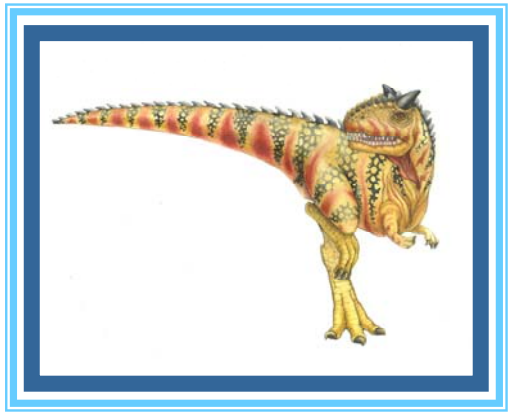
Bab 3: Proses



Bab 3: Proses

- Konsep Proses
- Penjadwalan Proses
- Operasi pada Proses Komunikasi
- Antarproses Contoh Komunikasi Sistem
- IPC dalam Sistem Client-Server
-

Tujuan

- Untuk memperkenalkan pengertian proses -- program dalam eksekusi, yang membentuk dasar dari semua perhitungan
- Untuk menggambarkan berbagai fitur proses, termasuk penjadwalan, pembuatan dan penghentian, dan komunikasi
- Untuk mengeksplorasi komunikasi antarproses menggunakan memori bersama dan pengiriman pesan
- Untuk menggambarkan komunikasi dalam sistem client-server

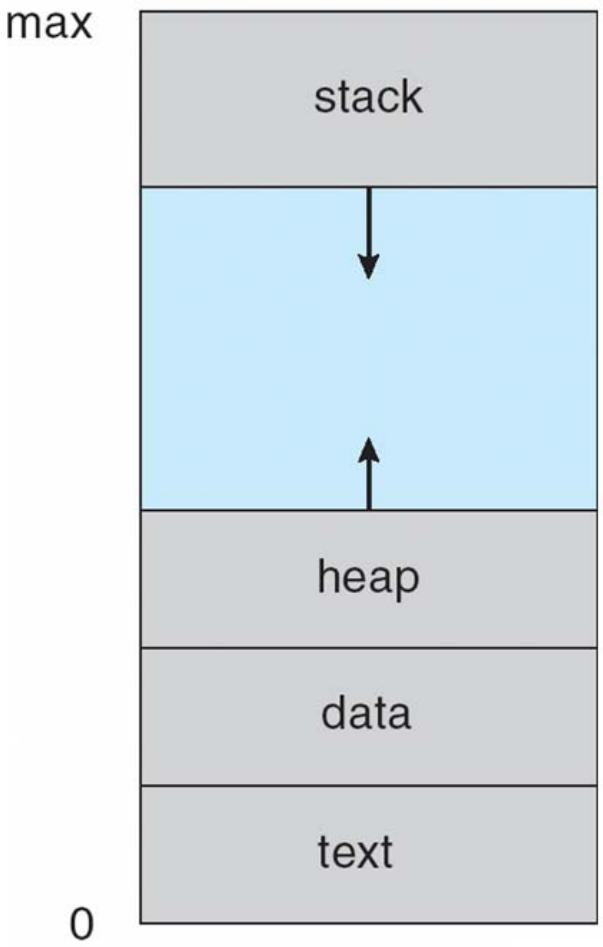
Konsep Proses

- Sebuah sistem operasi menjalankan berbagai program:
  - sistem batch-**pekerjaan**
  - Sistem pembagian waktu –**program penggunaan**atau**tugas**
- Buku teks menggunakan istilah***pekerjaan***Dan***proses***hampir bergantian
- **Proses**–sebuah program dalam eksekusi; eksekusi proses harus maju secara berurutan
- Beberapa bagian
  - Kode program, disebut juga**bagian teks**
  - Aktivitas saat ini termasuk**penghitung program**, register prosesor
  - **Tumpukan**berisi data sementara
    - Parameter fungsi, alamat pengirim, variabel lokal
  - **bagian Data**mengandung variabel global
  - **Tumpukan**berisi memori yang dialokasikan secara dinamis selama waktu berjalan

Konsep Proses (Lanjutan)

- Program adalah ***pasif*** entitas yang disimpan di disk (**file yang dapat dieksekusi**), proses adalah ***aktif***
  - Program menjadi proses ketika file yang dapat dieksekusi dimuat ke dalam memori
- Eksekusi program dimulai melalui klik mouse GUI, entri baris perintah dari namanya, dll
- Satu program bisa beberapa proses
  - Pertimbangkan banyak pengguna yang menjalankan program yang sama

Proses dalam Memori

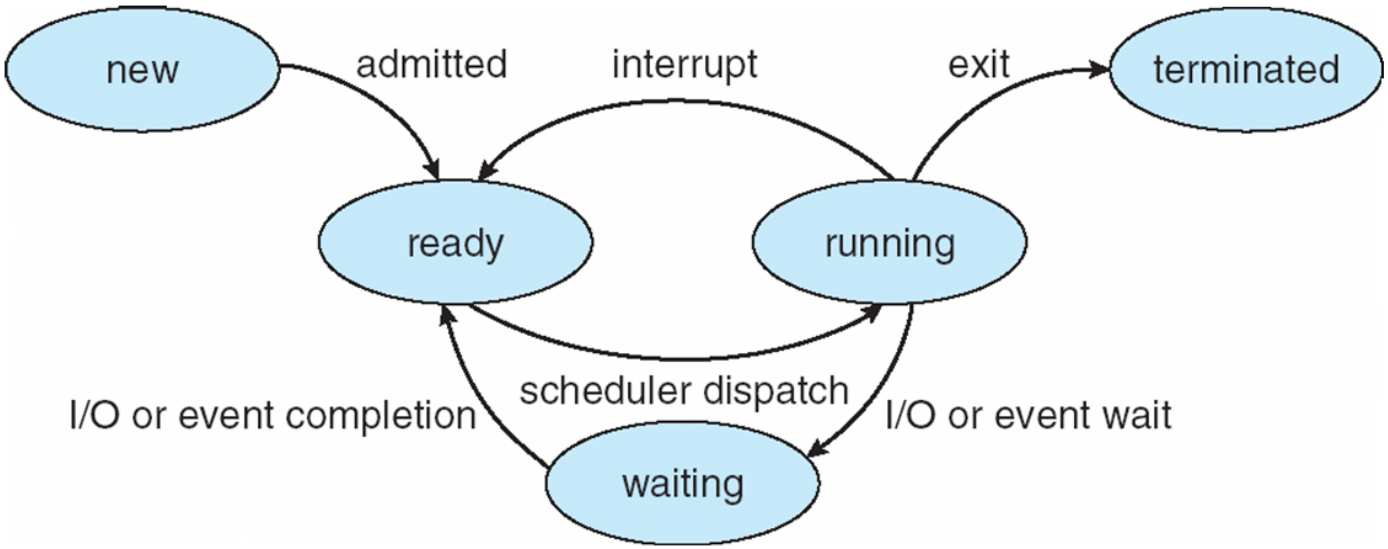


Status Proses

-Saat sebuah proses dieksekusi, proses itu berubah**negara**

- **baru**: Proses sedang dibuat **berlari**:
- Instruksi sedang dieksekusi
- **menunggu**: Proses sedang menunggu beberapa peristiwa terjadi
- **siap**: Proses sedang menunggu untuk ditugaskan ke prosesor
- **dihentikan**: Proses telah selesai dieksekusi

