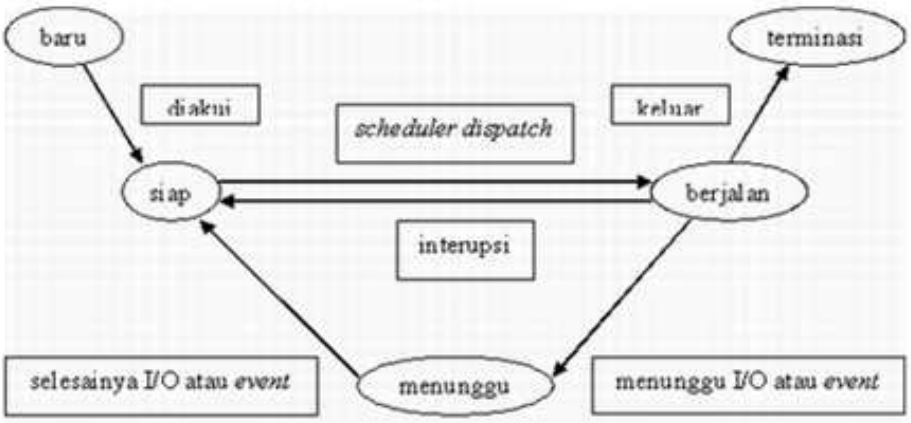
Proses berisi instruksi, data, program counter, register pemroses, stack data, alamat pengiriman dan variabel pendukung lainnya.

3. STATUS PROSES

Sebagaimana proses bekerja, maka proses tersebut merubah state (keadaan statis/ asal). Status dari sebuah proses didefinisikan dalam bagian oleh aktivitas yang ada dari proses tersebut. Tiap proses mungkin adalah satu dari keadaan berikut ini:

- New: Proses sedang dikerjakan/ dibuat.
- Running: Instruksi sedang dikerjakan.
- Waiting: Proses sedang menunggu sejumlah kejadian untuk terjadi (seperti sebuah penyelesaian I/Oatau penerimaan sebuah tanda/ signal).
- Ready: Proses sedang menunggu untuk ditugaskan pada sebuah prosesor.
- Terminated: Proses telah selsesai melaksanakan tugasnya/ mengeksekusi.

Nama-nama tersebut adalah arbitrer/ berdasar opini, istilah tersebut bervariasi disepanjang sistem operasi. Keadaan yang mereka gambarkan ditemukan pada seluruh sistem. Namun, sistem operasi tertentu juga lebih baik menggambarkan keadaan/ status proses. Adalah penting untuk menyadari bahwa hanya satu proses dapat berjalan pada prosesor mana pun pada waktu kapan pun. Namun, banyak proses yang dapat ready atau waiting. Keadaan diagram yang berkaitan dangan keadaan tersebut dijelaskan pada Gambar 1.



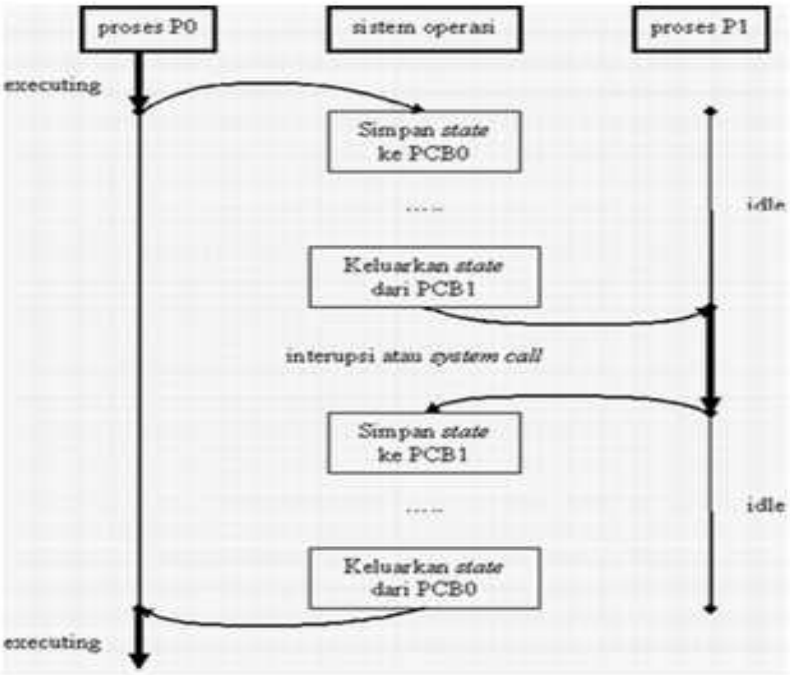
Gambar 1. Keadaan Proses

4. PROCESS CONTROL BLOCK

Tiap proses digambarkan dalam sistem operasi oleh sebuah process control block (PCB) - juga disebut sebuah control block. Sebuah PCB ditunjukkan dalam Gambar 2. PCB berisikan banyak bagian dari informasi yang berhubungan dengan sebuah proses yang spesifik, termasuk ini:

- Keadaan proses: Keadaan mungkin, new, ready, running, waiting, halted, dan juga banyak lagi.
- Program counter: Counter mengindikasikan address dari perintah selanjutnya untuk dijalankan untuk proses ini.
- CPU register: Register bervariasi dalam jumlah dan jenis, tergantung pada rancangan komputer.
- Register tersebut termasuk accumulator, index register, stack pointer, general-puposes register, ditambah code information pada kondisi apa pun. Besertaan dengan program counter, keadaan/ status informasi harus disimpan ketika gangguan terjadi, untuk memungkinkan proses tersebut berjalan/bekerja dengan benar setelahnya (lihat Gambar 3).
- Informasi managemen memori: Informasi ini dapat termasuk suatu informasi sebagai nilai dari dasar dan batas register, tabel page/ halaman, atau tabel segmen tergantung pada sistem memori yang digunakan oleh sistem operasi.
- Informasi pencatatan: Informasi ini termasuk jumlah dari CPU dan waktu riil yang digunakan, batas waktu, jumlah akun, jumlah job atau proses, dan banyak lagi.
- Informasi status I/O: Informasi termasuk daftar dari perangkat I/O yang digunakan pada proses ini, suatu daftar open berkas dan banyak lagi.
- PCB hanya berfungsi sebagai tempat menyimpan/ gudang untuk informasi apapun yang dapat bervariasi dari prose ke proses.

