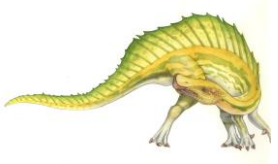
Operacinės sistemos

3. Procesų valdymas



3 skyriaus temos

- Proceso konceptas
- Procesų tvarkaraščių sudarymas
- Operacijos su procesais
- Tarpprocesinė komunikacija (IPC)
- IPC sistemų pavyzdžiai
- Komunikacija kliento-serverio sistemose
- Gijų apžvalga
- Keleto branduolių programavimas
- Multithreading modeliai
- Gijų bibliotekos
- Numanomas (angl. Implicit) gijų sudarymas (angl. Threading)
- Gijų sudarymo problemos
- OS pavyzdžiai

Procesų valdymo potemės tikslai

- Supažindinti su proceso sąvoka - vykdoma programa, kuri sudaro visų skaičiavimų pagrindą.
- Apibūdinti įvairias procesų savybes, įskaitant tvarkaraščių sudarymą, sukūrimą ir nutraukimą bei komunikaciją.
- Išnagrinėti tarpprocesinę komunikaciją, taikant bendrą atmintį ir žinučių perdavimą.
- Išnagrinėti komunikaciją kliento-serverio sistemose.

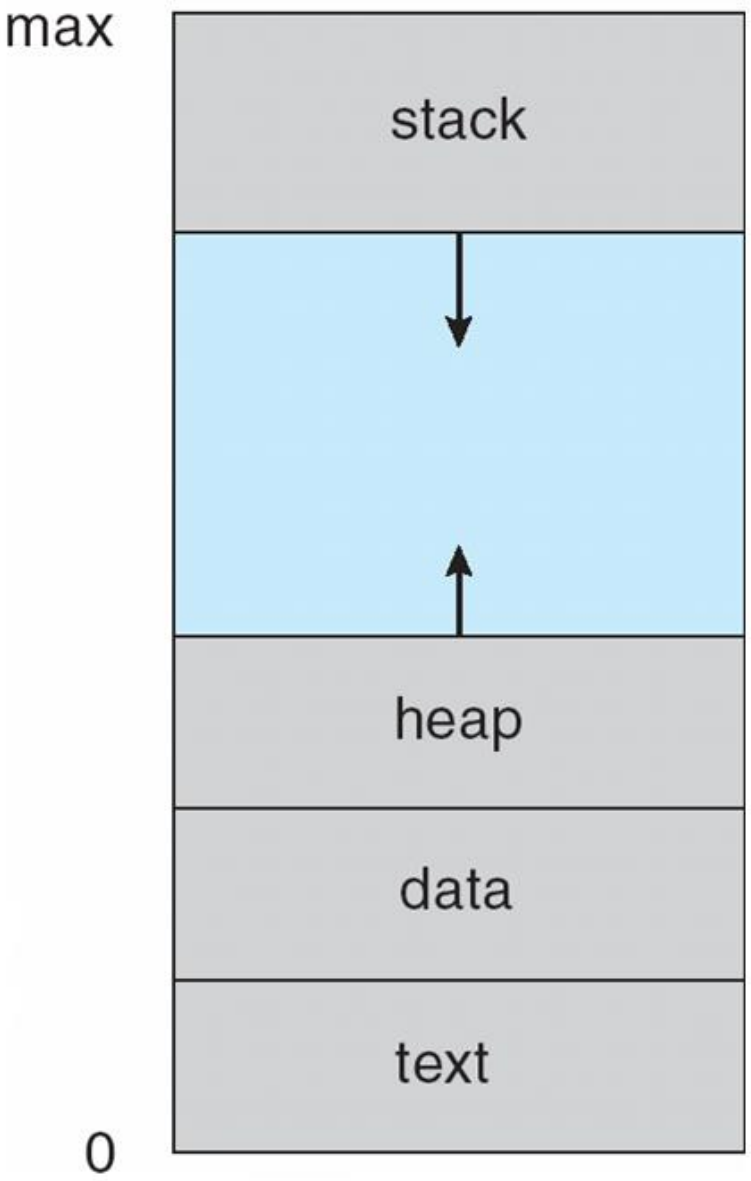
Proceso konceptas

- OS vykdo įvairias programas:
  - *Batch* sistemos – ***jobs.***
  - Bendro laiko sistemos (angl. *Time-shared systems*) – **vartotojo programas** (angl. *user programs*) ar **užduotis** (angl. *tasks*)
- Paskaitose terminas ***job*** ir ***process*** vartojamas, kaip sinonimai.
- **Procesas (*Process*)** – programa vykdymo metu; procesas turi būti vykdomas nuosekliai.
- Keletas dalių:
  - Programos kodas, dar vadinamas teksto sekcija (angl. ***text section***)
  - Esama veikla įskaitant programos skaitiklį (angl. ***program counter***), procesoriaus registerus.
  - **Stekas (angl. *Stack*)** turintis laikinus duomenis.
    - Funkcijų parameterus, grąžinamus adresus, vietinius kintamuosius
  - **Duomenų sekcija (*Data section*)**, turinti globalius kintamuosius (angl. global variables).
  - **Aibės (angl. Heap)** turinį, dinamiškai priskirtą atmintį vykdymo metu.

Proceso konceptas (tęs.)

- Programa yra ***pasyvi (passive)*** esybė, saugoma diske (paleidžiamasis failas - ***executable file***), procesas yra ***aktyvus (active)***
  - Programa tampa procesu, kai paleidžiamasis failas yra užkraunamas į atmintį.
- Programos paleidimas pradedamas per GUI, komandinę eilutę ir kt. būdais.
- Viena programa gali būti keletas procesų.