Diagram Status Proses



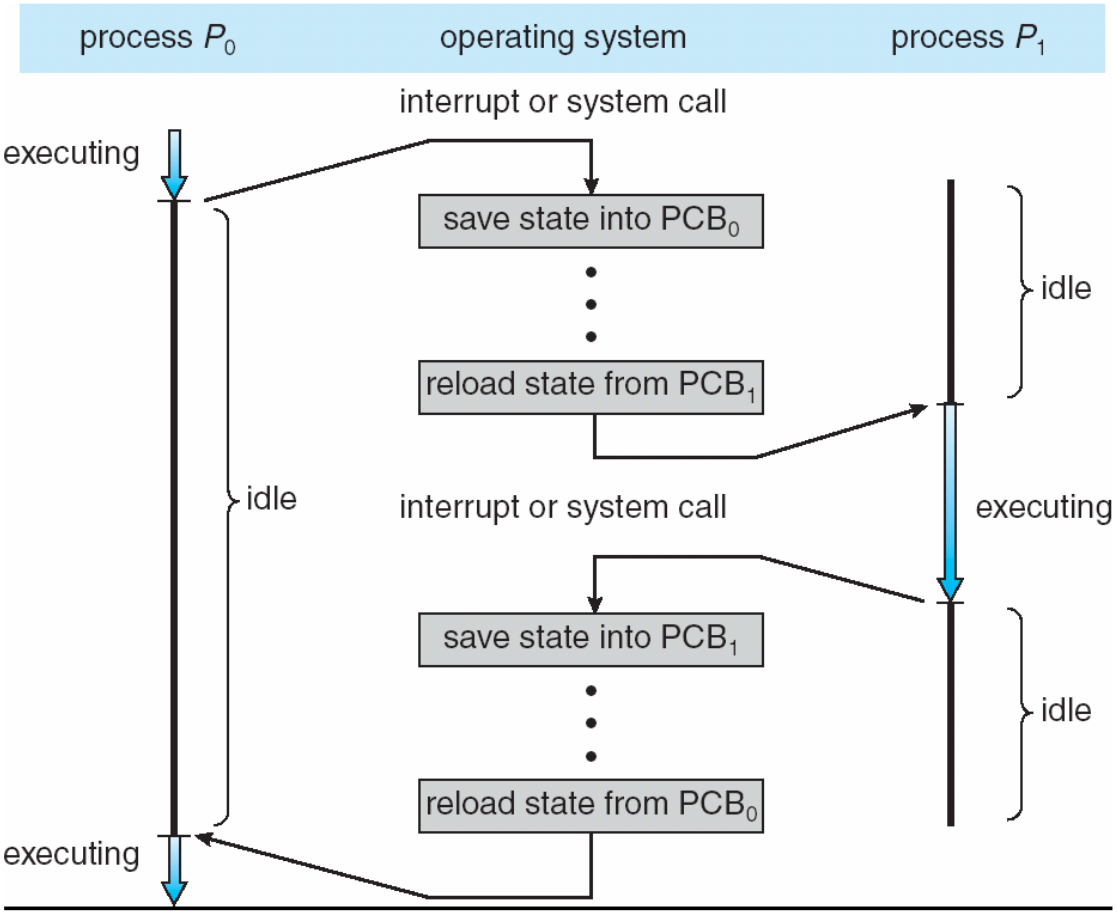
Blok Kontrol Proses (PCB)

Informasi yang terkait dengan setiap proses

(juga disebut**blok kontrol tugas**)

- Status proses - berjalan, menunggu, dll
- Penghitung program – lokasi instruksi untuk dieksekusi selanjutnya
- Register CPU – isi dari semua register yang berpusat pada proses
- Informasi penjadwalan CPU- prioritas, penunjuk antrian penjadwalan
- Informasi manajemen memori – memori yang dialokasikan untuk proses
- Informasi akuntansi – CPU digunakan, waktu jam berlalu sejak mulai, batas waktu
- Informasi status I/O – Perangkat I/O dialokasikan untuk proses, daftar file terbuka

CPU Beralih Dari Proses ke Proses



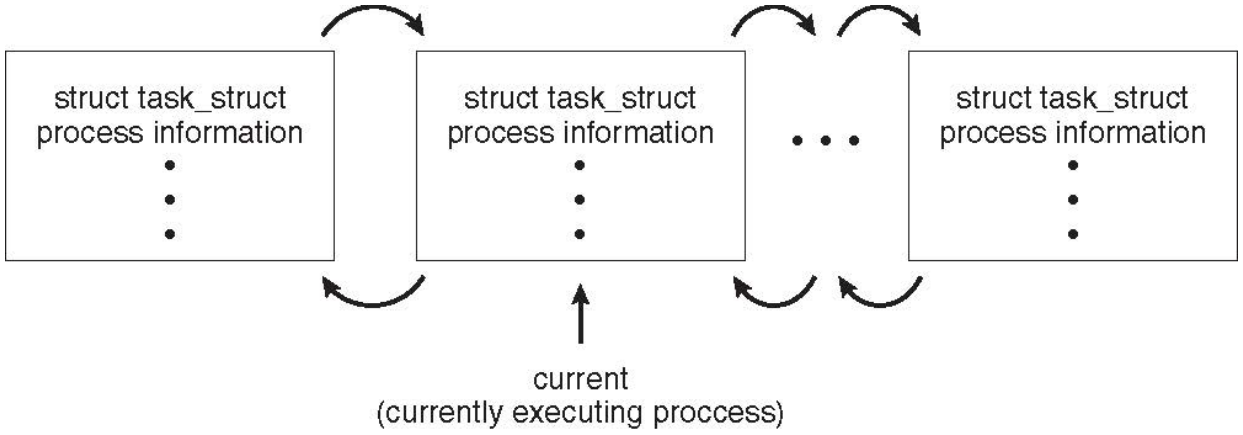
Utas

- Sejauh ini, proses memiliki satu utas eksekusi Pertimbangkan
- untuk memiliki beberapa penghitung program per proses
  - Beberapa lokasi dapat dijalankan sekaligus
    - Beberapa utas kontrol ->**benang**
- Kemudian harus memiliki penyimpanan untuk detail utas, beberapa penghitung program di PCB
- Lihat bab berikutnya

Representasi Proses di Linux

Diwakili oleh struktur Ctugas_struktur

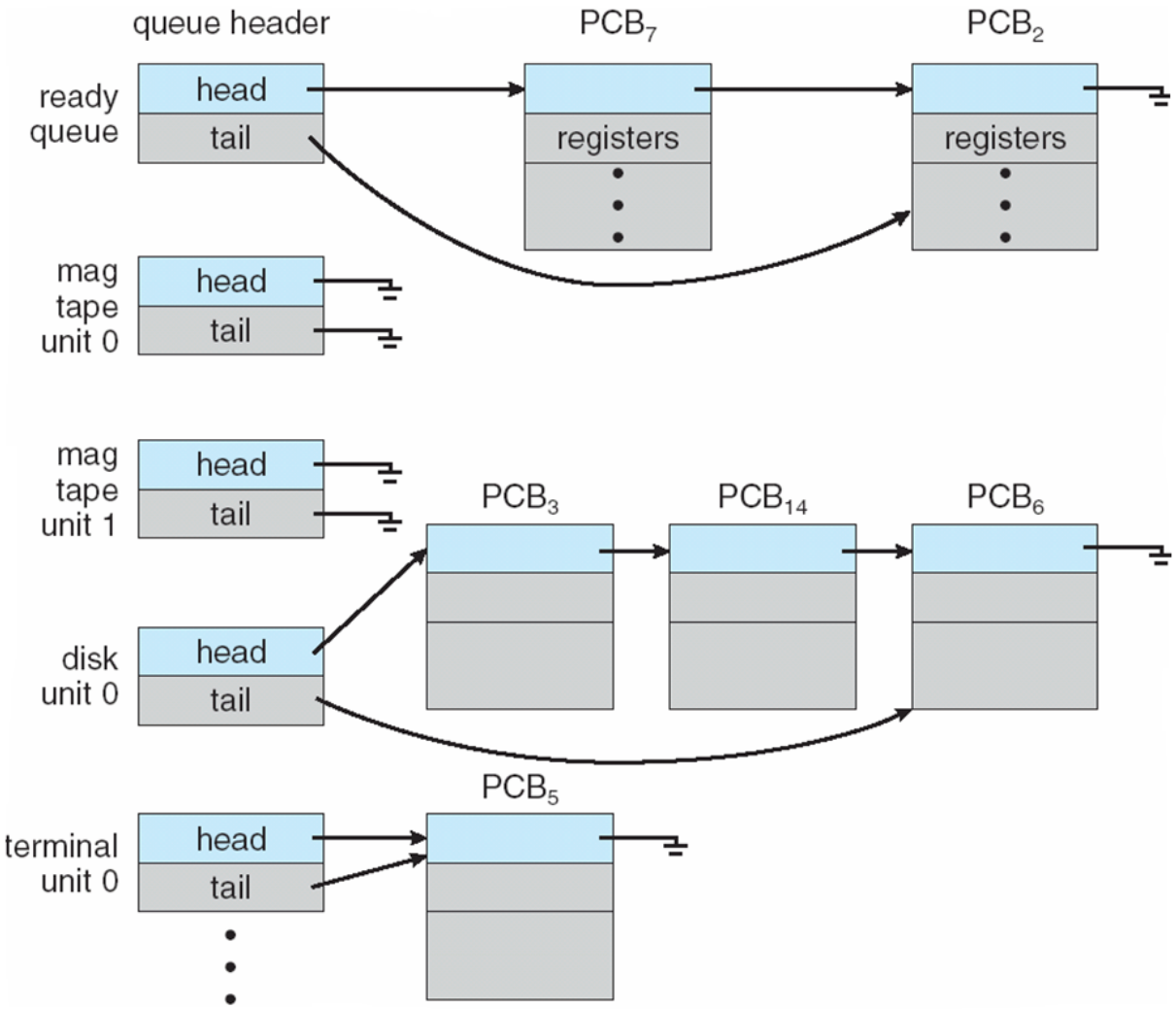
pid t_pid; /* pengidentifikasi proses */ status
panjang; /* status proses */
unsigned int time_slice /* informasi penjadwalan */ struct task_struct
parent; / induk proses ini */ struct list_head anak; /* anak proses ini */
struct files_struct *files; /* daftar file yang terbuka */ struct mm_struct *mm; /
* ruang alamat dari proses ini */