Gambar 2. PCB



Gambar 3. CPU Registers

5. PENJADWALAN PROSES

Tujuan dari multiprogramming adalah untuk memiliki sejumlah proses yang berjalan pada sepanjang waktu, untuk memaksimalkan penggunaan CPU.

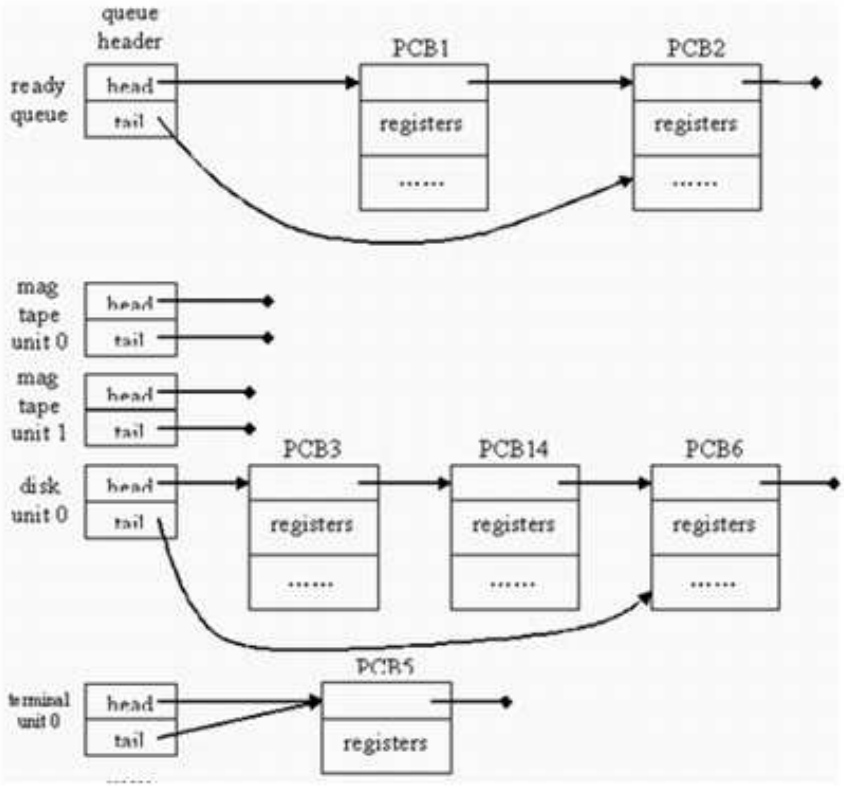
Tujuan dari pembagian waktu adalah untuk mengganti CPU diantara proses-proses yang begitu sering sehingga pengguna dapat berinteraksi dengan setiap program sambil CPU bekerja. Untuk sistem uniprosesor, tidak akan ada lebih dari satu proses berjalan. Jika ada proses yang lebih dari itu, yang lainnya akan harus menunggu sampai CPU bebas dan dapat dijadualkan kembali.

Terdapat 3 konsep dasar Penjadwalan proses yaitu :

5.1 Penjadualan Antrian (Scheduling Queue)

Ketika proses memasuki sistem, mereka diletakkan dalam antrian job. Antrian ini terdiri dari seluruh proses dalam sistem. Proses yang hidup pada memori utama dan siap dan menunggu/ wait untuk mengeksekusi disimpan pada sebuah daftar bernama ready queue. Antrian ini biasanya disimpan sebagai daftar penghubung. Sebuah header ready queue berisikan penunjuk kepada PCB-PCB awal dan akhir. Setiap PCB memiliki pointer field yang menunjukkan proses selanjutnya dalam ready queue.

Juga ada antrian lain dalam sistem. Ketika sebuah proses mengalokasikan CPU, proses tersebut berjalan/bekerja sebentar lalu berhenti, di interupsi, atau menunggu suatu kejadian tertentu, seperti penyelesaian suatu permintaan I/O. Pada kasus ini sebuah permintaan I/O, permintaan seperti itu mungkin untuk sebuah tape drive yang telah diperuntukkan, atau alat yang berbagi, seperti disket. Karena ada banyak proses dalam sistem, disket bisa jadi sibuk dengan permintaan I/O untuk proses lainnya. Maka proses tersebut mungkin harus menunggu untuk disket tersebut. Daftar dari proses yang menunggu untuk peralatan I/O tertentu disebut sebuah device queue. Tiap peralatan memiliki device queuenya sendiri (Lihat Gambar 4).

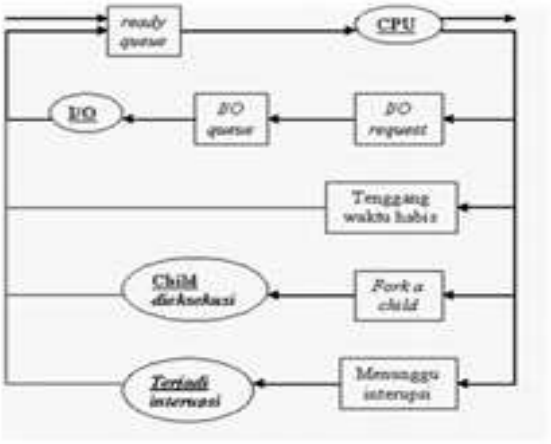


Gambar 4. Device Queue.

Reprensentasi umum untuk suatu diskusi mengenai penjadualan proses adalah diagram antrian, seperti pada Gambar 5. Setiap kotak segi empat menunjukkan sebuah antrian. Dua tipe antrian menunjukan antrian yang siap dan suatu perangkat device queues. Lingkaran menunjukkan sumber-sumber yang melayani sistem. Sebuah proses baru pertama-tama ditaruh dalam ready queue. Lalu menunggu dalam

ready queue sampai proses tersebut dipilih untuk dikerjakan/lakukan atau di dispatched. Begitu proses tersebut mengalokasikan CPU dan menjalankan/ mengeksekusi, satu dari beberapa kejadian dapat terjadi.

- Proses tersebut dapat mengeluarkan sebuah permintaan I/O, lalu di tempatkan dalam sebuah antrian I/O.
- Proses tersebut dapat membuat subproses yang baru dan menunggu terminasinya sendiri.
- Proses tersebut dapat digantikan secara paksa dari CPU, sebagai hasil dari suatu interupsi, dan diletakkan kembali dalam ready queue.



