Operacinės sistemos P175B304

3 paskaita

N. Sarafinienė

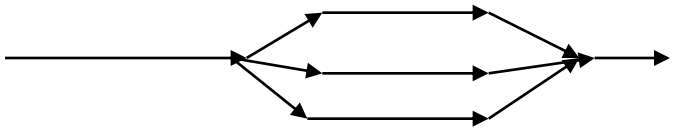
2014-02-25

Kalbėsime

- Gijos
- Jų tipai
- Realizacijos modeliai
- Proceso- gijos skirtumai
- Gijomis paremtų programų projektavimas

Konkurencija

- Nepriklausomi procesai (Independent process)
  - Tai nuosekli programa vykdymo eigoje
  - Privatus kontekstas
  - Rezultatai priklauso tik nuo įėjimo duomenų
- Konkuruojantys procesai (Concurrent processes)
  - Dalinimasis resursais (konkurencija)
  - Vykdymas reikalauja planavimo
- Kooperuojantys procesai (Cooperating processes)
  - Lygiagretaus principo programa jos vykdymo metu
  - Bendrai naudojamas kontekstas ar jo dalis
  - Rezultatai priklauso nuo vykdymo eigos

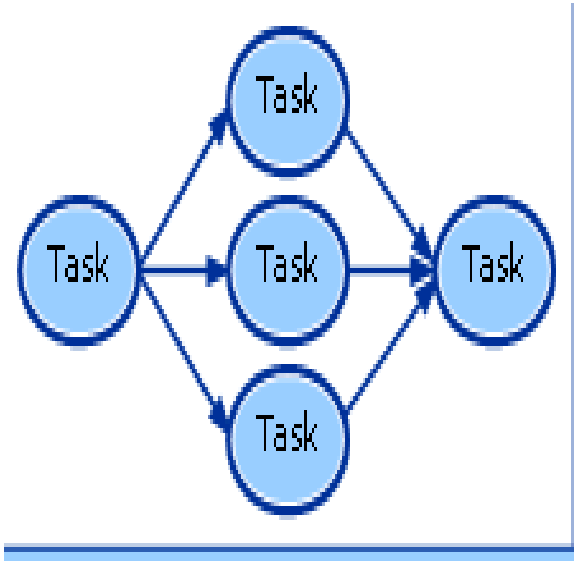


Konkurencija

- Tarp skirtingų procesų
- Tarp vienodų procesų
  - Norima vykdyti tą patį kodą,
    - Reikia: turėti galimybę fiksuoti skirtingą šio kodo vykdymo būklę, o tai apima:
      - steko informaciją
      - programos skaitliuko, registrų reikšmes.
  - Norima pasiekti tuos pačius duomenis,
  - Norima naudotis tais pačiais ištekliais, tokiais kaip kad atverti failai, tinklinės jungtys

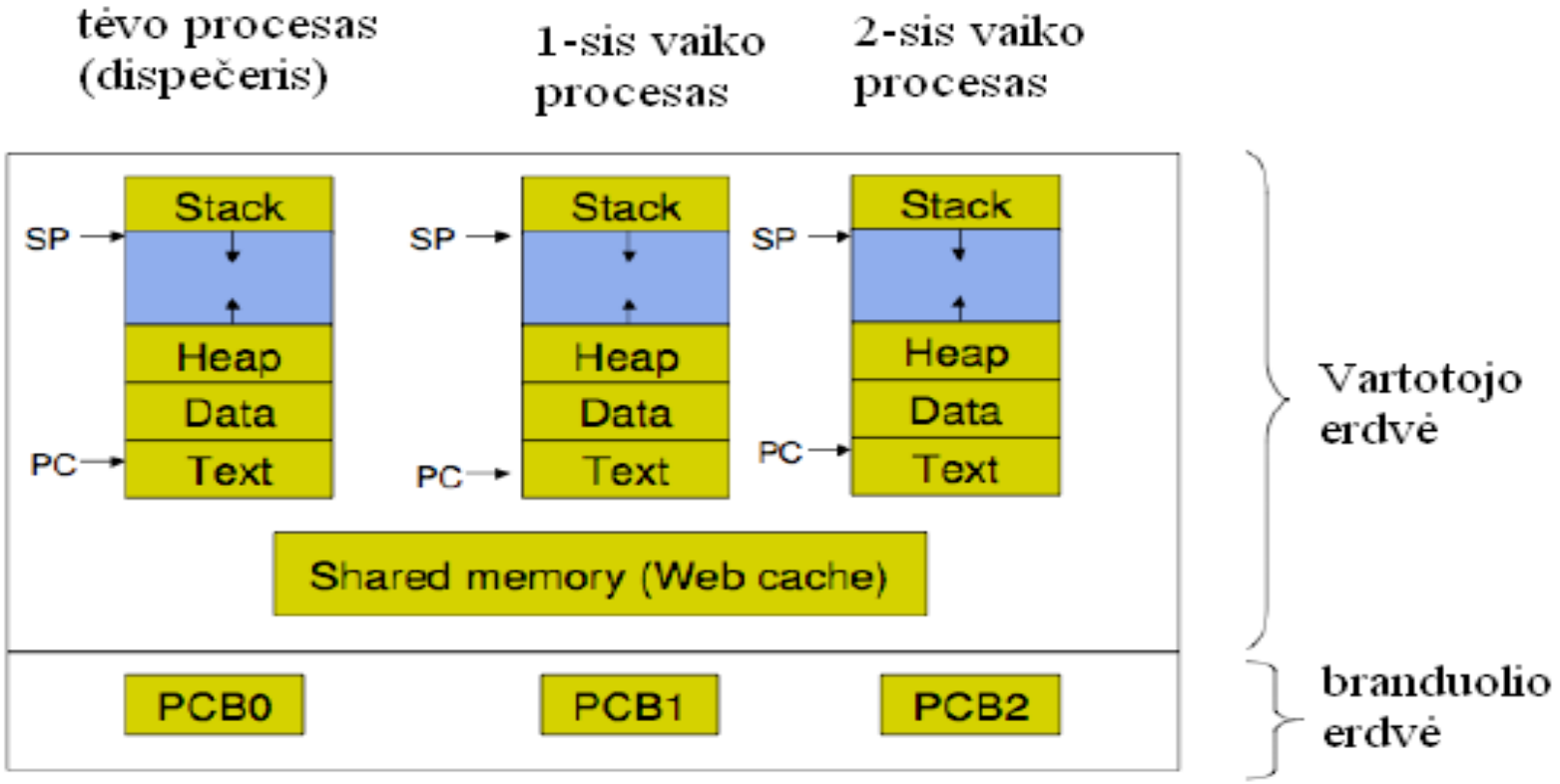
Lygiagretūs kooperuojantys procesai

- To paties kodo vykdymą pavedant lygiagretiems procesams (Web)
  - Reikia sukurti (fork)
  - Operacinė sistema turi planuoti lygiagretų jų vykdymą



- Situacija nėra efektyvi:
  - Adresų erdvė: Procesų PCB, puslapių lentelės, t.t.
  - Laikas: reikia sukurti duomenų struktūras, kopijuoti pačius procesus, t.t.
  - Persijungimas tarp procesų
  - Komunikacija tarp procesų

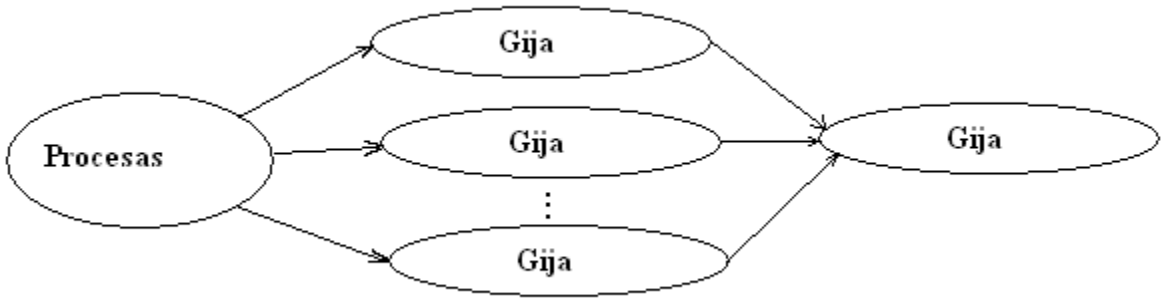
Lygiagretūs kooperuojantys procesai



Dar apie procesus...

- Kas bendra kooperuojantiems procesams?
  - Jie dalijasi tuo pačiu kodu ir duomenimis (adresų erdvė)
  - Jie visi dalosi tais pačiais ištekliais (failais, soketais, t.t)
- Kuo jie nesidalina?
  - Kiekvienas turi tik jam būdingą **vykdymo būvį**:
    - PC, SP, ir registrai
- Idėja: atskirti proceso koncepciją nuo vykdymo būvio
  - Procesas: adresų erdvė, privilegijos, ištekliai, t.t.
  - Vykdymo būvis: PC, SP, registrai
- Vykdymo būvis dar yra vadinamas kontrolės gija (the thread of control) arba tiesiog gija (thread)

Gijos



- *Tai nepriklausoma instrukcijų seka procese, kurios atžvilgiu galima taikyti vykdymo planavimą* (PC, SP, registrai). Jos naudojamos lygiagrečių skaičiavimų realizacijai.

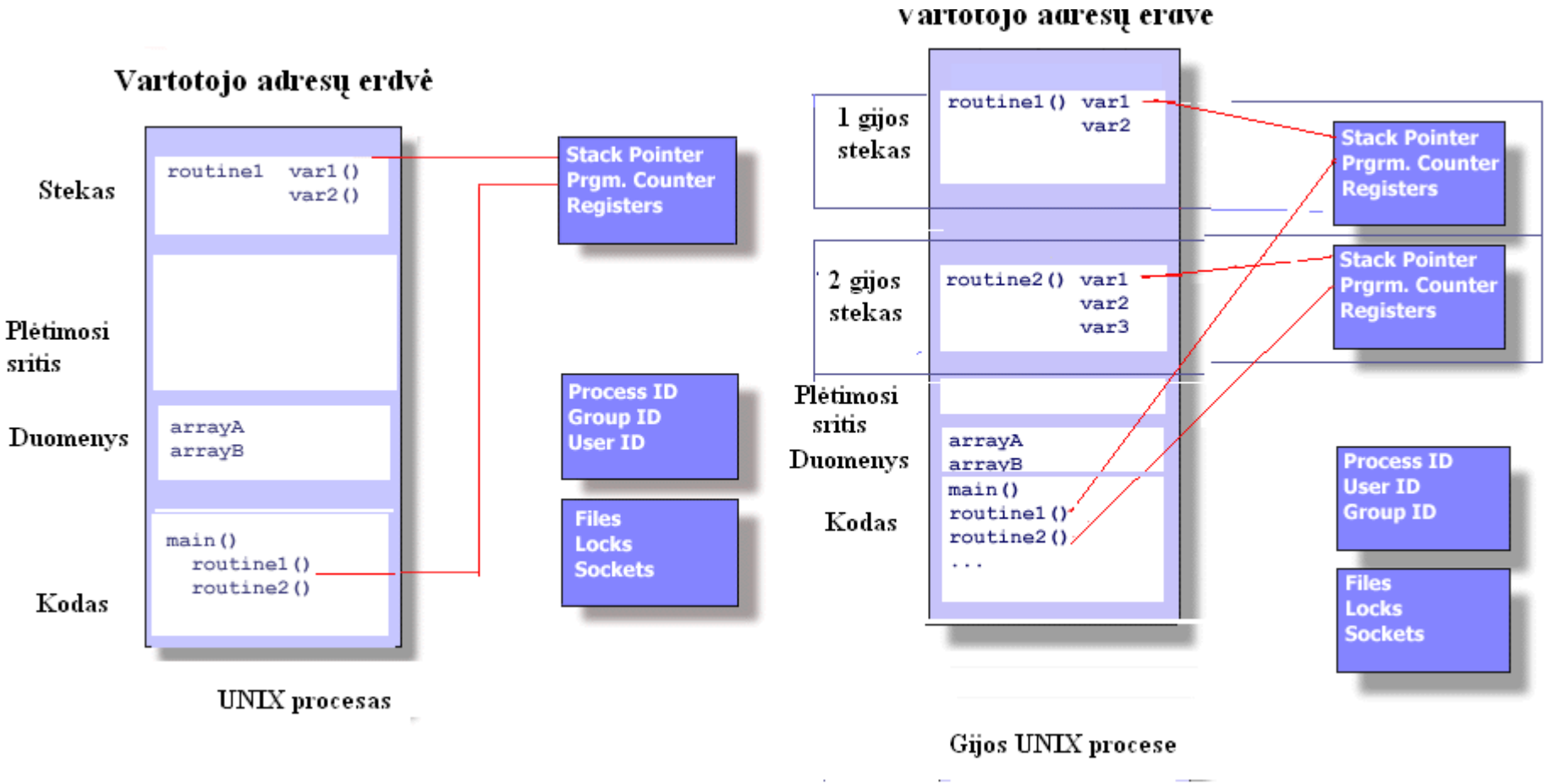
- Programuotojo požiūriu gijos koncepcija gali būti sutapatinama su nepriklausomai nuo pagrindinės programos vykdomos "procedūros" koncepcija.