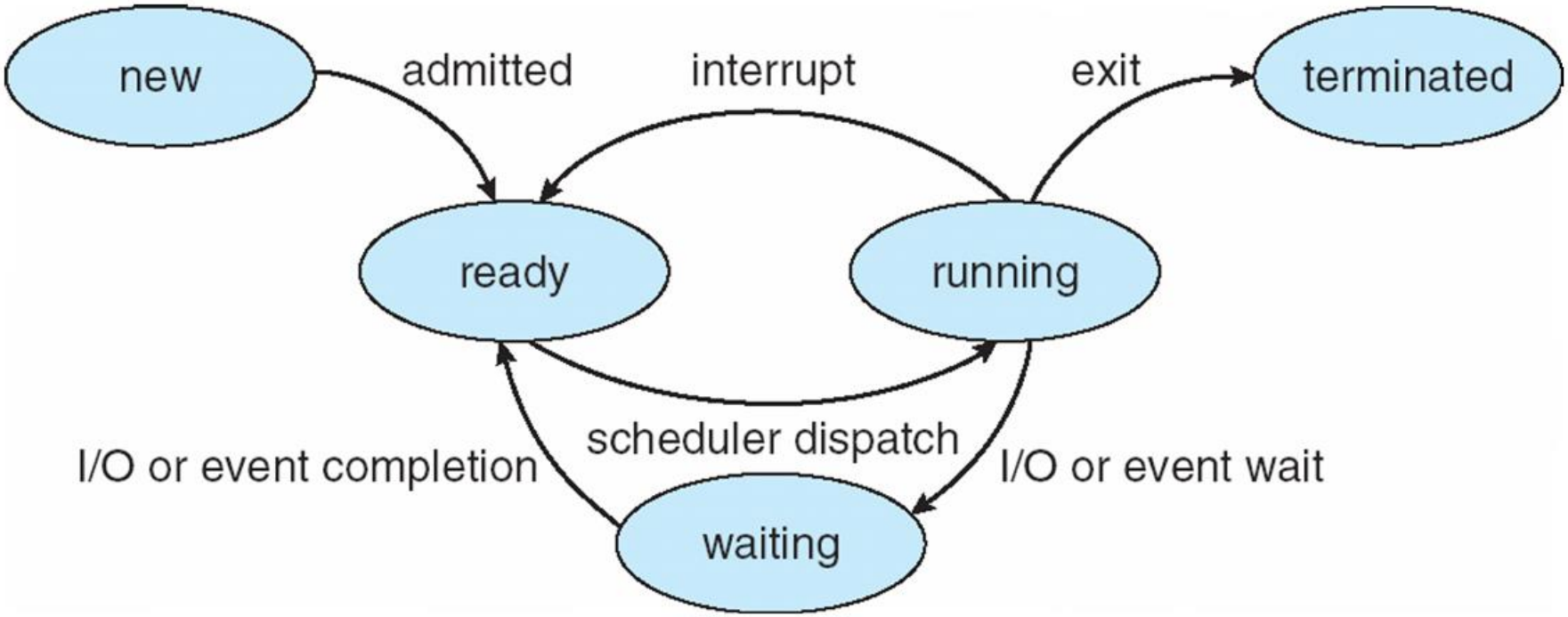
Procesas atmintyje



Proceso būsena

- Kai procesas yra vykdomas, keičiasi jo būsena (angl. **state**)
  - **Naujas (new)**: procesas pradedamas kurti
  - **Vykdomas (running)**: instrukcijos yra vykdomos
  - **Laukia (waiting)**: procesas laukia tam tikro įvykio
  - **Pasiruošęs (ready)**: procesas laukia, kol bus priskirtas procesoriui
  - **Nutrauktas (terminated)**: procesas užbaigęs veikimą

Proceso būsenos diagrama

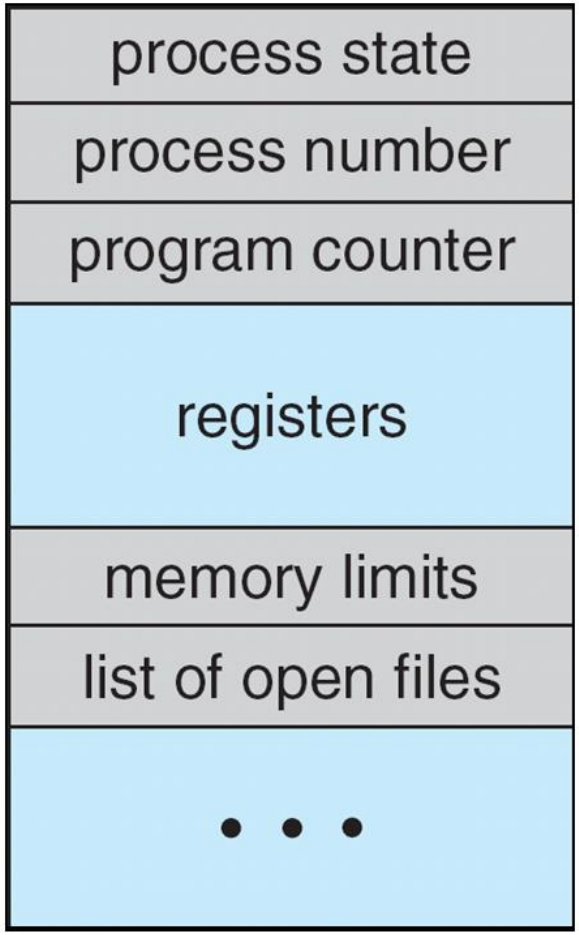


Proceso valdymo blokas (angl. *Process Control Block* (PCB))

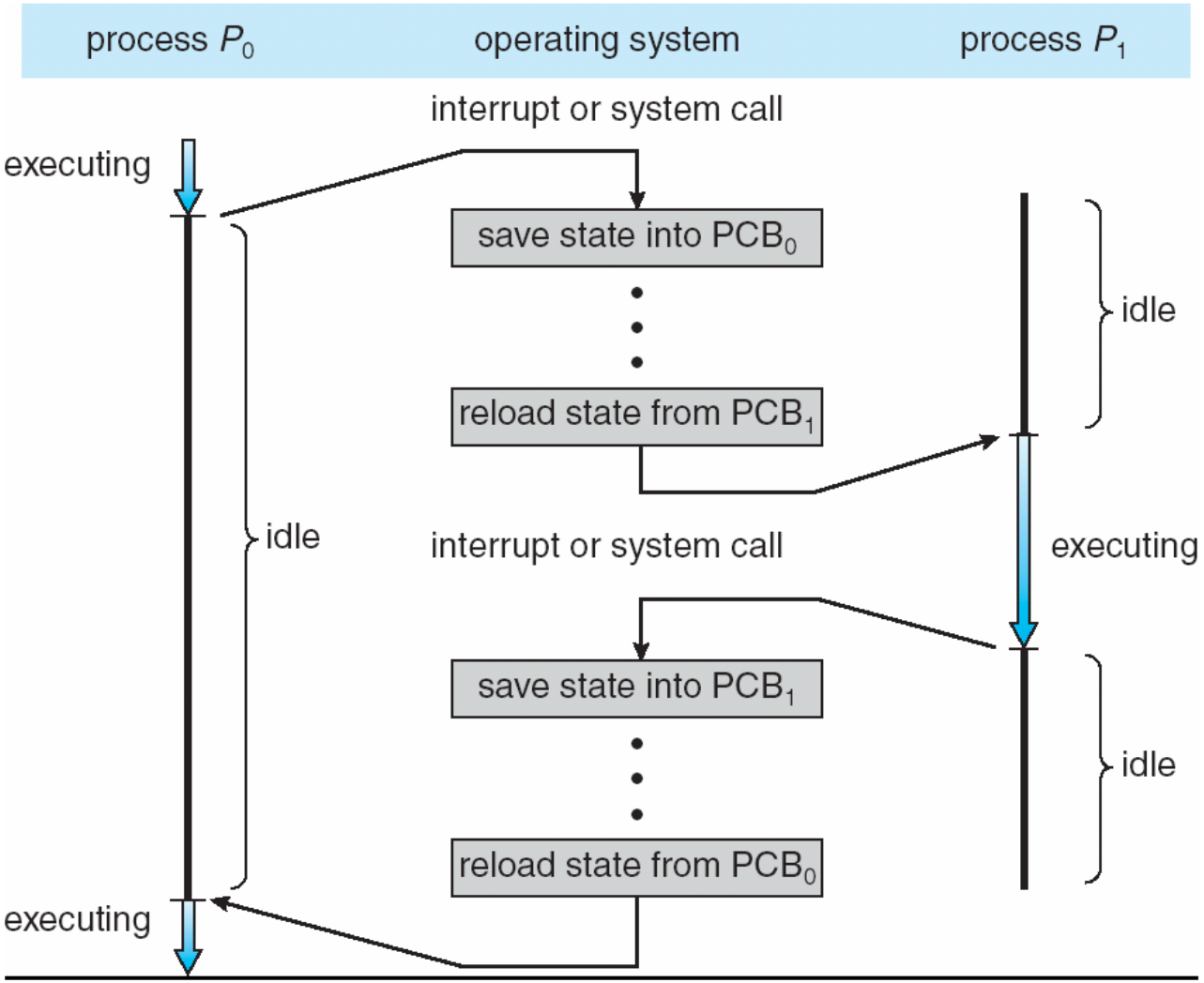
Informacija susieta su kiekvienu procesu

(taip pat vadinamas užduoties valdymo bloku (angl. ***task control block***)

- Proceso būsena – vykdymas, laukimas, kt.
- Programos skaitiklis – sekančios instrukcijos vieta.
- CPU registrai – CPU registrų turinys.
- CPU tvarkaraščių informacija - prioritetai, tvarkaraščių eilių rodyklės.
- Atminties valdymo informacija – atmintis paskirta procesui.
- Apskaitos informacija – CPU panaudojimas, laikrodžio informacija, laiko limitai.
- I/O būsenos informacija – I/O įrenginiai priskirti procesui, atidarytų failų sąrašas.