Gambar 5. Diagram Antrian

Dalam dua kasus pertama, proses akhirnya berganti dari waiting state menjadi ready state, lalu diletakkan kembali dalam ready queue. Sebuah proses meneruskan siklus ini sampai berakhir, disaat dimana proses tersebut diganti dari seluruh queue dan memiliki PCB nya dan sumber-sumber/ resources dialokasikan kembali.

5.2 Penjadual / Scheduler

Sebuah proses berpindah antara berbagai penjadualan antrian selama umur hidupnya. Sistem operasi harus memilih, untuk keperluan penjadualan, memproses antrian-antrian ini dalam cara tertentu. Pemilihan proses dilaksanakan oleh penjadual yang tepat/ cocok. Dalam sistem batch, sering ada lebih banyak proses yang diserahkan daripada yang dapat dilaksanakan segera. Proses ini dipitakan/ disimpan pada suatu alat penyimpan masal (biasanya disket), dimana proses tersebut disimpan untuk eksekusi dilain waktu. Penjadualan long term, atau penjadual job, memilih proses dari pool ini dan mengisinya kedalam memori eksekusi.

Sebuah proses dapat mengeksekusi untuk hanya beberapa milidetik sebelum menunggu permintaan I/O. Seringkali, penjadualan shorterm mengeksekusi paling sedikit sekali setiap 100 milidetik. Karena durasi waktu yang pendek antara eksekusi, penjadualan shorterm haruslah cepat. Jika

memerlukan 10 mili detik untuk menentukan suatu proses eksekusi selama 100 mili detik, maka 10/(100 + 10) = 9 persen CPU sedang digunakan (terbuang) hanya untuk pekerjaan penjadualan.

Penjadualan longterm pada sisi lain, mengeksekusi jauh lebih sedikit. Mungkin ada beberapa menit antara pembuatan proses baru dalam sistem. Penjadualan longterm mengkontrol derajat multiprogramming (jumlah proses dalam memori). Jika derajat multiprogramming stabil, lalu tingkat rata-rata dari penciptaan proses harus sama dengan tingkat kepergian rata rata dari proses yang meninggalkan sistem. Maka penjadualan longterm mungkin diperlukan untuk dipanggil hanya ketika suatu proses meninggalkan sistem. Karena interval yang lebih panjang antara eksekusi, penjadualan longterm dapat memakai waktu yang lebih lama untuk menentukan proses mana yang harus dipilih untuk dieksekusi.

Adalah penting bagi penjadualan longterm membuat seleksi yang hati-hati. Secara umum, kebanyakan proses dapat dijelaskan sebagai I/O bound atau CPU bound. Sebuah proses I/O bound adalah salah satu yang membuang waktunya untuk mengerjakan I/O dari pada melakukan perhitungan. Suatu proses CPU-bound, pada sisi lain, adalah salah satu yang jarang menghasilkan permintaan I/O, menggunakan lebih banyak waktunya melakukan banyak komputasi daripada yang digunakan oleh proses I/O bound. Penting untuk penjadualan longterm memilih campuran proses yang baik antara proses I/O bound dan CPU bound. Jika seluruh proses adalah I/O bound, ready queue akan hampir selalu kosong, dan penjadualan short term akan memiliki sedikit tugas. Jika seluruh proses adalah CPU bound, I/O waiting queue akan hampir selalu kosong, peralatan akan tidak terpakai, dan sistem akan menjadi tidak imbang. Sistem dengan kinerja yang terbaik akan memiliki kombinasi proses CPU bound dan I/O bound.

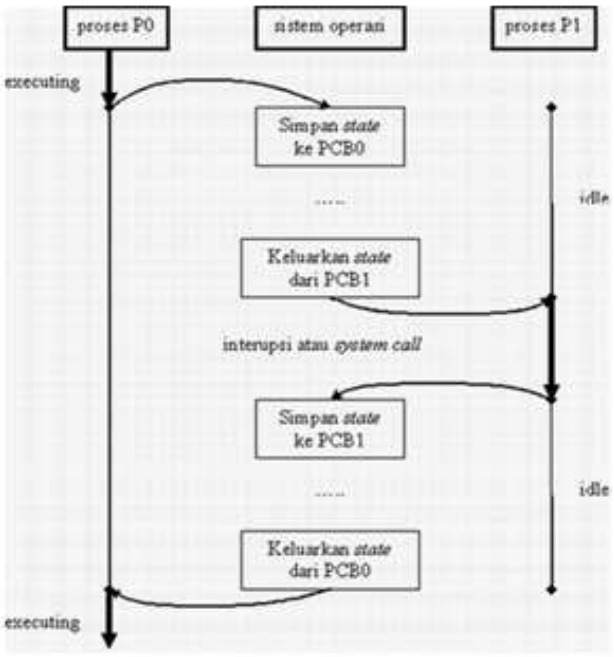
Gambar 6. Penjadual Medium-term.

Pada sebagian sistem, penjadual long term dapat tidak turut serta atau minimal. Sebagai contoh, sistem time-sharing seperti UNIX sering kali tidak memiliki penjadual long term. Stabilitas sistem-sistem ini bergantung pada keterbatasan fisik (seperti jumlah terminal yang ada) atau pada penyesuaian sendiri secara alamiah oleh manusia sebagai pengguna. Jika kinerja menurun pada tingkat yang tidak dapat diterima, sebagian pengguna akan berhenti.

Sebagian sistem operasi, seperti sistem time sharing, dapat memperkenalkan sebuah tambahan, penjadualan tingkat menengah. Penjadual medium-term ini digambarkan pada Gambar 6. Ide utama/kunci dibelakang sebuah penjadual medium term adalah kadang kala akan menguntungkan untuk memindahkan proses dari memori (dan dari pengisian aktif dari CPU), dan maka untuk mengurangi derajat dari multiprogramming. Dikemudian waktu, proses dapat diperkenalkan kedalam memori dan eksekusinya dapat dilanjutkan dimana proses itu di tinggalkan/ diangkat. Skema ini disebut swapping. Proses di swapped out, dan lalu di swapped in, oleh penjadual jangka menengah. Swapping mungkin perlu untuk meningkatkan pencampuran proses, atau karena suatu perubahan dalam persyaratan memori untuk dibebaskan. Swapping dibahas pada bab selanjutnya.

