PROCESET

KAPITULLI 4

Prof. Ass. Dr. Isak Shabani

Proceset

- Koncepti i proceseve
- Orari i Proceseve
- Operimet në Procese
- Komunikimi ndër-procese
- Shembuj të Sistemeve IPC
- Komunikimi në Sistemet Klient-Serverë

Objektivat

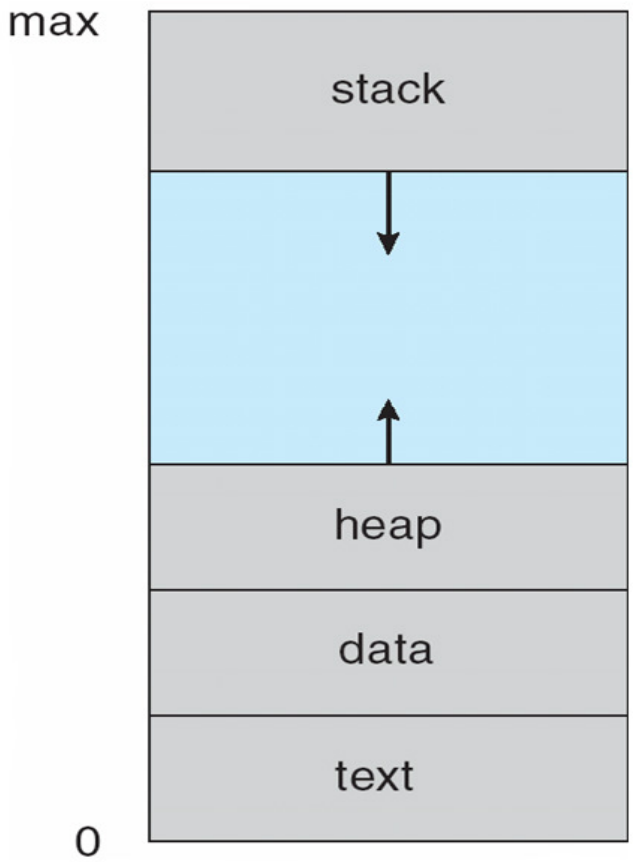
- Të përshkruaj nocionin e procesit, një program në ekzekutim, i cili formëson bazën e të gjitha kalkulimeve.
- Të përshkruaj mundësitë e ndryshme të proceseve, duke përfshirë oraret, krijimin dhe terminimin dhe komunikimin.
- Të përshkruaj komunikimin në sistemet klient-serverë.

Koncepti i Procesit

- Sistemi operativ ekzekuton një variacion programesh:
  - Sistemet bach, punë (jobs)
  - Sistemet me kohë të ndarë, programe të shfrytëzuesit ose detyra
- Termet *punë* dhe *proces* do të përdoren pothuajse pa ndonjë dallim, janë të ngjashëm !
- Procesi është një program në ekzekutimm
- Ekzekutimi i programit duhet të bëj progres në mënyrë sekuenciale
- Një proces përfshinë:
  - Numëruesin e programit
  - Stack-un
  - Seksionin e të dhënave

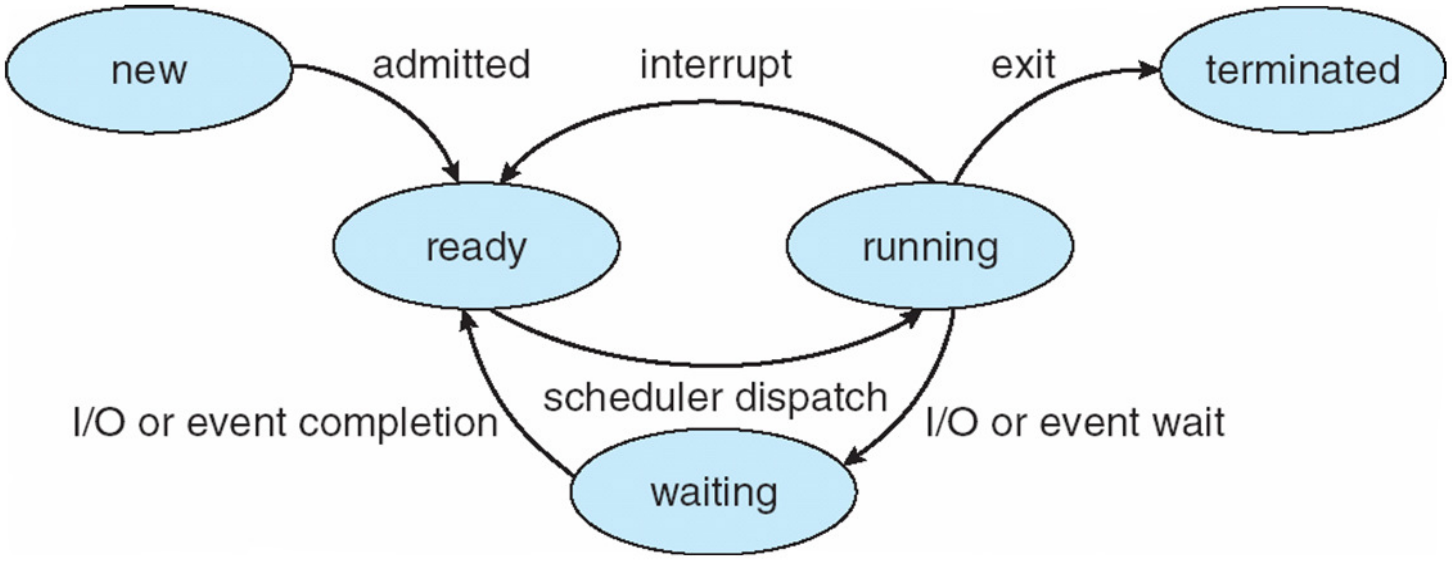
Procesi në memorie

- **Procesi në memorie shpërndahet neshumë pjesë**
  - **Kodi i programit**, i quajtur edhe **seksioni i tekstit**
  - Aktiviteti aktual duke përfshirë numëruesin e programit dhe regjistrat e procesit
  - **Stack** përshinë të dhëna të përkohshme
    - Parametrat e funksionit, adresat kthyese dhe variablat lokale
  - **Seksioni i të dhënave** përmban variablat globale
  - **Grumbulli (Heap)** përmban memorie të alokuar dinamike gjatë kohës së lëshimit (run-time)
- **Programi është entitet pasiv, procesi është aktiv**
  - *Programi bëhet proces kur fajlli ekzekutues të vendoset në memorie*
- P.Sh. Ekzekutimi i programit fillon përmes GUI me klikim të miut, përmes linjës së komandës përmes emrit të tij, etj
- **Një program mund të ketë disa procese**
  - Mendoni shumë shfrytëzues që e ekzekutojnë të njëjtin program



Gjendjet e Procesit

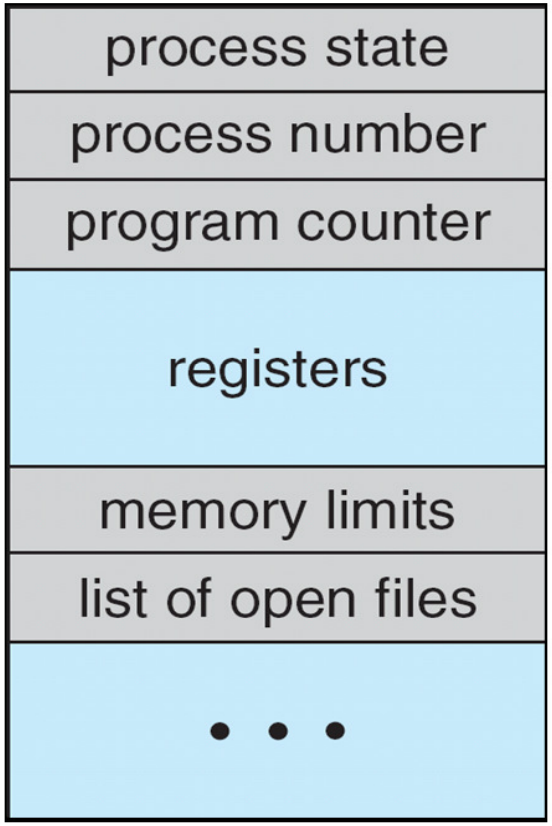
- Gjatë ekzekutimit të procesit, procesi e ndryshon gjendjen
  - **new**: procesi është duke u krijuar
  - **ready**: Procesi është në pritje të vendosjes në procesor
  - **running**: Instruksionet janë duke u ekzekutuar
  - **waiting**: Procesi është duke pritur për paraqitjen e ngjarjes
  - **terminated**: Procesi e ka përfunduar ekzekutimin



