Proses-Proses

Konsep Proses

- Proses merupakan semua aktifitas CPU seperti :
  - Jobs yg dieksekusi pd sistem batch
  - User program atau task pada sistem time-shared
  - Beberapa program yang dijalankan pada satu waktu : satu program interaktive dan beberapa program batch pada sistem single user seperti MS-DOS dan Macintosh O.S

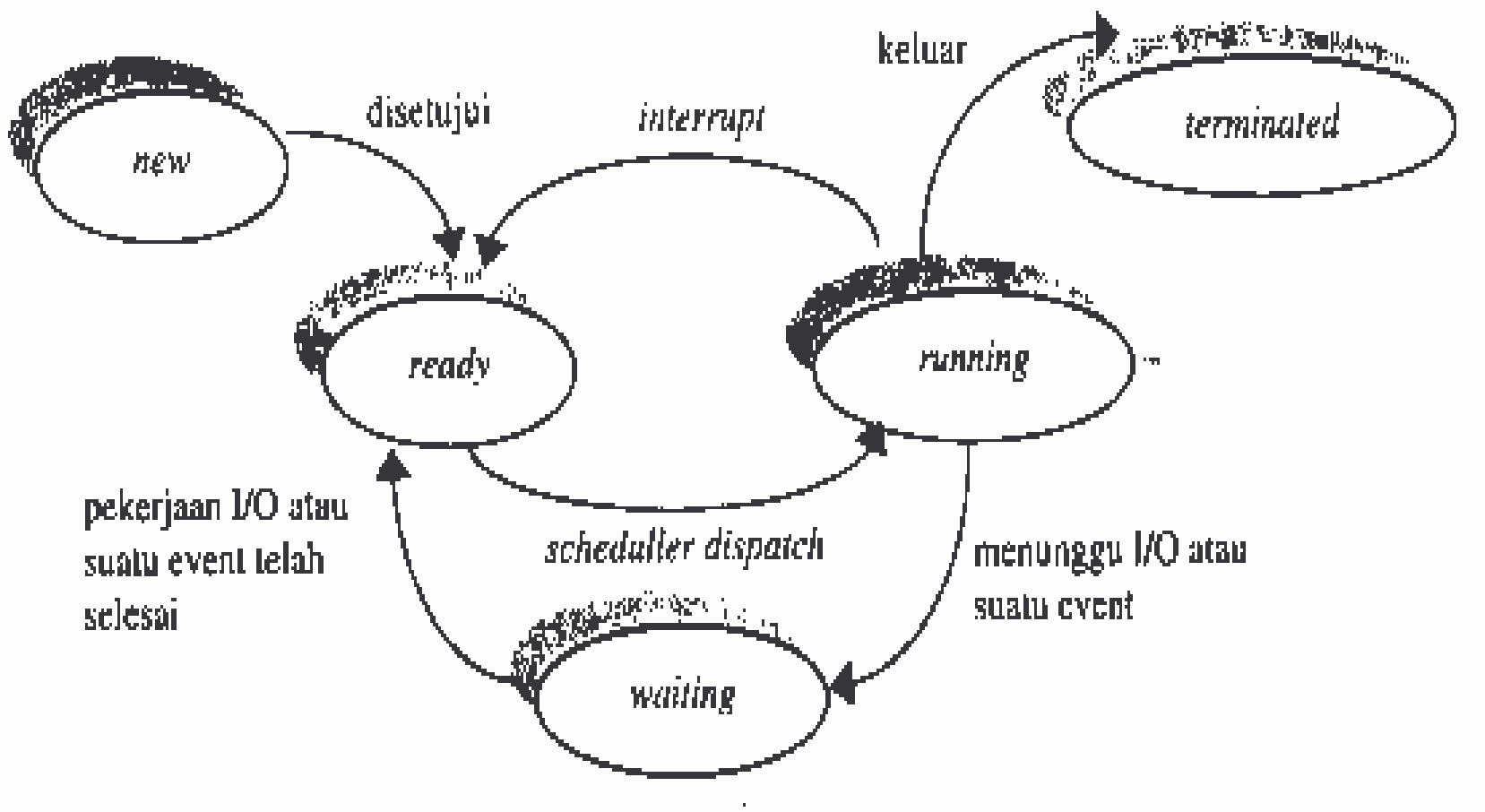
Proses

- Proses adalah program yang sedang dieksekusi. Eksekusi proses dilakukan secara berurutan sehingga satu instruksi dieksekusi sesuai kepentingan proses
- Proses bukan hanya "kode program" tapi juga termasuk di dalamnya :
  - Aktifitas saat ini yang merupakan nilai dari "program counter"
  - Isi dari register processor
  - Proses stack, berisi data sementara sebagai parameter subroutine, return address dan variabel sementara
  - "data section" yang berisi variabel global
- Sebuah program saja bukan sebuah proses. Program adalah entiti pasif seperti file yang disimpan pada disk. Proses adalah entiti aktif, mempunyai program counter yang menunjuk ke instruksi selanjutnya yg akan dieksekusi dan sekumpulan resource
- Dua proses yang merupakan program yang sama mempunyai urutan eksekusi yang terpisah