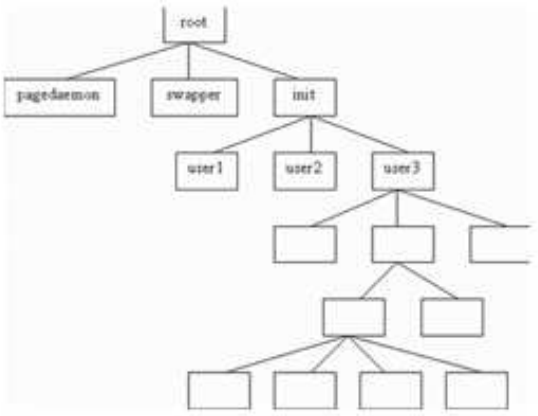
5.3 Alih Konteks / Switch Context

Mengganti CPU ke proses lain memerlukan penyimpanan suatu keadaan proses lama (state of old process) dan kemudian beralih ke proses yang baru. Tugas tersebut diketahui sebagai alih konteks (context switch). Alih konteks sebuah proses digambarkan dalam PCB suatu proses; termasuk nilai dari CPU register, status proses (lihat Gambar 7). dan informasi managemen memori. Ketika alih konteks terjadi, kernel menyimpan konteks dari proses lama kedalam PCB nya dan mengisi konteks yang telah disimpan dari process baru yang telah terjadual untuk berjalan. Pergantian waktu konteks adalah murni overhead, karena sistem melakukan pekerjaan yang tidak perlu. Kecepatannya bervariasi dari mesin ke mesin, bergantung pada kecepatan memori, jumlah register yang harus di copy, dan keberadaan instruksi khusus (seperti instruksi tunggal untuk mengisi atau menyimpan seluruh register). Tingkat kecepatan umumnya berkisar antara 1 sampai 1000 mikro detik



Gambar 7. Alih Konteks.

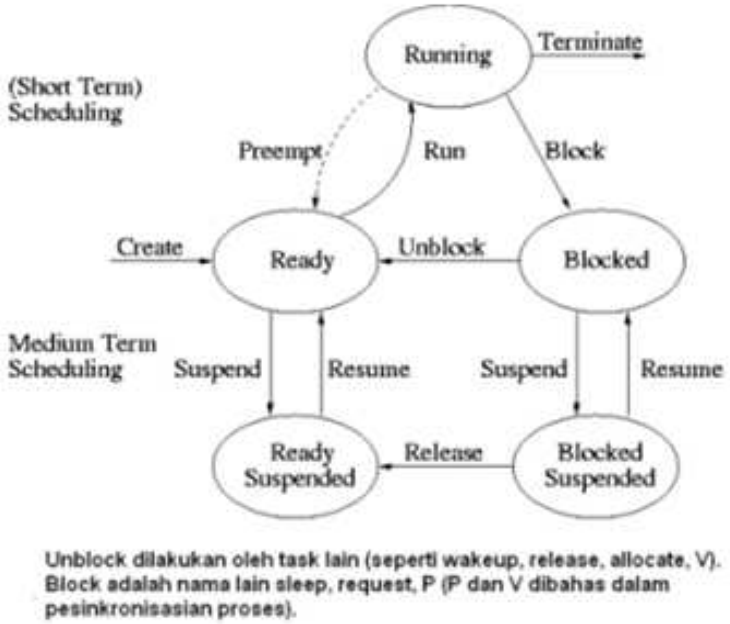
Waktu alih konteks sangat begantung pada dukungan perangkat keras. Sebagai contoh, prosesor seperti UltraSPARC menyediakan dua rangkap register. Sebuah alih konteks hanya memasukkan perubahan pointer ke perangkat register yang ada. Tentu saja, jika ada lebih proses-proses aktif yang ada dari pada yang ada di perangkat register, sistem menggunakan bantuan untuk meng-copy data register pada dan dari memori, sebagaimana sebelumnya. Semakin sistem operasi kompleks, makin banyak pekerjaan yang harus dilakukan selama alih konteks. Sebagaimana dilihat pada Bab 4, teknik managemen memori tingkat lanjut dapat mensyaratkan data tambahan untuk diganti dengan tiap konteks. Sebagai contoh, ruang alamat dari proses yang ada harus dijaga sebagai ruang pada pekerjaan berikutnya untuk digunakan. Bagaimana ruang alamat di jaga, berapa banyak pekerjaan dibutuhkan untuk menjaganya, tergantung pada metoda managemen memori dari sistem operasi. Sebagaimana akan kita lihat pada Bab 4, alih konteks telah menjadi suatu keharusan, bahwa programmer menggunakan struktur (threads) untuk menghindarinya kapan pun memungkinkan.



Gambar 8. Pohon Proses.

6. OPERASI PADA PROSES

Proses dalam sistem dapat dieksekusi secara bersama-sama, proses tersebut harus dibuat dan dihapus secara dinamis. Maka, sistem operasi harus menyediakan suatu mekanisme untuk pembuatan proses dan terminasi proses.



Gambar 9. Operasi pada Proses.

Sistem operasi dalam mengelola proses dapat melakukan operasi-operasi terhadap proses. Operasi tersebut adalah :

a. Penciptaan proses

b. Penghancuran/terminasi proses

c. Penundaan proses

d. Pelanjutan kembali proses

e. Pengubahan prioritas proses

f. Memblok proses

g. Membangunkan proses

h. Menjadwalkan proses

i. Memungkinkan proses berkomunikasi dengan proses lain

6.1. Pembuatan Proses

Melibatkan banyak aktivitas, yaitu :

a. Memberi identitas proses

b. Menyisipkan proses pada senarai atau tabel proses

c. Menentukan prioritas awal proses

d. Menciptakan PCB

e. Mengalokasikan sumber daya awal bagi proses

Ketika proses baru ditambahkan, sistem operasi membangun struktur data untuk mengelola dan mengalokasikan ruang alamat proses.

Kejadian yang dapat menyebabkan penciptaan proses :

a. Pada lingkungan batch, sebagai tanggapan atas pemberian satu kerja (job). Sistem operasi dengan kendali batch job, setelah menciptakan proses baru, kemudian melanjutkan membaca job berikutnya.

b. Pada lingkungan interaktif, ketika pemakai baru berusaha logon.

c. Sebagai tanggapan suatu aplikasi, seperti permintaan pencetakan file, sistem operasi dapat menciptakan proses yang akan mengelola pencetakan itu. Sistem operasi menciptakan proses untuk memenuhi satu fungsi pada program pemakai, tanpa mengharuskan pemakai menunggu.

d. Proses penciptaan proses lain (proses anak). Untuk mencapai modularitas atau mengeksploitasi kongkurensi, program pemakai memerintahkan pembuatan sejumlah proses.