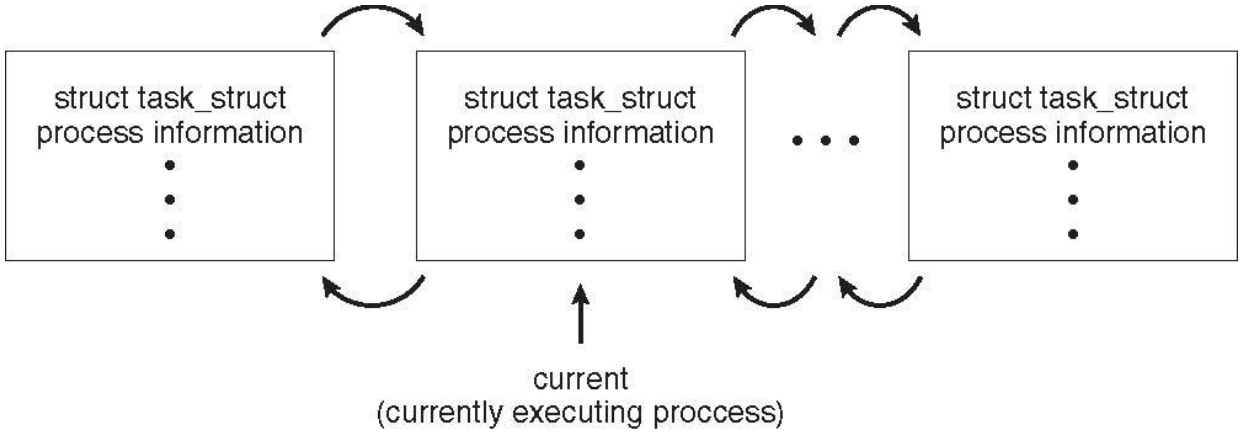
CPU perjungimas nuo vieno proceso prie kito proceso