Process State

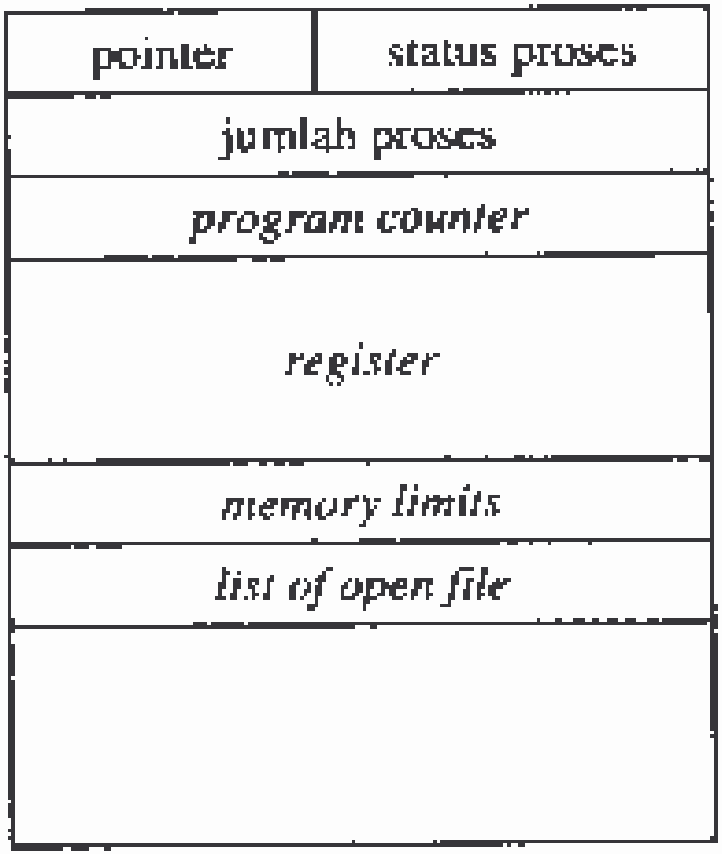
- Bila proses dieksekusi kemungkinan terjadi perubahan "state"
- State dari proses menjadi bagian dari aktifitas yang sedang dilakukan proses, state terdiri dari :
  - New : proses sedang dibuat
  - Running : proses bisa dieksekusi, karena CPU tidak sedang mengerjakan tugas yang lain
  - Waiting : proses sedang menunggu beberapa event yang akan terjadi seperti penyelesaian I/O atau menerima sinyal
  - Ready : proses menunggu jatah waktu dari prosessor
  - Terminated : proses selesai dieksekusi

Process State – diagram



Process Control Block (PCB) - 1

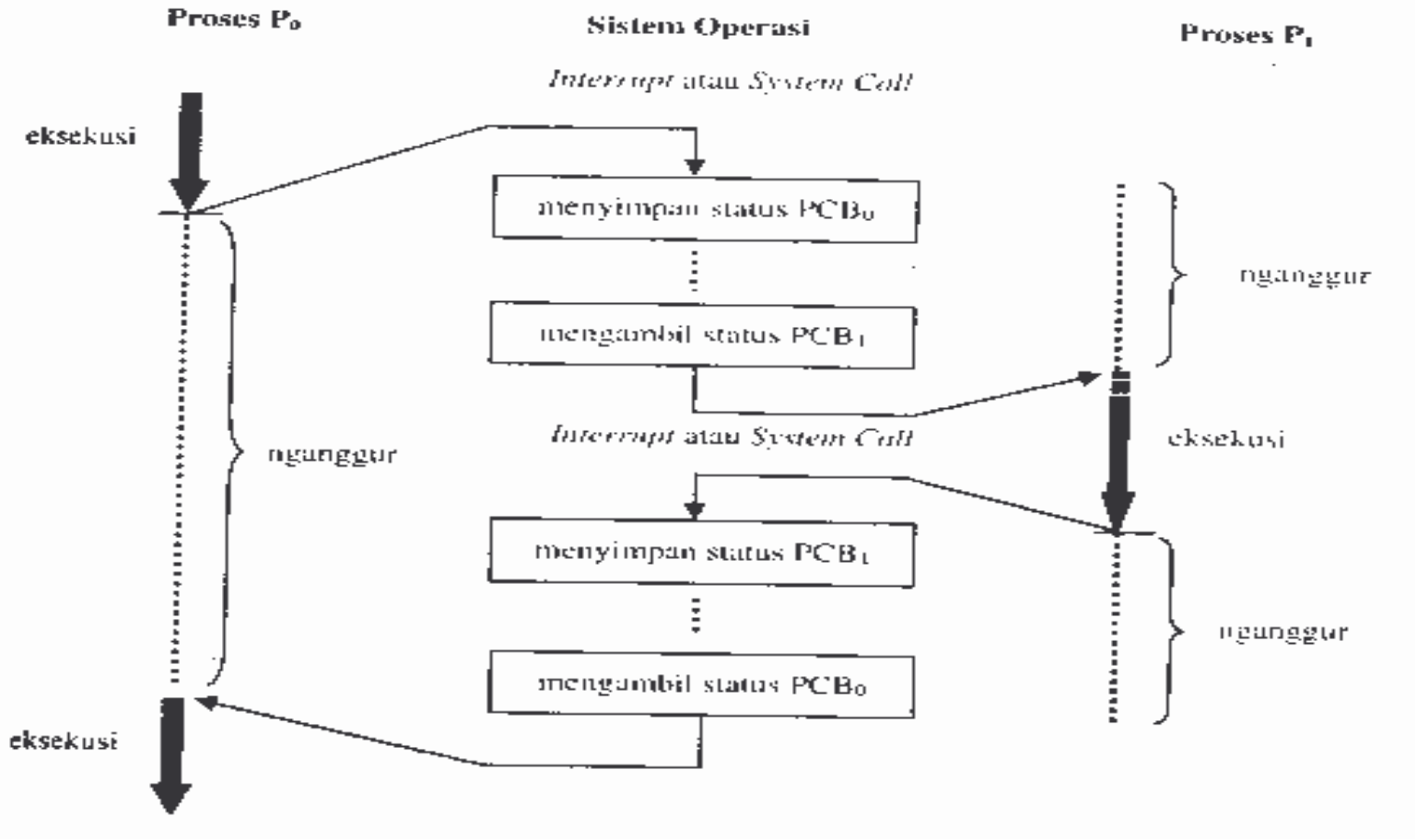
- Setiap proses direpresentasikan pada OS sebagai "Process Control Block (PCB)" atau "Task Control Block"