Tahap-tahap penciptaan proses

Penciptaan proses dapat disebabkan beragam sebab. Penciptaan proses meliputi beberapa tahap :

1. Beri satu identifier unik ke proses baru. Isian baru ditambahkan ke tabel proses utama yang berisi satu isian perproses.
2. Alokasikan ruang untuk proses.
3. PCB harus diinisialisasi.
4. Kaitan-kaitan antar tabel dan senarai yang cocok dibuat.
5. Bila diperlukan struktur data lain maka segera dibuat struktur data itu.

6.2. Penghancuran / Terminasi Proses

Penghancuran proses melibatkan pembebasan proses dari sistem, yaitu :

a. Sumber daya-sumber daya yang dipakai dikembalikan.

b. Proses dihancurkan dari senarai atau tabel sistem.

c. PCB dihapus (ruang memori PCB dikembalikan ke pool memori bebas).

Penghancuran lebih rumit bila proses telah menciptakan proses-proses lain. Terdapat dua pendekatan, yaitu :

a. Pada beberapa sistem, proses-proses turunan dihancurkan saat proses induk dihancurkan secara otomatis.

b. Beberapa sistem lain menganggap proses anak independen terhadap proses induk, sehingga proses anak tidak secara otomatis dihancurkan saat proses induk dihancurkan.

Alasan penghancuran proses :

no	penyebab/terminasi	KETERANGAN
1	Selesainya proses secara manual	Proses mengeksekusi panggilan layanan sistem operasi untuk menandakan bawah proses telah berjalan secara lengkap.
2	Batas waktu telah terlewati	Proses telah berjalan melebihi batas waktu total yang dispesifikasikan. Terdapat banyak kemungkinan untuk tipe waktu yang diukur,termasuk waktu total yang dijalani (wait clock time),jumlah waktu yang dipakai untuk eksekusi, dan jumlah waktu sejak pemakai terakhir kali memberi masukan (pada proses interaktif).
3	Memori tidak tersedia	Proses memerlukan memori lebih banyak daripada yang dapat disediakan sistem.
4	Pelanggaran terhadap batas memori	Proses mencoba mengakses lokasi memori yang tidak diijinkan diakses.
5	Terjadi kesalahan karena pelanggaran proteksi	Proses berusaha menggunakan sumber data atau file yang tidak diijinkan dipakainya, atau proses mencoba menggunakannya tidak untuk peruntukannya, seperti menulis file read only.
6	Terjadi kesalahan aritmatika	Proses memcoba perhitungan terlarang, seperti pembagian dengan nol,atau mencoba menyimpan angka yang lebih besar daripada yang dapat diakomodasi oleh perangkat keras
7	Waktu telah kedaluwarsa	Proses telah menunggu lebih lama dari pada maksimum yang ditentukan untuk terjadinya suatu kejadian spesifik.
8	Terjadi kegagalan Masukan/keluaran	Kesalahan muncul pada masukan/keluaran, seperti ketidakmampuan menemukan file, kegagalan membaca atau menulis setelah sejumlah maksimum percobaan yang ditentukan (area rusak didapatkan pada tape atau operasi tak valid seperti membaca dari line printer).
9	Instruksi yang tidak benar	Proses berusaha mengeksekusi intruksi yang tak ada (akibat percabangan ke daerah data dan dan berusaha mengeksekusi data itu)
10	Terjadi usaha memakai instruksi yang tidak Diijinkan	Proses berusaha menggunakan instruksi yang disimpan untuk sistem operasi.
11	Kesalahan Penggunaan data	Bagian data adalah tipe yang salah atau tidak diinisialisasi.
12	Diintervensi oleh sistem operasi atau operator	Untuk suatu alasan, operator atau sistem operasi mengakhiri proses (terjadi deadlock)
13	Berakhirnya proses induk	Ketika parent berakhir, sistem operasi mungkin dirancang secara otomatis mengakhiri semua anak proses dari parent itu.
14	Atas permintaan dari proses induk	Parent process biasanya mempunyai otoritas mengakhiri suatu anak proses.

7. HUBUNGAN ANTAR PROSES

Sebelumnya kita telah ketahui seluk beluk dari suatu proses mulai dari pengertiannya, cara kerjanya, sampai operasi-operasinya seperti proses pembentukannya dan proses pemberhentiannya setelah selesai melakukan eksekusi. Kali ini kita akan mengulas bagaimana hubungan antar proses dapat berlangsung, misal bagaimana beberapa proses dapat saling berkomunikasi dan bekerja-sama.

7.1. Proses yang Kooperatif

Proses yang bersifat simultan (concurrent) dijalankan pada sistem operasi dapat dibedakaan menjadi yaitu proses independent dan proses kooperatif. Suatu proses dikatakan independen apabila proses tersebut tidak dapat terpengaruh atau dipengaruhi oleh proses lain yang sedang dijalankan pada sistem.

Berarti, semua proses yang tidak membagi data apa pun (baik sementara/ tetap) dengan proses lain adalah independent. Sedangkan proses kooperatif adalah proses yang dapat dipengaruhi atau pun terpengaruhi oleh proses lain yang sedang dijalankan dalam sistem. Dengan kata lain, proses dikatakan kooperatif bila proses dapat membagi datanya dengan proses lain. Ada empat alasan untuk penyediaan sebuah lingkungan yang memperbolehkan terjadinya proses kooperatif:

1. Pembagian informasi: apabila beberapa pengguna dapat tertarik pada bagian informasi yang sama (sebagai contoh, sebuah berkas bersama), kita harus menyediakan sebuah lingkungan yang mengizinkan akses secara terus menerus ke tipe dari sumber-sumber tersebut.
2. Kecepatan penghitungan/ komputasi: jika kita menginginkan sebuah tugas khusus untuk menjalankan lebih cepat, kita harus membagi hal tersebut ke dalam subtask, setiap bagian dari subtask akan dijalankan secara parallel dengan yang lainnya. Peningkatan kecepatan dapat dilakukan hanya jika komputer tersebut memiliki elemen-elemen pemrosesan ganda (seperti CPU atau jalur I/O).
3. Modularitas: kita mungkin ingin untuk membangun sebuah sistem pada sebuah model modular-modular, membagi fungsi sistem menjadi beberapa proses atau threads.
4. Kenyamanan: bahkan seorang pengguna individu mungkin memiliki banyak tugas untuk dikerjakan secara bersamaan pada satu waktu. Sebagai contoh, seorang pengguna dapat mengedit, memcetak, dan meng-compile secara paralel.