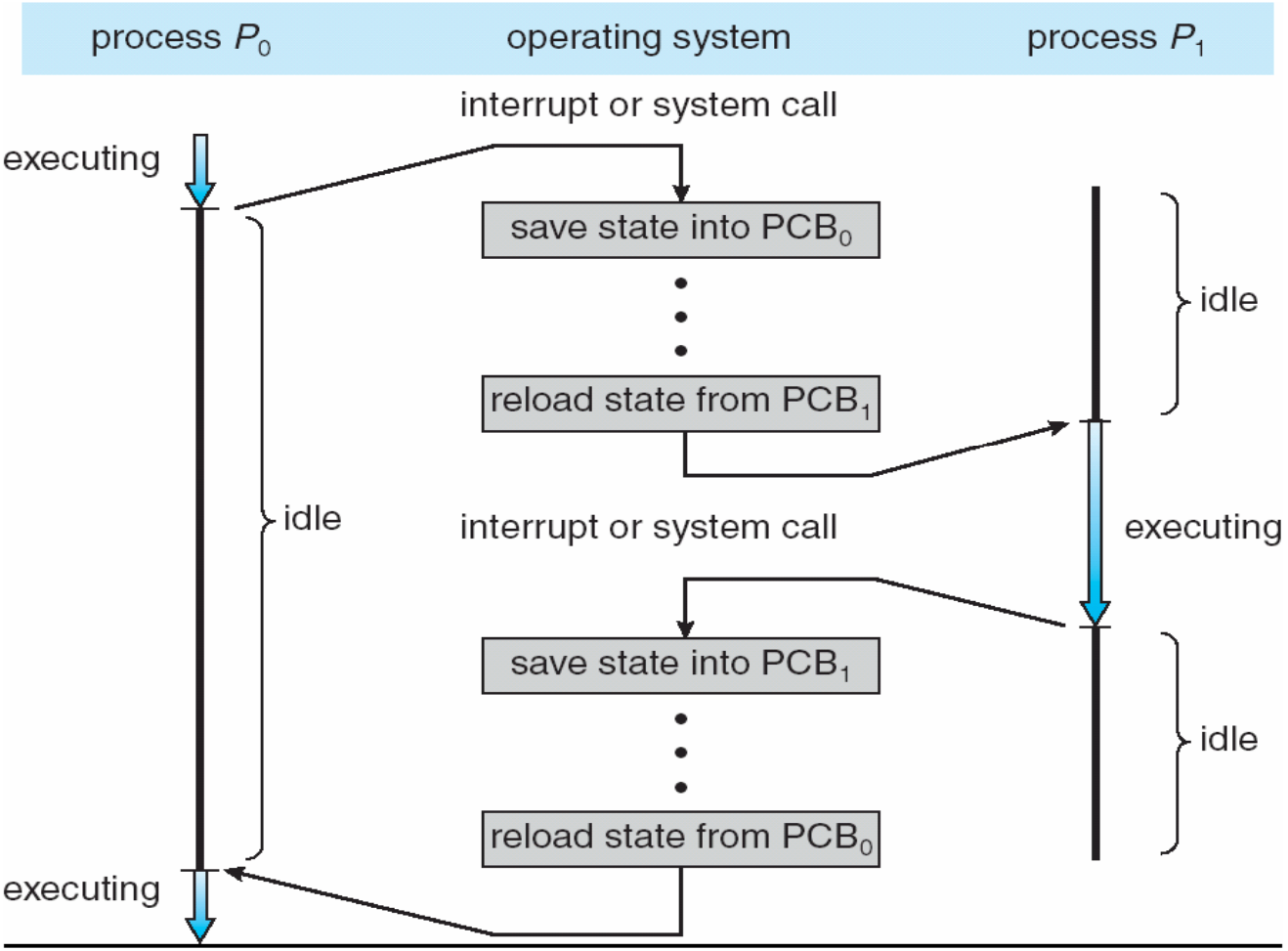
Bllok Kontrolli i Procesit (PCB)

Informatat e shoqëruara me secilin proces

- Gjendja e procesit
- Numruesi i programit
- Regjistrat e CPU-së
- Informatat për orarin e CPU-së
- Informatat mbi Menaxhimin e Memories
- Informata mbi Llogaritë (shfrytëzuesit)
- Informata mbi statutin e I/O



Kalimi i CPU nga Procesi në Proces



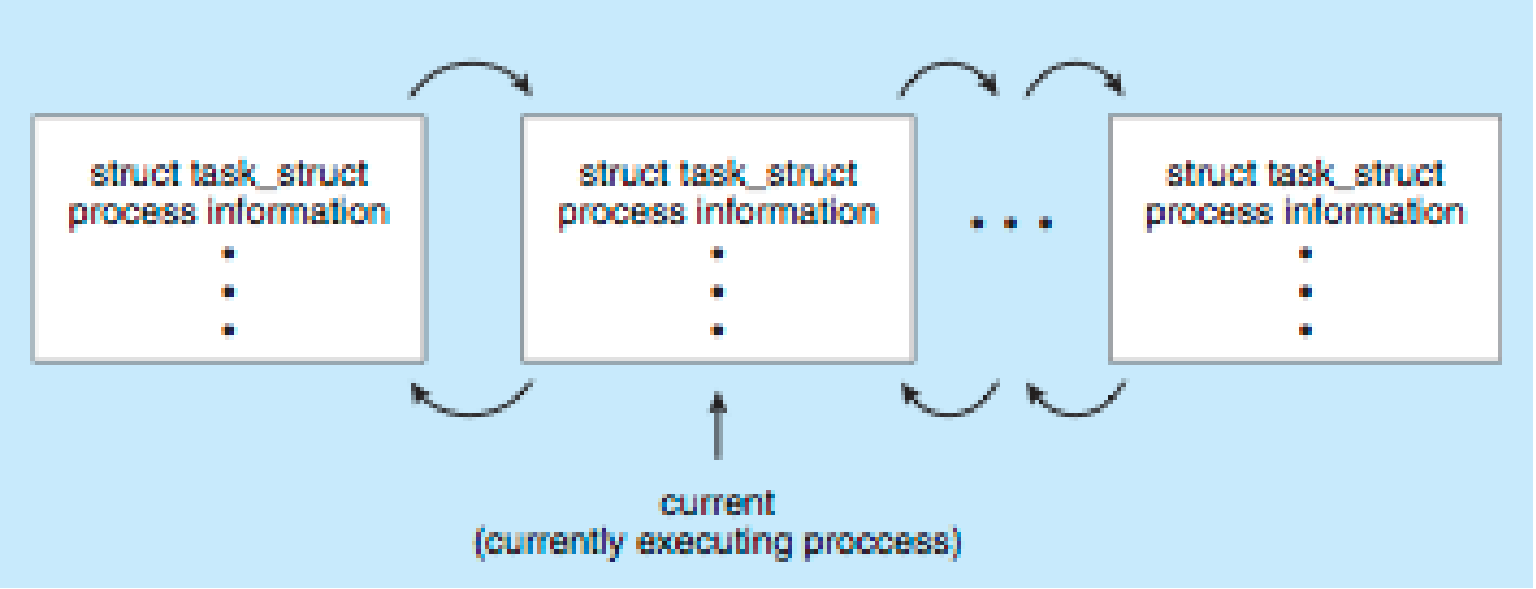
Orari i Proceseve

- Maksimizo shfrytëzimin e CPU, shpejt kalo proceset në CPU për ndarjen e kohës
- **Orari i proceseve** zgjedh në mes të proceseve të cilat janë të gatshme për ekzekutimin vijues në CPU
- Mirëmban **linjat e termineve** të proceseve
  - **Linja e punëve** – bashkësi e të gjitha proceseve në sistem
  - **Linja e gatshme (Ready queue)** – bashkësi e të gjitha proceseve të cilat gjenden në memorien kryesore, të gatshme dhe në pritje për ekzekutim
  - **Linja e pajisjeve (Device queues)** – bashkësi e proceseve në pritje të pajisjes I/O
  - Proceset migrojnë në mes të linjave të ndryshme