PCB - 2

- PCB berisi informasi dari proses tertentu sbb :
  - **Process State** : new, ready, running, waiting, terminated
  - **Program Counter** : menunjukkan alamat berikutnya yang akan dieksekusi oleh proses tersebut
  - **CPU register** : accumulator, index register, stack pointer, general purpose register, condition code information. Bila terjadi interrupt, informasi disimpan dan proses dilanjutkan
  - **Informasi Penjadwalan CPU** : berisi prioritas dari proses, pointer ke antrian penjadwalan, parameter penjadwalan lainnya
  - **Informasi Manajemen Memori** : berisi nilai (basis) dari limit register, page table atau segment table
  - **Informasi Accounting** : berisi jumlah CPU dan real time yang digunakan, time limits, account numbers, jumlah job atau proses dll
  - **Informasi Status I/O** : berisi deretan I/O device yang dialokasikan untuk proses, daftar file yang dibuka dll

PCB - switching process



Penjadwalan Proses

- Pada lingkungan multiprogramming, beberapa proses harus dijadwalkan untuk memaksimalkan utilitas CPU
- Pada lingkungan time-shared, CPU sering di-switch diantara proses-proses dimana user harus berinteraksi untuk setiap program saat di-running
- Pada sistem uniprocessor, tidak pernah terjadi lebih dari satu proses yang di-running. Bila terdapat banyak proses, sisanya harus menunggu sampai CPU bebas dan dapat dijadwal ulang

Scheduling Queue

- Secara umum, antrian penjadwalan dapat diklasifikasikan dalam 3 kategori :
  - Job queue : antrian berisi semua proses yang memasuki sistem
  - Ready queue : tempat proses-proses di memori utama yang menunggu dieksekusi
  - Device queue : deretan proses yang sedang menunggu peralatan I/O tertentu
- Setiap antrian disimpan sebagai linked list. Header berisi prointer PCB pertama dan terakhir dari list. Setiap PCB mempunayi pointer yang menunjuk ke proses selanjutnya pada antrian
- Proses baru mula-mula diletakkan di ready queue dan menunggu sampai dipilih untuk dieksekusi (dispatched) CPU
- Ketika proses dialokasikan CPU dan dieksekusi, terjadi satu dari event berikut :
  - Proses meminta I/O dan kemudian ditempatkan pada I/O queue
  - Proses membuat sub proses baru dan menunggu diterminasi
  - Proses dihapus dari CPU karena di-interrupt dan dikembalikan ke ready queue