Penjadwalan Proses

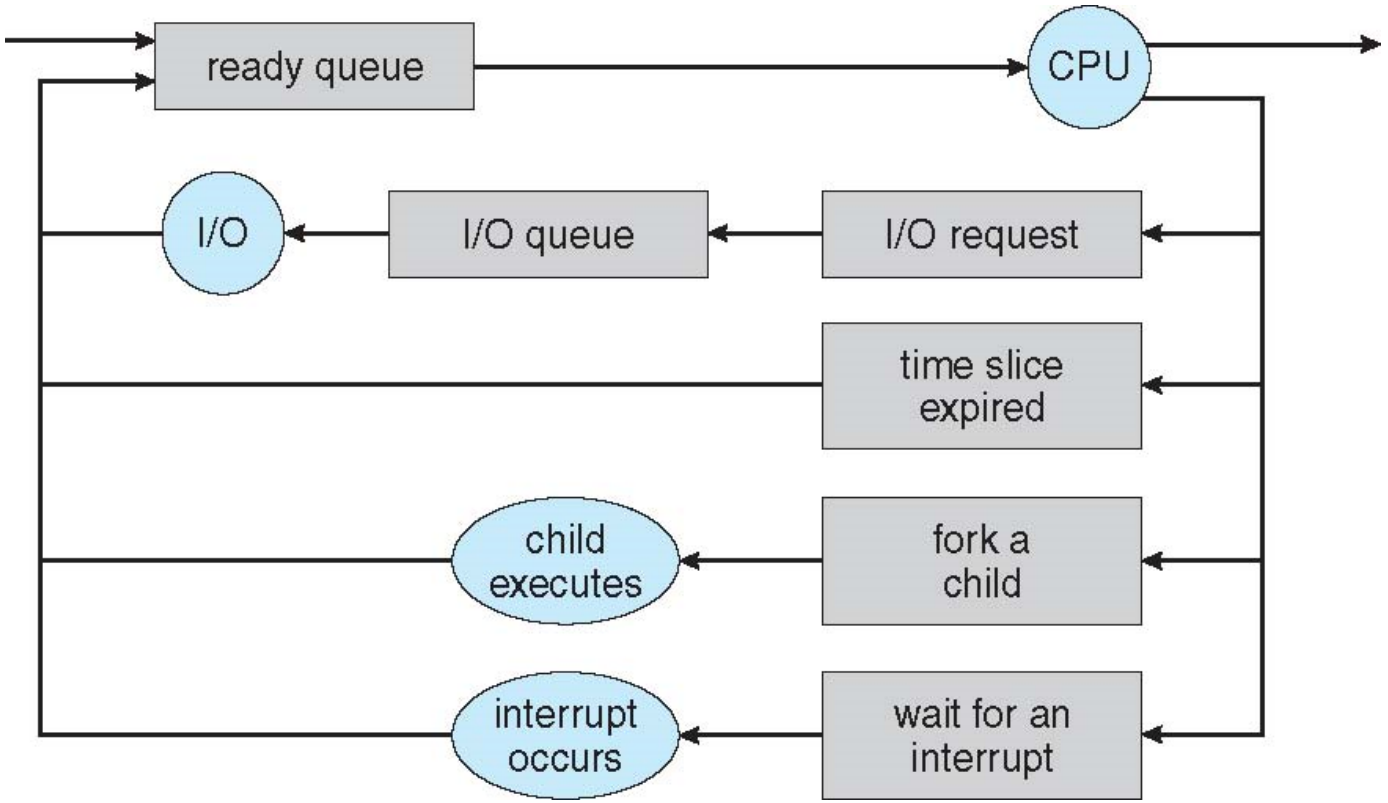
- Maksimalkan penggunaan CPU, alihkan proses dengan cepat ke CPU untuk pembagian waktu
- **Penjadwal proses** memilih di antara proses yang tersedia untuk eksekusi berikutnya pada CPU
- Mempertahankan **penjadwalan antrian** proses
  - **Antrean pekerjaan** – mengatur semua proses dalam sistem
  - **Antrian siap** – mengatur semua proses yang berada di memori utama, siap dan menunggu untuk dieksekusi
  - **Antrean perangkat** – set proses menunggu perangkat I/O
  - Proses bermigrasi di antara berbagai antrian