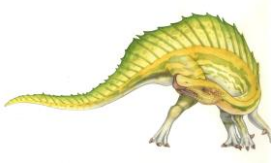


Procesų atvaizdavimas Linux

Atvaizduojama C struktūros `task_struct`

```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```

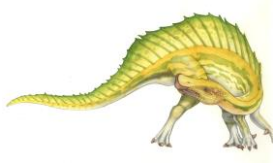




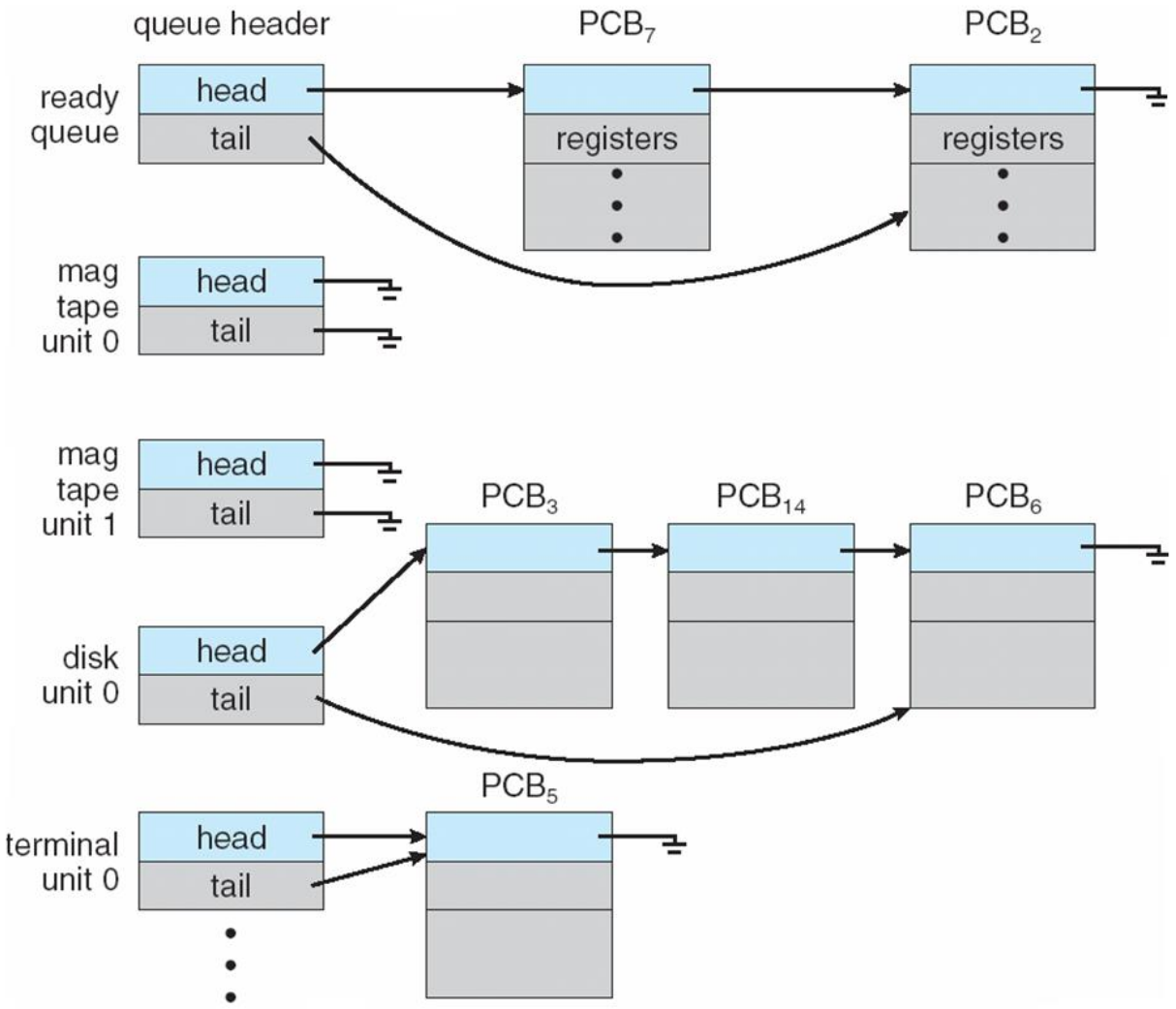
Procesų tvarkaraščių sudarymas

(angl. Process Scheduling)

- Maksimizuojamas CPU panaudojimas, greitas persijungimas tarp procesų CPU laiko dalijumui.
- **Procesų tvarkaraščių sudarytojas (*Process scheduler*)** iš prieinamų procesų pasirenka tą, kuris bus vykdomas sekantis.
- Yra išlaikomos procesų tvarkaraščių eilės (***scheduling queues***).
  - ***Job queue*** – visų sistemos procesų aibė.
  - ***Ready queue*** – visų procesų, esančių atmintyje ir pasiruošusių vykdymui aibė.
  - ***Device queues*** – aibė procesų, laukiančių I/O įrenginių.
  - Procesai migruoja tarp įvairių eilių.

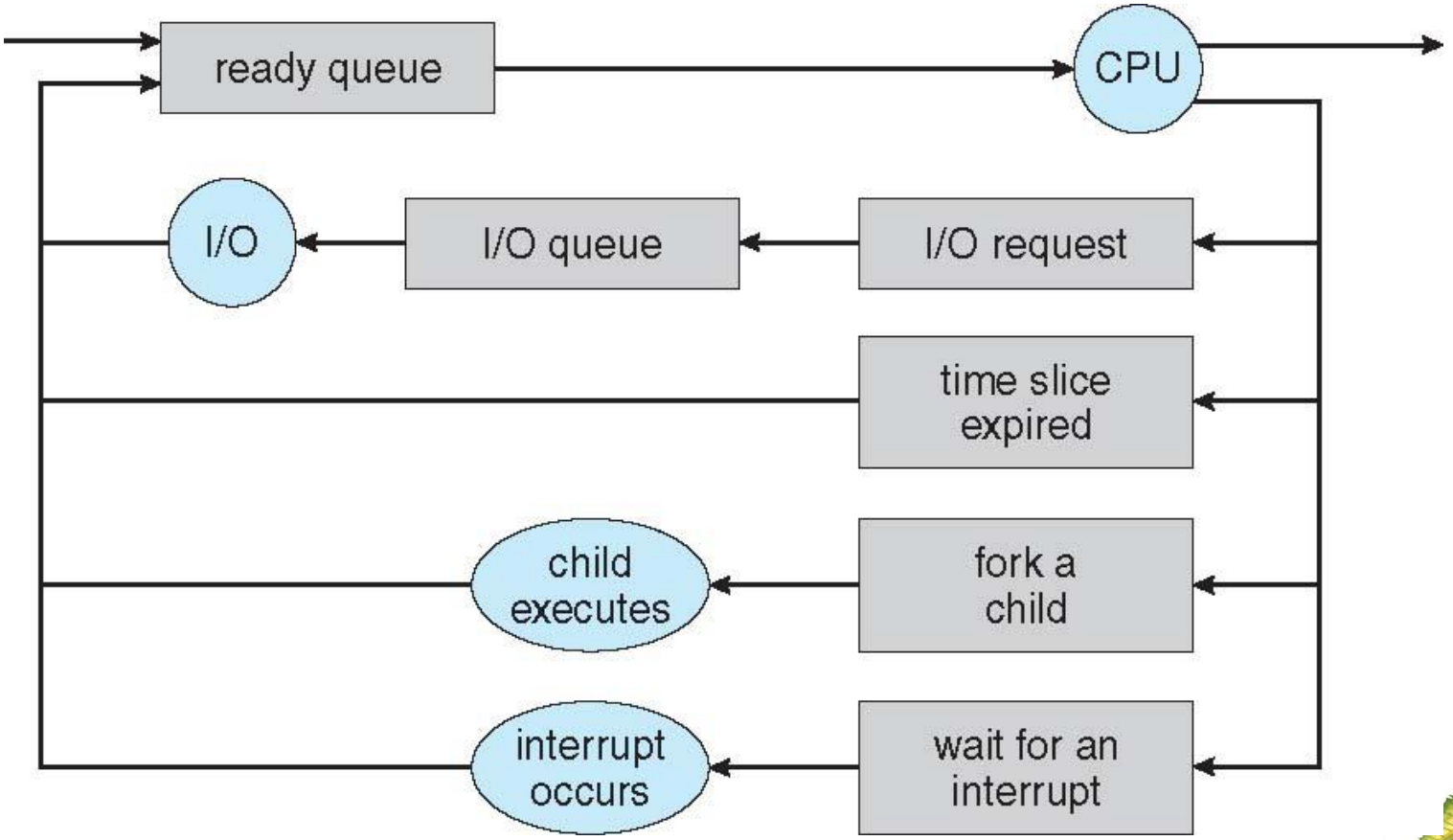


Ready Queue ir įvairios I/O įrenginių eilės



Procesų tvarkaraščių atvaizdavimas

- **Tvarkaraščių diagrama (Queueing diagram)** atvaizduojanti eiles, resursus ir srautus.

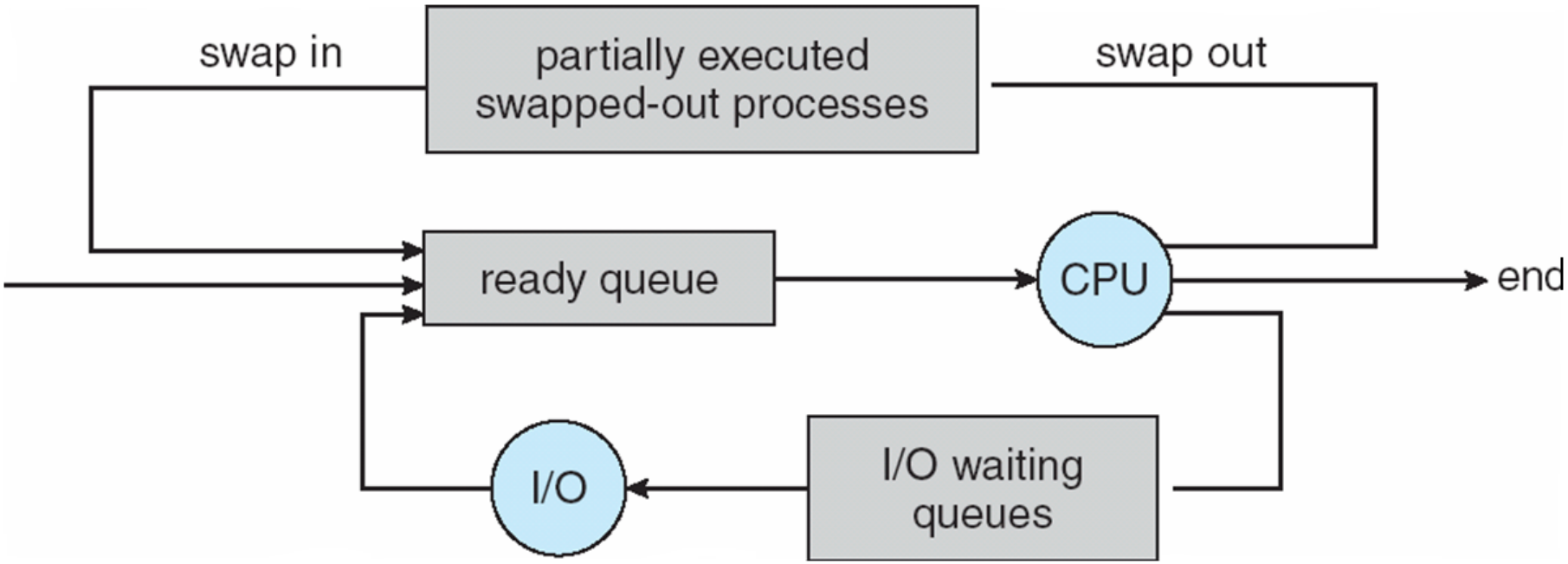


Tvarkaraščių sudarytojai

- **Trumpo laiko tvarkaraščių sudarytojai (*Short-term scheduler*)** (ar ***CPU scheduler***) – parenka kuris procesas turi būti paleistas sekantis ir priskiria CPU.
  - Kartais tai yra vienintelis tvarkaraščių sudarytojas sistemoje.
  - Trumpo laiko tvarkaraščių sudarytojas yra iškviečiamas dažnai (milisekundės) ⇒ (privalo būti greitas).
- **Ilgo laiko tvarkaraščių sudarytojai (*Long-term scheduler*)** (ar ***job scheduler***) – paranka kurie procesai turi būti įkelti į pasiruošusių eilę.
  - Ilgo laiko tvarkaraščių sudarytojai yra iškviečiami retai (sekundės, minutės) ⇒ (gali būti lėti)
  - Ilgo laiko tvarkaraščių sudarytojai valdo multiprogramavimo laipsnį (angl. ***degree of multiprogramming***).
- Procesai gali būti apibūdinami, kaip:
  - **I/O susieti procesai (*I/O-bound process*)** – daugiau laiko praleidžia vykdydami I/O operacijas, negu skaičiavimus. Daug trumpų CPU spends more time doing I/O than computations, many short CPU pkiūpsnių.
  - **CPU susieti procesai (*CPU-bound process*)** – daugiau laiko praleidžia atlikdami skaičiavimus, mažai labai ilgų CPU pliūpsnių.
- Ilgo laiko tvarkaraščių sudarytojai siekia gero procesų mišinio (***process mix)***

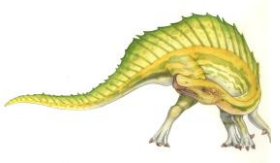
Priedas: vidutinės trukmės tvarkaraščių sudarymas (Medium Term Scheduling)

- **Vidutinės trukmės tvarkaraščių sudarymas** gali būti pridedamas, jei multiprogramavimo laipsnis turi būti sumažintas
  - Reikia pašalinti procesus iš atminties, išsaugoti juos diske, sugrąžinti iš disko ir tęsti vykdymą - **swapping**



Daugiaprogramis režimas (Multitasking) mobiliose sistemose