Ready Queue dan Device Queue

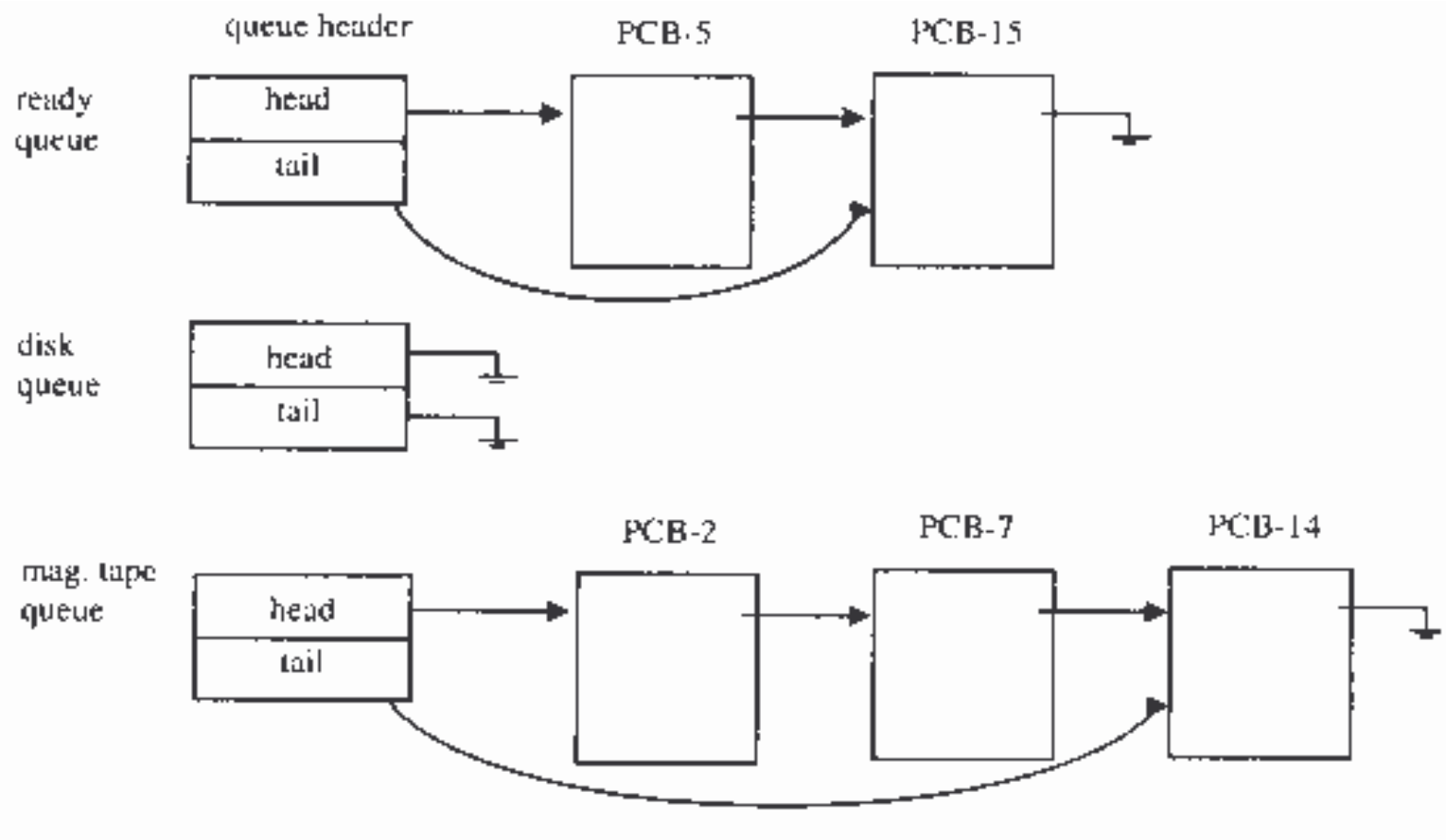
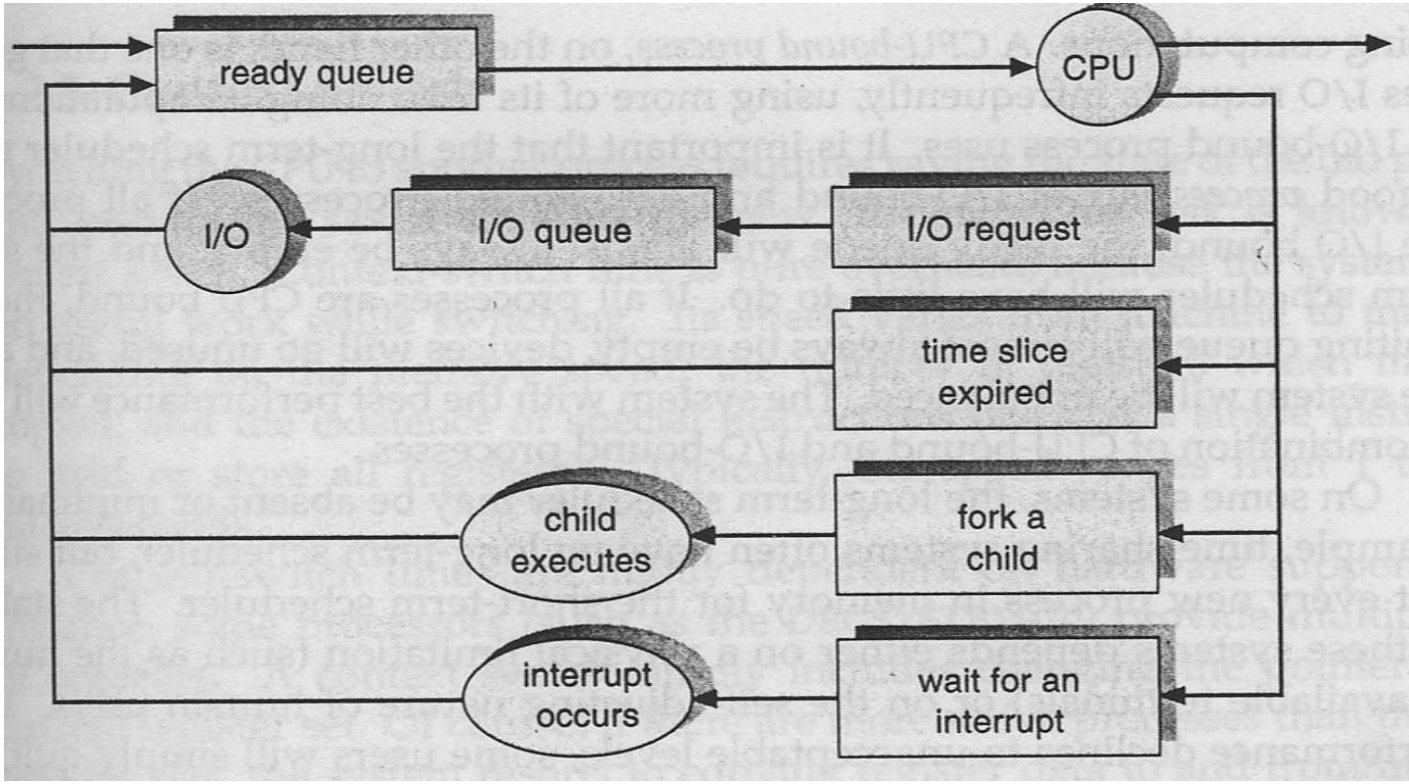


Diagram Antrian Penjadwalan Proses



Scheduler

- Scheduler dilibatkan untuk memilih proses dari scheduling queue yang berbeda
- Secara umum terdapat 3 tipe scheduler :
  - Long-term Schedulers (job scheduler)
  - Short-term Schedulers (CPU scheduler)
  - Medium-term Schedulers