- Modernios OS (Mach, Chorus, NT, modernios Unix) atskiria procesų bei gijų koncepcijas

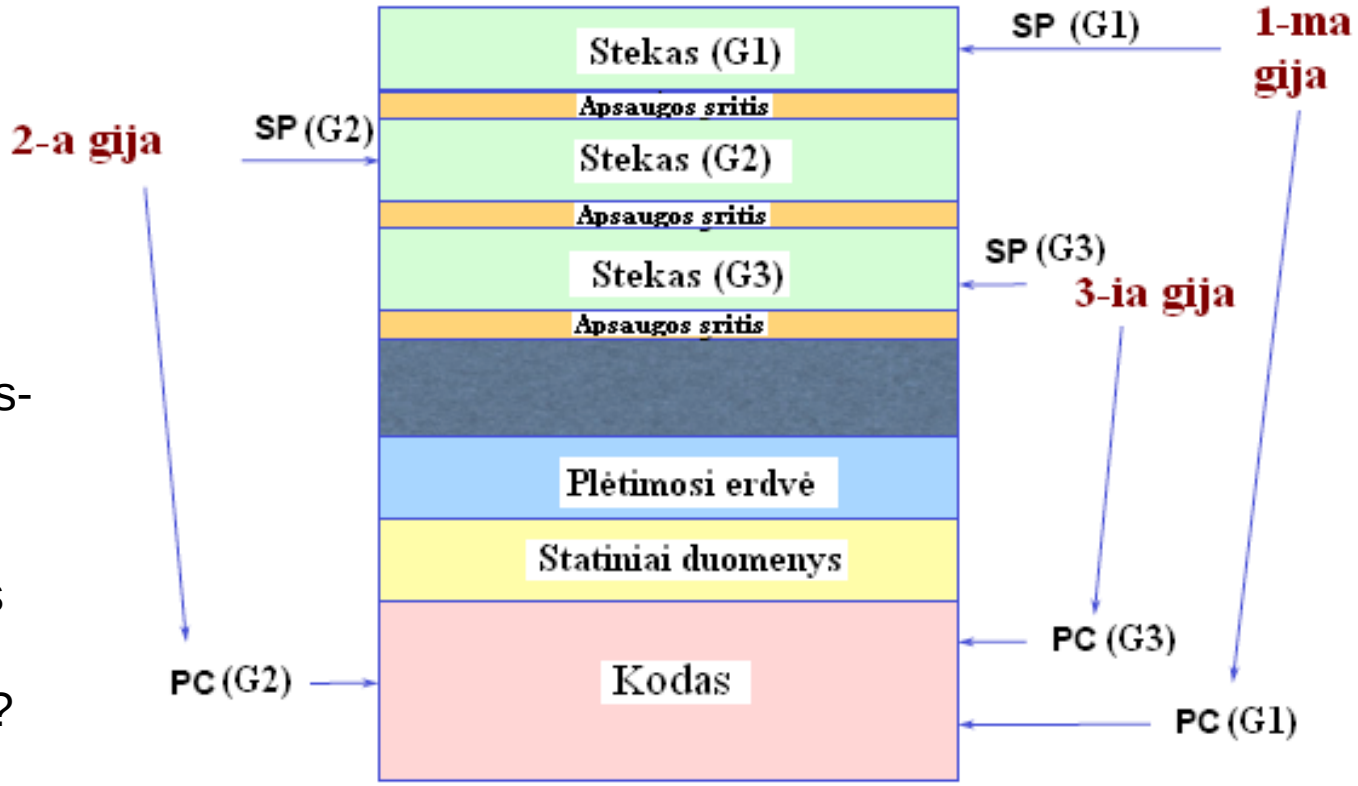
Planavimas

- Kadangi kiekviena gija turi savo vykdymo būvį, tai kiekvienai iš jų turi būti priskiriamas vykdymo resursas - CPU
- Gijos tampa **planavimo** vienetu
  - Procesai tokiu atveju tampa tam tikra ***terpe***, kurioje vykdomos gijos
  - Procesai tampa ***statišku*** elementu, o gijos – dinaminiu vienetu
- Kiekvienas procesas turi bent vieną giją

Gijos ir procesas



Gijos procese



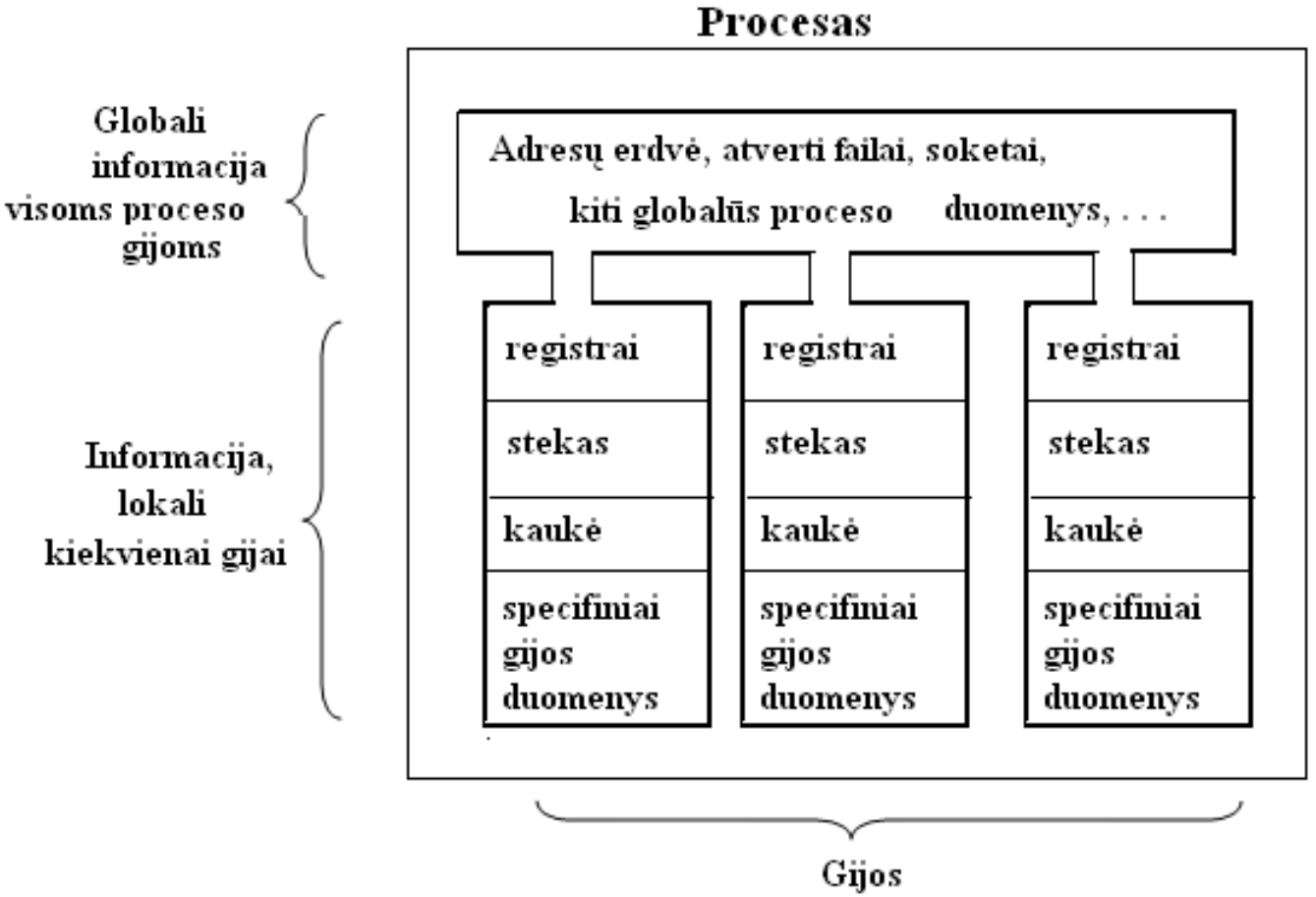
Steko dydis-kas jį nusako?

Kas atsitiks jei stekas persipildys?

Gija

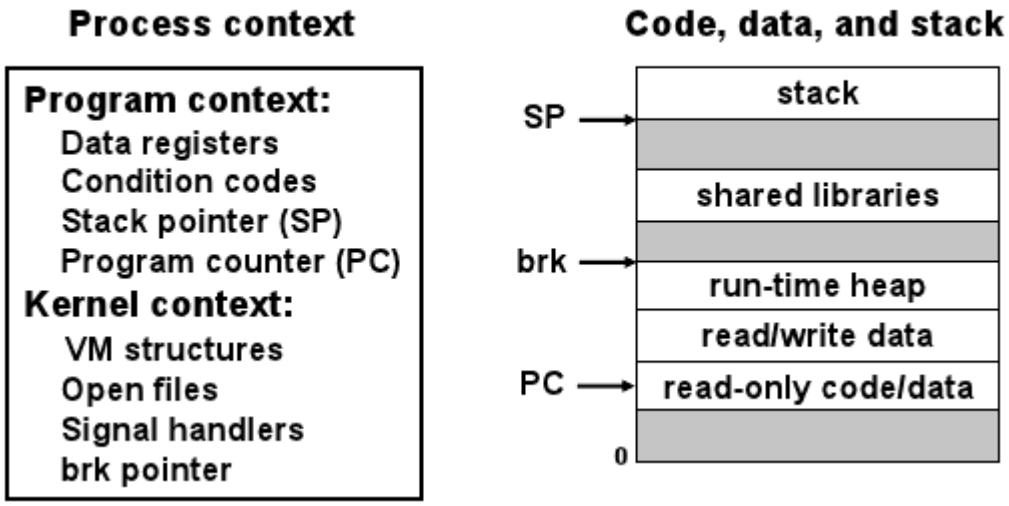
- Savybės:
  - Egzistuoja proceso aplinkoje ir naudoja proceso resursus
  - Turi nepriklausomą vykdymo seką (proceso vykdymo metu)
  - Dubliuoja tik tuos pagrindinius resursus, kurių reikia kad būtų nepriklausomai vykdoma
  - Gali dalintis proceso resursais su kitom gijom
  - “Miršta” jei procesas miršta.
  - Yra lengvasvorė (lightweight), kadangi yra sukuriama paprasčiau.
- Vieno proceso gijos dalinasi bendrais resursais:
  - Vienos gijos daromi pakeitimai matomi kitų gijų.
  - Jei nuorodų reikšmė ta pati – jos rodo į tuos pačius duomenis.
  - Galimas skaitymas-rašymas į tas pačias atmintinės sritis – reikia sinchronizacijos iš programuotojo pusės.