- Kai kurios mobiliosios sistemos (pvz., pirmosios iOS verisjos) leido veikti tik vienam procesui vienu metu, kiti buvo suspenduojami.
- Dėl riboto ekrano ploto, iOS vartotojo sąsaja yra apribota:
  - Vienas **priekinis** procesas valdomas per vartotojo sąsają.
  - Keletas **foninių** procesų yra atmintyje vykdomi, tačiau ne ekrane (su apribojimais).
  - Apribojimus apima viena trumpa užduotis – įvykių pranešimai, specifinės ilgai vykstančios užduotis, pvz. muzikos grojimas.
- Android vykdomi priekiniai ir foniniai procesai su mažiau apribojimų:
  - Foniniai procesai naudoja servisus užduotims atlikti.
  - Servisai gali veikti toliau, nors foniniai procesai ir sustabdomi.
  - Servisai neturi vartotojo sąsajos, naudoja mažai atminties.



Konteksto perjungimas

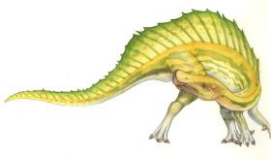
- Kai CPU persijungia prie kito proceso, sistema turi išsaugoti **senojo proceso būseną** ir paleisti išsaugotą būseną naujam procesui per konteksto perjungimą (angl. **context switch**)
- Proceso **kontekstas (Context)** yra atvaizduojamas PCB - proceso valdymo bloke (process control block)
- Konteksto perjungimo laikas yra papildomas, sistema neatlieka naudingo darbo persijungdama.
  - Kuo sudėtingesnė OS ir PCB➔ tuo ilgiau vyksta konteksto perjungimas.
- Laikas yra priklausomas nuo techninės įrangos palaikymo.
  - Kai kurti techninė įranga turi keletą registrų rinkinių vienam CPU ➔ keletas kontekstų paleisti vienu metu.

Operacijos su procesais

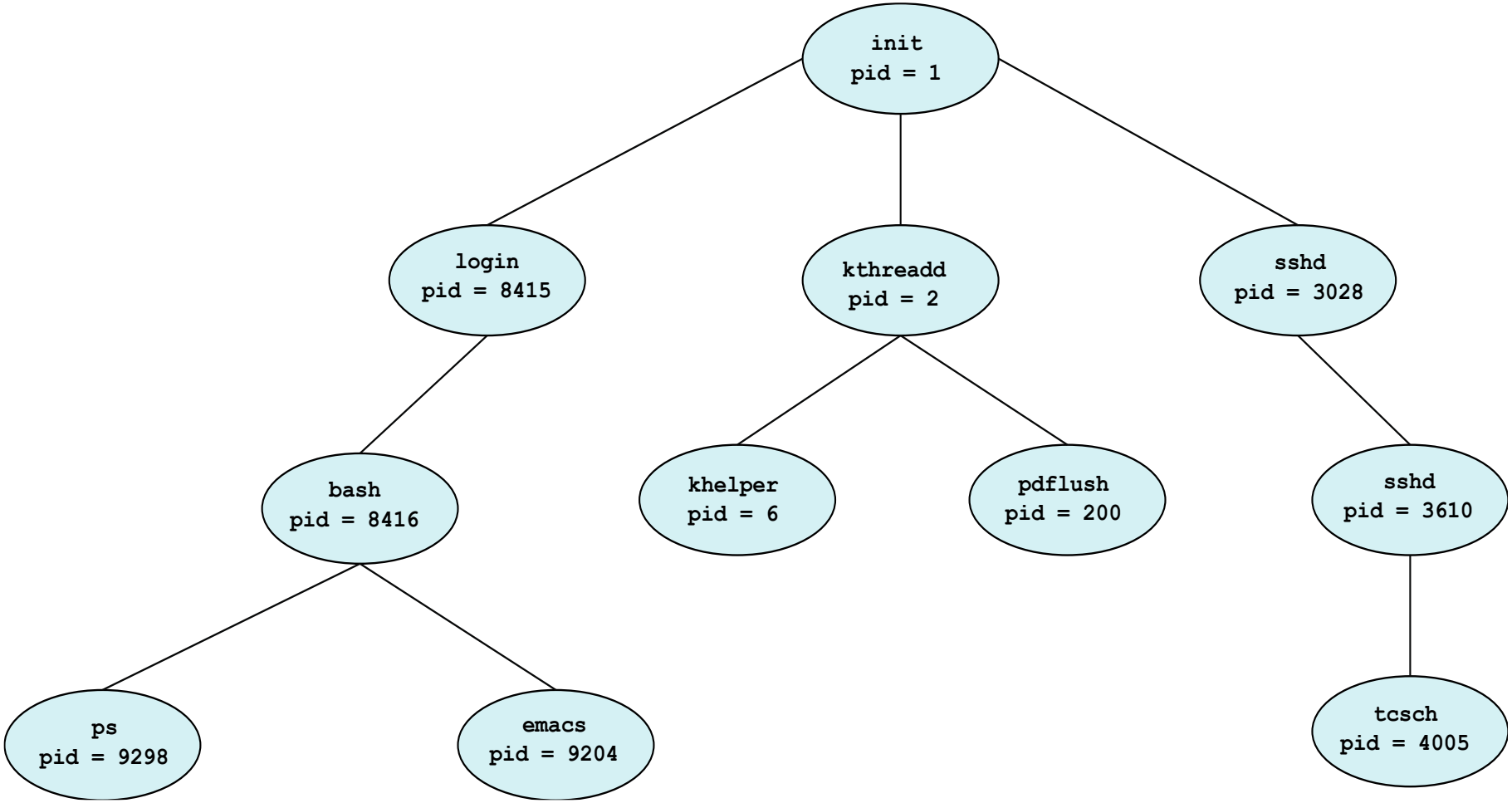
- Sistema turi užtikrinti mechanizmus:
  - Procesų sukūrimui,
  - Procesų nutraukimui

Procesų sukūrimas

- **Tėvo (Parent)** procesas sukuria **vaikų** (**children)** procesus, kurie savo ruožtu sukuria dar kitus procesus, taip suformuojamas **procesų medis** (tree of processes).
- Bendrai procesai yra identifikuojami ir valdomi, pagal **procesų identifikatorių** (process identifier (pid)).
- Resursų dalijimosi opcijos:
  - Tėvo ir vaiko procesai dalijasi visais resursais.
  - Vaikas naudojasi dalimi tėvo resursų.
  - Tėvas ir vaikas nesidalija resursais.
- Vykdymo opcijos
  - Tėvo ir vaiko procesai vykdomas lygiagrečiai.
  - Tėvo procesas laukia, kol vaiko procesas bus sustabdytas.

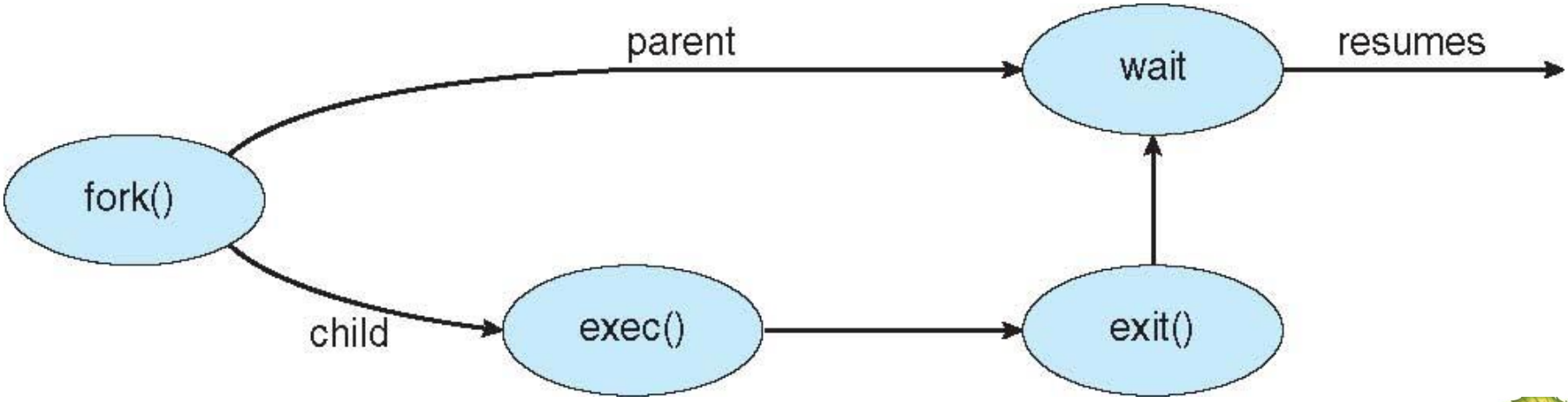


Linux procesų medis



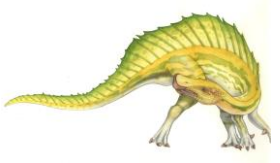
Procesų sukūrimas (tęs.)

- Adresų erdvė
  - Vaiko procesas dubliuoja tėvo.
  - Vaiko procesas turi programą paleistą joje.
- UNIX pavyzdžiai
  - **`fork()`** sisteminis iškvietimas sukuria naują procesą.
  - **`exec()`** sisteminis iškvietimas panaudotas po **`fork()`**, kad pakeisti proceso atminties erdvę nauja programa



C programa šakojanti (Forking) atskirus procesus