Long- term Scheduler

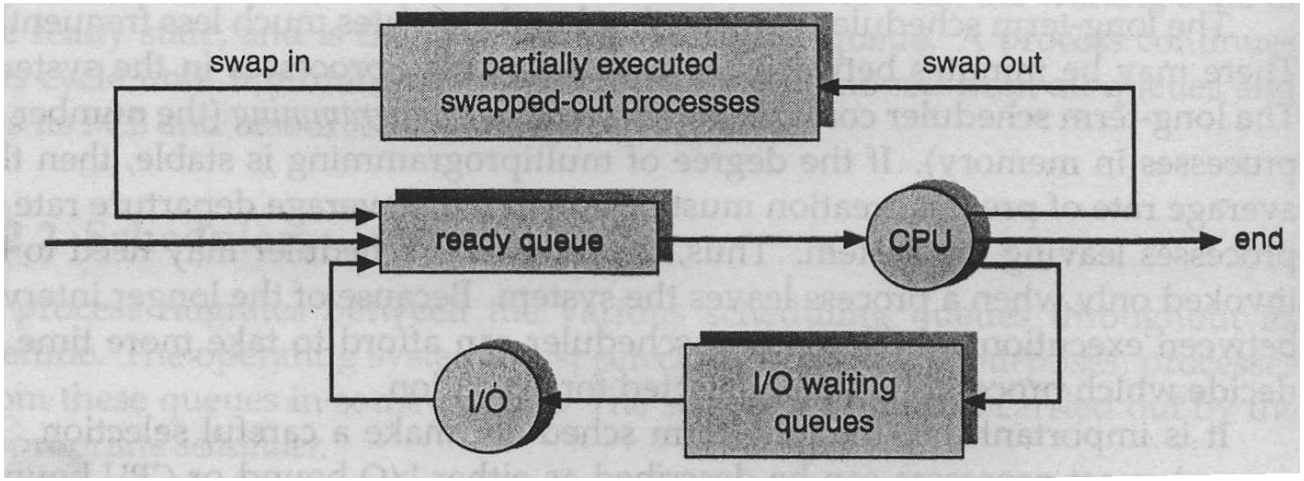
- Proses-proses pada sistem batch di-spool ke mass-storage device (disk), disimpan sebagai eksekusi selanjutnya
- Long-term scheduler digunakan untuk memilih proses dari pool dan menyimpannya ke memory
- Long-term scheduler tidak sering mengeksekusi, digunakan hanya jika proses meninggalkan sistem
- Karena antar eksekusi terjadi interval yang panjang, long-term scheduler mempunyai waktu lebih banyak untuk memutuskan proses mana yang dipilih untuk eksekusi
- Long-term scheduler memilih dengan baik "process mix" antara *I/O-bound* dan *CPU-bound*
  - Bila semua proses adalah I/O-bound, ready queue hampir selalu kosong
  - Bila semua proses adalah CPU-bound, I/O queue hampir selalu kosong
- Pada beberapa sistem, long-term scheduler tidak digunakan (misalnya pada time-sharing system) atau minimal

Short-term Scheduler

- Short-term scheduler digunakan untuk memilih diantara proses-proses yang siap dieksekusi dan salah satunya dialokasikan ke CPU
- Short-term scheduler sering digunakan untuk memilih proses baru untuk CPU. Proses dieksekusi hanya beberapa milidetik sebelulm menunggu I/O
- Karena durasi yang pendek antara eksekusi, short-tem scheduler harus sangat cepat
  - Contoh : jika short-term scheduler membutuhkan 10ms untuk memutuskan mengeksekusi proses 100 ms, maka 10/110 = 9 % CPU digunakan untuk menjadwalkan pekerjaan
- Pada sistem time-sharing, setiap proses baru ditempatkan di memori. Short-term scheduler digunakan untuk memilih dari proses-proses tersebut di memory untuk dieksekusi

Medium-term Scheduler

- Beberapa OS seperti sistem time-sharing, membutuhkan penjadwalan level tambahan (intermediate), yang disebut "medium-term scheduler"
- Memperkenalkan konsep swapping proses : proses di "swap out" dan di "swap in" pada medium-term scheduer
- Swapping diperlukan untuk meningkatkan "process mix" atau karena perubahan pada kebutuhan memori melebihi memory yang tersedia, memory perlu dibebaskan



Contex Switch

- Switching CPU ke proses yang lain membutuhkan menyimpan state dari proses yang lama dan mengambil state untuk proses yang baru. Pekerjaan ini yang disebut "context switch"
- Context switch merupakan waktu tambahan di luar sistem, karena sistem tidak berguna selama switching. Biasanya kecepatan berkisar antara 1 s/d 1000 ms, tergantung dukungan h/w, seperti kecepatan memori, jumlah register yang harus dicopy dll
- Context switch berisi perubahan pointer ke register set saat ini. Jika terdapat proses aktif yang lebih banyak dibandingkan register set, sistem mengcopy data register ke dan dari memory.
- Semakin kompleks suatu OS, semakin banyak pekerjaan yang dilakukan selama context switch

Operasi Pada Proses

- OS menyediakan mekanisme untuk membuat (create) dan menhentikan (terminate) proses, karena proses-proses pada sistem dapat dieksekusi secara konkuren dan harus dibuat dan dihapus secara dinamis

Pembuatan Proses - 1

- Selama eksekusi, proses kemungkinan membuat beberapa proses baru menggunakan system call create-process
  - Proses yang dibuat disebut proses "parent"
  - Proses baru yang dibuat disebut "child" dari proses
- Setiap proses baru membuat proses lain membentuk "tree" dari proses
- Beberapa event yang terjadi ketika proses membuat sub-proses :
  - Sub-proses dapat memperoleh resource langsung dari OS, atau terbatas pada subset dari resource pada proses parent
  - Parent harus membagi resource diantara child-nya atau dapat membagi beberapa resource (seperti memory) diantara sejumlah child-nya
  - Resource fisik dan logika yang berbeda yang diperoleh saat dibuat, inisialisasi data (input) dilebatkan oleh proses parent ke proses child
- Dua kemungkinan yang terjadi saat eksekusi ketika proses membuat proses baru :
  - Parent melanjutkan eksekusi beriringan dengan child-nya
  - Parent menunggu sampai beberapa atau semua child-nya dihentikan
- Dua kemungkinan lokasi alamat proses baru :
  - Proses child merupakan duplikasi dari proses parent
  - Proses child mempunyai program untuk menyimpannya