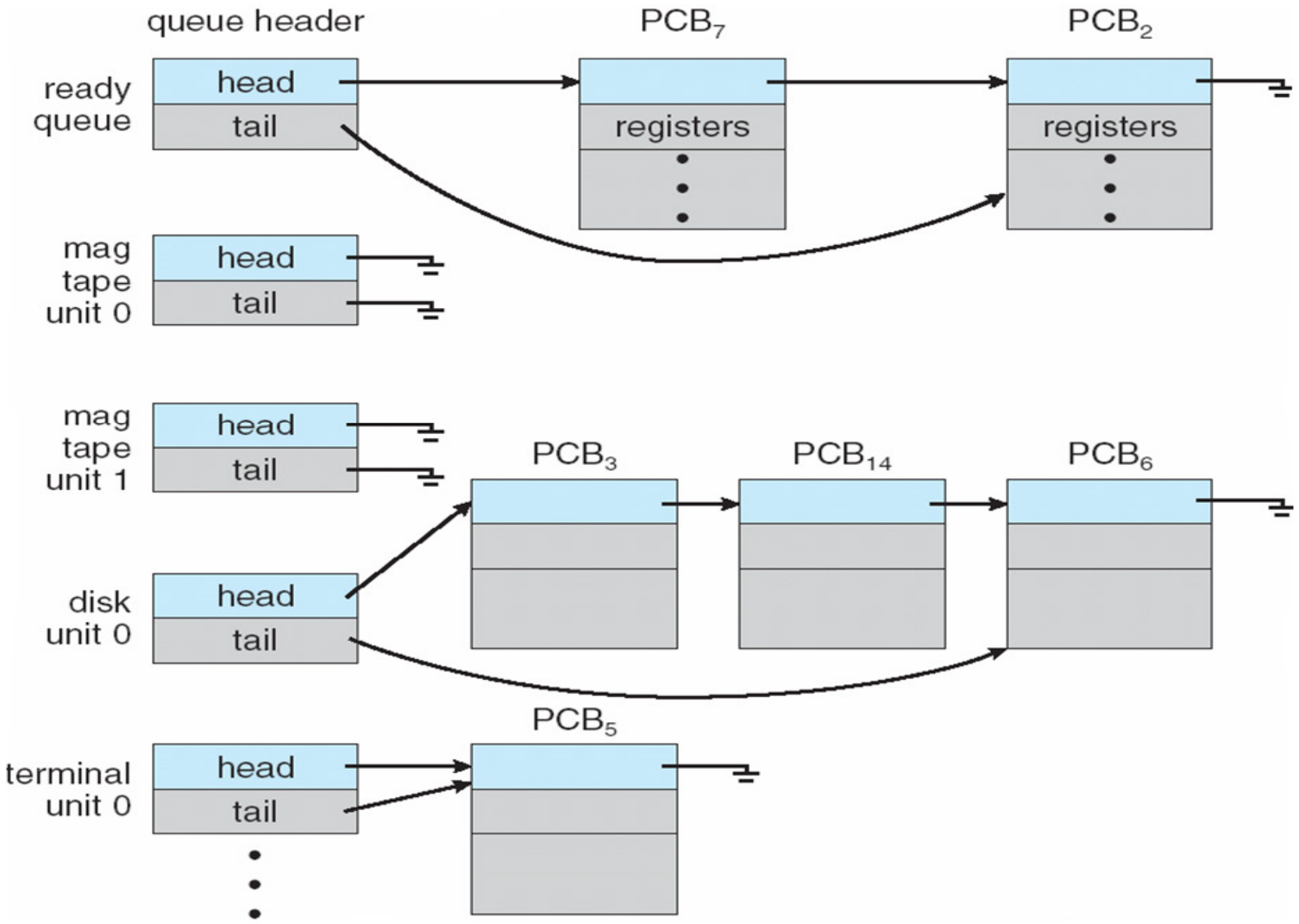
Përfaqësimi i Proceseve në Linux

- Të përfaqësuara në strukturën e C `task_struct`

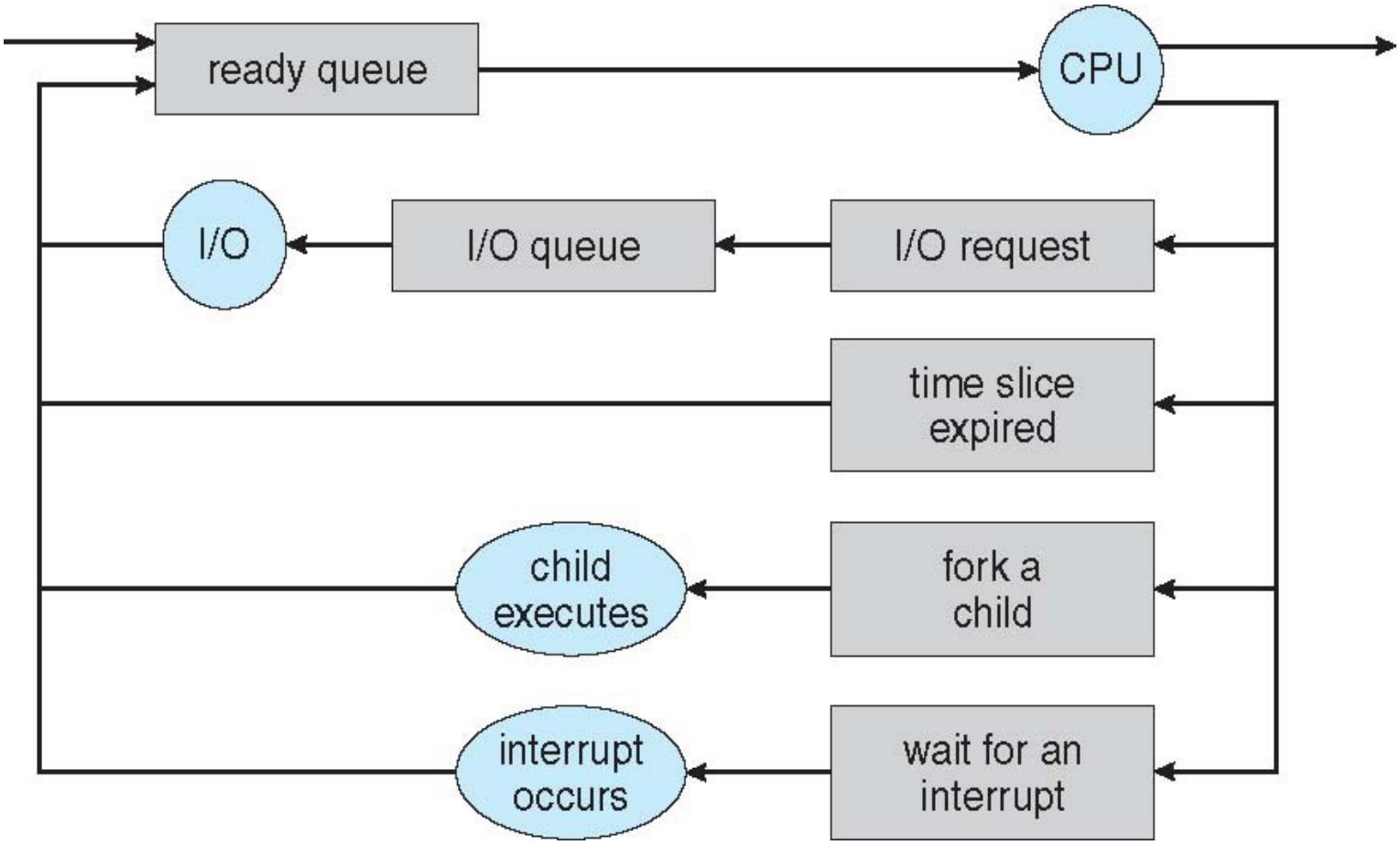
```
pid t pid;  /* identifikuesi i procesit*/
long state;  /* gjendja e procesit*/
unsigned int time slice  /* plani i informacionit*/
struct task struct *parent;  /* procesi prind */
struct list head children;  /* procesi fëmmij */
struct files struct *files; /* lista me fajllat e hapur*/
struct mm struct *mm;  /* hapësira e adresave*/
```



