Protsessihaldus

- Protsessi mõiste
- Protsesside planeerimine
- Operatsioonid protsessidega
- Protsesside koostöö
- Protsessidevaheline side
- Side klient-serversüsteemides

Paralleeltöötlus

- Opsüsteem võib täita mitmesuguseid töid:
  - Pakktööd (*batch jobs*) — täidetakse järjekorras ükshaaval
  - Aega jagavad tööd (*time-sharing tasks*)
- Ajajaotussüsteem tekitab näilise paralleeltöötluse isegi ühe protsessori korral
- Tänapäeval lasevad mitu protsessorit (tuuma) ka reaalselt mitmel tööl korraga käia
  - Iga protsessori peal on sellegipoolest ajajaotussüsteem

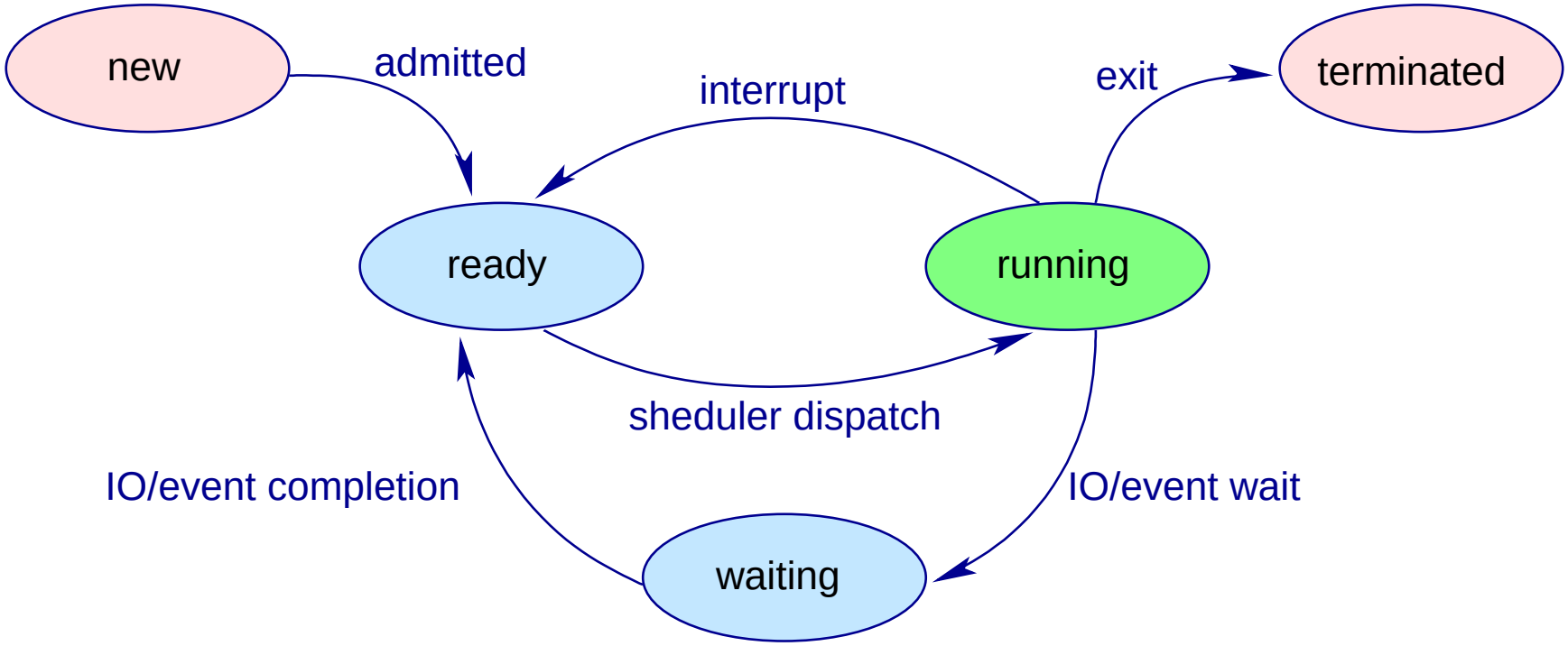
Protsessi mõiste

- Protsess — täitmisel olev programm
- Programmi täidetakse järjest (programm kui käsujada, retsept)
- Protsess koosneb:
  - Protsessori seisust (registrid)
    - Sealhulgas käsuloendurist (*program counter, instruction pointer*)
  - Mälu sisust:
    - Koodisektsioonist
    - Andmesektsioonist
    - Magasinist (*stack*)

Protsessi olek

- Täitmise käigus satub protsess järgmistesse olekutesse:
  - Uus (*new*) — protsess luuakse
  - Töös (*running*) — selle protsessi käske täidetakse
  - Ootel (*waiting*) — protsess ootab mingi sündmuse toimumist
  - Valmis (*ready*) — protsess ootab, et tema käske täitma hakataks
  - Lõpetatud (*terminated*) — protsessis olev programm on täidetud (edukalt või ebaedukalt)