```
#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;
}
```

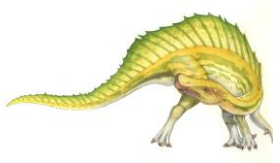


Atskiro proceso kūrimas per Windows API

```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
     "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
     NULL, /* don't inherit process handle */
     NULL, /* don't inherit thread handle */
     FALSE, /* disable handle inheritance */
     0, /* no creation flags */
     NULL, /* use parent's environment block */
     NULL, /* use parent's existing directory */
     &si,
     &pi))
    {
      fprintf(stderr, "Create Process Failed");
      return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Proceso nutraukimas

- Procesas įvykdo paskutinį sakinį ir paprašo OS jį ištrinti, panaudodamas **`exit()`** sisteminį iškvietimą.
  - Grąžina būsenos duomenis iš vaiko tėvui (per **`wait()`**)
  - Proceso resursai yra atlaisvinai OS.
- Tėvas gali nutraukti vaiko proceso veikimą, naudodamas **`abort()`** sisteminį iškvietimą. Kai kurios to priežastys:
  - Vaikas pereikvojo priskirtus resursus.
  - Užduotis priskirta vaikui yra nebereikalinga.
  - Tėvo procesas išjungiamas, o OS neleidžia vaikui tęsti darbo, jei tėvo procesas nutraukiamas.



Proceso nutraukimas

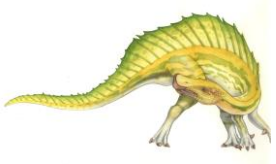
- Kai kurios OS neleidžia egzistuoti vaiko procesams, jei tėvo procesas yra nutraukiamas.
  - **Kaskadinis nutraukimas (cascading termination).** Visi vaikų, jų vaikų ir t.t. procesai yra nutraukiami.
  - Nutraukimas inicijuojamas OS.
- Tėvo procesas gali laukti, kol bus nutraukti vaikų procesai, naudodamas **`wait()`** sisteminį iškvietimą. Iškvietimas grąžina būsenos informaciją ir nutraukto proceso pid.

```
pid = wait(&status);
```

- Jei nėra laukiančių tėvų (nebuvo **`wait()`**) iškvietimo, tai procesas yra **zombis** (**zombie**).
- Jei tėvo procesas buvo nutrauktas be **`wait`** iškvietimo, tai procesas yra **našlaitis** (**orphan**).

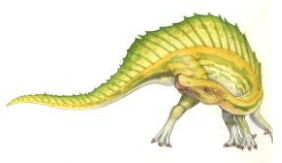
Keleto procesų architektūrą - Chrome Browser

- Daugelis interneto naršyklių veikdavo, kaip vienas procesas (kai kurios iki šiol veikia).
  - Jei vienas web puslapis sukelia problemų, visa naršyklė pakimba arba nulūžta.
- Google Chrome Browser yra daugiaprocesė su 3 skirtingais procesų tipais:
  - **Browser** procesas valdo vartotojo sąsają, diskus ir tinklo I/O
  - **Renderer** procesas generuoja web puslapius, apdoroja HTML, Javascript. Po naują renderer yra sukuriama kiekvienam atidarytam puslapiui
    - Veikia **sandbox** režimu, apribojant disko ir tinkloI/O, minimizuojant saugumo spragų poveikį.
  - **Plug-in** procesas kiekvienam plug-in tipui.