Pembuatan Proses - 2

- Pembuatan Proses di UNIX :
  - Setiap proses diidentifikasi oleh "proses indentifier" (unique integer)
  - Proses baru dibuat dengan system call "fork"
    - Proses baru berisi copy jika ruang alamat dari proses original, proses parent lebih mudah berkomunikasi dengan proses child
    - Baik proses (parent maupun child) melanjutkan eksekusi setelah "fork" dengan satu perbedaan : return code "fork" bernilai zero untuk proses child baru dan proses identifier dari child bernilai non-zero jika kembali ke parent
- MS-Windows / NT :
  - Ruang alamat parent diduplikasi atau
  - Parent menentukan nama program untuk OS yang disimpan ke ruang alamat dari proses baru

Penghentian Proses - 1

- Proses dihentikan apabila selesai mengeksekusi pernyataan terakhir dan meminta OS untuk menghapusnya menggunakan system call "exit
- Beberapa event yang mungkin terjadi saat terminasi :
  - Proses menghasilkan data (output) ke proses parent (melalui system call "fork")
  - Semua resource proses, termasuk memory fisik dan virtual, open file dan I/O buffer didealokasi oleh OS
  - Sebuah proses dapat menyebabkan terminasi proses lainnya melalui system call tertentu (contoh : abort). Biasanya, system call ini capat dihasilkan hanya oleh parent dari proses yang diterminasi
- Parent menhentikan eksekusi satu dari child untuk sejumlah alasan, misalnya :
  - Child melebihi penggunakan dari beberapa resource yang sudah dialokasikan
  - Task diserahkan ke child yang tidak lagi diperlukan
  - Parent tersedia, OS tidak mengijinkan child melanjutkan jika parent diterminasi

Penghentian Proses - 2

- Pada UNIX
  - Proses mungkin diterminasi dengan menggunakan system call “exit” dan parent menunggu event dengan menggunakan system call “wait”
  - System call “wait” menghasilkan identifier proses dari child yang diterminasi, sehingga parent dapat memberitahu beberapa child yang mungkin diterminasi
  - JIka parent diterminasi, semua child diterminasi dengan OS, karena tanpa parent, UNIX tidak mengetahui siapa yang melaporkan aktifitas dari child

Alasan Penghentian Proses

- Proses selesai mengerjakan tugasnya (selesai normal)
- Proses berjalan melebihi batas waktu
- Memori tidak tersedia
- Proses mengakses kawasan memori yang tidak boleh diakses
- Terjadi kesalahan karena pelanggaran proteksi
- Terjadi kesalahan perhitungan
- Proses menunggu terlalu lama
- Terjadi kegagalan I/O
- Proses mengeksekusi instruksi yang tidak ada
- Proses menggunakan instruksi yang disimpan untuk SO
- Terjadi kesalahan penggunaan data
- Terjadi intervensi dari operator atau SO (contoh : deadlock)
- Proses induk berakhir
- Atas permintaan proses induk

Proses yang Saling Bekerja Sama - 1 (Cooperating Process)

- Proses-proses konkuren yg dieksekusi di OS berupa proses-proses yang terpisah (independence) atau saling bekerjasama (cooperate)
  - Proses terpisah (independent) bila tidak dapat berakibat atau diakibatkan oleh proses lain yang dieksekusi di sistem, berarti proses yg tidak membagi semua data dengan proses lain
  - Proses bekerjasama (cooperate) jika dapat berakibat atau diakibatkan oleh proses lain yang dieksekusi di sistem, berarti proses yang membagi data dengan proses lain
- Lingkungan yang memungkinkan proses bekerjasama dilatarbelakangi beberapa alasan sbb :
  - Sharing informasi : karena beberapa user tertarik pada sebagian informasi (seperti file sharing), harus disediakan lingkungan untuk mengijinkan akses konkuren untuk tipe resource ini
  - Kecepatan komputasi : jika kita ingin menjalankan task tertentu lebih cepat, kita harus membagi menjadi sejumlah sub task, yang dieksekusi paralel. Kecepatan dapat dicapai jika komputer mempunyai elemen multiple processing
  - Modularity : jika ingin membangun sistem modular, fungsi sistem dibagi ke beberapa proses terpisah
  - Kenyamanan : user individual mungkin mempunyai beberapa task yang bekerja pada satu waktu, contohnya user edit, print, compile secara paralel
- Eksekusi konkurent yang membutuhkan kerjasama diantara proses membutuhkan mekanisme untuk mengijinkan proses berkomunikasi satu sama lain dan sinkronisasi proses

Cooperating Process – 2

Contoh : Permasalahan Producer-Consumer (PC)

- Proses producer menghasilkan informasi yang dikonsumsi oleh proses consumer, contoh :
  - program untuk mencetak menghasilkan karakter-karakter yang akan dikonsumsi oleh printer driver;
  - compiler menghasilkan kode assembly yang akan dikonsumsi assembler;
  - assembler menghasilkan object modul yang akan dikonsumsi loader;
- Buffer yang berisi item dapat diisi oleh producer dan dikosongkan oleh consumer harus dibuat sehingga proses producer dan comsumer berjalan konkuren
  - Unbounded-buffer : permasalahan PC tidak terbatas ukuran buffer. Producer dapat selalu menghasilkan item baru, tetapi consumer harus menunggu item baru
  - Bounded-buffer : permasalahan PC mengasumsikan terdapat ukuran buffer tertentu (fixed). Dalam hal ini producer harus menunggu jika buffer penuh dan consumer harus menuggu jika buffer kosong
  - Buffer disediakan oleh OS melalui penggunaan IPC (Inter-process communication) atau secara eksplisit disediakan oleh programmer dengan menggunakan shared memory

Cooperating Process – 3

Penyelesaian dg Shared Memory untuk permasalahan Bounded-buffer

- Variabel :

```
CONST   n = …;
TYPE    item = …;
VAR     buffer: ARRAY [0..n-1] OF item;
        in, out: 0..n-1
```