Protsessi olekuskeem



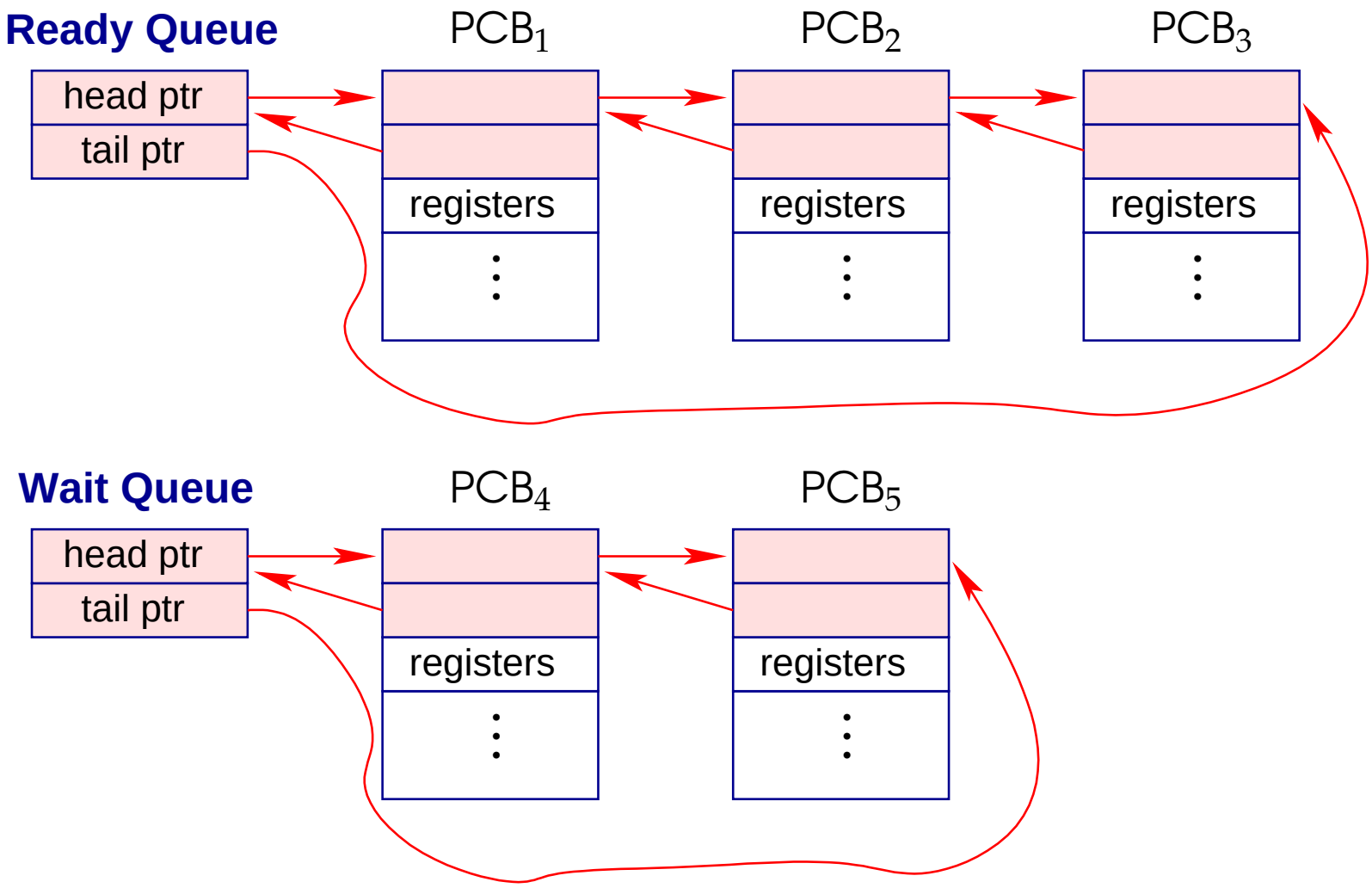
Protsessi andmed operatsioonisüsteemis

- Iga protsessiga on seotud protsessi juhtplokk (PCB — *Process Control Block*)
- Protsessi juhtploki sisu:
  - (Protsessi number)
  - Protsessi olek
  - Käsuloendur
  - Protsessori registrite seis
  - Protsessoriaja planeerimise info
  - Mäluhaldusinfo
  - Arvepidamisinfo
  - I/O staatus
  - Omanikuinfo
  - . . .

Planeerimisjärjekorrad

- Operatsioonisüsteem haldab kõigi protsesside juhtplokke erinevates järjekordades:
  - Tööde järjekord — kõik süsteemi protsessid
  - Töövalmis tööde järjekord — kõik protsessid, mis on mälus ning valmis täitmiseks
  - Seadmete järjekorrad — protsessid, mis ootavad antud seadme taga I/O lõppemist
- Protsesse liigutatakse vastavalt vajadusele järjekordade vahel
- Erinevate järjekordade jaoks võib opsüsteem kasutada erinevat halduspoliitikat
- *Load average* — töövalmis tööde järjekorra keskmine pikkus

Planeerimisjärjekorrad



Planeerijad

- Lühiajaline planeerija
  - Valmis protsesside hulgast sobiva tööks valimine