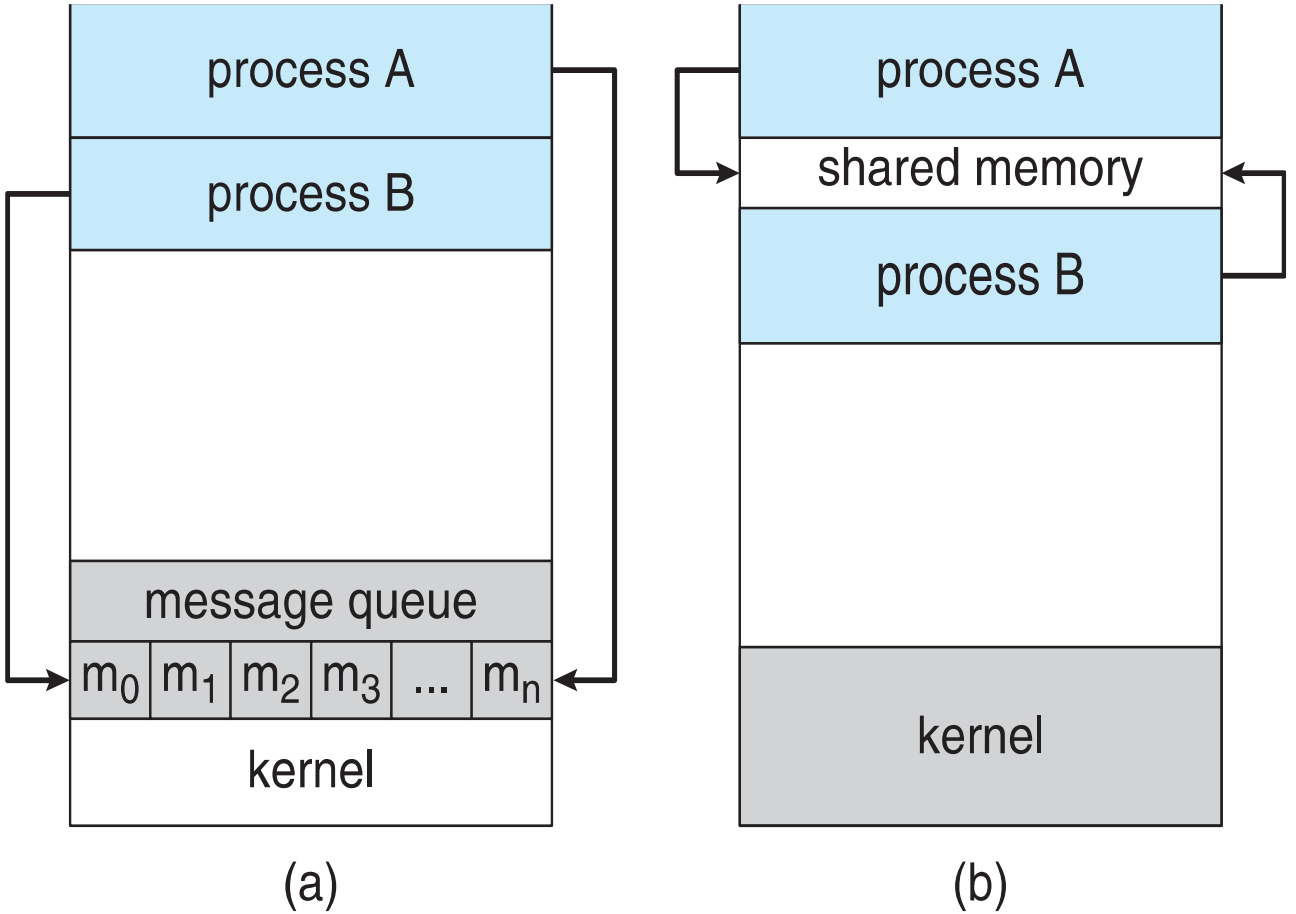
Tarpprocesinė komunikacija

- Procesai sistemoje gali būti ***nepriklausomi*** (***independent***) arba ***kooperuojantys*** (***cooperating***).
- Kooperuojantys procesai gali paveikti arba būti paveikiami kitų procesų, įskaitant bendrus duomenis.
- Procesų kooperavimo priežastys:
  - Informacijos bendrinimas
  - Skaičiavimų pagreitinimas
  - Moduliškumas
  - Patogumas
- Kooperuojantiems procesams reikalinga **tarpprocesinė komunikacija** (angl. interprocess communication (**IPC**))
- Du IPC modeliai:
  - **Bendroji atmintis** (angl. Shared memory)
  - **Žinučių perdavimas** (angl. Message passing)



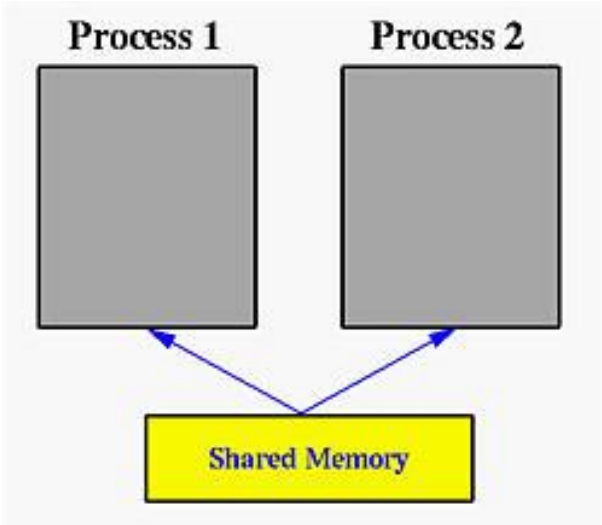
Komunikacijos modeliai

(a) Žinučių perdavimas. (b) bendroji atmintis.



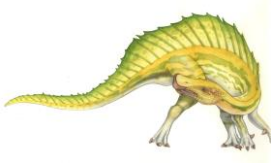
Tarpprocesinė komunikacija – bendroji atmintis

- Atminties dalis bendrinama tarp norinčių komunikuoti procesų.
- Komunikacija yra pavaldi vartotojo procesams, o ne OS.
- Didžiausios problemos yra užtikrinti mechanizmus, kurie leistų vartotojo procesams sinchronizuoti jų veiksmus, kai pasiekiama bendroji atmintis.
- Sinchronizavimas aptariamas plačiau sekančiose temose.



Tarpprocesinė komunikacija – žinučių perdavimas

- Tai mechanizmas procesams komunikuoti ir sinchronizuoti veiksmus.
- Žinučių sistema – procesai komunikuoja vieni su kitais, nenaudojant bendrų kintamųjų.
- IPC priemonės suteikia dvi operacijas:
  - **`send`**(*message*)
  - **`receive`**(*message*)
- Žinutės dydis gali būti fiksuotas arba kintantis.

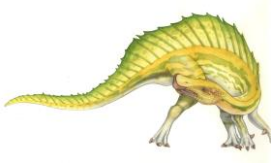


Žinučių perdavimas (tęs.)