Antrean Siap dan Berbagai Antrian Perangkat I/O



Representasi Penjadwalan Proses

- **Diagram antrian**mewakili antrian, sumber daya, arus

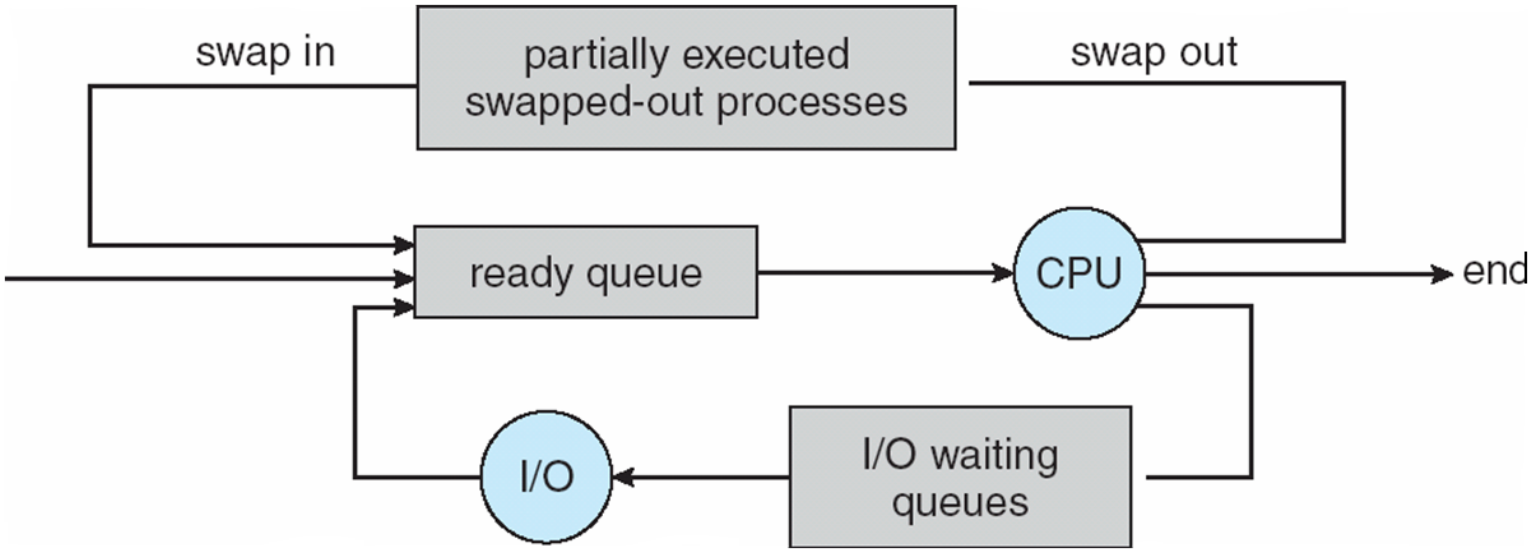


Penjadwal

- **Penjadwal jangka pendek**(atau**Penjadwal CPU**) – memilih proses mana yang harus dijalankan selanjutnya dan mengalokasikan CPU
  - Terkadang satu-satunya penjadwal dalam suatu sistem
  - Penjadwal jangka pendek sering dipanggil (milidetik) - (harus cepat)
- **Penjadwal jangka panjang**(atau**penjadwal pekerjaan**) – memilih proses mana yang harus dibawa ke ready queue
  - Penjadwal jangka panjang jarang dipanggil (detik, menit) - (mungkin lambat)
  - Penjadwal jangka panjang mengontrol**derajat multiprogramming**
- Proses dapat digambarkan sebagai:
  - **proses terikat I/O**–menghabiskan lebih banyak waktu untuk melakukan I/O daripada perhitungan, banyak ledakan CPU singkat
  - **Proses terikat CPU**–menghabiskan lebih banyak waktu melakukan perhitungan; beberapa semburan CPU yang sangat panjang
- Penjadwal jangka panjang berusaha untuk kebaikan***campuran proses***

Penambahan Penjadwalan Jangka Menengah

- **Penjadwal jangka menengah**dapat ditambahkan jika degree of multiple programming perlu dikurangi
  - Hapus proses dari memori, simpan di disk, bawa kembali dari disk untuk melanjutkan eksekusi:**bertukar**



Multitasking dalam Sistem Seluler

- Beberapa sistem seluler (misalnya, versi awal iOS) hanya mengizinkan satu proses untuk dijalankan, yang lainnya ditangguhkan
- Karena ruang layar, antarmuka pengguna membatasi yang disediakan iOS untuk a
  - Lajang**latar depan**dikendalikan oleh proses melalui antarmuka pengguna
  - Banyak**latar belakang**proses– dalam memori, berjalan, tetapi tidak pada tampilan, dan dengan batasan
  - Batasan mencakup tugas tunggal dan singkat, menerima pemberitahuan acara, tugas khusus yang berjalan lama seperti pemutaran audio
- Android menjalankan latar depan dan latar belakang, dengan batasan yang lebih sedikit
  - Proses latar belakang menggunakan a**melayani**untuk melakukan tugas
  - Layanan dapat tetap berjalan meskipun proses latar belakang ditangguhkan
  - Layanan tidak memiliki antarmuka pengguna, penggunaan memori kecil

Beralih Konteks

- Ketika CPU beralih ke proses lain, sistem harus**menyelamatkan negara**dari proses lama dan memuat**keadaan tersimpan**untuk proses baru melalui a**saklar konteks**
- **Konteks**proses yang direpresentasikan dalam PCB
- Waktu pengalihan konteks adalah overhead; sistem tidak melakukan pekerjaan yang berguna saat beralih
  - Semakin kompleks OS dan PCB - semakin lama sakelar konteksnya
- Waktu tergantung pada dukungan perangkat keras
  - Beberapa perangkat keras menyediakan beberapa set register per CPU -beberapa konteks dimuat sekaligus

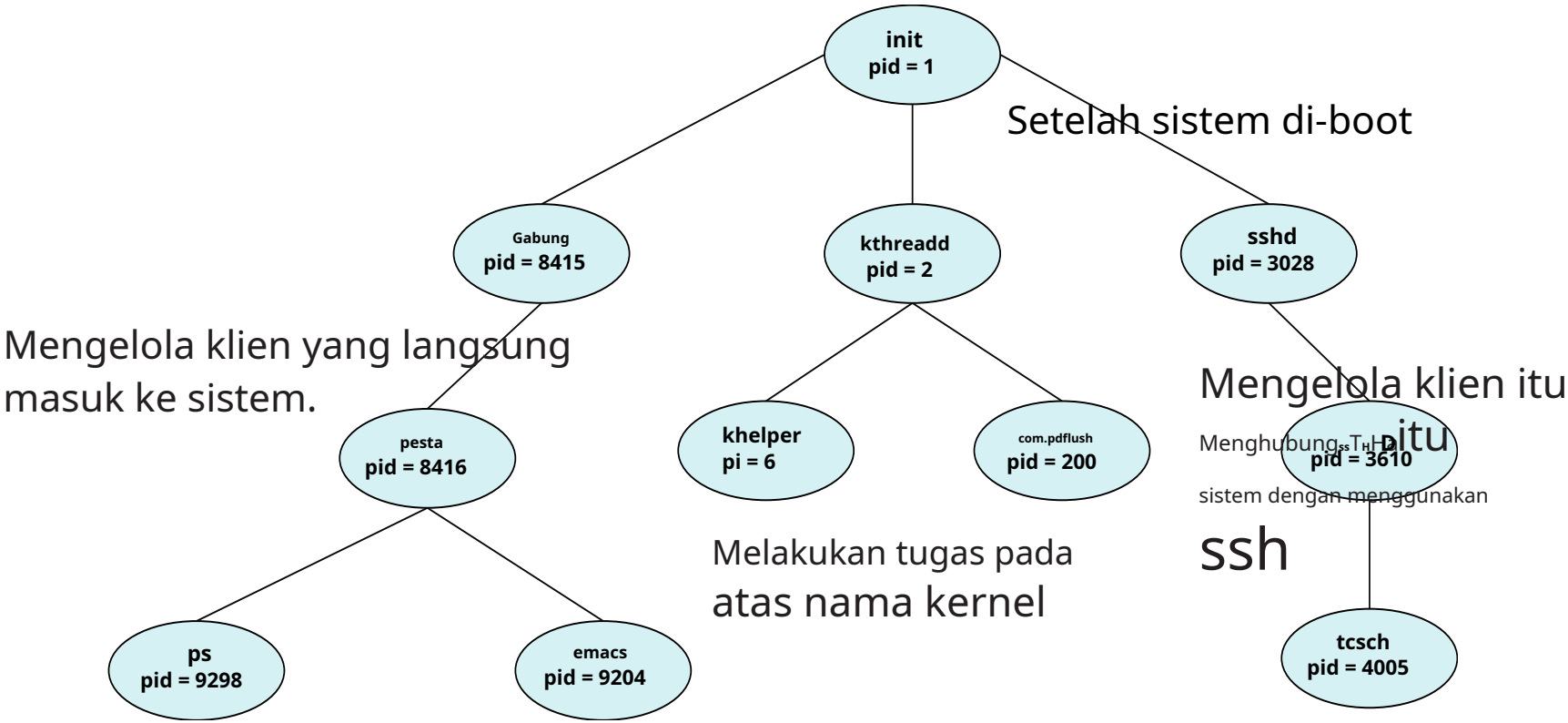
Operasi pada Proses

- Sistem harus menyediakan mekanisme untuk:
  - pembuatan proses,
  - penghentian proses,
  - dan seterusnya seperti yang dijelaskan selanjutnya