Linja e gatshme dhe variacionet e llinjave të Pajisjeve I/O



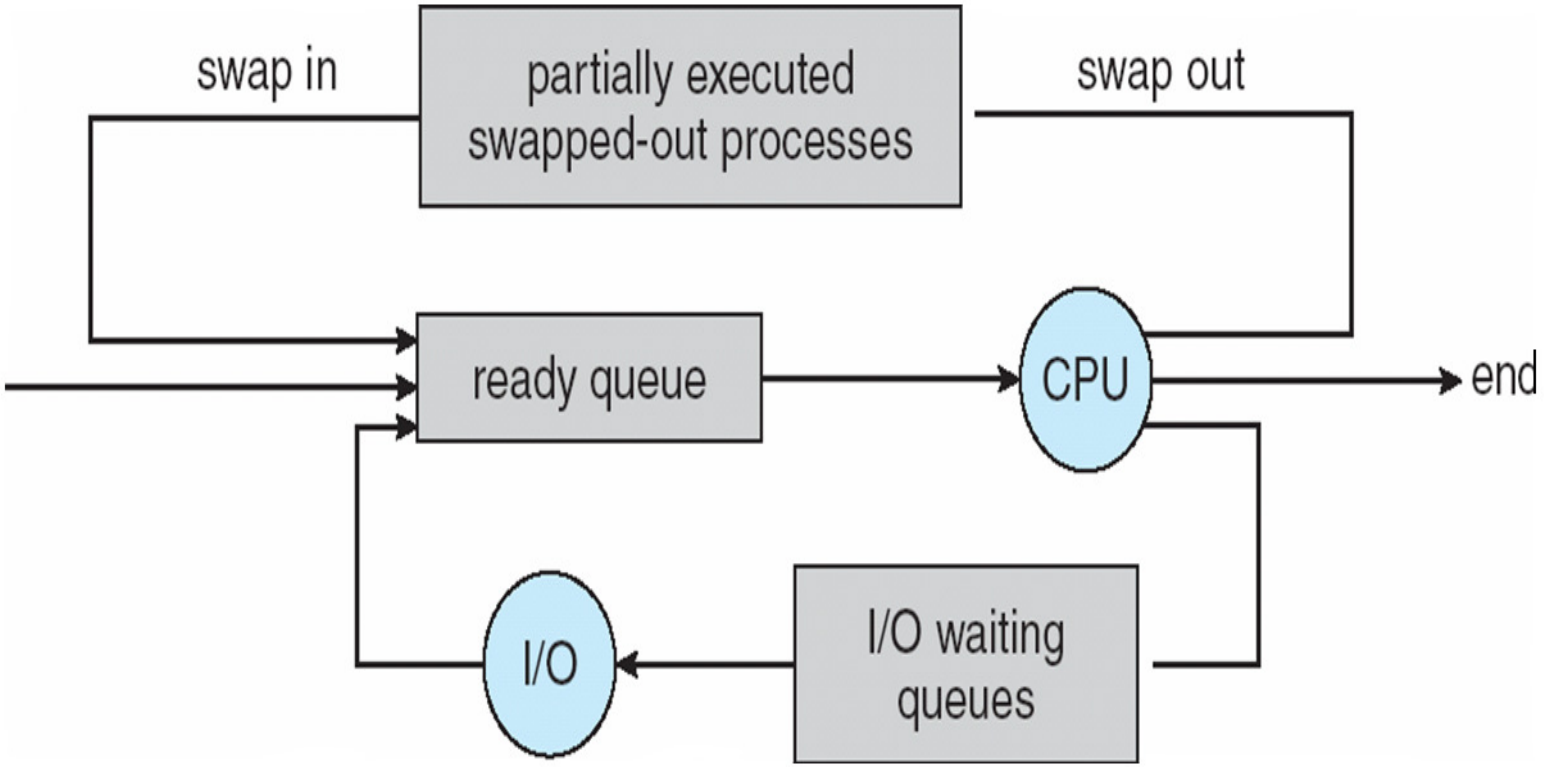
Paraqitja e Orarit të Proceseve



Orarët

- **Orari afatgjatë** (ose orari i punëve) – zgjedhë se cili proces duhet të sillet në linjën e gatshme
- **Orari afatshkurtër** (ose orari i CPU) – zgjedhë se cili proces tjetër duhet ekzekutuar dhe alokon CPU-në
  - Nganjëherë i vetmi orar në sistem
- Orari afatshkurtër thirret shumë shpesh (milisekonda) ⇒ (duhet të jetë i shpejtë)
- Orari afatgjatë thirret më pak (sekonda, minuta) ⇒ (mund të jetë i ngadalshëm)
- Orari afatgjatë kontrollon shkallën e *multiprogramimit*
- Proceset mund të përshkruhen si:
  - **Procese të lidhura I/O** – shpenzon më shumë kohë për I/O se sa llogaritje, shumë CPU burst të shkurtër
  - **Procese të lidhura të CPU** – shpenzon më shumë kohë në llogaritje; vetëm disa CPU burst

Mbledhja e orarit afatmesëm



Ndryshimi i Kontekstit

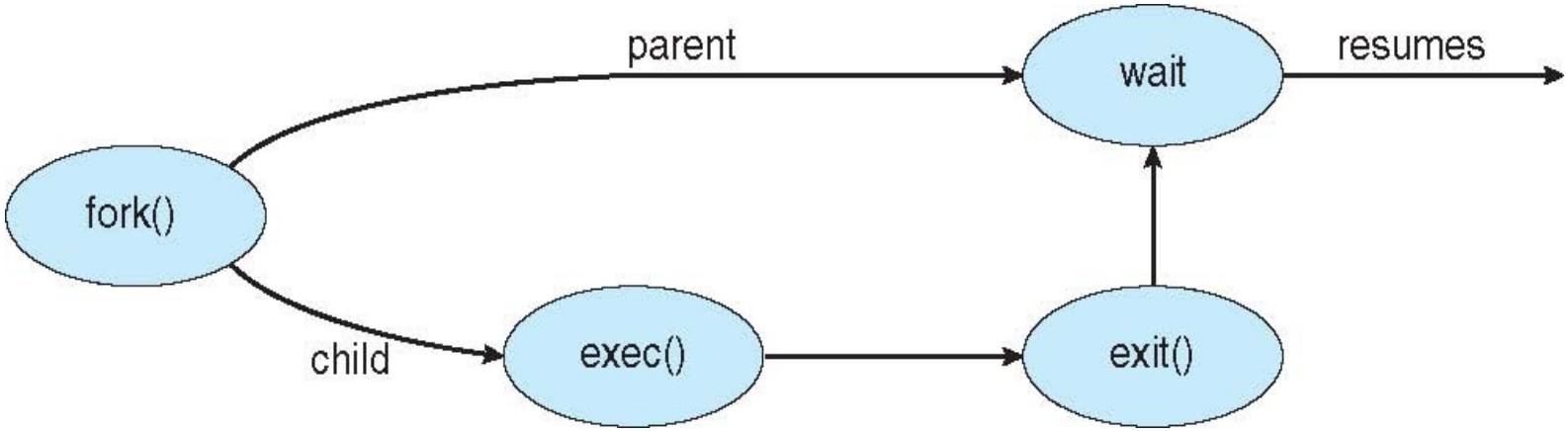
- Kur CPU kalon në procesin tjetër, sistemi duhet të ruaj gjendjen e procesit të vjetër dhe ta vendos gjendjen e ruajtur për procesin e ri përmes ndryshimit të kontekstit (**context switch**).
- **Konteksti** i procesit i paraqitur në PCB
- Ndryshimi i kontekstit është i ngarkueshëm; sistemi nuk bën ndonjë punë të arsyeshme gjerësa bëhet ndryshimi
  - Sa më kompleks të jetë OS dhe PCB -> ndryshimi i kontekstit do të jetë më i gjatë
- Koha e varur nga mbështetja harduerike
  - Disa harduera ofrojnë bashkësi regjistrash për CPU -> shumë kontekste të vendosura përnjëherë

Krijimi i Procesit

- Procesi **prind** krijon procese **fëmijë**, të cilët, në këtë mënyrë krijojnë procese të tjera, duke formuar një pemë të proceseve
- Në përgjithësi, procesi identifikohet dhe menaxhohet përmes **identifikuesit të procesit - process identifier** (**pid**)
- Ndarja e resurseve
  - Prindi dhe fëmijët ndajnë të gjitha resurset
  - Fëmijët ndajnë nënbashkësi të resurseve të prindit
- Ekzekutimi
  - Prindi dhe fPrindi pret gjerësa fëmijët të mbarojnë
  - ëmijët ekzekutohen në mënyrë konkurrente

Krijimi i Procesit (vazhd.)