Proceso ir gijos sąryšis



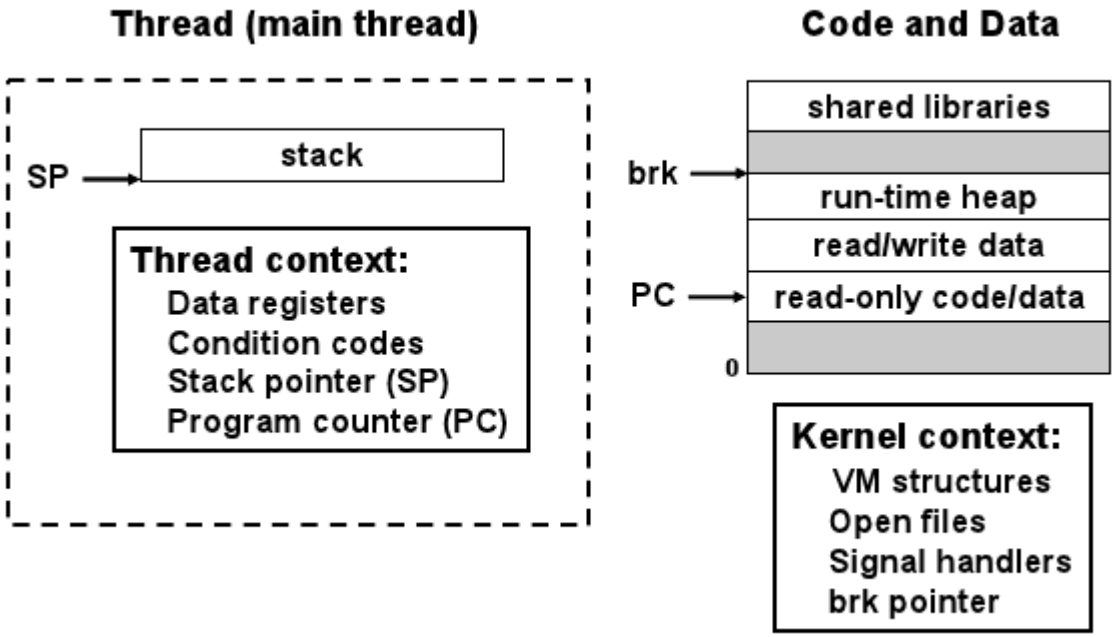
Procesas

- Procesas = proceso kontekstas + kodas, duomenys ir stekas



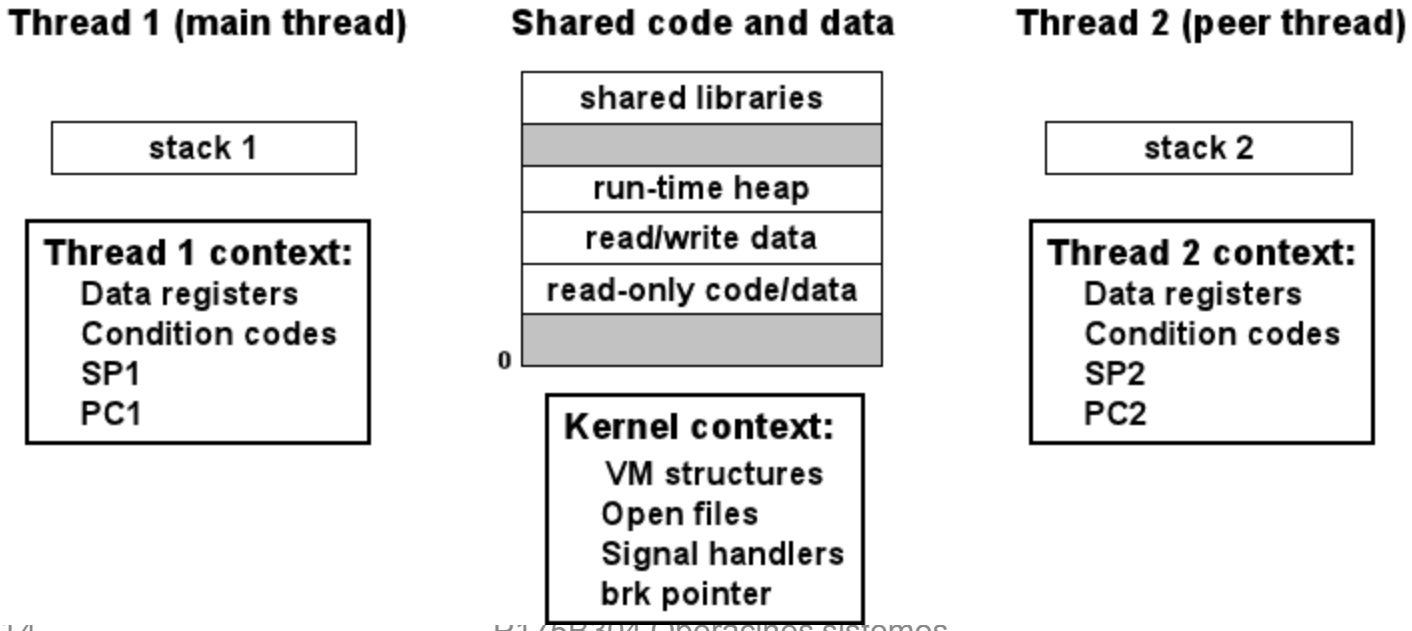
Modernus procesas

- Procesas = gija + kodas, duomenys ir branduolio kontekstas



Procesas su daugeliu gijų

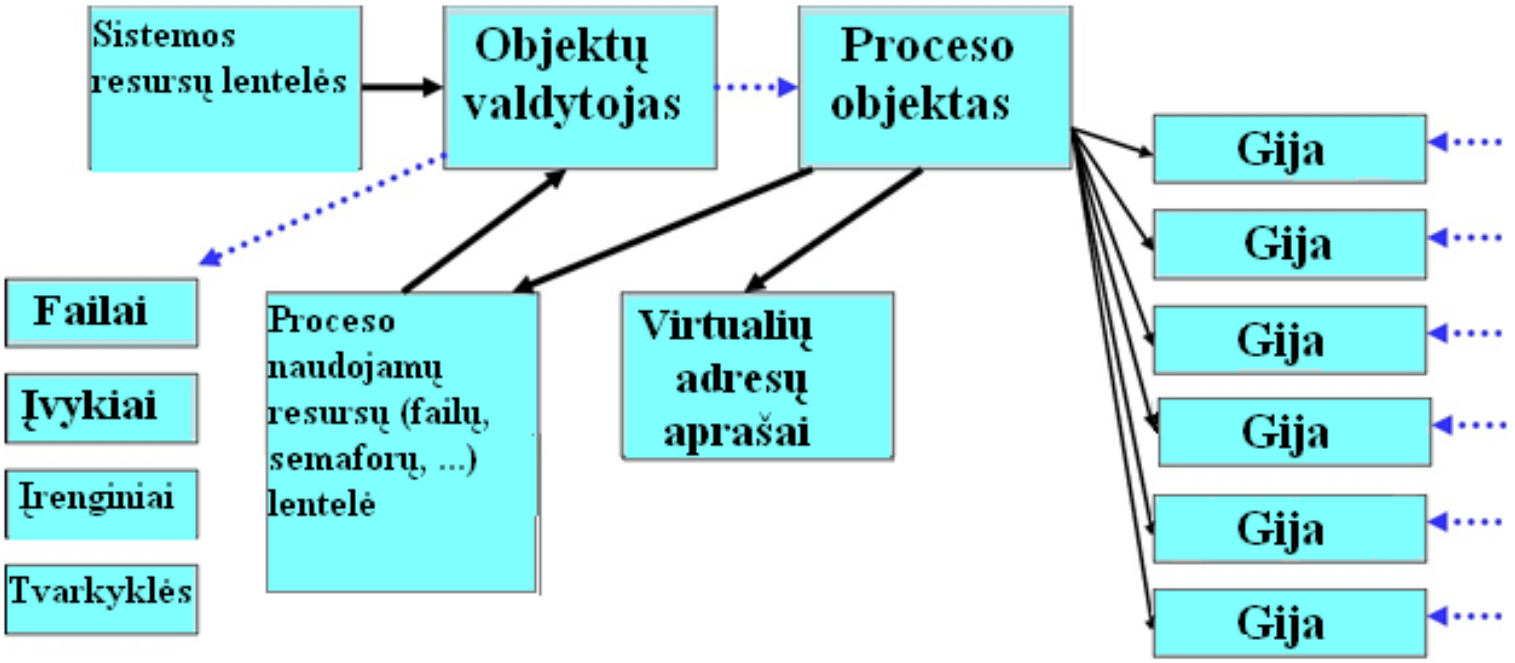
- Kiekvienos gijos vykdymas turi savo vykdymo eigą
- Kiekviena gija turi savą steką
- Kiekviena gija bendrai naudojasi tuo pačiu kodu, duomenimis ir branduolio kontekstu



Procesas, gija (UNIX)

- **Procesas**
  - Proceso ID, proceso grupės ID, vart. ID, ir grupės ID
  - Aplinka
  - Darbinis katalogas.
  - Programos kodas
  - Registrai
  - Stekas
  - Plėtimosi sritis
  - Failų deskriptoriai
  - Signalus apdorojančios f-jos
  - Bendrai naudojamos bibliotekos
  - Tarp-procesinių komunikacijų įrankiai (pranešimų eilės, vamzdžiai, semaforai arba bendrai naudojama atmintinė).
- **Gijos egzistuoja procese ir naudojasi jo ištekliais**
  - Steko rodyklė
  - Registrai
  - Planavimo parametrai (planavimo algoritmas, prioritetas)
  - Grupė laukiančių arba blokuotų signalų
  - Vidiniai, lokalūs gijų duomenys.

Proceso-gijos struktūra Windows sistemoje

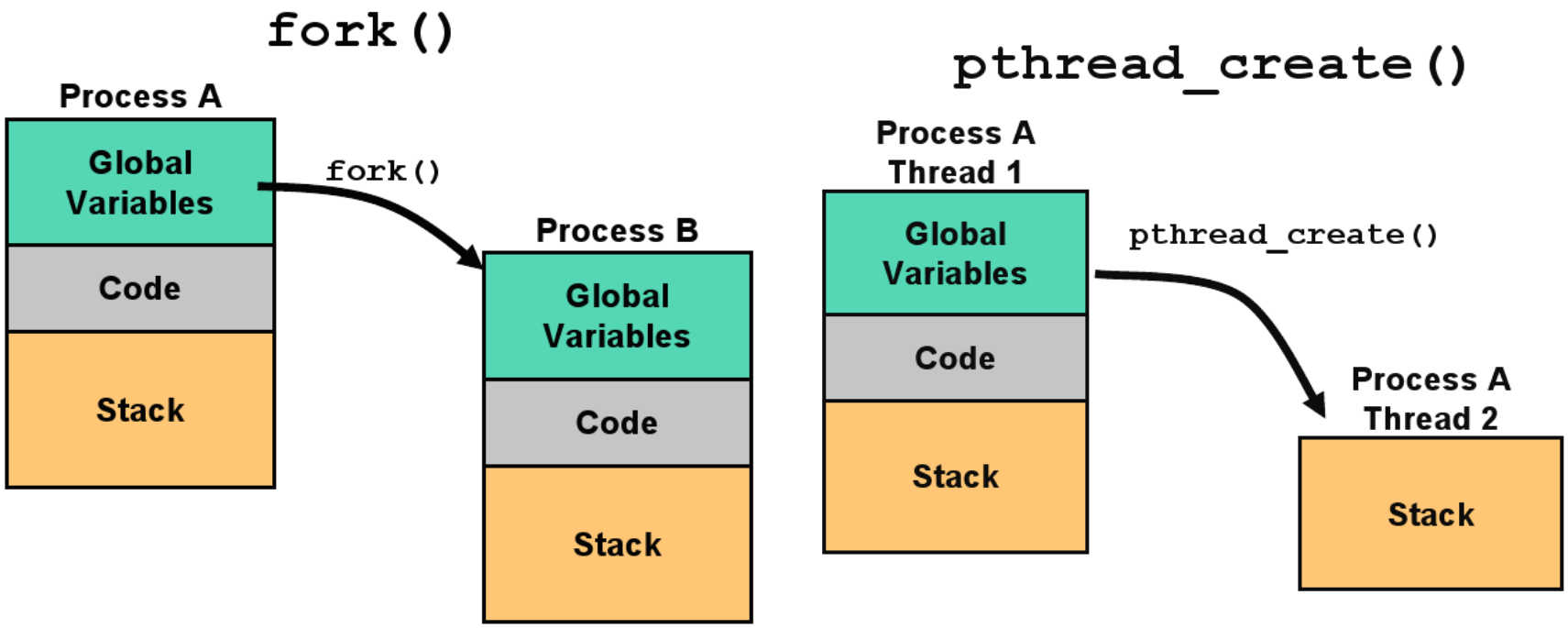


Web serverio pavyzdys

- **while (1) {**
- **int sock = accept();**
- **if ((child_pid = fork()) == 0) {**
- Apdoroti kliento užklausą
- **} else {**
- close (sock)
- **}**
- **}**

- **while (1) {**
- **int sock = accept();**
- **thread_fork(handle_request, sock);**
- **}**
- **}**
- **handle_request(int sock) {**
- Apdoroti užklausą
- **close(sock);**
- **}**

UNIX: naujas procesas ir nauja gija

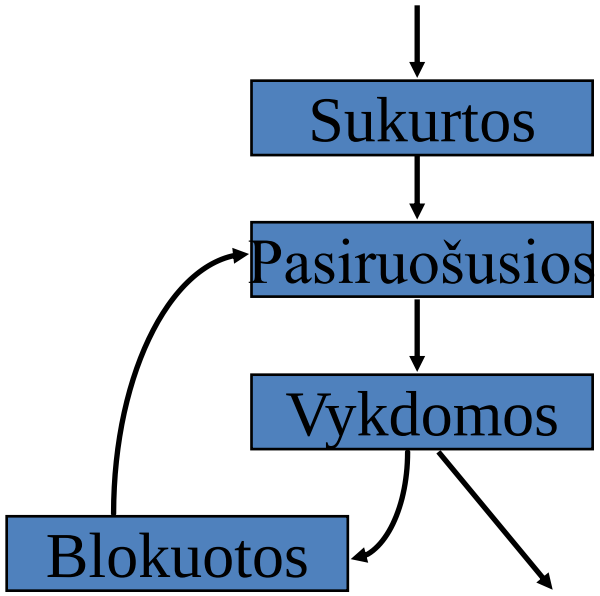


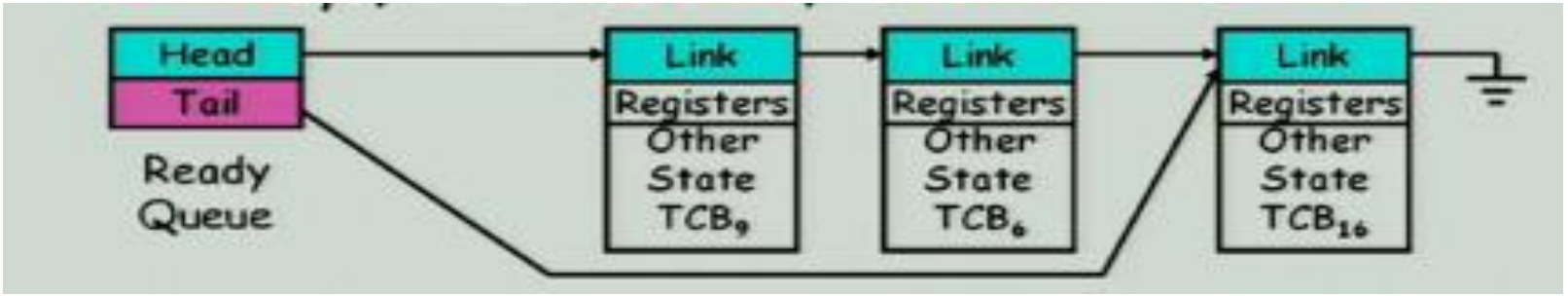
Operation Microseconds
Create unbound thread 52 -
Create bound thread 350
fork() 1700

Gijų ir procesų panašumai/skirtumai

- Panašumai:
  - Kaip ir procesai, gijos gali būti ***sukuriamos***, ***atidedamos*** ar ***užbaigiamos***.
  - Kaip ir procesai, gijos ***dalinasi procesoriumi*** (CPU) ir tik viena gija (esant vienam CPU) yra aktyvi (vykdoma) bet kuriuo laiko momentu.
  - Kaip ir procesai, proceso gijos, esant vienam CPU, yra vykdomos ***nuosekliai***.
  - Kaip ir procesai, gijos gali ***sukurti „vaikus“***.
  - Kaip ir procesų atveju, jei viena ***branduolio*** gija yra ***blokuojama***, gali būti vykdoma ***kita gija***.
- Skirtumai
  - Skirtingai nei procesai proceso gijos ***nėra tarpusavyje nepriklausomos***.
  - Proceso gijos turi ***prieigą prie bet kurio adreso*** proceso erdvėje.
  - Proceso gijos yra projektuojamos siekiant, kad jos ***padėtų*** viena kitai, o projektuojant procesus to nėra siekiama, nes jie gali būti skirtingų vartotojų.