Penciptaan Proses

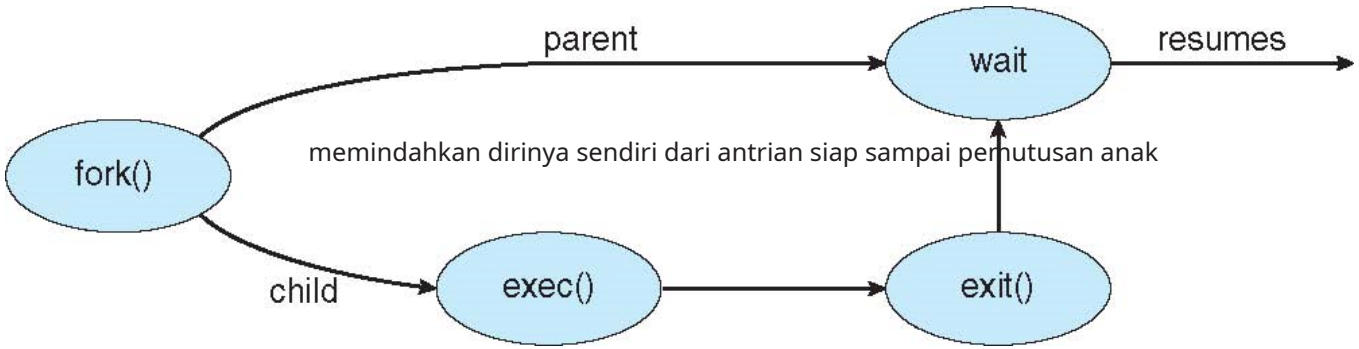
- **Induk**proses buat**anak-anak**proses, yang, pada gilirannya, menciptakan proses lain, membentuk a**pohon**proses
- Secara umum, proses diidentifikasi dan dikelola melalui a **pengenal proses**(**pid**)
- Opsi berbagi sumber daya
  - Orang tua dan anak-anak berbagi semua sumber daya
  - Anak-anak berbagi subset dari sumber daya orang tua
  - Orang tua dan anak tidak berbagi sumber daya
- Opsi eksekusi
  - Induk dan anak-anak mengeksekusi secara bersamaan
  - Induk menunggu sampai anak-anak berhenti

Sebuah Pohon Proses di Linux



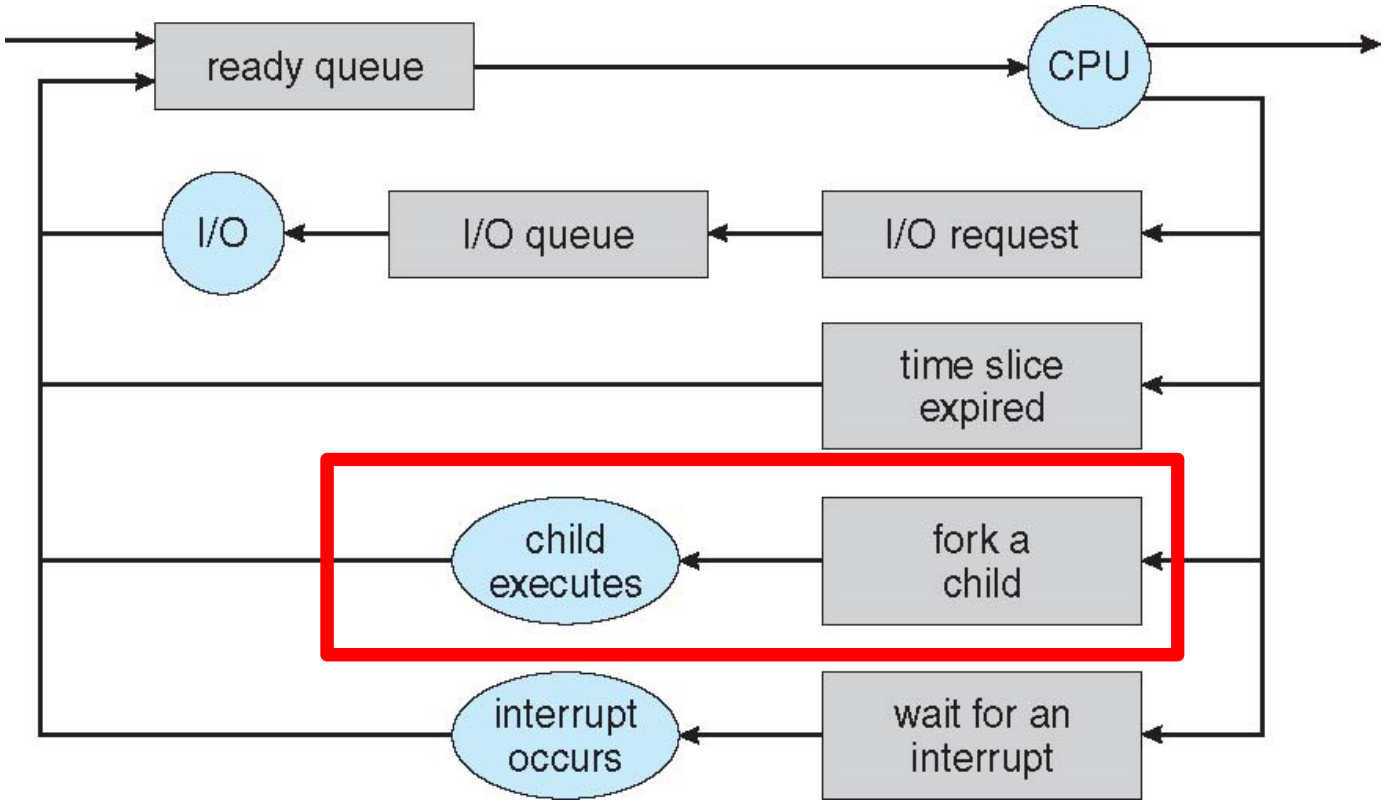
Penciptaan Proses (Lanjutan)

- Ruang alamat
  - Anak duplikat dari induk (memiliki program yang sama dengan induknya)
  - Anak memiliki program yang dimuat ke dalamnya
- Contoh UNIX
  - **garpu()** panggilan sistem menciptakan proses baru. Proses baru terdiri dari salinan ruang alamat dari proses asli.
  - **eksekusi()** panggilan sistem digunakan setelah a **garpu()** untuk mengganti ruang memori proses dengan program baru



Representasi Penjadwalan Proses

- **Diagram antrian**mewakili antrian, sumber daya, arus



Proses Terpisah Program Forking C

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

int main()
{
pid_t pid;