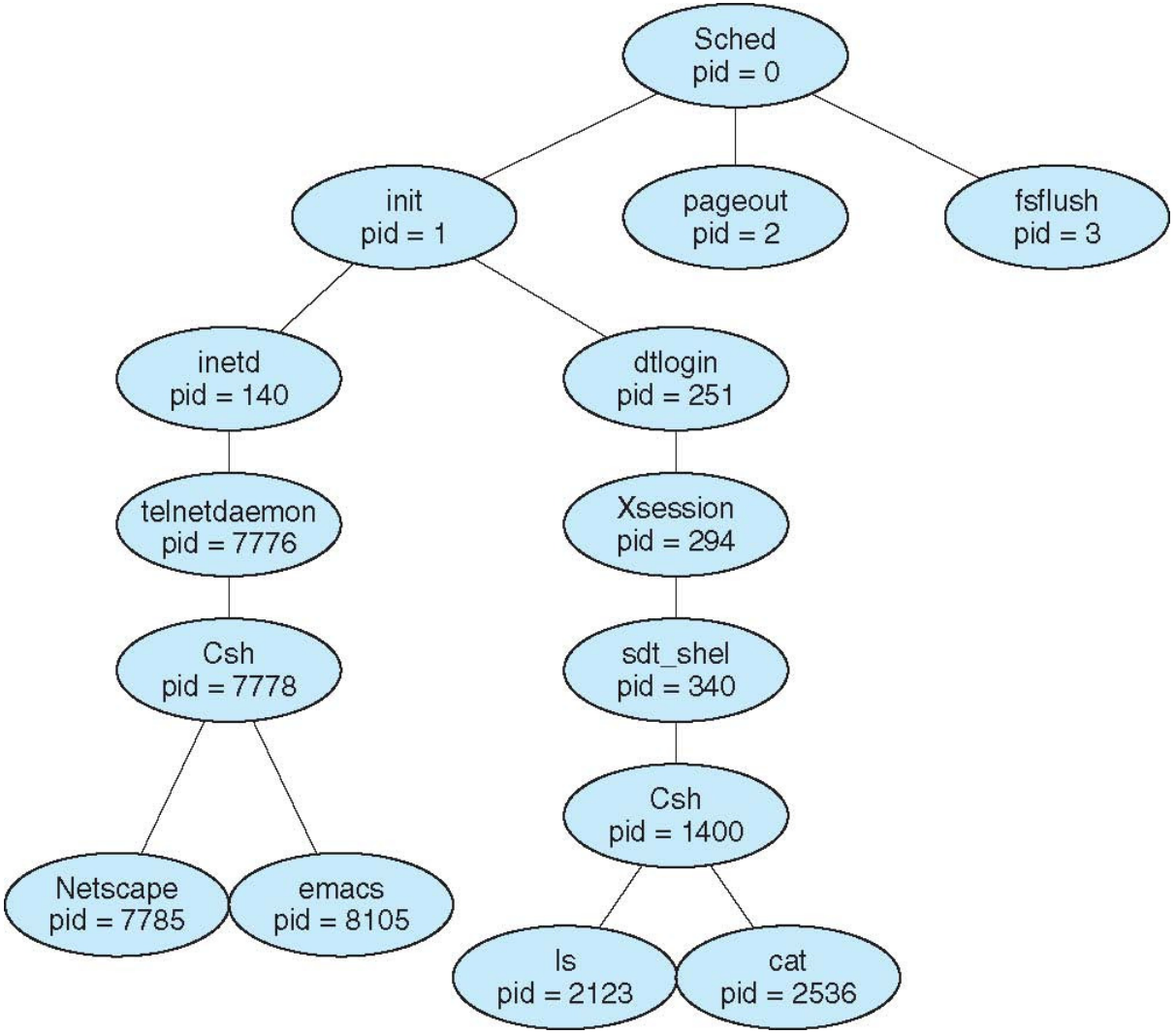
- Hapësira e adresave
  - Fëmija duplikat i prindit
  - Fëmija ka program të vendosur në të
- Shembuj të UNIX
  - Thirrja e sistemit **fork** e krijon procesin e ri
  - Thirrja sistemit **exec** përdoret pas **fork** për të zëvendësuar hapësirën memorike me program të ri



Programi në C për Krijimin e Procesit të ndarë

```
#include <sys/types.h>
#include <studio.h>
#include <unistd.h>
int main()
{
pid_t  pid;
   /* fork another 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 */
                wait (NULL);
                printf ("Child Complete"); }
   return 0;
}
```

Një Pemë e proceseve në Solaris



Përfundimi i Procesit

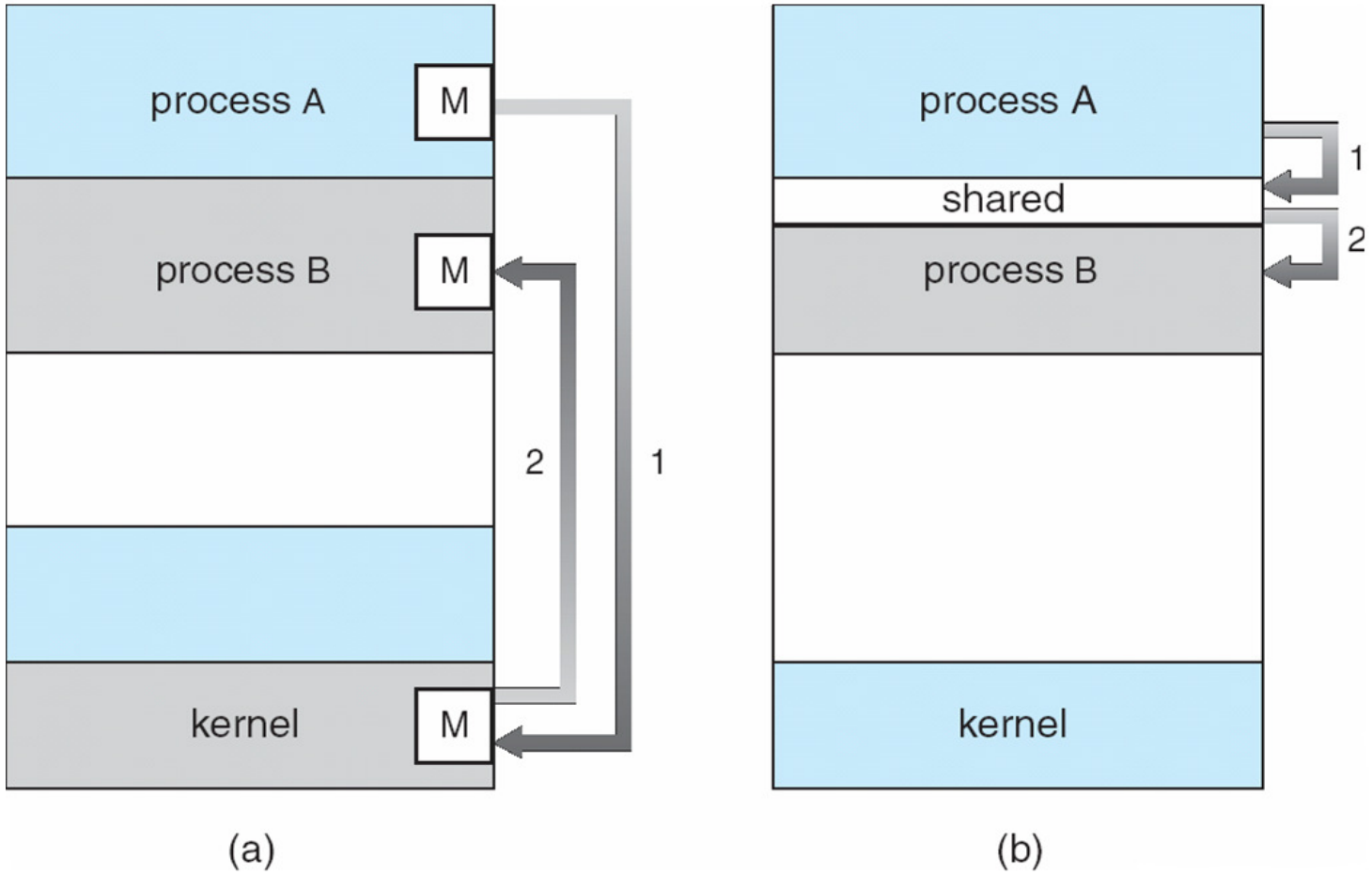
- Procesi e ekzekuton formulimin e fundit dhe i kërkon sistemit operativ ta fshij atë (**exit**)
  - I shkruan të dhënat nga fëmija tek prindi (përmes **wait**)
  - Resurset e proceseve de-alokohen nga sistemi operativ