Gijų būsenos

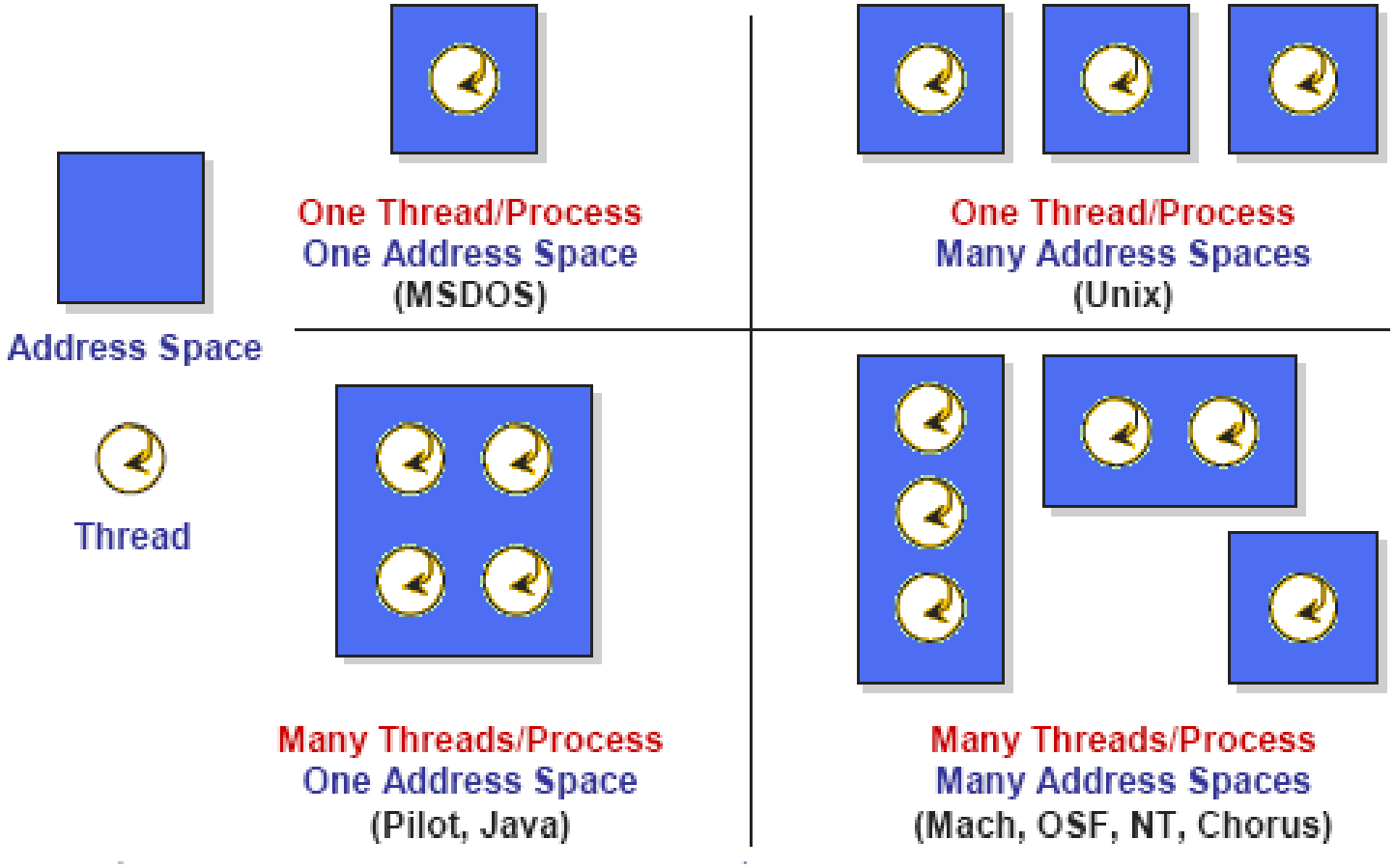




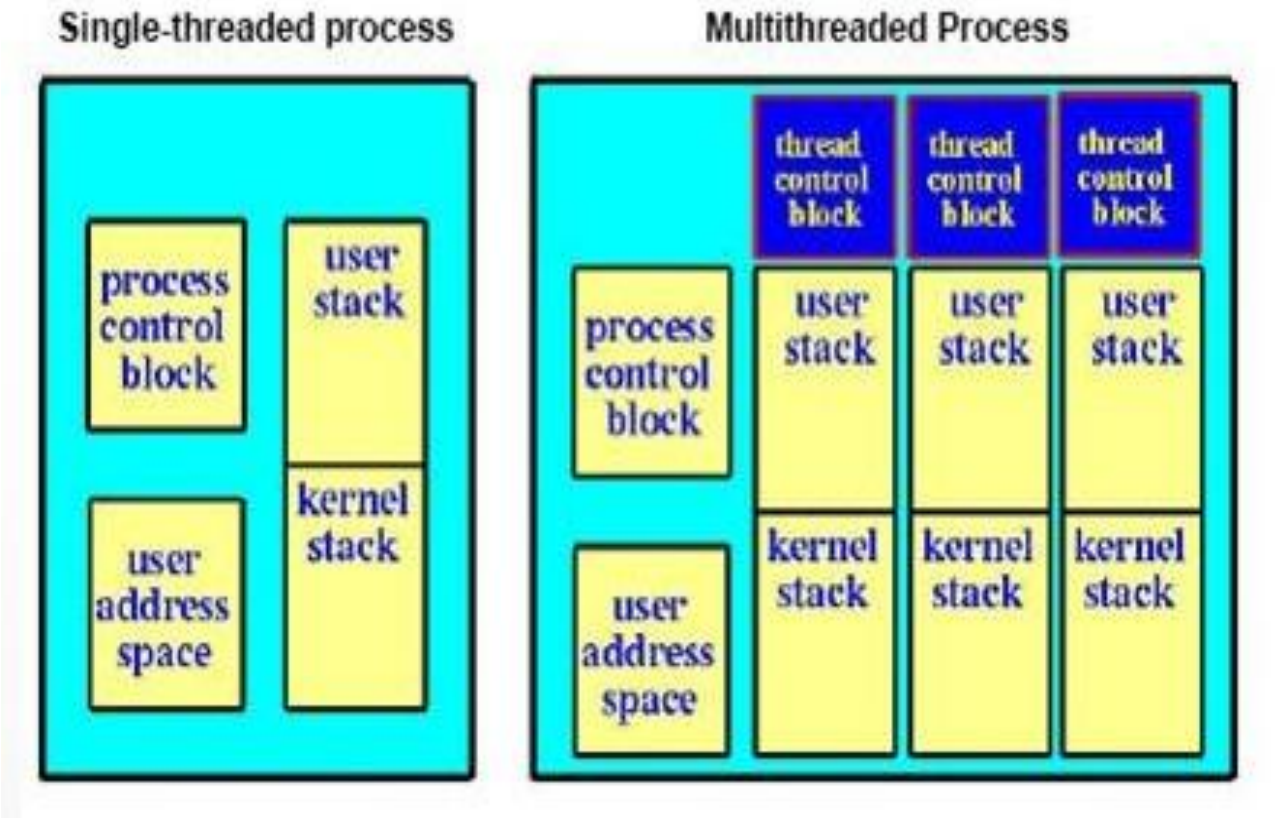
Gijų projektavimo poreikis

- Gijos daug natūraliau atitinka lygiagrečius skaičiavimus: naudodami procesą su daugeliu gijų galime gauti puikų serverį.
- Joms nereikia priemonių, leidžiančių komunikacijas tarp procesų, jos gali komunikuoti per bendrą atmintinę.
- Gijos gali lengvai pasinaudoti nauda, kurią teikia daugiaprocesorinės sistemos.
- Gijos yra pigus, pseudolygiagretumą (arba lygiagretumą daugelio procesorių atveju) užtikrinantis mechanizmas ta prasme, kad:
  - Gijoms reikia tik **steko** ir **atmintinės** srities registrams, tai jas lengviau sukurti.
  - gijoms yra nereikalinga nauja adresų erdvė, globalių duomenų, programos kodo ar naujų operacinės sistemos išteklių.
  - **Konteksto perjungimas** yra žymiai greitesnis pereinant nuo gijos prie gijos nei nuo proceso prie proceso. Tai užtikrinama todėl, kad tenka išsaugoti (arba atstatyti) tik PC, SP ir registrus.

Thread Design Space



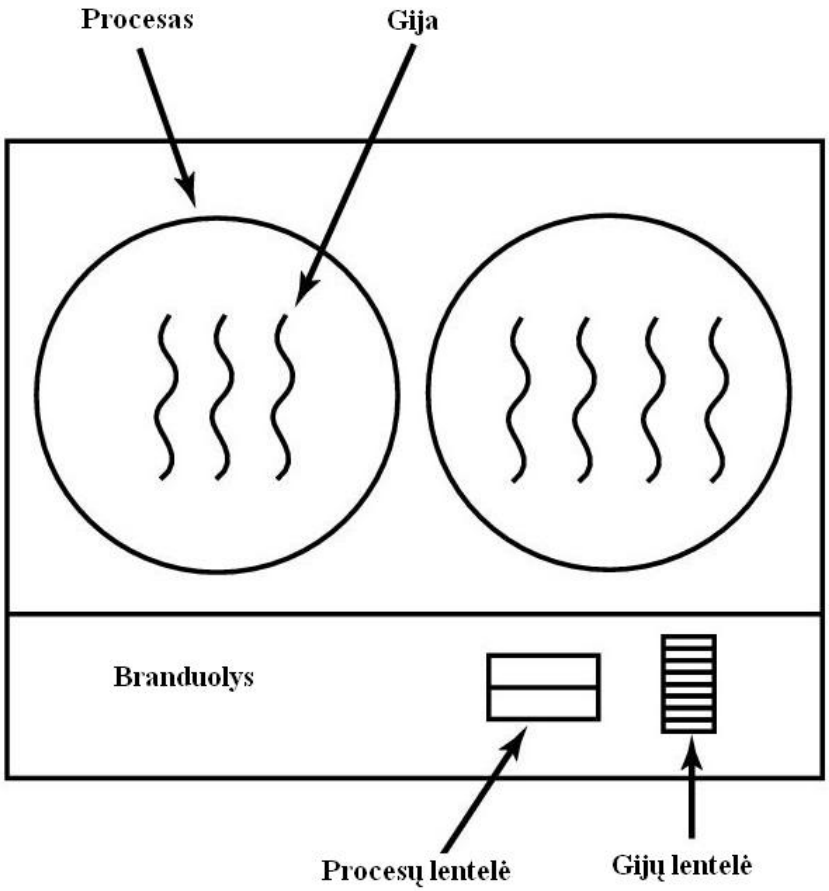
Procesai ir gijos



Gijų realizacijos

- Egzistuoja trys galimos gijų realizacijos. Tai:
  - ***Branduolio*** *lygio gijos*
    - Naudingos esant lygiagretaus darbo galimybei (daug procesorių, branduolių)
    - pav.: *Windows 9x/NT/2000, Linux, Solaris,Tru64 UNIX, MacOS X*
  - ***Vartotojo*** *lygio gijos*
    - Planuojamos vartotojo procese
    - Branduoliui nematomos
    - pav.: *POSIX Pthreads, Win32 threads, Java threads*
  - ***Kombinacija*** *tarp vartotojo lygio ir branduolio lygio gijų*

Branduolio lygio gijos



OS valdo gijas bei procesus
• Visos operacijos, susijusios su gijomis įdiegiamos branduolyje
• OS planuoja gijų vykdymą
• OS valdomos gijos yra vadinamos branduolio lygio gijomis arba lengvasvoriais procesais
• Windows XP
• Solaris: lengvasvoriai procesai (LWP)