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

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

    return 0;
}
```

Satu-satunya perbedaan adalah bahwa nilai pid untuk proses anak adalah nol, sedangkan untuk induk adalah pid sebenarnya dari proses anak.

Penghentian Proses

- Proses mengeksekusi pernyataan terakhir dan kemudian meminta sistem operasi untuk menghapusnya menggunakan**KELUAR()**panggilan sistem.
  - Mengembalikan data status dari anak ke orang tua (melalui**Tunggu()**
  -) Sumber daya proses tidak dialokasikan oleh sistem operasi
- Induk dapat menghentikan eksekusi proses anak-anak menggunakan **menggugurkan()**panggilan sistem. Beberapa alasan untuk melakukannya:
  - Anak telah melampaui sumber daya yang dialokasikan Tugas
  - yang diberikan kepada anak tidak lagi diperlukan
  - Induk keluar dan sistem operasi tidak mengizinkan anak untuk melanjutkan jika induknya berhenti

Penghentian Proses

- Beberapa sistem operasi tidak mengizinkan anak untuk ada jika induknya telah dihentikan. Jika suatu proses berakhir, maka semua anaknya juga harus diakhiri.
  - **penghentian kaskade.**Semua anak, cucu, dll diberhentikan.
  - Pengakhiran dimulai oleh sistem operasi.
- Proses induk dapat menunggu penghentian proses anak dengan menggunakan **Tunggu()**panggilan sistem**.**Panggilan mengembalikan informasi status dan pid dari proses yang dihentikan

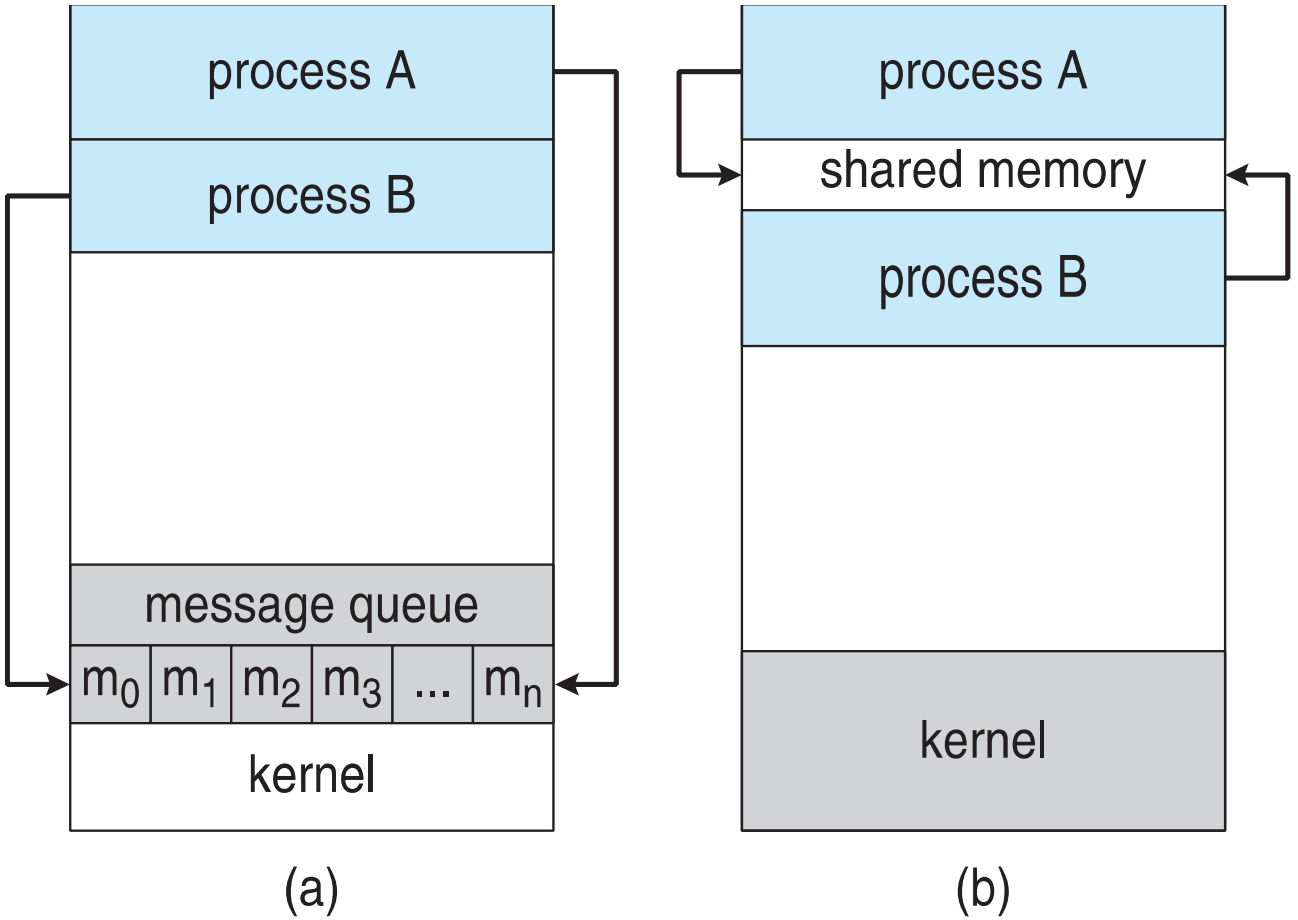
  pid = tunggu(&status);
- Jika tidak ada orang tua yang menunggu (tidak memohon**Tunggu()**) proses adalah a**zombie**
  - Setelah induk memanggil wait(), pengidentifikasi proses dari proses zombie dan entrinya di tabel proses dilepaskan.
- Jika orang tua diakhiri tanpa memohon**Tunggu**,proses adalah suatu**yatim piatu**
  - Menetapkan proses init sebagai induk baru, memanggil wait() secara berkala

Komunikasi Antarproses

- Proses dalam sistem mungkin ***mandiri*** atau ***bekerja sama***
- Proses kerja sama dapat memengaruhi atau dipengaruhi oleh proses lain, termasuk berbagi data
- Alasan untuk proses kerja sama:
  - Berbagi informasi (file bersama) Percepatan
  - komputasi (subtugas paralel)
  - Modularitas (fungsi sistem dibagi menjadi proses terpisah)
  - Kenyamanan
- Proses kerja sama membutuhkan **komunikasi antarproses** (**IPC**)
- Dua model IPC
  - **Berbagi memori**
  - **Pesan lewat**

Model Komunikasi

(a) Pesan lewat. (b) memori bersama.



Komunikasi Antarproses – Memori Bersama

- Area memori yang digunakan bersama di antara proses yang ingin berkomunikasi
  - Biasanya, wilayah memori bersama berada di ruang alamat dari proses yang membuat segmen memori bersama. Proses lain yang ingin berkomunikasi menggunakan segmen memori bersama ini harus melampirkannya ke ruang alamat mereka.
- Komunikasi berada di bawah kendali proses pengguna bukan sistem operasi.
- Masalah utama adalah menyediakan mekanisme yang memungkinkan proses pengguna menyinkronkan tindakan mereka saat mereka mengakses memori bersama.
- Sinkronisasi dibahas dengan sangat rinci di Bab 5.

Masalah Produsen-Konsumen

- Paradigma untuk proses kerja sama, *produsen* proses menghasilkan informasi yang dikonsumsi oleh a *konsumen* proses
  - **buffer tak terbatas** tidak menempatkan batasan praktis pada ukuran buffer
    - Konsumen mungkin harus menunggu barang baru, tetapi produsen selalu dapat memproduksi barang baru.
  - **buffer terbatas** mengasumsikan bahwa ada ukuran buffer tetap
    - Konsumen harus menunggu jika buffer kosong, dan produsen harus menunggu jika buffer penuh.

Bounded-Buffer – Solusi Memori Bersama

- Data bersama

```
# mendefinisikan  UKURAN BUFFER  10
typedef     struktur {
    . . .
} barang;

buffer item[BUFFER_SIZE]; int di =
0;
int keluar = 0;
```

Bounded-Buffer – Produser

```
item selanjutnya_diproduksi;

sementara (benar) {
        /* menghasilkan item dalam produksi berikutnya
        */ while (((in + 1) % BUFFER_SIZE) == out)
                ; /* tidak melakukan apapun */
        buffer[in] = next_produced; dalam =
        (dalam + 1) % BUFFER_SIZE;
}
```

Buffer Terikat – Konsumen

```
barang    selanjutnya_dikonsumsi;

ketika    (BENAR) {
              sementara (masuk == keluar)
                        ; /* tidak melakukan apa-apa
              */ next_consumed = buffer[keluar];
              keluar = (keluar + 1) % BUFFER_SIZE;