7.2. Komunikasi Proses Dalam Sistem

Cara lain untuk meningkatkan efek yang sama adalah untuk sistem operasi yaitu untuk menyediakan alat-alat proses kooperatif untuk berkomunikasi dengan yang lain lewat sebuah komunikasi dalam proses (IPC = Inter-Process Communication). IPC menyediakan sebuah mekanisme untuk mengizinkan proses-proses untuk berkomunikasi dan menyelaraskan aksi-aksi mereka tanpa berbagi ruang alamat yang sama. IPC adalah khusus digunakan dalam sebuah lingkungan yang terdistribusi dimana proses komunikasi tersebut mungkin saja tetap ada dalam komputer-komputer yang berbeda yang tersambung dalam sebuah jaringan. IPC adalah penyedia layanan terbaik dengan menggnakan sebuah sistem penyampaian pesan, dan sistem-sistem pesan dapat diberikan dalam banyak cara.

7.2.1. Sistem Penyampaian Pesan

Fungsi dari sebuah sistem pesan adalah untuk memperbolehkan komunikasi satu dengan yang lain tanpa perlu menggunakan pembagian data. Sebuah fasilitas IPC menyediakan paling sedikit dua operasi yaitu kirim (pesan) dan terima (pesan). Pesan dikirim dengan sebuah proses yang dapat dilakukan pada ukuran pasti atau variabel. Jika hanya pesan dengan ukuran pasti dapat dikirimkan, level sistem implementasi adalah sistem yang sederhana. Pesan berukuran variabel menyediakan sistem implementasi level yang lebih kompleks.

Berikut ini ada beberapa metode untuk mengimplementasikan sebuah jaringan dan operasi pengiriman/penerimaan secara logika:

- Komunikasi langsung atau tidak langsung.
- Komunikasi secara simetris/ asimetris.
- Buffer otomatis atau eksplisit.
- engiriman berdasarkan salinan atau referensi.
- Pesan berukuran pasti dan variabel.

7.2.2. Komunikasi Langsung

Proses-proses yang ingin dikomunikasikan harus memiliki sebuah cara untuk memilih satu dengan yang lain. Mereka dapat menggunakan komunikasi langsung/ tidak langsung.

Setiap proses yang ingin berkomunikasi harus memiliki nama yang bersifat eksplisit baik penerimaan atau pengirim dari komunikasi tersebut. Dalam konteks ini, pengiriman dan penerimaan pesan secara primitive dapat dijabarkan sebagai:

- Send (P, message) - mengirim sebuah pesan ke proses P.
- Receive (Q, message) - menerima sebuah pesan dari proses Q.

Sebuah jaringan komunikasi pada bahasan ini memiliki beberapa sifat, yaitu:

- Sebuah jaringan yang didirikan secara otomatis diantara setiap pasang dari proses yang ingin dikomunikasikan. Proses tersebut harus mengetahui identitas dari semua yang ingin dikomunikasikan.
- Sebuah jaringan adalah terdiri dari penggabungan dua proses.
- Diantara setiap pesan dari proses terdapat tepat sebuah jaringan.

Pembahasan ini memperlihatkan sebuah cara simetris dalam pemberian alamat. Oleh karena itu, baik keduanya yaitu pengirim dan penerima proses harus memberi nama bagi yang lain untuk berkomunikasi, hanya pengirim yang memberikan nama bagi penerima sedangkan penerima tidak menyediakan nama bagi pengirim. Dalam konteks ini, pengirim dan penerima secara sederhana dapat dijabarkan sebagai:

- Send (P, message) - mengirim sebuah pesan kepada proses P.
- Receive (id, message) - menerima sebuah pesan dari semua proses.

Variabel id diatur sebagai nama dari proses dengan komunikasi.

7.2.3. Komunikasi Tidak Langsung

Dengan komunikasi tidak langsung, pesan akan dikirimkan pada dan diterima dari/ melalui mailbox(kotak surat) atau terminal-terminal, sebuah mailbox dapat dilihat secara abstrak sebagai sebuah objek didalam setiap pesan yang dapat ditempatkan dari proses dan dari setiap pesan yang bias dipindahkan. Setiap kotak surat memiliki sebuah identifikasi (identitas) yang unik, sebuah proses dapat berkomunikasi dengan beberapa proses lain melalui sebuah nomor dari mailbox yang berbeda. Dua proses dapat saling berkomunikasi apabila kedua proses tersebut sharing mailbox. Pengirim dan penerima dapat dijabarkan sebagai:

- Send (A, message) - mengirim pesan ke mailbox A.
- Receive (A, message) - menerima pesan dari mailbox A.

Dalam masalah ini, link komunikasi mempunyai sifat sebagai berikut:

- Sebuah link dibangun diantara sepasang proses dimana kedua proses tersebut membagi mailbox.
- Sebuah link mungkin dapat berasosiasi dengan lebih dari dua proses.
- Diantara setiap pasang proses komunikasi, mungkin terdapat link yang berbeda-beda, dimana setiap link berhubungan pada satu mailbox.

Misalkan terdapat proses P1, P2 dan P3 yang semuanya share mailbox. Proses P1 mengirim pesan ke A, ketika P2 dan P3 masing-masing mengeksekusi sebuah kiriman dari A. Proses mana yang akan menerima pesan yang dikirim P1? Jawabannya tergantung dari jalur yang kita pilih:

- Mengizinkan sebuah link berasosiasi dengan paling banyak 2 proses.
- Mengizinkan paling banyak satu proses pada suatu waktu untuk mengeksekusi hasil kiriman (receive operation).
- Mengizinkan sistem untuk memilih secara mutlak proses mana yang akan menerima pesan (apakah itu P2 atau P3 tetapi tidak keduanya, tidak akan menerima pesan). Sistem mungkin mengidentifikasi penerima kepada pengirim.