  - Töötab tihti, peab olema kiire
- Pikaajaline planeerija
  - Uute protsesside loomine
  - Hoolitseb multiprogrammeerimise astme eest
  - Käivitub harva, pole kiirusekriitiline
  - Õige protsessisegu hoidmine (I/O taga ootavad ja protsessori taga ootavad protsessid)
- Vahepealne planeerija — protsesside välja ja sisse saalimine (*swapping*)

Kontekstivahetus

- *Context switch*
- Ühelt protsessilt teisele lülitumisel tuleb salvestada vana protsessi olek ja laadida uue protsessi salvestatud olek
- See aeg on lisakulu, vähendab rakendustele jäävat protsessoriaega
- Kuluv aeg sõltub protsessori võimalustest

Protsesside loomine

- Vanemprotsessid võivad luua lapsprotsesse
- Iga lapsprotsess võib olla teistele protsessidele omakorda vanemaks
- Nii tekib protsesside puu
- Ressursside jagamine
  - Vanem ja laps jagavad kõiki ressursse
  - Laps saab alamhulga vanema ressurssidest
  - Vanem ja laps ei jaga mingeid ressursse
- Töötamine
  - Vanem ja laps töötavad paralleelselt
  - Vanem ootab, kuni laps töö lõpetab
- Eri mudelid: fork+exec, CreateProcess