- Jei procesai *P* ir *Q* nori komunikuoti, jiems reikia:
  - Užmegzti ***ryšio liniją*** (angl. communication link) tarp jų.
  - Apsikeisti žinutėmis per send/receive
- Įgyvendinimo problemos:
  - Kiek ryšio linijų yra užmegzta?
  - Ar gali ryšio linija būti asocijuota su daugiau nei dviem procesais?
  - Kiek ryšio linijų gali būti tarp kiekvienos komunikuojančių procesų poros?
  - Kokia linijos talpa?
  - Ar žinučių dydis yra fiksuotas ar kintantis?
  - Ar ryšio linija yra vienakryptė, ar dvikryptė?

Žinučių perdavimas (tęs.)

- Ryšio linijos įgyvendinimas
  - Fizinis:
    - Bendroji atmintis
    - Techninės įrangos magistralė
    - Tinklas
  - Loginė:
    - Tiesioginė ar netiesioginė
    - Sinchroninė ar asinchroninė
    - Automatinė ar aiškiai buferinė



Tiesioginė komunikacija

- Procesai turi aiškiai vieni kitus įvardinti:
  - **`send`** (*P, message*) – siunčiama žinutė procesui P.
  - **`receive`**(*Q, message*) – gaunama žinutė iš proceso Q.
- Ryšio linijos savybės:
  - Linijos sudaromos automatiškai.
  - Linija yra susieta su tiksliai viena komunikuojančių procesų pora.
  - Tarp kiekvienos poros yra lygiai viena linija.
  - Linija gali būti vienakryptė, tačiau dažniausiai būna dvikryptės.

Netiesioginė komunikacija

- Žinutės yra nukreipiamos ir gaunamos iš pašto dėžučių (angl. mailboxes) (dar žymima, kaip ports)
  - Kiekviena dėžutė turi unikalų id.
  - Procesai gali komunikuoti tik, jei jie dalijasi dėžute.
- Ryšio linijos savybės:
  - Ryšio linija sukuriama tik jei procesai turi bendrą dėžutę.
  - Linija gali būti susieta su daugeliu procesų.
  - Kiekviena procesų pora gali dalintis keletu ryšio linijų.
  - Linija gali būti vienakryptė arba dvikryptė.

Netiesioginė komunikacija

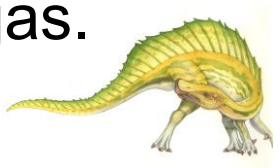
- Operacijos
  - Sukurti naują mailbox (port).
  - Siųsti ir priimti žinutes per mailbox.
  - Sunaikinti mailbox.
- Primityvai apibrėžiami kaip:

  `send`(*A, message*) – siųsti žinutę į mailbox A.

  `receive`(*A, message*) – gauti žinutę iš mailbox A.

Netiesioginė komunikacija

- Mailbox dalijimasis
  - P_1, P_2, ir P_3 dalijasi mailbox A.
  - P_1, siunčia; P_2 *ir* P_3 gauna
  - Kas gaus žinutę?
- Sprendimai:
  - Leisti linijai būti susietai su daugiausiai dviem procesais.
  - Leisti tik vienam procesui vienu metu vykdyti priėmimo operaciją.
  - Leisti sistemai savarankiškai parinkti gavėją. Siuntėjas yra informuojamas, kad buvo gavėjas.

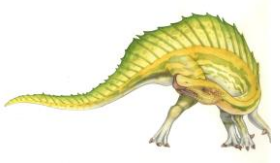


Sinchronizavimas

- Žinučių perdavimas gali būti blokuojantis arba ne blokuojantis.
- **Blokuojantis** yra laikomas **sinchroniniu.**
  - **Blokuojantis siuntimas** – siuntėjas yra blokuojamas, kol žinutė yra gaunama.
  - **Blokuojantis gavimas** – gavėjas yra blokuojamas, kol žinutė yra prieinama.
- **Neblokuojantis** yra laikomas **asinchroniniu.**
  - **Neblokuojantis siuntimas** siuntėjas siunčia žinutę ir tęsia darbą.
  - **Neblokuojantis gavimas** – gavėjas priima:
    - Tinkamą žinutę, arba
    - Null žinutę.
- Galimos skirtinos kombinacijos.
  - Jei tiek siuntimas, tiek gavimas yra blokuojantis, turime „pasimatymą“ (**rendezvous**).

Buferizavimas

- Žinučių eilė ryšio linijoje.
- Įgyvendinama trimis būdais:
  1. Nulinės talpos – ryšio linijoje žinutės nededamos į eilę. Siuntėjas turi laukti gavėjo (rendezvous)
  2. Ribota talpa – baigtinis žinučių skaičius *n.* Siuntėjas turi laukti, jei linija pilna.
  3. Neribota talpa – begalinis ilgis. Siuntėjas niekada nelaukia.



IPC sistemų pavyzdžiai - POSIX

- POSIX bendroji atmintis
  - Pirmiausia, procesas sukuria bendrosios atminties segmentą

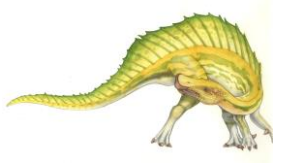
    ```
    shm_fd = shm_open(name, O CREAT | O RDWR, 0666);
    ```

  - Taip pat, naudojama egzistuojančio segmento bendrinimui
  - Nustatomas objekto dydis

    ```
    ftruncate(shm fd, 4096);
    ```

  - Dabar, procesai gali rašyti į bendrąją atmintį

    ```
    sprintf(shared memory, "Writing to shared memory");
    ```

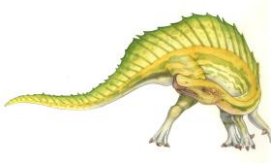


IPC sistemų pavyzdžiai - Mach

- Mach komunikacija yra grindžiama žinutėmis.
  - Net sisteminiai iškvietimai yra žinutės.
  - Kiekviena užduotis gauna dvi pašto dėžutes sukūrimo metu: Kernel ir Notify
  - Žinučių perdavimui reikalingi tik trys sisteminiai iškvietimai:

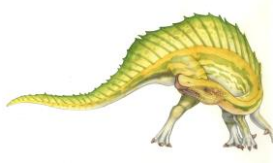
    `msg_send(), msg_receive(), msg_rpc()`
  - Pašto dėžutės reikalingos komunikacijai, sukuriamai per

    `port_allocate()`
  - Siuntimas ir priėmimas yra lankstus, pvz., galimi 4 variantai, jei dėžutė yra pilna:
    - Laukti Wait neribotą laiką,
    - Laukti tam tikrą milisekundžių laiką
    - Grįžti nedelsiant
    - Laikinai kešuoti žinutę.



IPC sistemų pavyzdžiai – Windows

- Žinučių perdavimas centralizuotas per **advanced local procedure call (LPC)** priemonę.
  - Veikia tik tarp procesų toje pačioje sistemoje.
  - Naudoja uostus (ports) (kaip pašto dėžutes) užmegzti ir išlaikyti komunikacijai.
  - Komunikacija veikia sekančiai:
    - Klientas nurodo subsistemos **jungties uostą** (angl. connection port) objekte.
    - Klientas išsiunčia ryšio prašymą.
    - Serveris sukuria du privačius komunikacijos uostus ir grąžina nuorodą vienam iš klientų..
    - Klientas ir serveris naudoja atitinkamą uosto rodyklę žinučių siuntimui ir atsakymams.



Lokalūs procedūrų iškvietimai Windows