  - Variabel `in, out` diinisialisasi 0
  - Solusi paling banyak `n-1` item pada buffer pada waktu yg sama
  - Buffer shared diimplementasikan sebagai circular array dengan pointer logika `in` dan `out`

Cooperating Process – 4

Penyelesaian dg Shared Memory untuk permasalahan Bounded-buffer

- Producer process :

(nextp adalah variabel lokal yang digunakan untuk menyimpan item baru yang diproduksi)

```
REPEAT
        …
        memproduksi item dalam nextp
        …
        WHILE in+1 MOD n = out DO nothing;
        buffer[in] := nextp;
        in := (in+1) MOD n;
UNTIL false
```

- Consumer process :

(nextc adalah variabel lokal yang digunakan untuk menyiimpan item yang dikonsumsi)

```
REPEAT
        WHILE in =out DO nothing;
        nextc := buffer[out];
        out := (out+1) MOD n;
        …
        mengkonsumsi item pada nextc
        …
UNTIL false
```

Komunikasi Antar Proses

- Sistem operasi menyediakan layanan proses untuk berkomunikasi dengan proses lain melalui *Interprocess communication* (IPC)
- IPC menyediakan mekanisme untuk berkomunikasi antar proses dan mensinkronikasinya. IPC terbaik disediakan dengan sistem pesan (message system)
- Sebagai catatan, skema komunikasi shared memory dan message system tidak eksklusif, tetapi dapat digunakan secara simultan dalam single OS atau single proses

Struktur Dasar

- Fungsi message system memungkinkan komunikasi antar proses tanpa perlu tempat untuk sharing variabel
- Fasilitas IPC menyediakan sedikitnya 2 operasi : **send(message)** dan **receive(message)**
- Pesan dikirim oleh proses berupa :
  - **Fixed size**. Implementasi fisik mudah, tetapi programming task menjadi lebih sulit
  - **Variable size**. Membutuhkan implementasi fisik yang lebih komplek, tetapi programming task lebih mudah
- Saluran komunikasi harus terdapat antara 2 proses yang akan berkomunikasi. Saluran dapat diimplementasikan dalam beberapa cara.

Komunikasi Langsung - 1 (Direct Communication)

- Setiap proses yang ingin berkomunikasi harus mencantumkan nama penerima atau pengirim secara eksplisit.
- Primitif send dan receive didefinisikan dengan (symmetric addressing) :
  send(*P, pesan*) : mengirim pesan ke proses P
  receive(*Q, pesan*) : menerima pesan dari proses Q
- Properti :
  - Saluran otomatis tersedia antara setiap pasangan proses yang ingin berkomunikasi. Proses hanya perlu mengetahui identitas proses lain untuk berkomunikasi
  - Saluran dihubungkan tepat 2 proses
  - Antara setiap pasangan proses, tersedia satu saluran
  - Saluran kemungkinan unidirectional dan bidirectional
- Contoh : Producer-consumer problem

Proses pada producer :

```
REPEAT
        (* memproduksi satu item dalam nextp *)
        send(consumer, nextp);
UNTIL false;
```

Proses pada consumer :

```
REPEAT
        receive(producer, nextc);
        (* mengkonsumsi item pada nextc *)
UNTIL false;
```

Komunikasi Langsung – 2

- Selain skema diatas juga terdapat skema *asymmetric addressing*. Hanya pengirim yang memberi nama penerima, tetapi penerima tidak perlu memberi nama pengirim. Primitif send dan receive didefinisikan sbb :
  - **send**(*P*,pesan)

    mengirim pesan ke proses P
  - **receive**(*id*, pesan)

    menerima pesan dari sembarang proses. Variabel "id" di-set nama proses dimana komunikasi terjadi
- Kerugian kedua skema (symmetric dan asymmetric) adalah modularity terbatas dari definisi proses hasil. Mengubah nama proses harus memeriksa semua definisi proses lain.

Komunikasi Tak Langsung – 1 (Indirect Communication)

- Message dikirim atau diterima dari *mailbox* (disebut juga *port*)
- Mailbox dapat dipandang sebagai obyek dimana pesan dapat ditempatkan oleh proses dan pesan dihapus oleh proses
- Setiap mailbox mempunyai identifikasi yang unik
- Primitif send dan receive didefinisikan sbb :

  send(*A, pesan*) : mengirim pesan ke mailbox A

  receive(*A, pesan*) : menerima pesan dari mailbox A
- Properti :
  - Saluran tersedia antara pasangan proses hanya jika mempunyai sharing dengan mailbox
  - Saluran dihubungkan dengan lebih dari 2 proses
  - Antara setiap pasangan proses yang berkomunikasi, terdapat sejumlah saluran yang berbeda, dimana setiap saluran berkorespondensi ke satu mailbox
  - Saluran baik unidirectional maupun bidirectional

Komunikasi Tak Langsung – 2

- Misalnya *P1*, *P2* dan *P3* berhubungan dengan mailbox *A*. Proses *P1* mengiriim pesan ke *A*, sementara *P2* dan *P3* masing masing menjalankan **`receive`** dari *A*. Proses mana yang akan menerima pesan *P1* ? Pemecahannya bisa beberapa macam sbb :
  - Mengijinkan sebuah link dihubungkan paling banyak 2 proses
  - Mengijinkan satu proses dalam satu waktu mengeksekusi operasi receive
  - Mengijinkan sistem memilih proses mana yang menerima pesan (*P2* atau *P3* tetapi bukan keduanya). Sistem menyebutkan penerima ke pengirimnya