Protsesside loomine: näide

```
main () {
    int parentID = getpid();
    char prgname[1024];
    fgets(prgname, 1024, stdin); /* input new program name */
    int cid = fork();
    if (cid == 0) { /* I'm the child process */
       execlp(prgname, prgname, 0); /* Load the program */
       perror("Can't execute program %s", prgname);
    } else {        /* I'm the parent process */
       waitpid(cid, 0, 0); /* Wait the child to terminate */
       printf("Program %s finished\n", prgname);
    }
}
```

Protsessi lõpetamine

- Protsess võib lõppeda vabatahtlikult (`exit`)
  - Vanemale antakse tagasi väljundinfo (`wait`)
  - Operatsioonisüsteem vabastab protsessiga seotud ressursid
- Sunniviisiliselt — vanemprotsess tapab (`kill`), opsüsteem tapab, vanemprotsess sureb ja opsüsteem tapab seetõttu

Protsesside lõpetamine: näide

```
main () {
    int cid = fork();
    if (cid == 0) { /* I'm the child process */
       sleep(5); /* I'll exit myself after 5 seconds */
       printf("Quitting child.\n");
       exit(0);
       printf("Oops! Shouldn't be here!\n");
    } else {        /* I'm the parent process */
       printf("Type any character to kill the child.\n");
       char c = getchar();
       if (!kill(cid, SIGKILL))
          printf("Killed the child.\n");
    }
}
```

Koostööd tegevad protsessid

- Sõltumatud protsessid ei saa teisi protsesse mõjutada ega teiste protsesside poolt mõjutatud saada
- Koostööd tegevad protsessid saavad teisi protsesse mõjutada ja/või teiste protsesside poolt mõjutatud saada
- Miks protsesside koostööd vaja on?
  - Info jagamine
  - Töö kiirendamine
  - Modulaarsus
  - Mugavus

Tootja-tarbija probleem

- Tootja — *producer*, tarbija — *consumer*
- Protsesside koostöö paradigma
- Tootjaprotsess toodab mingit infot ja tarbijaprotsess saab selle kätte ja kasutab
- Puhverdamine
  - Piiramata puhvriga — praktiliselt pole piire kahe protsessi vahelisel puhvril
  - Piiratud puhvriga — sidepuhvri(te)l on fikseeritud suurus

Näide piiratud puhvrist: jagatud mälu

```
#define BUFFER_SIZE 10
typedef struct {
. . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Töötab, aga kasutada saab kuni
`BUFFER_SIZE - 1` elementi

Piiratud puhver: tootja ja tarbija

- Tootja protsess

```
item nextProduced;
while (1) {
  while (((in + 1) % BUFFER_SIZE) == out)
  ; /* do nothing */
  buffer[in] = nextProduced;
  in = (in + 1) % BUFFER_SIZE;
}
```

Piiratud puhver: tootja ja tarbija

- Tarbija protsess