Mailbox mungkin dapat dimiliki oleh sebuah proses atau sistem operasi. Jika mailbox dimiliki oleh proses, maka kita mendefinisikan antara pemilik (yang hanya dapat menerima pesan melalui mailbox) dan pengguna dari mailbox (yang hanya dapat mengirim pesan ke mailbox). Selama setiap mailbox mempunyai kepemilikan yang unik, maka tidak akan ada kebingungan tentang siapa yang harus menerima

pesan dari mailbox. Ketika proses yang memiliki mailbox tersebut diterminasi, mailbox akan hilang. Semua proses yang mengirim pesan ke mailbox ini diberi pesan bahwa mailbox tersebut tidak lagi ada.

Dengan kata lain, mempunyai mailbox sendiri yang independent, dan tidak melibatkan proses yang lain. Maka sistem operasi harus memiliki mekanisme yang mengizinkan proses untuk melakukan hal-hal dibawah ini:

- Membuat mailbox baru.
- Mengirim dan menerima pesan melalui mailbox.
- Menghapus mailbox.

Proses yang membuat mailbox pertama kali secara default akan memiliki mailbox tersebut. Untuk pertama kali, pemilik adalah satu-satunya proses yang dapat menerima pesan melalui mailbox ini. Bagaimana pun, kepemilikan dan hak menerima pesan mungkin dapat dialihkan ke proses lain melalui sistem pemanggilan.

7.2.4. Sinkronisasi

Komunikasi antara proses membutuhkan place by calls untuk mengirim dan menerima data primitive. Terdapat rancangan yang berbeda-beda dalam implementasi setiap primitive. Pengiriman pesan mungkin dapat diblok (blocking) atau tidak dapat dibloking (nonblocking) - juga dikenal dengan nama sinkron atau asinkron.

- Pengiriman yang diblok: Proses pengiriman di blok sampai pesan diterima oleh proses penerima (receiving process) atau oleh mailbox.
- Pengiriman yang tidak diblok: Proses pengiriman pesan dan mengkalkulasi operasi.
- Penerimaan yang diblok: Penerima mem blok samapai pesan tersedia.
- Penerimaan yang tidak diblok: Penerima mengembalikan pesan valid atau null.

7.2.5. Buffering

Baik komunikasi itu langsung atau tak langsung, penukaran pesan oleh proses memerlukan antrian sementara. Pada dasarnya, terdapat tiga jalan dimana antrian tersebut diimplementasikan:

- Kapasitas nol (zero capacity): antrian mempunyai panjang maksimum 0, maka link tidak dapat mempunyai penungguan pesan (message waiting). Dalam kasus ini, pengirim harus memblok sampai penerima menerima pesan.
- Kapasitas terbatas (Bounded capacity): antrian mempunyai panjang yang telah ditentukan, paling banyak n pesan dapat dimasukkan. Jika antrian tidak penuh ketika pesan dikirimkan, pesan yang baru akan

menimpa, dan pengirim pengirim dapat melanjutkan eksekusi tanpa menunggu. Link mempunyai kapasitas terbatas.

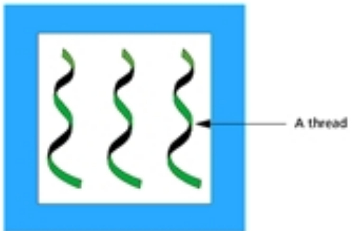
- Jika link penuh, pengirim harus memblok sampai terdapat ruang pada antrian.
- Kapasitas tak terbatas(Unbounded capacity): antrian mempunyai panjang yang tak terhingga, maka, semua pesan dapat menunggu disini. Pengirim tidak akan pernah di blok.

8. THREAD

Model proses yang didiskusikan sejauh ini telah menunjukkan bahwa suatu proses adalah sebuah program yang menjalankan eksekusi thread tunggal. Sebagai contoh, jika sebuah proses menjalankan sebuah program Word Processor, ada sebuah thread tunggal dari instruksi-instruksi yang sedang dilaksanakan.

Kontrol thread tunggal ini hanya memungkinkan proses untuk menjalankan satu tugas pada satu waktu. Banyak sistem operasi modern telah memiliki konsep yang dikembangkan agar memungkinkan sebuah proses untuk memiliki eksekusi multithreads, agar dapat dapat secara terus menerus mengetik dalam karakter dan menjalankan pengecek ejaan didalam proses yang sama. Maka sistem operasi tersebut memungkinkan proses untuk menjalankan lebih dari satu tugas pada satu waktu.

Thread, atau kadang-kadang disebut proses ringan (lightweight), adalah unit dasar dari utilisasi CPU. Di dalamnya terdapat ID thread, program counter, register, dan stack. Dan saling berbagi dengan thread lain dalam proses yang sama.



Gambar 10. Thread

8.1. Konsep Dasar

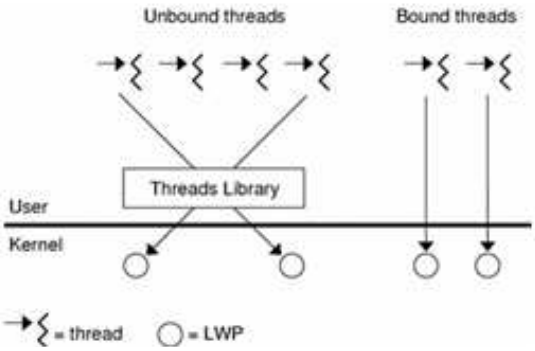
Secara informal, proses adalah program yang sedang dieksekusi. Ada dua jenis proses, proses berat (heavyweight) atau biasa dikenal dengan proses tradisional, dan proses ringan atau kadang disebut thread.

Thread saling berbagi bagian program, bagian data dan sumber daya sistem operasi dengan thread lain yang mengacu pada proses yang sama. Thread terdiri atas ID thread, program counter, himpunan register, dan stack. Dengan banyak kontrol thread proses dapat melakukan lebih dari satu pekerjaan pada waktu yang sama.

8.2. Keuntungan

1. Tanggap: Multithreading mengizinkan program untuk berjalan terus walau pun pada bagian program tersebut di block atau sedang dalam keadaan menjalankan operasi yang lama/ panjang. Sebagai contoh, multithread web browser dapat mengizinkan pengguna berinteraksi dengan suatu thread ketika suatu gambar sedang diload oleh thread yang lain.
2. 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 memori yang sama.
3. Ekonomis: Mengalokasikan memori dan sumber daya untuk membuat proses adalah sangat mahal. Alternatifnya, karena thread membagi sumber daya dari proses, ini lebih ekonomis untuk membuat threads.
4. Pemberdayaan arsitektur multiprosesor: Keuntungann dari multithreading dapat ditingkatkan dengan arsitektur multiprosesor, dimana setiap thread dapat jalan secara parallel pada prosesor yang berbeda. Pada arsitektur prosesor tunggal, CPU biasanya berpindah-pindah antara setiap thread dengan cepat, sehingga terdapat ilusi paralelisme, tetapi pada kenyataannya hanya satu thread yang berjalan di setiap waktu.