- Prindi mund të përfundojë ekzekutimin e proceseve fëmijë (**abort**)
  - Fëmija ka tejkaluar resurset e alokuara
  - Puna e dhënë tek fëmija nuk është më e nevojshme
  - Nëse prindi është në mbyllje
    - Disa sisteme operative nuk lejojnë që fëmija të vazhdojë nëse prindi përfundon
      - Të gjithë fëmijët përfundohen (terminohen) - përfundim ujvarë (**cascading termination**)

Komunikimi ndërprocese

- Proceset në një sistem mund të jenë të **pavarura** ose **kooperuese**
- Proceset kooperuese mund të prekin ose të preken nga sistemet e tjera, duke përfshirë këtu edhe ndarjen e të dhënave
- Arsyet për kooperimin e proceseve:
  - Ndarja e informatave
  - Ngritje e shpejtësisë në llogaritje
  - Modulariteti
  - Komoditeti
- Proceset kooperuese kanë nevojë për **komunikimin ndërprocese** (**IPC**)
- **Jane dy modele të komunikimin ndërprocese**
  - **Memorie të ndarë**
  - **Pasimi i mesazheve**

Modelet e komunikimit



Proceset Kooperuëse

- Proceset e **pavarura** nuk mund të ndikohen nga ekzekutimi i procesit tjetër
- Proceset **kooperuëse** mund të ndikojnë ose të ndikohen nga ekzekutimi i procesit tjetër
- Përparësitë e kooperimit të proceseve
  - Ndarja e informatave
  - Ngritje e shpejtësisë në llogaritje
  - Modulariteti
  - Komoditeti

Problemi Prodhuesi-Konsumatori

- Paradigmë për kooperimin e proceseve, *prodhuësi* i proceseve prodhon informata të cilat konsumohen nga procesi *konsumuës*
  - *Buferi i pakufizuar (unbounded-buffer)* nuk vendos ndonjë limit praktik në madhësinë e buferit
  - *Buferi i kufizuar (bounded-buffer*) para-supozon që madhësia e buferit të është fikse

Buferi-Kufizuar – Zgjidhja me Memorje të Ndarë

- Të dhënat e ndara

```
#define BUFFER_SIZE 10
typedef struct {
   . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Zgjidhja është e saktë, por mund të përdorë vetëm elemente BUFFER_SIZE-1

Buferi i Kufizuar– Prodhuësi

```
while (true) {
   /* Produce an item */
      while (((in = (in + 1) % BUFFER SIZE count)  == out)
      ;   /* do nothing -- no free buffers */
    buffer[in] = item;
    in = (in + 1) % BUFFER SIZE;
   }
```

Buferi i Kufizuar– Konsumatori

```
while (true) {
        while (in == out)
                ; // do nothing -- nothing to consume

    // remove an item from the buffer
    item = buffer[out];
    out = (out + 1) % BUFFER SIZE;
return item;
  }
```

Komunikimi ndërprocese - Pasimi i Mesazheve

- Mekanizëm për proceset që të komunikojnë dhe të sinkronizojnë veprimet e tyre
- Sistemi i mesazheve – komunikimi i proceseve në mes njëri tjetrit pa përdorur variablat e ndara
- Struktura IPC ofron dy operacione:
  - **Dërgo dhe merr**(*mesazhin*)
  - madhësia e mesazhit fikse ose variabile
- Nëse *P* dhe *Q* dëshirojnë të komunikojnë, atyre ju duhet që:
  - të themelojnë *lidhje komunikimi* në mes tyre
  - të shkëmbejnë mesazhe përmes dërgo/merr
- Implementimi i lidhjes komunikiuëse
  - fizik (p.sh., memoria e ndarë, busi harduerik)
  - logjik (p.sh., vetitë logjike)

Pyetjet e Implementimit

- Si themelohen lidhjet?
- A mund të shoqërohet lidhja me më shumë se dy procese?
- Sa lidhje mund të jenë në mes të secilës palë të proceseve komunikuëse?
- Sa është kapaciteti i lidhjes?
- A është madhësia e mesazhit që lidhja mund të akomodojë e fiksuar apo variabile?
- A është linku njëdrejtimësh apo dydrejtimësh?

Komunikimi Direkt

- Proceset duhet të emrojnë njëri tjetrin në mënyrë eksplicite:
  - **dërgo** (*P, mesazhi*) – dërgo mesazh tek procesi P
  - **merr**(*Q, mesazhi*) – merr mesazh nga procesi Q
- Vetitë e lidhjes komunikuëse
  - Lidhjet themelohen automatikisht
  - Lidhja shoqërohet saktësisht me një palë të procesit komunikues
  - Në mes të secilës palë ekziston saktësisht një lidhje
  - Lidhja mund të jetë një dretimëshe, por zakonisht është dy-drejtimëshe

Komunikimi indirekt

- Mesazhet drejtohen dhe pranohen nga mailbox-at (gjithashtu të referuara si porta)
  - Secili mailbox ka një ID unike
  - Proceset mund të komunikojmë vetëm nëse kanë mailbox të ndarë
- Vetitë e lidhjes komunikuëse
  - Lidhja themelohet nëse proceset ndajnë mailbox të përbashkët
  - Lidhja mund të shoqërohet me disa procese
  - Secila palë e proceseve mund të ndajë disa lidhje komunikuëse
  - Lidhjet mund të jenë njëdrejtimëshe dhe dy drejtimëshe

Komunikimi indirekt

- Operacionet
  - krijo një mailbox të ri
  - dërgo dhe prano mesazhe përmes mailbox
  - shkatërro mailbox-in

- Baza është e definuar si vijon:

  dërgo(A, *mesazhi*) – dërgo mesazh në mailbox-in A

  prano(*A, mesazhi*) – prano mesazh nga mailbox-i A

Komunikimi indirekt

- Ndarja e mailbox-it
  - P_1, P_2, dhe P_3 ndajnë mailbox-in A
  - P_1, dërgon; P_2 dhe P_3 pranon
  - Kush e merr mesazhin?

- Zgjidhjet
  - Lejo lidhjes të jetë e shoqëruar me jo më shumë se dy procese
  - Lejo vetëm një proces në kohë të ekzekutojë operacionin e pranimit
  - Lejo sistemin të zgjedhë marrësin në mënyrë arbitrare. Dërguesi duhet të njoftohet se kush ishte marrësi.

Sinkronizimi

- Pasimi i mesazheve mund të jetë bllokues ose jo-bllokues

- **Bllokimi** konsiderohet **sinkron**
  - **Bllokimi i dërgo** e bllokon dërguesin derisa të pranohet mesazhi
  - **Bllokimi i prano** e bllokon marrësin gjerësa mesazhi të jetë i gatshëm

- **Jo-bllokues** konsiderohet **asinkron**
  - dërgo **Jo-bllokues** e lejon dëerguesin të dërgoj mesazhin dhe të vazhdojë
  - pranimi **jo-bllokues** e lejon marrësin të marrë mesazhin e vlefshëm ose të zbrazët (null)

Baferimi

- Varg i mesazheve të lidhura me link; të implementuara në një nga tri mënyrat
  1. Me zero kapacitet – 0 mesazhe
     Dërguesi duhet të pres për marrësin (rendezvous)
  2. Kapacitet i limituar – gjatësi finite e *n* mesazheve
     Dërguesi duhet të pres nëse linku është i mbushuar (full)
  3. Kapacitet i pa limituar – gjatësi e pafundme
     Dërguesi nuk pret kurrë!

Shembuj të Sistemit IPC - POSIX

- Memorja e ndarë POSIX
  - Procesi së pari e krijon segmentin për memorie të ndarë

```
segment id = shmget(IPC PRIVATE, size, S IRUSR
  | S IWUSR);
```

  - Procesi i cili dëshiron të ketë qasje në atë memorje të ndarë duhet të lidhet për të

```
shared memory = (char *) shmat(id, NULL, 0);
```

  - Tani pasi procesi mund të shkruaj në memorjen e ndarë

```
sprintf(shared memory, "Duke shkruar në
  memorjen e ndarë!");
```

  - Kur të mbaroj procesi mund të largoj memorjen e ndarë nga hapsira adresuëse e tij