Branduolio lygio gijos

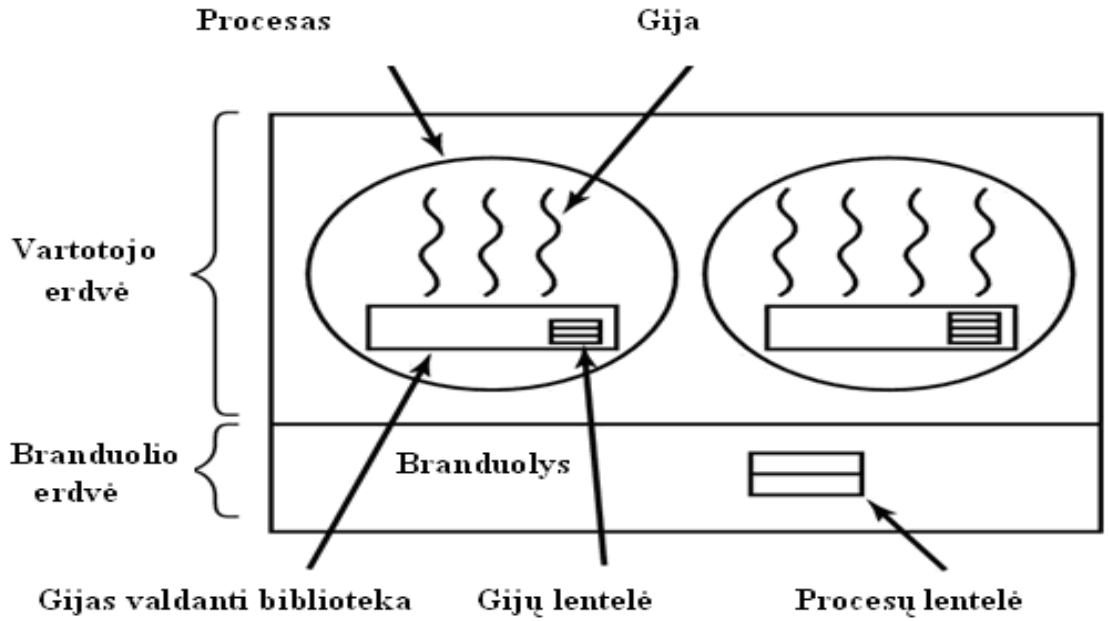
- Gijas nusako:
  - grupė registrų, stekas bei keletas branduolio struktūrų, jos reikalauja mažiau sąnaudų nei procesai.
- Operacinė sistema turi kiekvienai gijai priskirtą deskriptorių ir atskirai planuoja kiekvienos gijos vykdymą

- Branduolio lygio gijos yra palaikomos paties OS branduolio.
  - Branduolys vykdo gijų *sukūrimą, planavimą bei valdymą* .
  - Kadangi gijų valdymą vykdo OS, jų *palaikymas reikalauja papildomų veiksmų (reikalingi sisteminiai kvietiniai)*.
  - Kaip ir procesų atveju, užsiblokavus šio tipo gijai, *blokuojama tik ši gija* ir branduolys gali planuoti kitos gijos vykdymą.

LINUX branduolio lygio gijos

- Tradicinė UNIX sistema deleguoja tam tikras užduotis procesams:
  - Diskų “cach’ų” išvalymas(Flushing), iškėlimas nenaudojamų puslapių į swap sritį, tinklinių susijungimų valdymas, t.t.
  - Yra daug efektyviau šias užduotis vykdyti asinchroniškai
- Kadangi dauguma šių užduočių vykdoma branduolio režime, Linux sistemoje tai pavedama branduolio lygio gijoms.
  - Kiekviena branduolio gija vykdo specifinę branduolio funkciją
  - Kiekviena branduolio gija yra vykdoma tik branduolio režime
  - Kiekviena branduolio gija turi ribotą adresų erdvę

Vartotojo lygio gijos

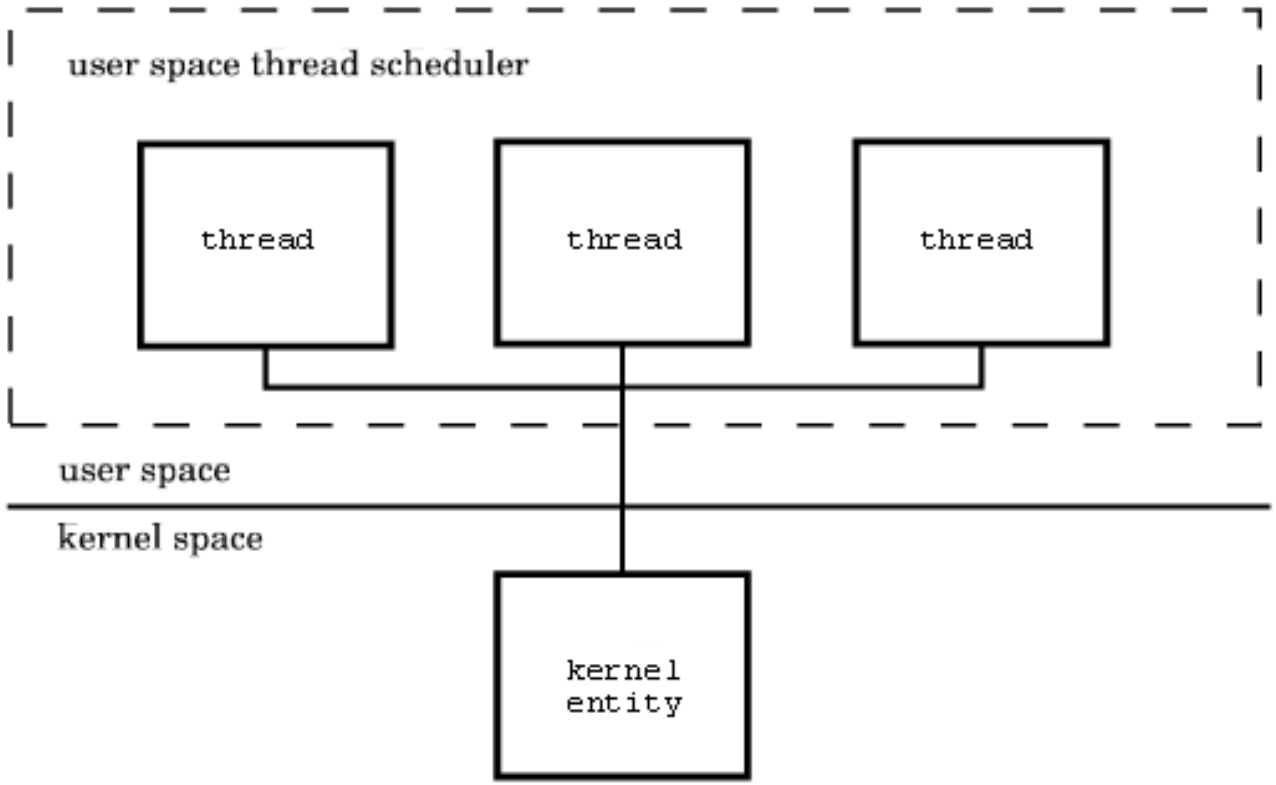


Vartotojo lygio gijos yra įdiegiamos naudojant vartotojo lygio bibliotekas, o ne per sisteminius kvietinius, taigi persijungimo tarp gijų metu nėra reikalo kviesti operacinę sistemą bei vykdyti branduolio pertraukimus. (Gijos greitai sukuriamos ir nesunkiai valdomos).

Naudojant vartotojo lygio gijas, branduolys nežino apie jų egzistavimą.

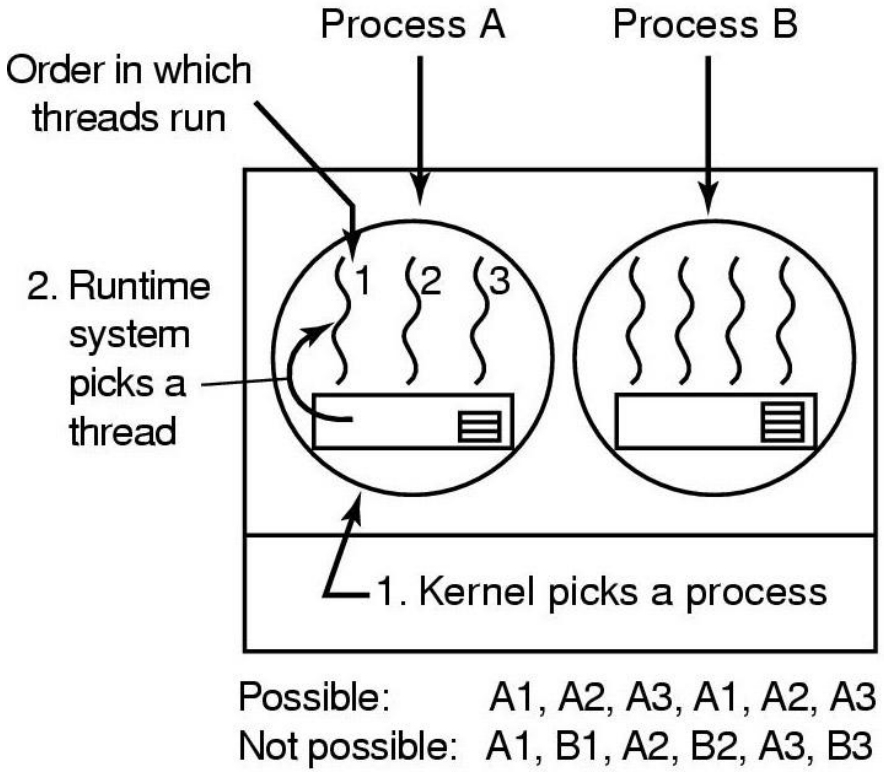
Realizacijos: GNU Portable Threads (Pth) , FreeBSD's userland threads, QuickThreads, Charm++ systemos gijos

Architecture



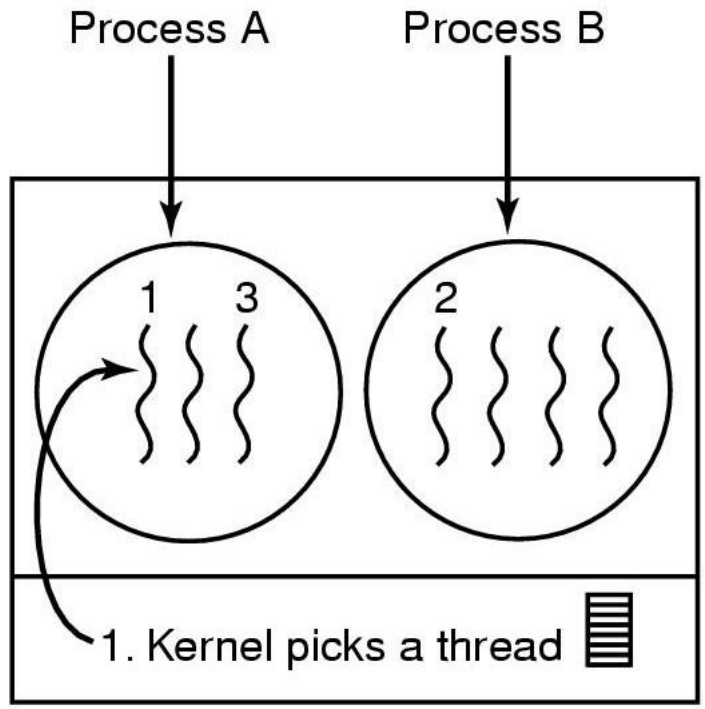
Advantages	*Disadvantages*
Context-switches performed entirely in user mode	Poor performance due to blocking system calls and/or associated code complexity to avoid them
Straight forward implementation	Unable to use multiple processors

Vartotojo lygio gijų planavimas



Galimas vartotojų gijų planavimas :
50-msec laiko kvantas procesui;
Gijos vykdomos po 5 msec

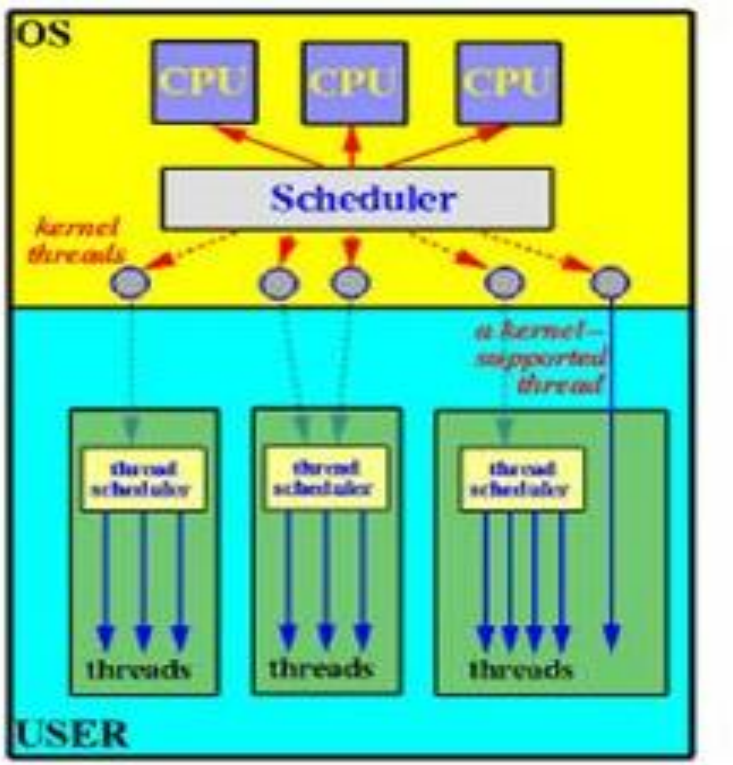
Branduolio lygio gijų planavimas



Possible: A1, A2, A3, A1, A2, A3
Also possible: A1, B1, A2, B2, A3, B3

Galimas branduolio gijų planavimas : 50-msec laiko kvantas procesui; Gijos vykdomos po 5 msec

Gijų planavimas multiprocesorinėse sistemose



Gijų rikiavimas

- Konceptualiu požiūriu: OS dispečeriavimo ciklas atrodo taip

Ciklas {

```
Vykdyti giją();
Išsaugoti CPU būseną (einamosios Gijos kontrolės bloke);
Išrinkti sekančią giją();

Įkelti CPU būseną (naujas GKB);
```

}

Tai begalinis ciklas

Kada bus išeita iš šio ciklo??? (kada tai bus?)