```
item nextConsumed;
while (1) {
  while (in == out)
  ; /* do nothing */
  nextConsumed = buffer[out];
  out = (out + 1) % BUFFER_SIZE;
}
```

Protsessidevaheline side

- IPC — *Interprocess Communication*
- Operatsioonisüsteemi poolt pakutav mehhanism protsesside vahel side pidamiseks ja tegevuste sünkroniseerimiseks
- Jagatud mälu või teadete edastamine
- Näide: System V IPC — jagatud mälu, semaforid, teadete järjekorrad
- Näide: Unixi torud (*pipe*)
- Näide: Windowsi aknateated (*window messages*)

Teadete saatmine

- IPC mehhanism pakub kahte operatsiooni:
  - `send(message)` — teate suurus fikseeritud või vaba
  - `receive(message)`
- Kui protsessid P ja Q tahavad suhelda, siis peavad nad
  - Omavahel mingi ühenduse looma
  - Vahetama teateid `send/receive` abil
- Ühenduse omadusi:
  - Otsene või kaudne suhtlus?
  - Sümmeetriline või asümmeetriline suhtlus?
  - Automaatne või käsitsi toimuv puhverdamine?
  - Koopia või viida saatmine?
  - Kindla või suvalise pikkusega teated?

Otsene suhtlus

- Protsessid saadavad teateid otse üksteisele
- Peavad üksteise nimesid teadma:
  - `send(P, message)` — teate saatmine protsessile P
  - `receive(Q, message)` — teate vastuvõtt protsessilt Q
- Sidakanali omadused
  - Ühendused luuakse automaatselt
  - Ühendus on seotud täpselt ühe paari protsessidega
  - Iga protsessipaari vahel on täpselt üks ühendus
  - Ühendus võib olla ühesuunaline, enamasti on kahesuunaline

Kaudne suhtlus

- Teated saadetakse postkastidesse ja võetakse vastu postkastidest (*mailbox*, tihti ka port)
  - Igal postkastil oma ID
  - Protsessid saavad suhelda ainult siis, kui nad jagavad mingit postkasti
  - Postkast võib kuuluda nii rakendusele kui operatsioonisüsteemile
- Sidakanali omadused
  - Ühendus luuakse ainult siis, kui protsessid jagavad postkasti
  - Ühendus võib olla paljude protsesside vahel
  - Iga protsesside paar võib jagada mitut postkasti
  - Ühendus võib olla nii ühe- kui kahesuunaline

Postkastide kasutamine

- Operatsioonid postkastidega:
  - `send(A, message)` — teate saatmine postkasti A
  - `receive(A, message)` — teate vastuvõtt postkastist A
  - Kui postkastid kuuluvad opsüsteemile, on olemas primitiivid ka postkastide loomiseks ja kustutamiseks
- Postkastil on omanik (protsess)
- Lugeda saab kas ainult omanik või ka omaniku poolt lubatud teised protsessid
- Omanikuõigust ja lugemisõigust saab edasi saata
- Mis juhtub, kui postkasti tuleb teade ja lugejaid on mitu?
- Näited: Mach, Windows 2000

Sünkroniseerimine

- Teadete saatmine võib olla nii blokeeruv kui mitteblokeeruv
- Blokeerumine annab sünkroonse suhtluse
- Mitteblokeerumine annab asünkroonse suhtluse
- Saatmine ja vastuvõtt võivad teineteisest sõltumatult olla blokeeruvad või mitteblokeeruvad

Puhverdamine

- Ühendusega on seotud teadete järjekord
- Järjekorra pikkuse jaoks mitu võimalust:
  - Null — mitte ühtegi teadet, käsitsi puhverdamine
  - Piiratud maht — etteantud arv teateid, kui saatmisel on järjekord täis, siis sünkroonne saatja blokeerub ruumi vabanemiseni
  - Piiramatu maht — saatja ei blokeeru kunagi

Klient-server mudel: soklid

- Sokkel — side otspunkt
- Side toimib kahe sokli vahel
- Sokli aadressiks on masina aadressi ja protsessi identifikaatori komplekt
- Näiteks TCP/IP puhul IP aadress + pordi number
- Näide: sokkel 161.25.19.8:1625 ühes otsas ja sokkel 193.40.5.125:80 teises otsas.

Klient-server mudel: kaugprotseduurid

- Kaugprotseduuri väljakutse (RPC — *Remote Procedure Call*) üldistab protseduuride väljakutseid erinevate arvutite protsesside vahel
- Tüügas (*stub*) — kliendipoolne proksi, mida kutsutakse välja tegeliku protseduuri asemel
- Tüügas otsib serveri ja kodeerib parameetrid õiges formaadis
- Serveripoolne tüügas saab teate, kodeerib parameetrid lahti ja kutsub tegeliku protseduuri välja
- Näide: SunRPC (+XDR), DCE RPC
- Portmapper (ka *matchmaker*, *rendezvous daemon*)