              /* konsumsi item pada konsumsi berikutnya */
}
```

Solusi benar, tetapi hanya dapat menggunakan elemen BUFFER_SIZE-1. Bagaimana merancang solusi di mana item BUFFER SIZE dapat berada di buffer pada saat yang bersamaan?

Komunikasi Antarproses – Penyampaian Pesan

- Mekanisme proses untuk berkomunikasi dan menyinkronkan tindakan mereka
- Sistem pesan - proses berkomunikasi satu sama lain tanpa menggunakan variabel bersama

-Fasilitas IPC menyediakan dua operasi:

- **mengirim**(*pesan*)
- **menerima**(*pesan*)

- Sangat berguna dalam lingkungan terdistribusi The
- *pesan*ukurannya tetap atau berubah-ubah

Penyampaian Pesan (Lanjutan)

- Jika proses *P* Dan *Q* ingin berkomunikasi, mereka perlu:
  - Menetapkan a ***tautan komunikasi*** di antara
  - mereka Bertukar pesan melalui kirim/terima
- Masalah implementasi:
  - Bagaimana tautan dibuat?
  - Bisakah tautan dikaitkan dengan lebih dari dua proses?
  - Berapa banyak hubungan yang dapat terjadi di antara setiap pasang proses komunikasi?
  - Berapa kapasitas sebuah link?
  - Apakah ukuran pesan yang dapat diakomodasi oleh tautan tetap atau variabel?
  - Apakah tautan searah atau dua arah?

Penyampaian Pesan (Lanjutan)

- Pelaksanaan link komunikasi
  - Fisik:
    - Berbagi memori
    - Bus perangkat keras
    - Jaringan
  - Logis:
    - Langsung atau tidak langsung
    - Sinkron atau asinkron
    - Penyangga otomatis atau eksplisit

Komunikasi Langsung

- Proses harus saling memberi nama secara eksplisit:
  - **mengirim**(*P, pesan*) – mengirim pesan ke proses P
  - **menerima**(*P, pesan*) – menerima pesan dari proses Q
- Properti tautan komunikasi
  - Tautan dibuat secara otomatis
  - Tautan dikaitkan dengan tepat satu pasang proses komunikasi
  - Di antara setiap pasangan terdapat tepat satu mata rantai
  - Tautan mungkin searah, tetapi biasanya dua arah

Komunikasi Tidak Langsung

- Pesan diarahkan dan diterima dari kotak surat (juga disebut sebagai port)
  - Setiap kotak surat memiliki id unik
  - Proses dapat berkomunikasi hanya jika mereka berbagi kotak surat
- Properti tautan komunikasi
  - Tautan dibuat hanya jika proses berbagi kotak surat umum
  - Tautan mungkin terkait dengan banyak proses
  - Setiap pasangan proses dapat berbagi beberapa tautan
  - komunikasi Tautan mungkin searah atau dua arah

Komunikasi Tidak Langsung

- Operasi
  - buat kotak surat baru (port)
  - mengirim dan menerima pesan melalui kotak surat
  - menghancurkan kotak surat
- Primitif didefinisikan sebagai:

  mengirim(*Sebuah pesan*) – mengirim pesan ke kotak surat A **menerima**(

  Sebuah pesan) – menerima pesan dari kotak surat A

Komunikasi Tidak Langsung

- Berbagi kotak surat
  - P_1, P_2, Dan P_3 berbagi kotak surat A P_1,
  - mengirim; P_2 Dan P_3 menerima
  - Siapa yang mendapat
- pesan? Solusi
  - Izinkan tautan dikaitkan dengan paling banyak dua proses
  - Izinkan hanya satu proses pada satu waktu untuk mengeksekusi operasi terima
  - Izinkan sistem untuk memilih penerima secara sewenang-wenang. Pengirim diberitahu siapa penerima itu.

Sinkronisasi

- Pengiriman pesan dapat berupa pemblokiran atau non-
- pemblokiran **Pemblokiran**dianggap**sinkronis**
  - **Memblokir pengiriman**--pengirim diblokir sampai pesan diterima
  - **Memblokir menerima**--penerima diblokir sampai pesan tersedia
- **Non-pemblokiran**dianggap**asinkron**
  - **Pengiriman tanpa pemblokiran**--pengirim mengirim pesan dan melanjutkan
  - **Penerimaan tanpa pemblokiran**--penerima menerima:
    - Pesan yang valid,
    - atau pesan Null
- Kombinasi yang berbeda mungkin

Sinkronisasi (Lanjutan)

- Produsen-konsumen menjadi sepele

```
pesan selanjutnya_diproduksi;
sementara (benar) {
        /* menghasilkan item dalam produksi berikutnya */
kirim(berikutnya_diproduksi);
}

pesan next_consumed;
sementara (benar) {
    terima(next_consumed);

    /* konsumsi item pada konsumsi berikutnya */
}
```

Penyangga

- Antrean pesan yang dilampirkan ke tautan.
- diimplementasikan dalam salah satu dari tiga cara
  1. Kapasitas nol – tidak ada pesan yang diantrekan di tautan. Pengirim harus menunggu penerima (rendezvous)
  2. Kapasitas terbatas – panjang terbatas *N* pesan Pengirim harus menunggu jika tautan penuh
  3. Kapasitas tak terbatas – panjang tak terbatas Pengirim tidak pernah menunggu

Komunikasi dalam Sistem Client-Server

- Soket
- Panggilan Prosedur Jarak Jauh
- Pipa
- Pemanggilan Metode Jarak Jauh (Java)

Soket

- A**stopkontak**didefinisikan sebagai titik akhir untuk komunikasi

  - Rangkaian alamat IP dan**pelabuhan**–nomor yang disertakan pada awal paket pesan untuk membedakan layanan jaringan pada host

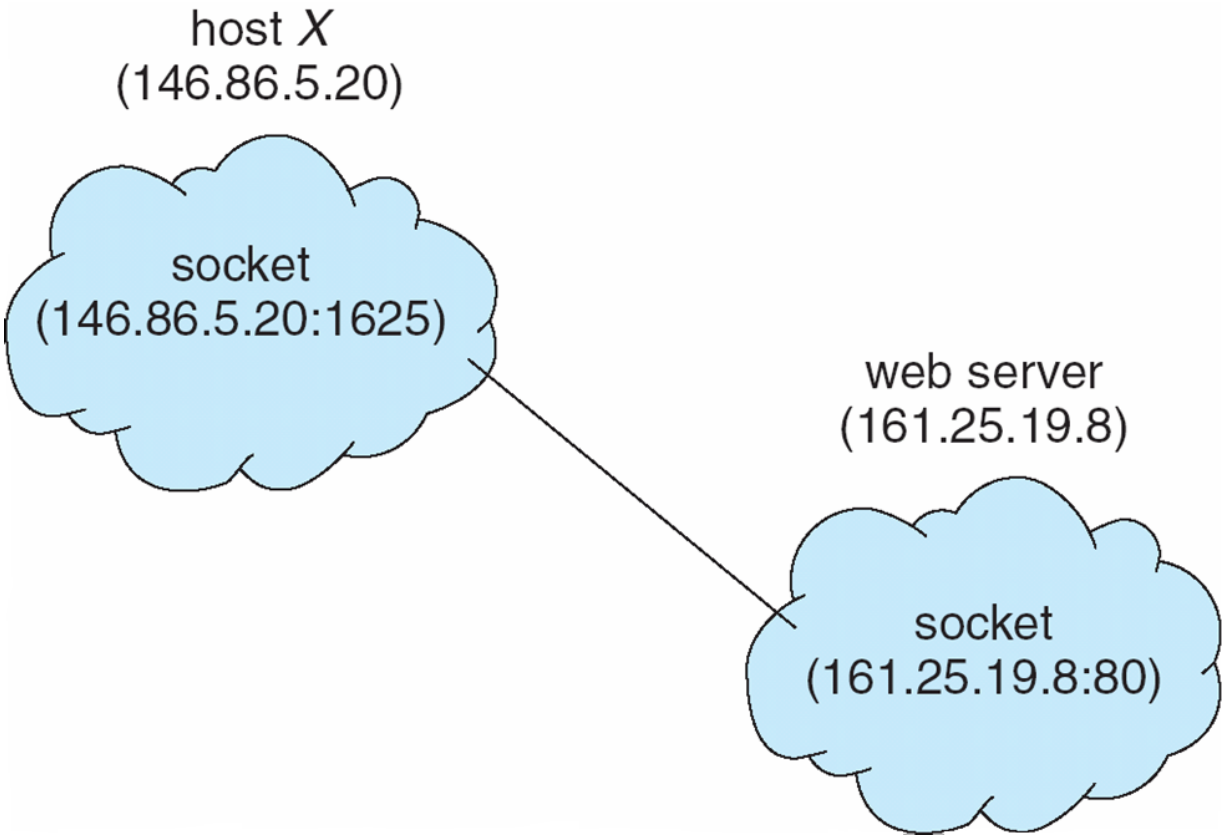
  - Soket**161.25.19.8:1625**mengacu pada pelabuhan**1625**pada tuan rumah **161.25.19.8**

- Komunikasi terdiri antara sepasang soket

  - Semua port di bawah 1024 adalah***terkenal***, digunakan untuk layanan standar

  - Alamat IP khusus 127.0.0.1 (**loopback**) untuk merujuk ke sistem tempat proses sedang berjalan

Komunikasi Soket



Pemrograman soket

Dua jenis soket untuk dua layanan transportasi:

- *UDP:* datagram yang tidak dapat diandalkan
- *TCP:* andal, berorientasi aliran byte

Contoh Aplikasi:

1. Klien membaca sebaris karakter (data) dari keyboardnya dan mengirimkan data tersebut ke server.
2. Server menerima data dan mengubah karakter menjadi huruf besar. Server
3. mengirimkan data yang dimodifikasi ke klien.
4. Klien menerima data yang dimodifikasi dan menampilkan garis di layarnya.

Pemrograman soket *dengan UDP*

UDP: tidak ada "koneksi" antara klien & server

- tidak ada jabat tangan sebelum mengirim data
- pengirim secara eksplisit melampirkan alamat tujuan IP dan port # ke setiap paket
- rcvr mengekstrak alamat IP pengirim dan port # dari paket yang diterima

UDP: data yang dikirimkan mungkin hilang atau diterima tidak sesuai pesanan

Sudut pandang aplikasi:

- UDP menyediakan *tidak bisa diandalkan* transfer kelompok byte (“datagrams”) antara klien dan server

Interaksi soket klien/server: UDP

server (berjalan di serverIP)

buat soket, port = x:
serverSocket =
soket(AF_INET,SOCK_DGRAM)

membaca datagram dari
serverSockeT

tulis balasan
serverSocket
menentukan
alamat klien,
nomor pelabuhan

klien

buat soket:
clientSocket =
soket(AF_INET,SOCK_DGRAM)

Buat datagram dengan IP server dan port=x;
mengirim datagram melalui
clientSocket

membaca datagram dari
clientSocket

menutup
clientSockeT

Contoh aplikasi: klien UDP

UDPClient Python

termasuk perpustakaan soket Python → dari impor soket *

serverName = 'nama host'

serverPort = 12000

buat soket UDP untuk server → clientSocket = soket(socket.AF_INET,
soket.SOCK_DGRAM)

dapatkan keyboard pengguna memasukkan → pesan = raw_input('Masukkan kalimat dengan huruf kecil:')

Lampirkan nama server, port ke pesan; kirim ke soket → clientSocket.sendto(pesan,(serverName, serverPort))

baca karakter balasan dari soket ke dalam string → pesan yang dimodifikasi, alamatserver =
clientSocket.recvfrom(2048)

cetak string yang diterima dan tutup soket → cetak pesan yang dimodifikasi

clientSocket.close()

Contoh aplikasi: server UDP

Server UDPS Python