Gijos vykdymas

- Panagrinėkim pirmą dalį: **`Vykdyti giją();`**
- Kaip ji yra vykdoma?
  - Įkeliama būvio informacija (registrai, PC, steko nuoroda) į CPU
  - Įkeliama aplinkos informacija (virtuali atmintinės erdvė, ir t.t)
  - Pereinama prie PC
- Kada dispečeris atgauna kontrolę
- - vidiniai įvykiai: gija grąžina kontrolę geranoriškai
- - išoriniai įvykiai : CPU perėmimas

Vidiniai įvykiai

- Blokavimasis dėl I/O
  - I/O užprašymas tiesiogiai veda prie to, kad CPU yra grąžinamas
- Signalo iš kitos gijos laukimas
  - Gija paprašo laukti (wait) ir tuo grąžina CPU
- Gija vykdo f-ją **`yield()`**
  - gija laisva valia grąžina CPU

Būvio išsaugojimas/atstatymas –konteksto perjungimas

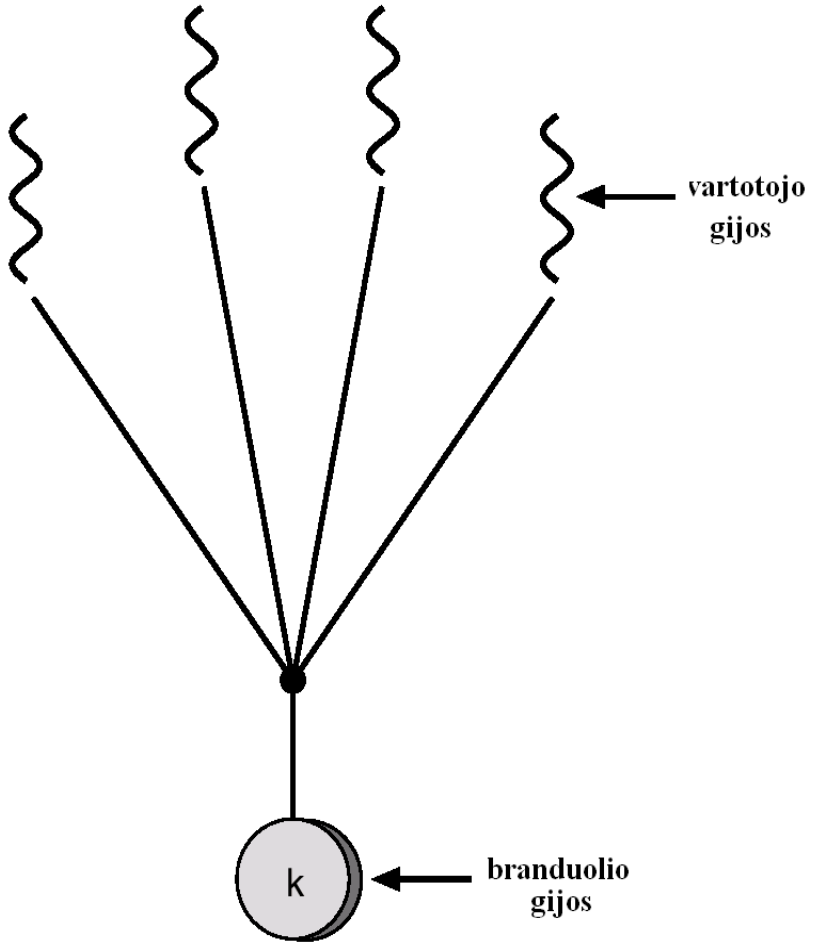
```
Switch (tCur,tNew)  {
/* Unload old thread  */
    TCB[tCur].reg.r7 = CPU.r7;
          …
    TCB[tCur].reg.r0 = CPU.r0;
    TCB[tCur].reg.sp = CPU.sp;
    TCB[tCur].reg.retpc = CPU.retpc;/*grįžimo adresas*/
/*Load and execute new thread */
      CPU.r7 = TCB[tNew].reg.r7 ;
      ...
      CPU.r0 = TCB[tNew].reg.r0 ;
      CPU.sp = TCB[tNew].reg.sp ;
      CPU.retpc = TCB[tNew].reg.retpc ;
```

Modeliai

- 1:1 (One-to-One)
- N:1 (Many-to-One)
- N:M (Many-to-Many)

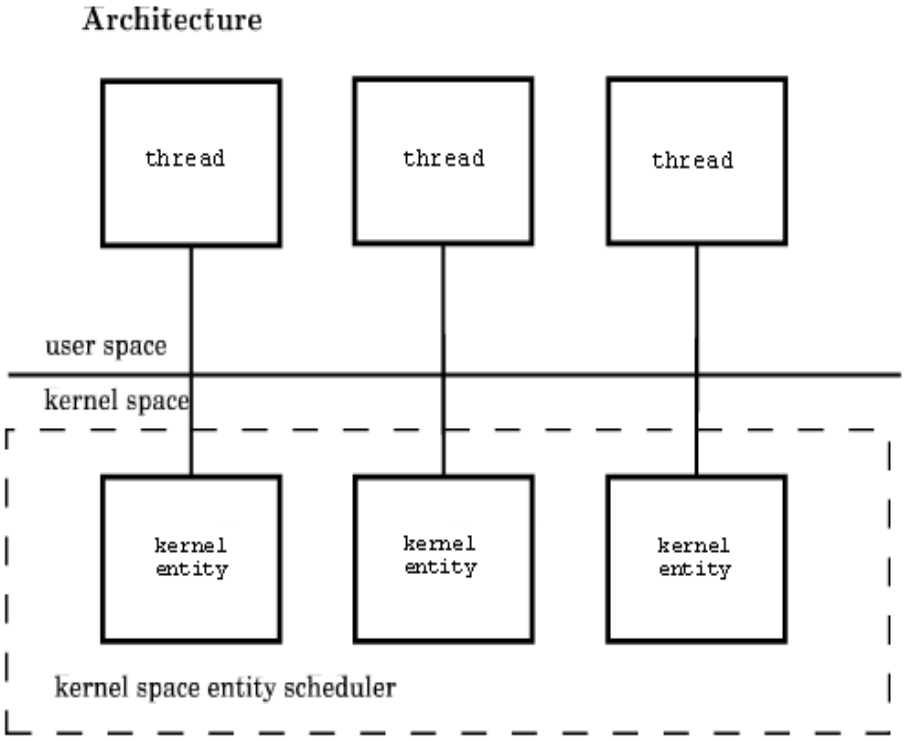
N:1 (Many-to-One)

- Naudojama sistemose, kurios nepalaiko branduolio lygio gijų
  - Paprasta , nebrangi sinchronizacija
  - Nebrangus gijų sukūrimas
  - Efektyviai naudojami resursai
- Trūkumai:
  - Blokavimas blokuoja visą procesą.
  - Multi-procesorinėse sistemose procesas nėra vykdomas greičiau

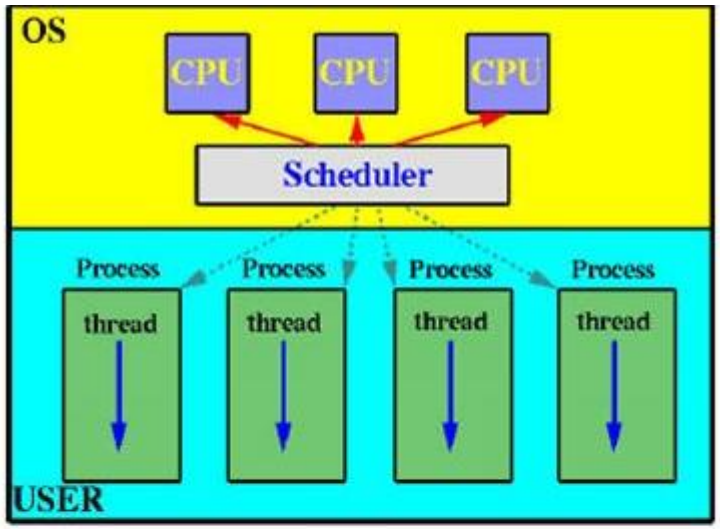


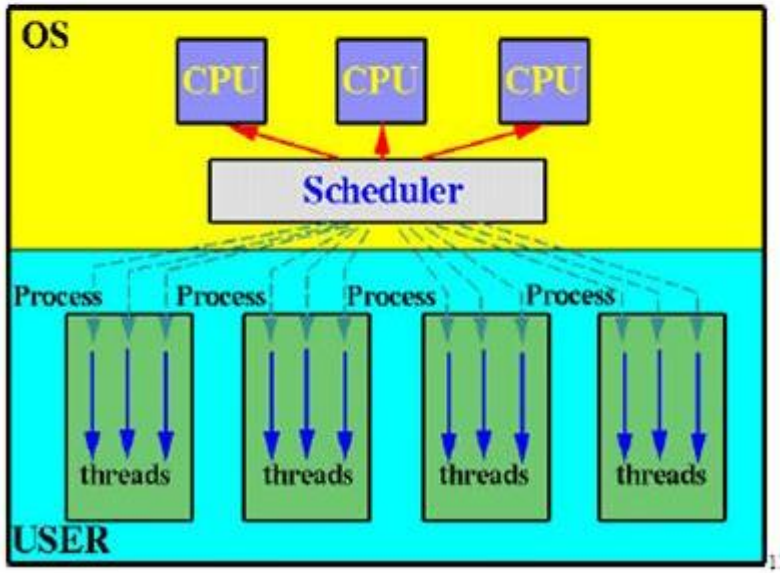
1:1 (One-to-One)

- Kiekvienai vartotojo lygio gijai atitinka viena branduolio gija.
- Gijos gali būt išlygiagretinamos multi-procesorinėse sistemose
- Trūkumai:
  - Sinchronizacijai reikia branduolio įsikišimo
  - Branduolio gijos sukūrimas reikalauja net 3-10 kartų daugiau laiko nei vartotojo gijos
  - Reikalauja branduolio atmintinės stekui ir su gija surištom struktūrom
- Windows 95/98/NT/2000; Linux; Solaris 9 ir vėlesnės



Advantages	*Disadvantages*
Able to use multiple processors	Context switches performed entirely in kernel mode
Improved performance when coping with blocked system calls, etc	Decreased performance with large number of threads