8.3. User Threads



Gambar 16. User dan Kernel Thread.

User thread didukung oleh kernel dan diimplementasikan oleh thread library ditingkat pengguna. Library mendukung untuk pembentukan thread, penjadualan, dan managemen yang tidak didukung oleh kernel.

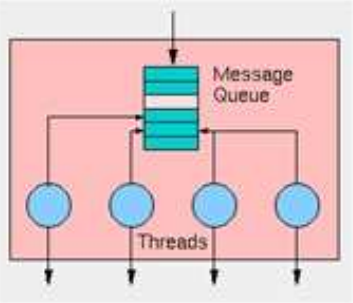
8.4. Kernel Threads

Kernel thread didukung secara langsung oleh sistem operasi: pembentukan thread, penjadualan, dan managemen dilakukan oleh kernel dalam ruang kernel. Karena managemen thread telah dilakukan oleh sistem operasi, kernel thread biasanya lebih lambat untuk membuat dan mengelola daripada pengguna thread. Bagaimana pun, selama kernel mengelola thread, jika suatu thread di block tehadap sistem pemanggilan, kernel dapat menjadualkan thread yang lain dalam aplikasi untuk dieksekusi. Juga, di dalam lingkungan multiprosesor, kernel dapat menjadualkan thread dalam prosesor yang berbeda. Windows NT, Solaris, dan Digital UNIX adalah sistem operasi yang mendukung kernel thread.

9. Model Multithreading

Dalam sub bab sebelumnya telah dibahas pengertian dari thread, keuntungannya, tingkatan atau levelnya seperti pengguna dan kernel.

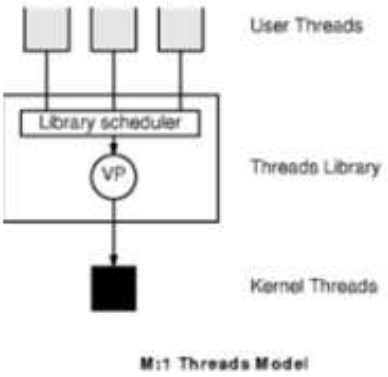
Sistem-sistem yang ada sekarang sudah banyak yang bisa mendukung untuk kedua pengguna dan kernel thread, sehingga model-model multithreading-nya pun menjadi beragam. Implementasi multithreading yang umum akan kita bahas ada tiga, yaitu model many-to-one, one-to-one, dan many-to-many.



Gambar 17. Model Multithreading.

9.1. Model Many to One

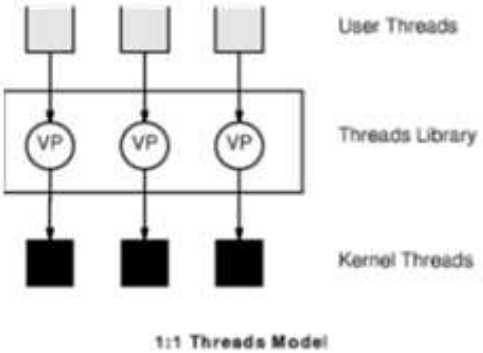
Model many-to-one ini memetakan beberapa tingkatan pengguna thread hanya ke satu buah kernel thread. Managemen proses thread dilakukan oleh (di ruang) pengguna, sehingga menjadi efisien, tetapi apabila sebuah thread melakukan sebuah pemblokingan terhadap sistem pemanggilan, maka seluruh proses akan berhenti (blocked). Kelemahan dari model ini adalah multihreads tidak dapat berjalan atau bekerja secara paralel di dalam multiprosesor dikarenakan hanya satu thread saja yang bisa mengakses kernel dalam suatu waktu.



Gambar 18. Model Many to One.

9.2. Model One to One

Model one-to-one memetakan setiap thread pengguna ke dalam satu kernel thread. Hal ini membuat model one-to-one lebih sinkron daripada model many-to-one dengan mengizinkan thread lain untuk berjalan ketika suatu thread membuat pemblokingan terhadap sistem pemanggilan; hal ini juga mengizinkan multiple thread untuk berjalan secara parallel dalam multiprosesor. Kelemahan model ini adalah dalam pembuatan thread pengguna dibutuhkan pembuatan korespondensi thread pengguna. Karena dalam proses pembuatan kernel thread dapat mempengaruhi kinerja dari aplikasi maka kebanyakan dari implementasi model ini membatasi jumlah thread yang didukung oleh sistem. Model one-to-one diimplementasikan oleh Windows NT dan OS/2.

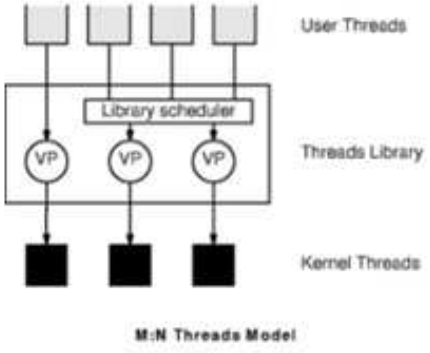


Gambar 19. Model One to One.

9.3. Model Many to Many

Beberapa tingkatan thread pengguna dapat menggunakan jumlah kernel thread yang lebih kecil atau sama dengan jumlah thread pengguna. Jumlah dari kernel thread dapat dispesifikasikan untuk beberapa aplikasi dan beberapa mesin (suatu aplikasi dapat dialokasikan lebih dari beberapa kernel thread dalam multiprosesor daripada dalam uniprosesor) dimana model many-to-one mengizinkan pengembang untuk membuat thread pengguna sebanyak mungkin, konkurensi tidak dapat tercapai karena hanya satu thread yang dapat dijadualkan oleh kernel dalam satu waktu. Model one-to-one mempunyai konkurensi yang

lebih tinggi, tetapi pengembang harus hati-hati untuk tidak membuat terlalu banyak thread tanpa aplikasi dan dalam kasus tertentu mungkin jumlah thread yang dapat dibuat dibatasi.



Gambar 20. Model Many to Many