```
dari impor soket *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
cetak "Server siap menerima"
sementara 1:
    pesan, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

buat soket UDP

ikat soket ke nomor port lokal 12000

lingkaran selamanya

Baca dari soket UDP ke dalam pesan, dapatkan alamat klien (IP dan port klien)

kirim string huruf besar kembali ke klien ini

Pemrograman soket *dengan TCP*

klien harus menghubungi server

- proses server harus dijalankan terlebih dahulu
- server harus membuat soket (pintu) yang menyambut kontak klien

klien menghubungi server dengan:

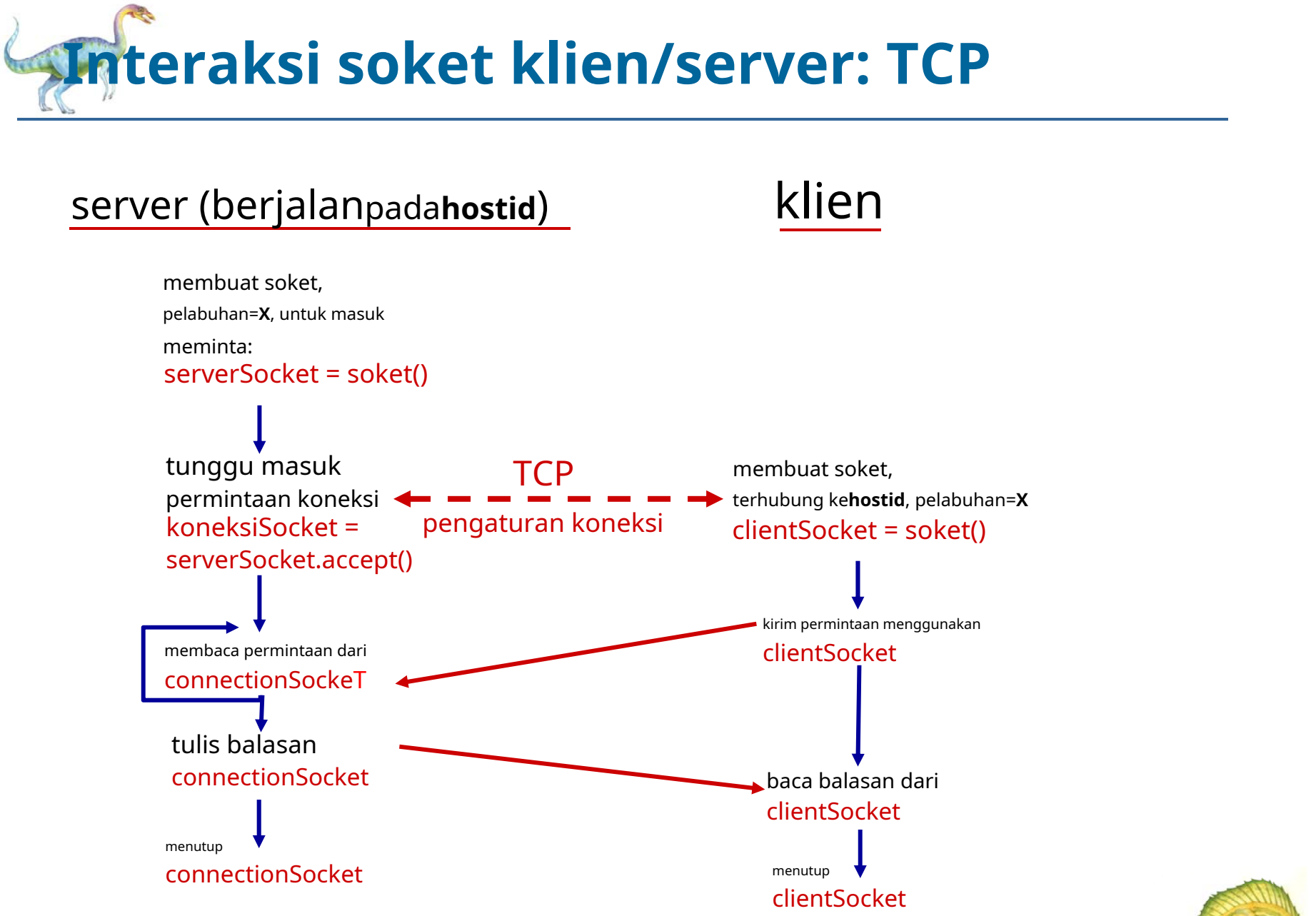
- Membuat soket TCP, menentukan alamat IP, nomor port proses server
- *ketika klien membuat soket:* klien TCP menetapkan koneksi ke server TCP

- saat dihubungi oleh klien, *server TCP membuat soket baru* untuk proses server untuk berkomunikasi dengan klien tertentu
  - memungkinkan server untuk berbicara dengan banyak klien
  - nomor port sumber yang digunakan untuk membedakan klien (selengkapnya di Bab 3)

sudut pandang aplikasi:

TCP menyediakan transfer byte-stream ("pipa") yang andal dan berurutan antara klien dan server

Interaksi soket klien/server: TCP



Contoh aplikasi: Klien TCP

Klien TCP Python

```
dari soket impor *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
kalimat = raw_input('Masukkan kalimat huruf kecil:')
clientSocket.send(kalimat)
modifiedSentence = clientSocket.recv(1024)
cetak 'Dari Server:', modifiedSentence
clientSocket.close()
```

crmakan soket TCP untuk Server, port jarak jauh 12000

No perlu melampirkan server Name, port

Contoh aplikasi: server TCP

Server TCP Python

```
 dari impor soket *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.mendengarkan(1)
print 'Server siap menerima'
sedangkan 1:
    connectionSocket, addr = serverSocket.accept()

    kalimat = connectionSocket.recv(1024)
    kapitalisasiSentence = kalimat.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

buat penyambutan TCP
stopkontak

server mulai mendengarkan permintaan TCP yang masuk

lingkaran selamanya

server menunggu menerima () untuk permintaan masuk, soket baru dibuat saat kembali

baca byte dari soket (tetapi bukan alamat seperti di UDP)

dekat koneksi ke klien ini (tapi *bukan* soket penyambutan)

Akhir Bab 3