```
shmdt(shared memory);
```

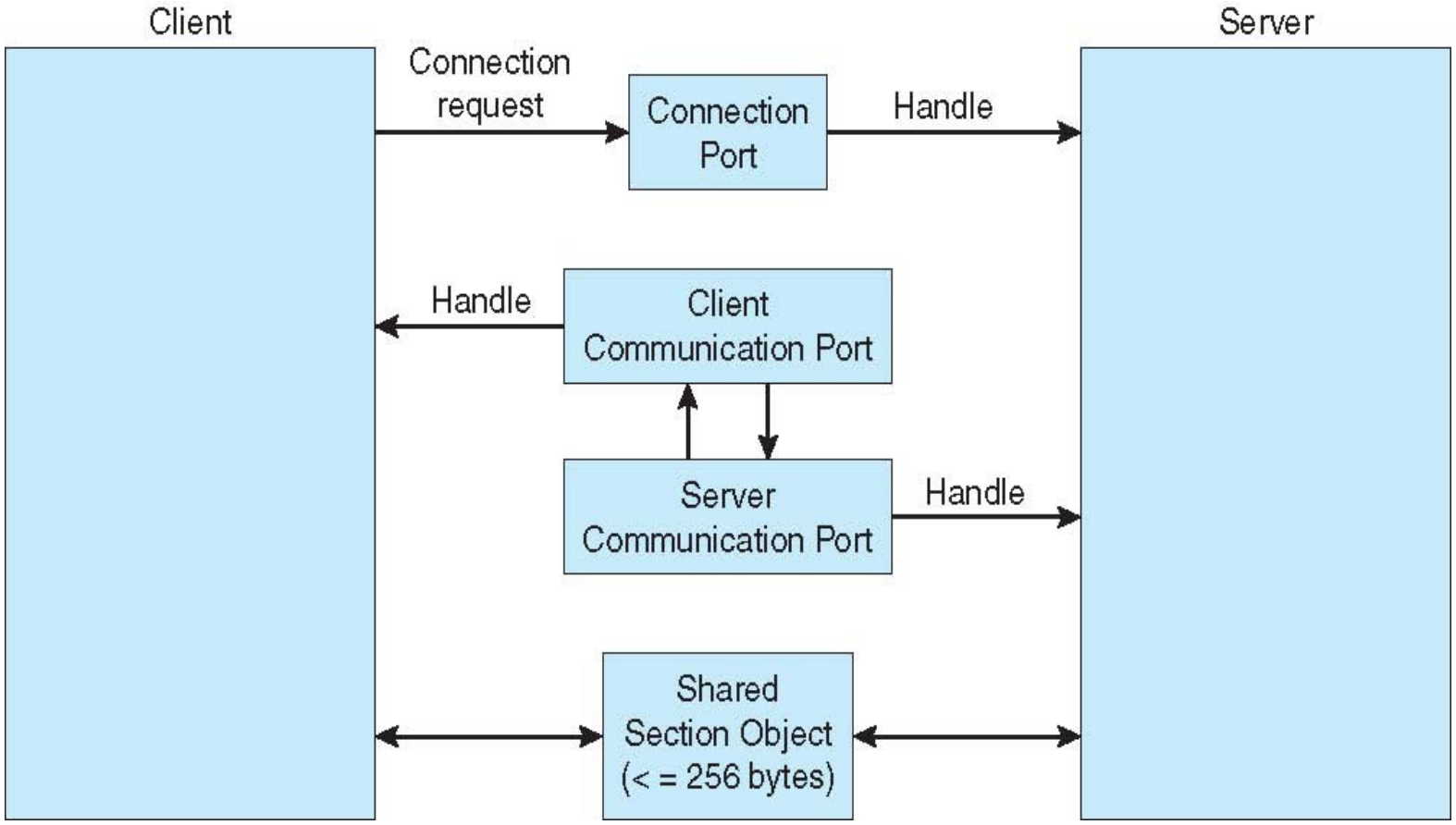
Shembuj të Sistemit IPC - Mach

- Komunikimi i Mach është i bazuar në mesazhe
  - Edhe thirrjet e sistemit janë mesazhe
  - Secila punë merr dy mailboxa në krijim – Kerneli dhe Lajmrimi
  - Vetëm tri thirrje të sistemit nevojiten për transferimin e mesazhit `msg_send()`, `msg_receive()`, `msg_rpc()`
  - Mailbox-at e nevojshëm për komunikim, krijohen përmes `port_allocate()`

Shembuj të Sistemit IPC– Windows XP

- Pasimi i mesazheve qëndror përmes thirrjes së procedurave lokale - **local procedure call (LPC)**
  - Punon vetëm për proceset në sistemin e njejtë
  - Përdor porta (ngjashëm me mailbox-a) për të themeluar dhe mirëmbajur kanalet komunikuëse
  - Komunikimi punon si vijon:
    - Klienti hapë një trajtim tek porti i objektit komunikues të nën-sistemit
    - Klienti dërgon kërkesën për lidhje.
    - Serveri krijon dy porta për komunikime private dhe kthen trajtimin tek një nga klientët.
    - Klienti dhe serveri përdorin porta korresponduëse të trajtimit për të dërguar ose callbacks dhe të ndëgjoj për përgjigje(replies).

Thirrja e Procedurave Lokale në Windows XP



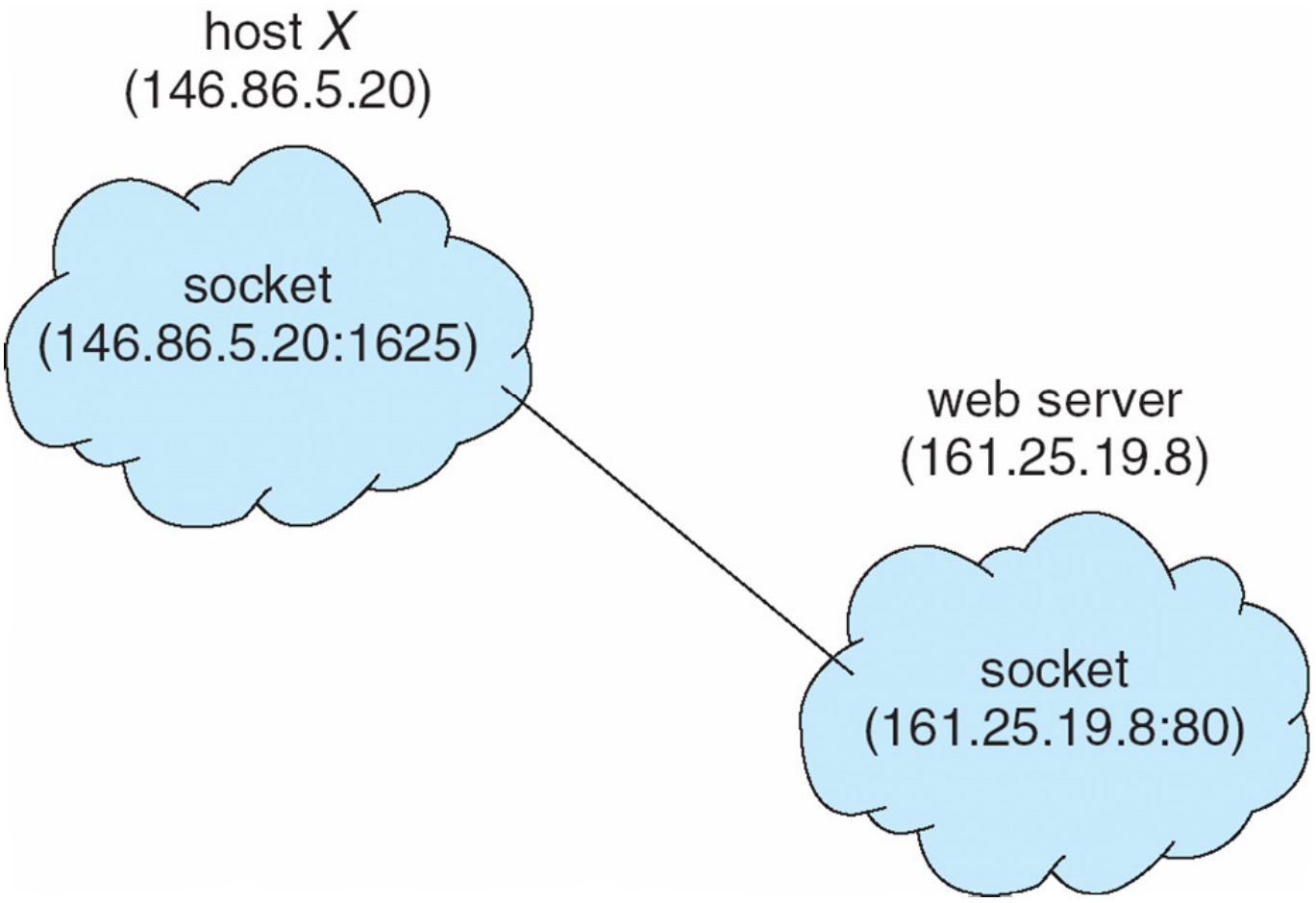
Komunikimet në Sistemet Klient-Serverë

- Soketa
- Thirrja e Procedurave në Largësi (Remote Procedure Calls)
- Tubat (Pipes)
- Thirrja e Metodave në Largësi (Remote Method Invocation) - Java

Soketat

- **Soketi** definohet si *pikë fundore e komunikimit*
- Lidhja e IP adresës dhe portit
- Soketi **161.25.19.8:1625** i referohet portit **1625** në hostin **161.25.19.8**