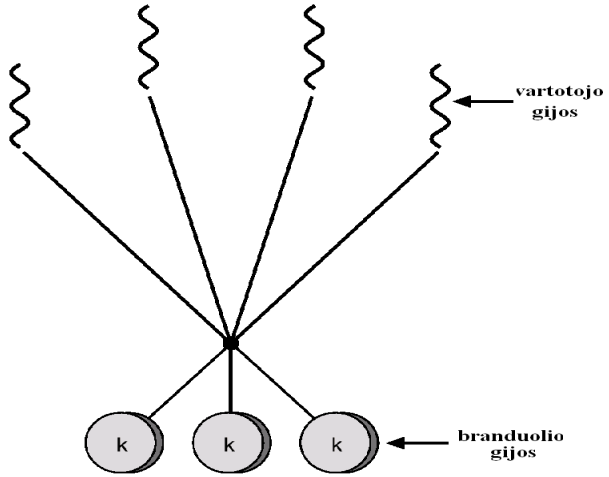


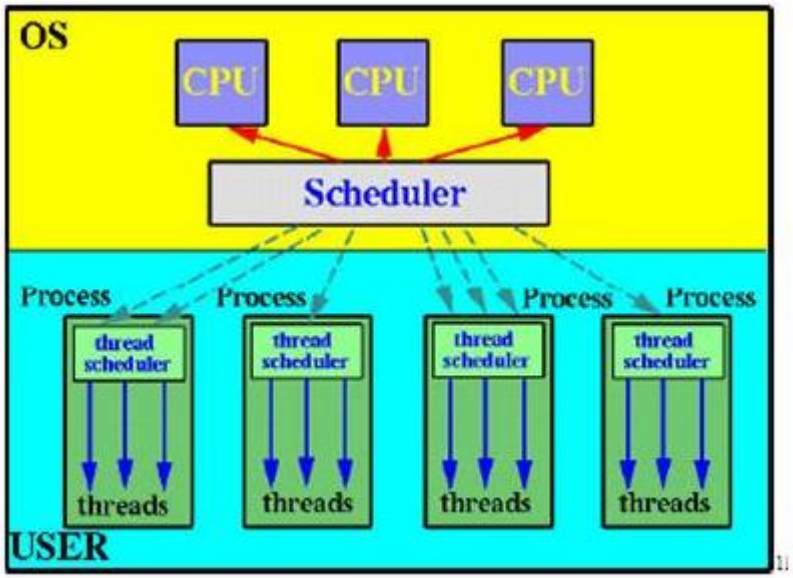
1:1

- Kiekviena nauja vartotojo lygmens gija reikalauja naujos branduolio gijos

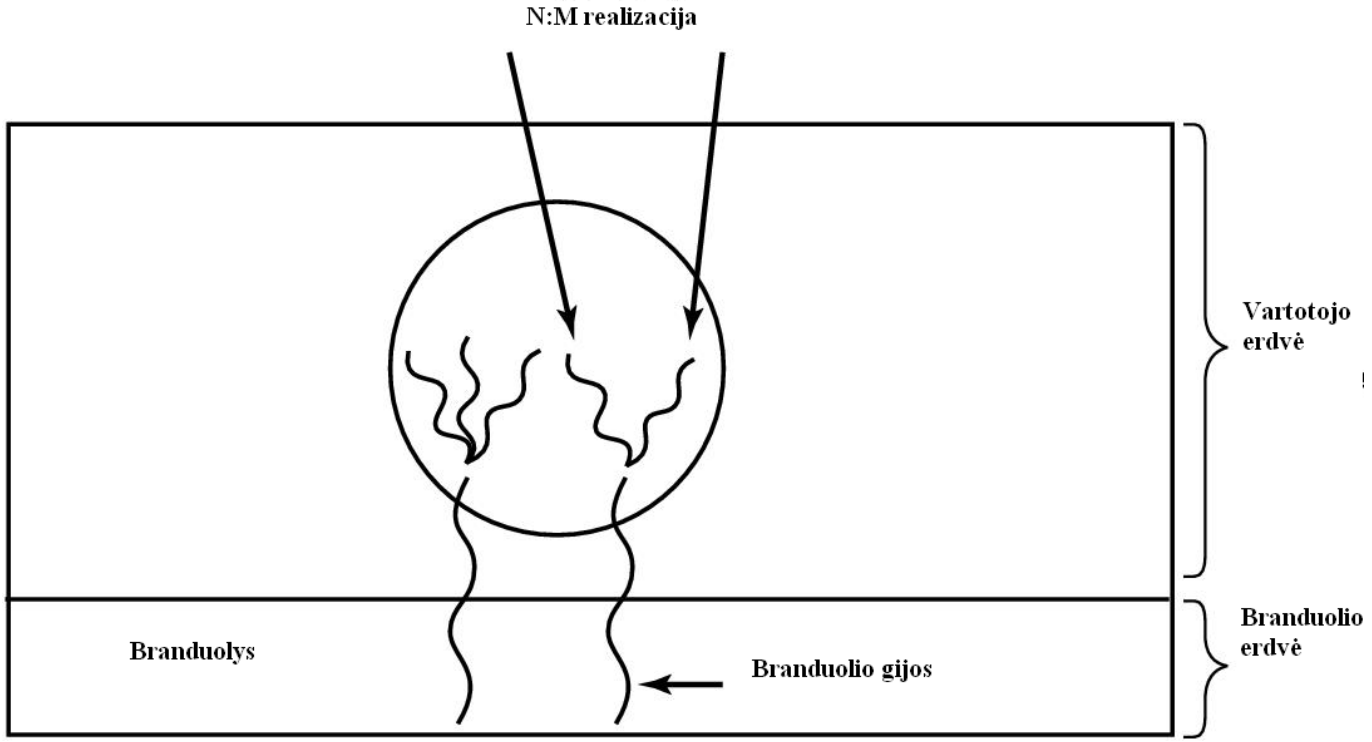
N:M Many-to-Many Model

- Daug vartotojo lygio gijų atitinka daug branduolio lygio gijų
- PAV:
  - Solaris 2, AIX,IRIX, HP-UX
  - Windows NT/2000 su *ThreadFiber* paketu
- Pagrindinis privalumas:
  - Pakankamai nedidele kaina (resursų požiūriu) galima palaikyti daug gijų;
  - sinchronizacija gali būti sprendžiama vartotojo lygyje





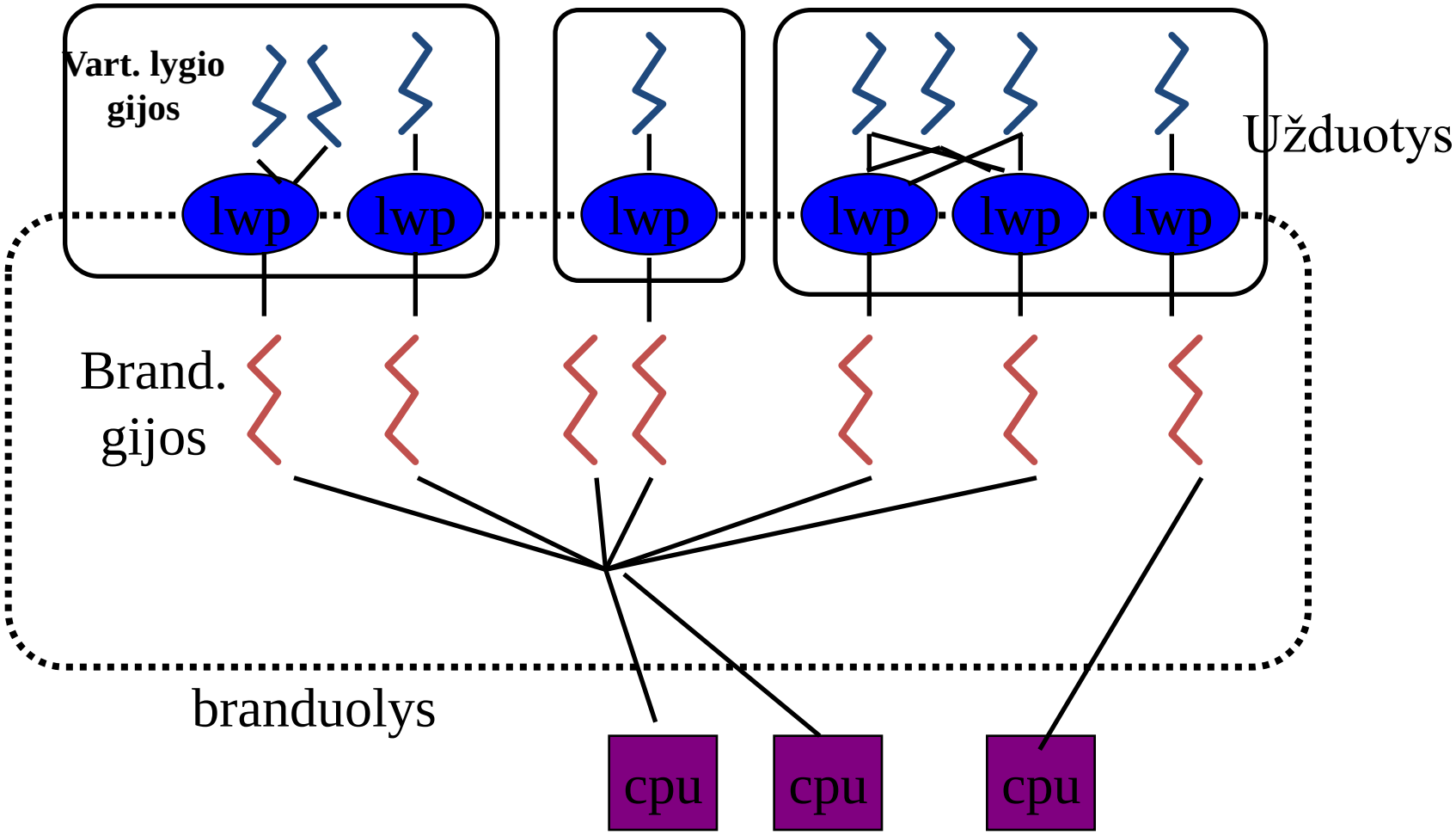
Hibridinė realizacija



kiekviena branduolio lygio gija gali vykdyti persijungimus tarp vartotojo lygio gijų.

Vartotojo gijos gali “klajoti tarp branduolio gijų, arba pririšamos “kietai” prie kažkurios branduolio gijos, ją sukuriant.

Solaris 2


Vart. lygio
gijos
lwp
lwp
lwp
lwp
lwp
lwp
Užduotys
Brand.
gijos
branduolys
cpu
cpu
cpu

N:M Solaris OS

- **Vartotojo lygio gijų** *planavimą* tvarko planavimui skirta vartotojo lygio gija, sukurta gijų bibliotekos.
- Planavimui naudojama tam tikra **prioritetinė disciplina**
- Taikomajai programai leidžiama “paleisti” **tūkstančius** gijų, neapkraunat branduolio.
- Branduolys nežino apie jas, kol jos nesurišamos su lengvasvoriu procesu.
- Planavimo gija vykdo gijų surišimą su LPW procesu – jis turi tam tikrą statusą sistemoje.
- Kiekvienam LPW atitinka viena branduolio gija, tačiau branduolio gijai nebūtinai turi atitikti LPW

Pthread

- IEEE POSIX Section 1003.1c
- IEEE -Institute of Electric and Electronic Engineering
- POSIX (Portable Operating System Interface)
- Pthread tai standartizuotas modelis, nusakantis *gijų kūrimą, valdymą ir sinchronizavimą.*
  - *API nusako bibliotekos elgesį, o realizacija priklauso nuo projektuotojų.*
  - *Naudojama UNIX operacinėse sistemose.*
  - *Tai C funkcijų rinkinys*.

Pthreads API

- Pthreads sąsaja (API) yra apibrėžta ANSI/IEEE POSIX 1003.1 - 1995 standartu.
- Paprogramės, kurios sudaro Pthreads API yra suskirstytos į 3 grupes:
  1. ***Gijų valdymas:*** jų sukūrimas, priskyrimas joms veiksmų, jų sujungimas, gijų atributų nusakymas. . .
  2. ***Sinchronizacija (Mutexes):*** sudaro f-jos skirtos veiksmams su kritine sekcija.
  3. ***Fukcijos, surištos su sąlyginiais kintamaisiais:*** tai funkcijos, kurios aprašo komunikacijas tarp gijų, susijusių su “mutex” situacija.

Gijų sukūrimas

- Pradžioje main() programa atitinka vieną giją. Kitos gijos turi būti sukuriamos.
- Funkcija:

pthread_create (thread, attr, start_routine, arg)

  - Sukuria naują giją ir padaro ją vykdomąja, visos gijos yra lygiavertės.
  - gijos ID grįžta per *thread* argumentą.
  - *start_routine:*
    - tai C kalbos f-ja, kurią vykdys sukurta gija.
  - arg :
    - Tai argumentas, perduodamas *start_routine* per *arg*.

Gija baigiasi

- Pthread nusako keletą galimybių :
  - Gija baigia iškviestos funkcijos veiksmus
  - Sutinkamas kvietinys pthread_exit
  - Gijos veiksmus nutraukia kita gija per funkciją pthread_cancel
  - Baigiasi visas procesas įvykdant exec ar exit

Pavyzdys

```
#include <pthread.h>
#define NUM_THREADS 5
    void *PrintHello(void *threadid)
    { printf("\n%d: Hello World!\n", threadid);
      pthread_exit(NULL);
    }
int main (int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc, t;
      for(t=0;t < NUM_THREADS;t++) {
       printf("Creating thread %d\n", t);
       rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
       if (rc) {
          printf("ERROR; return code from pthread_create() is %d\n",
      rc); exit(-1);
       }
      }
      pthread_exit(NULL);
}
```

Procesas ar gija

- Visus veiksmus galima realizuot procese:
  - Mainams tarp procesų -bendrai naudojama atmintinė, IPC
- Procesų perjungimas sudėtingesnis nei gijų
- Atskirų procesų atveju geresnė apsauga
- Gijos suteikia lygiagretaus projektavimo galimybes – tačiau iškyla sinchronizacijos problemos:
  - Kritinės sekcijos
  - Lenktynių situacijos