Komunikasi Tak Langsung – 3

- Mailbox dimiliki oleh proses atau oleh sistem
- Jika mailbox dimiliki proses, maka owner (yang hanya menerima pesan) dan user (yang hanya mengirim pesan) dari mailbox harus berbeda
  - Terdapat beberapa cara berbeda untuk menandakan owner dan user dari mailbox tertentu. Satu kemungkinan adalah mengijinkan proses untuk mendeklarasikan variabel dari tipe mailbox. Kemudian proses lain yang mengetahui nama mailbox ini dapat menggunakan mailbox ini.
- Jika mailbox dimiliki OS, maka bersifat independent dan tidak diperuntukkan untuk proses tertentu. Tetapi OS perlu menyediakan mekanisme yang mengijinkan proses :
  - Membuat mailbox baru
  - Mengirim dan menerima pesan melalui mailbox
  - Menghapus mailbox

Komunikasi Tak Langsung – 4

- Proses yang membuat mailbox baru secara default merupakan owner dari mailbox yang hanya dapat menerima pesan melalui mailbox.
- Proses juga bisa menggunakan sebuah mailbox bersama-sama melalui fasilitas pembuatan proses
  - Contoh : proses P membuat mailbox A dan kemudian dibuat proses baru Q, P dan Q bisa menggunakan mailbox A bersama-sama

Buffering - 1

- Saluran (link) mempunyai kapasitas yang menentukan jumlah pesan yang dapat ditampung sementara. Properti ini dapat dipandang sebagai antrian dari pesan dalam bentuk link diimplementasikan dalam 3 cara :
  - **Zero capacity (no buffering)**
    - Antrian mempunyai maksimum panjang 0 sehingga link tidak mempunyai pesan yang menunggu
    - Pengirim pesan harus menunggu sampai penerima pesan menerima pesan sebelum mengirim pesan lagi
    - 2 proses harus disinkronisasi pada saat transfer pesan
  - **Bounded capacity (automatic buffering)**
    - Antrian memiliki panjang maksimum tertentu (n)
    - Jika antrian belum penuh pada saat pesan baru dikirim, pesan baru menempati antrian paling akhir dan pengirim dapat melanjutkan eksekusi tanpa menunggu
    - Bila antrian penuh, pengirim harus menunggu sampai tersecdia tempat kosong pada antrian
  - **Unbounded capacity (automatic buffering)**
    - Antrian memiliki panjang tidak terbatas, sehingga sembarang pesan dapat menunggu di link dan pengiriman tidak pernah menunda pekerjaan.

Buffering - 2

- Pada nonzero-capacity buffering, proses tidak mengetahui apakah pesan sudah diterima oleh tujuan setelah operasi send selesai
- Jika informasi ini penting untuk komputasi, pengirim harus berkomunikasi secara eksplisit dengan penerima untuk menemukan apakah pesan sudah diterima
- Sebagai contoh, misalnya proses P mengirim pesan ke proses Q dan dapat melanjutkan eksekusi hanya setelah pesan diterima
  - Proses P mengeksekusi urutan :

    send(Q, pesan);
    receive(Q, pesan);
  - Proses Q mengeksekusi :

    receive(P, pesan);
    send(P, "acknowledgment");

  Kedua proses diatas dikatakan sedang berkomunikasi secara "asynchronous"

Kondisi Pengecualian - 1

- Sistem pesan sangat bermanfaat pada lingkungan terdistribusi, dimana proses berada pada site (mesin) yang berbeda
- Pada lingkungan ini, kemungkinan error terjadi selama komunikasi (dan proses) lebih besar daripada pada lingkungan single-machine
- Jika terjadi kegagalan baik pada sistem sentalisasi maupun terdistribusi, beberapa recovery error (exception-condition handling) harus ada

Kondisi Pengecualian - 2

- **Terminasi proses**
  - Baik pengirim maupun penerima bisa menghentikan sebelum sebuah pesan diproses. Situasi ini akan meninggalkan pesan yang tidak pernah diterima atau proses menunggu pesan yang tidak pernah dikirim
  - Dua kemungkinan yang terjadi :
    - Proses penerima P menunggu untuk sebuah pesan dari proses Q yang diterminasi. Jika tidak ada aksi, P akan diblok selamanya. Dalam hal ini sistem akan menghentikan P atau memberitahu P bahwa Q sudah berhenti
    - Proses P mengirim pesan ke proses Q yang diterminasi. Pada skema buffering otomatis, P masih dapat melanjutkan eksekusi. Sebaliknya, pada skema no-buffering, P akan diblok selamanya

Kondisi Pengecualian - 3

- **Kehilangan pesan**
  - Pesan dari proses P ke proses Q bisa hilang di jaringan komunikasi, karena hadware atau kabel komunikasi gagal
  - Terdapat 3 metode dasar untuk menangani event ini :
    - OS bertanggung jawab untuk mendeteksi event ini dan mengirim pesan lagi
    - Proses pengirim bertanggung jawab untuk mendeteksi event ini dan mentransmisikan pesan lai
    - OS bertanggung jawab untuk mendeteksi event ini; kemudian memberitahu proses pengirim bahwa pesan hilang. Proses pengirim dapat memulai lagi
  - Bagaimana pesan hilang dapat dideteksi ? Metode deteksi yang umum digunakan adalah "timeouts"
    - Jika pesan dikirim, pesan acknowledgment selalu dikirim kembali
    - OS atau proses menentukan interval waktu selama menunggu pesan acknowledgment tiba. Jika periode waktu habis sebelum acknowledgment tiba, OS (atau proses) mengasumsikan bahwa pesan hilang dan pesan akan dikirim kembali

Kondisi Pengecualian - 4

- **Pesan Teracak**
  - Pada saat pesan dikirim ke tujuan, tetapi diacak ditengah jalan (sebagai contoh, karena noise pada channel komunikasi)
  - Hal ini sama kasusnya dengan kehilangan pesan, OS akan mentransmisikan kembali pesan original
  - Tipe error biasanya dideteksi menggunakan kode checking error, seperti checksum, parity dan CRC