- Komunikimi përbëhet nga palët e soketave

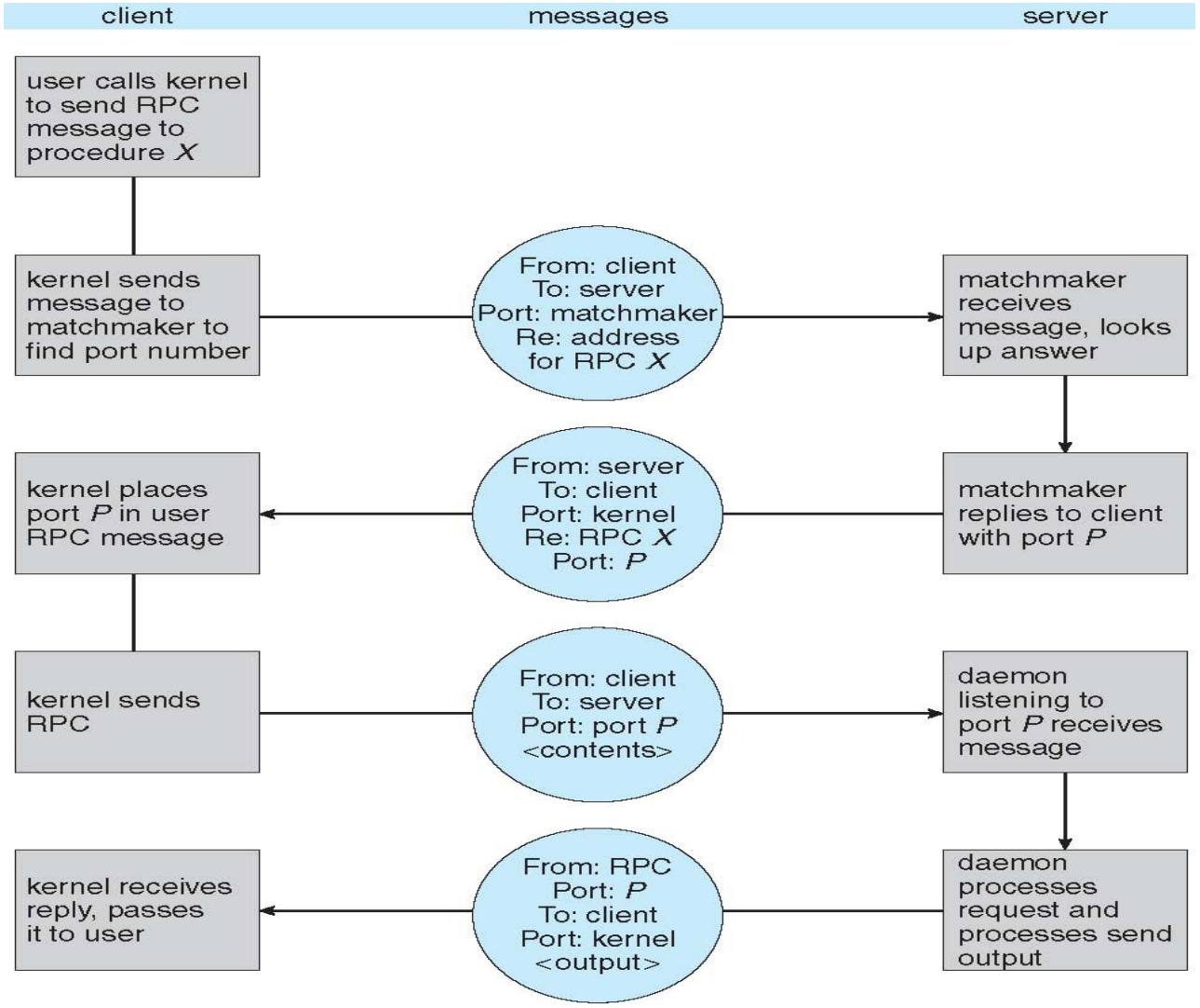
Komunikimi i Soketave



Thirrja e Procedurave në Largësi

- Thirrja e procedurave në Largësi (RPC) abstrakton thirrjen e procedurave në mes të procesve në sistemet në rrjetë
- **Cungat (Stubs)** – proksi në anën e klientit për procedurën aktuale në server
- Cungu në anën e klientit e lokalizon serverin dhe *marshallon* parametrat
- Cungu në anën e serverit e merr këtë mesazh, e shpaketon,dhe kryen procedurën në server

Ekzekutimi i RPC



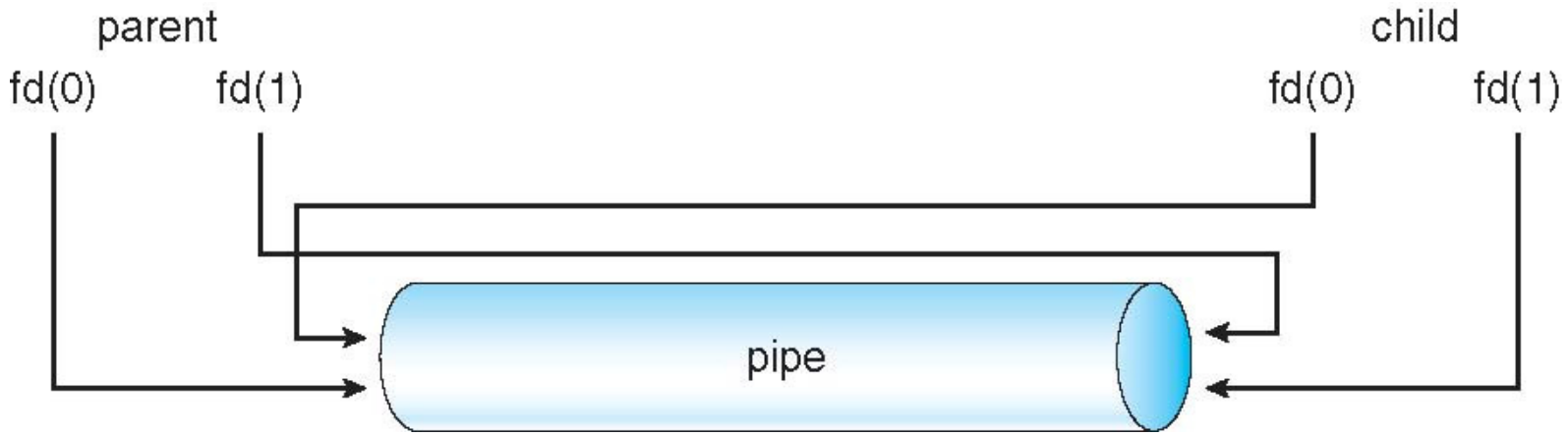
Tubat

- Vepron si tub lidhës i cili i lejon dy procese që të komunikojnë

- **Çështjet**
  - A është komunikimi një kahësh apo dykahësh?
  - Në rastin e komunikimit me dy kalime, a është dyfish (duplex) i përgjysmuar apo i plotë?
  - A duhet patjetër të ekzistoj një lidhje (p.sh., prind-fëmijë) në mes të proceseve komunikuëse?
  - A mund të përdoren tubat në rrjetë?

Tubat e Zakonshme

- **Tubat e Zakonshme** lejojnë komunikimin në stilin standard të prodhuesit-konsumatorit
- Prodhuësi shkruan në një fund (*write-end* i tubës)
- Konsumatori lexon nga fundi i tjetër (*read-end* i tubës)
- Tubat e zakonshme janë njëdrejtimëshe
- Kërkojnë relacionin prind-fëmijë në proceset komunikuëse



Tubat e Emëruara

- Tubat e Emëruara janë më të fuqishme sesa tubat e zakonshme
- Komunikimi është njëkahësh
- Nuk është i nevojshëm relacioni prind-fëmijë në mes të proceseve komunikuëse
- Disa procese mund të përdorin tuba të emëruara për komunikim
- Të ofruara në sistemet UNIX dhe Windows