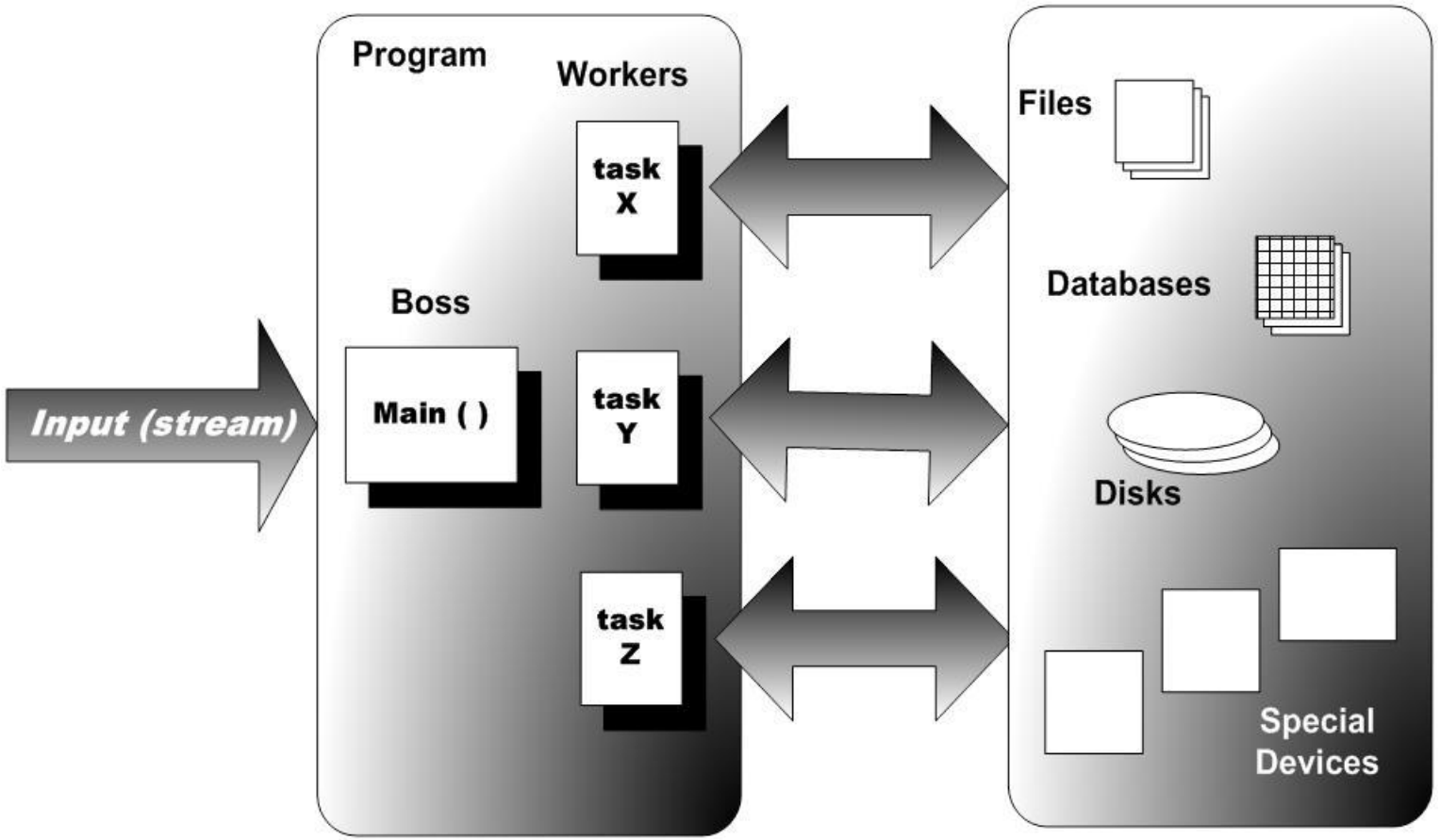
Palyginimas

- Vartotojo lygio gijos:
  - Efektyvesnės sukūrimo požiūriu – tai tiesiog kaip procedūros kvietimas
  - Vartotojas gali kontroliuoti jų planavimo algoritmą
  - Tačiau užsiblokavus vienai – blokuojasi visas procesas
- Branduolio lygio gijos
  - Vienos gijos blokavimas neliečia kitų gijų
  - Planuojama kaip ir procesai

Gijomis paremtų programų projektavimas

- Naudojami keli modeliai:
  - ***Valdytojo/ darbininkų (Manager/worker):***
    - Valdytojo gija skiria darbus (įėjimo duomenis) kitoms gijoms.
  - ***Grandinėlė (Pipeline)***
    - Užduotis sudaloma į smulkesnes, kurios atliekamos viena po kitos, skirtingų gijų (pvz automobilio surinkimo linija)
  - ***Lygiavertės gijos (peer):***
    - Panašu į valdytojo-darbininkų, tik pagrindinė gija po gijų sukūrimo dalyvauja darbe.

Valdytojo/ darbininkų Manager/Worker modelis

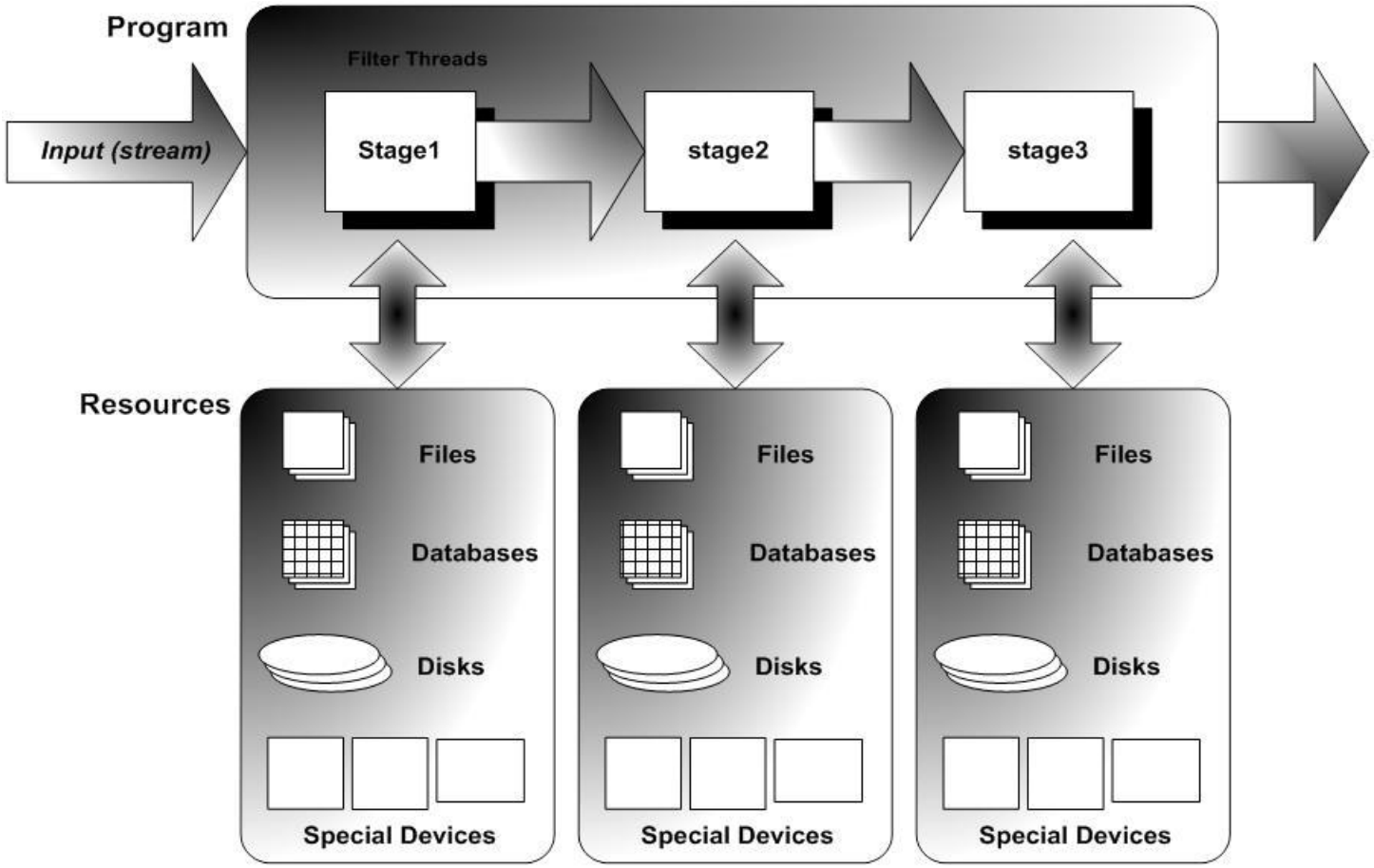


Manager/Worker modelis

- Pavyzdys (manager/worker model programa)

```
main ( void )  /* the manager */
{
   forever  {
        get a request;
        switch request
            case X : pthread_create ( … taskX);
             case Y : pthread_create ( … taskY);
             ……..
   }
}
taskX ()  /* Workers processing requests of type X  */
{
   perform the task, synchronize as needed if accessing shared
   resources;
   done;
}
```

Grandinėlės (Pipeline) modelis



Pipeline modelis

- Pavyzdys (pipeline model io programa)

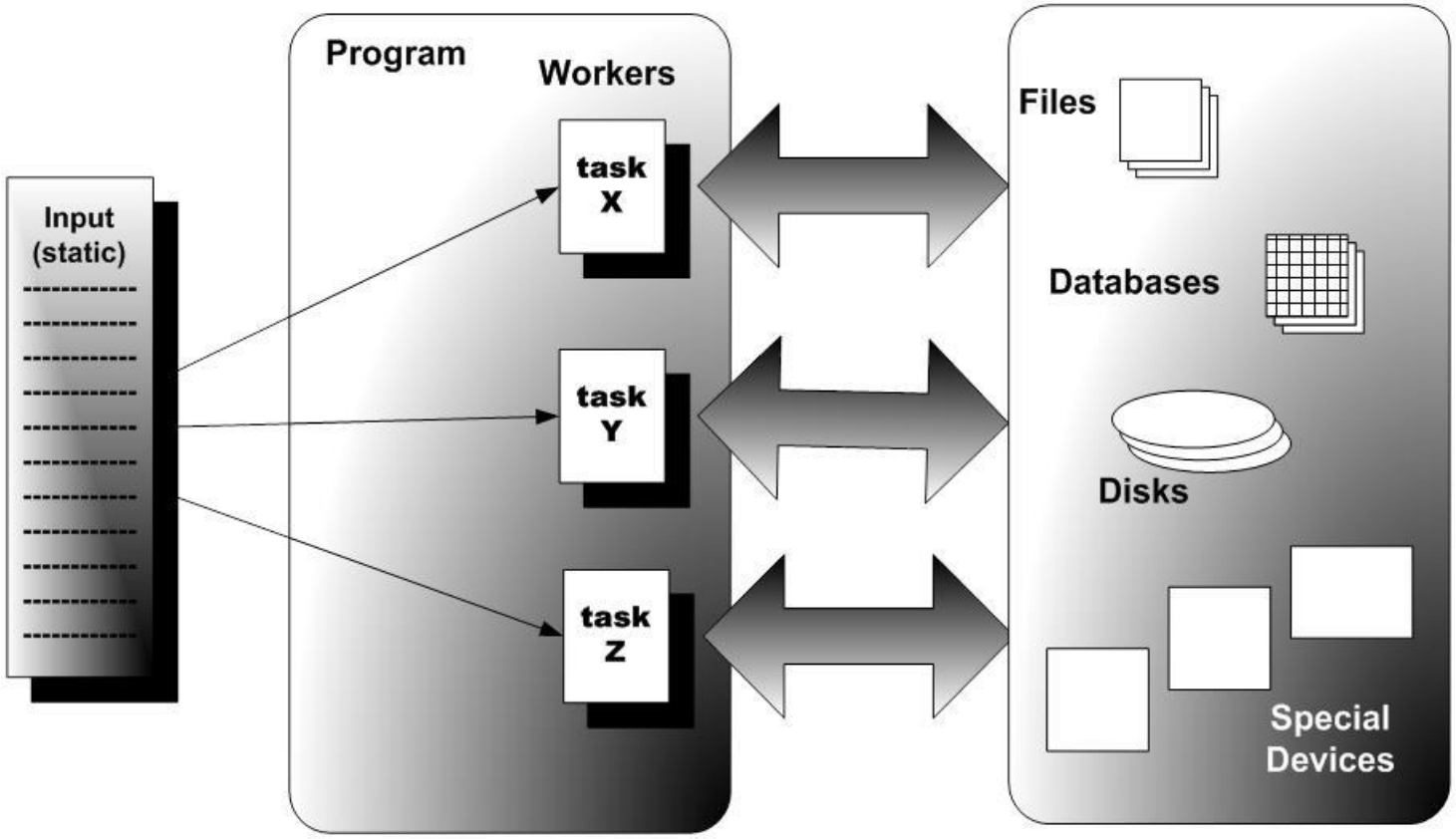
```
main (void)
{
   pthread_create( …stage1 );
   pthread_create( …stage2);
   …
   wait for all pipeline threads to finish;
   do any clean up;
}
Stage1 ()
{
   forever {
         get next input for the program;
         do stage1 processing of the input;
         pass result to next thread in pipeline;
   }
}
```

Pipeline modelis

- Pavyzdys (pipeline model io programa) –tęsinys

```
Stage2 ()
{
   forever {
        get input from previous thread in pipeline;
        do stage2 processing of the input;
       pass result to next thread in pipeline;
   }
}
stageN ()
{
   forever {
       get input from previous thread in pipeline;
       do stageN processing of the input;
      pass result to program output;
   }
}
```

Lygiaverčių gijų modelis



Peer modelio pavyzdys

- Pavyzdys (peer model programa)

```
main (void)
{
   pthread_create ( …thread1 … task1 );
   pthread_create ( …thread2 … task2 );
   …
   signal all workers to start;
   wait for all workers to finish;
   do any clean up;
}
task1 ()
{
   wait for start;
   perform task, synchronize as needed if accessing shared
   resources;
   done;
}
```