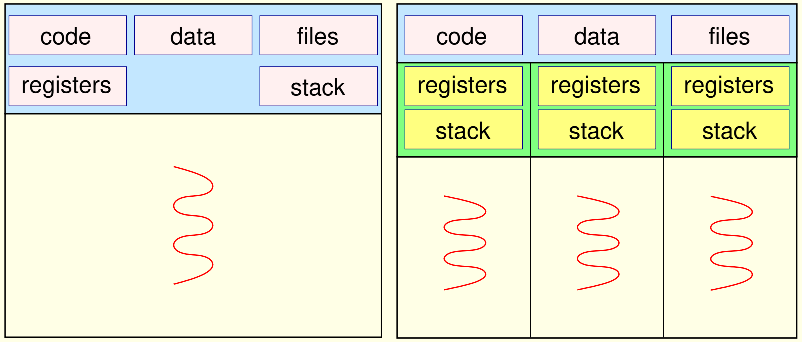
Lõimed

Ülevaade

- Lõime mõiste
- Lõimede mudelid
- Probleemid lõimedega seoses
- Pthreads
- Solarise lõimed
- Windows 2000 lõimed
- Linuxi lõimed
- Java lõimed

Ühe- ja mitmelõimelised protsessid



Lõimed: kolm vaatepunkti

- Abstraktsioon algoritmide esitamiseks
- Programmeerimisliides
- Operatsioonisüsteemi ajajaotusühik

Lõimed – milleks?

- Reageerimiskiirus – parema interaktiivsuse saamiseks võib kasutajaliidese ja muu töö eraldi lõimedesse panna
- Ressursside jagamine – aadressipiirkond programmi mitme täitja poolt kasutatav
- Kokkuhoid – hoiame kokku jagatud ressursside, uute protsesside loomise ning kontekstivahetuste pealt
- Mitme protsessori ära kasutamine ühe programmi poolt

Kasutaja taseme lõimed

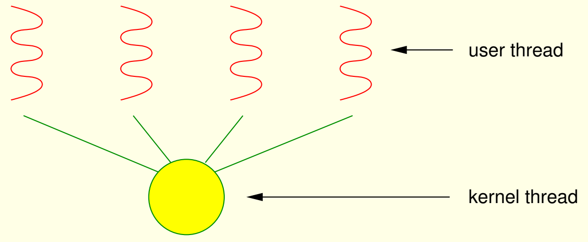
- Lõimede haldus realiseeritud kasutaja tasemel teegina
- Tuum ei tea midagi, tema jaoks üks ühelõimeline protsess
- Haldus kiire, kuna toimub kasutaja tasemel
- Näiteks
  - POSIXi Pthreads
  - Mach'i C-threads
  - Solarise lõimed (UI-threads)
  - FreeBSD oma lõimed
  - Windows NT fiibrid (kiud - osa lõimest)

Tuuma lõimed

- Toetatud tuuma poolt, tuum jagab nende lõimede vahel aega
- Haldus kulukam kui kasutaja tasemel
- Näiteks
  - Windows 95/98/NT/2000
  - Solaris
  - Tru64 UNIX
  - BeOS
  - Linux

Lõimede mudelid

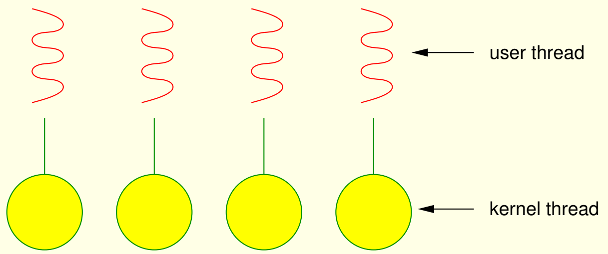
- Mitu-ühele — protsessi kõigi kasutaja tasemel lõimede jaoks on üks tuuma lõim



  - Haldus kiire, kuna toimub kasutaja tasemel
  - Korraga saab töötada kuni üks lõim
  - Blokeeruva süsteemifunktsiooni kasutamisel blokeerub kogu protsess

Lõimede mudelid

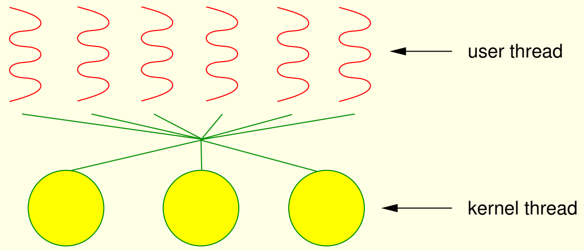
- Üks-ühele — igale kasutaja taseme lõimele vastab tuuma lõim



  - Haldus kulukam
  - Saab kasutada mitut protsessorit
  - Saab korraga teha mitut blokeeruvat operatsiooni
  - Ei maksa liiga palju lõimi teha

Lõimede mudelid

- Mitu-mitmele — protsessi paljudele kasutaja taseme lõimedele vastab palju (kuni samapalju, võib ka vähem) tuuma lõimesid



  - Kombineerib eelmiste head küljed
  - Keerukam

Probleemid lõimedega seoses

- fork() semantika muutus – kas alamprotsess tehakse paljulõimeline või kopeeritakse üks lõim?
- exec() semantika muutus – kas asendatakse kogu protsess või üks lõim?
- Signaalide püüdmine – missugune lõim saab mitmelõimelise protsessi signaalid?
- Lõimede tekitamine ja loomine → lõimevaru (*thread pool*)
- Lõimede lõpetamine (*cancellation*): asünkroonne ja sünkroonne
- Lõime kohalikud andmed

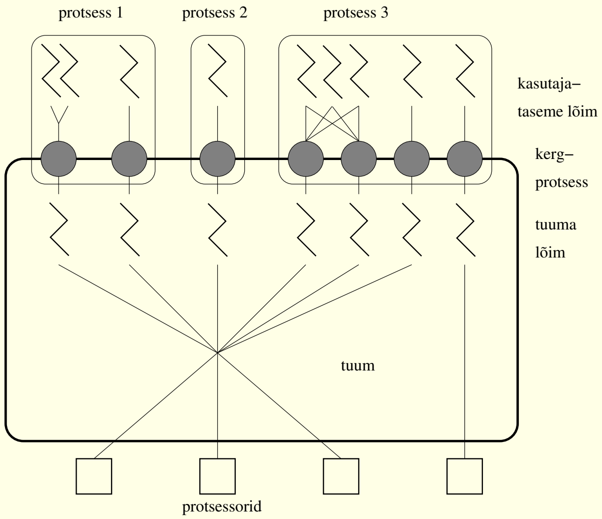
Pthreads

- POSIXi standard (IEEE 1003.1c, POSIX 1c)
- Defineerib programmeerimisliidese lõimedele (C tasemel)
- Kasutaja tasemel, tuuma lõimedega soetud ainult konkreetse implementatsiooni detailide kaudu
- pthread.h, pthread_* funktsioonid
- *De facto* standard UNIXites
- Signaalitöötlus kohmakas, muidu normaalne API

Solarise lõimed

- Läbi ja lõhki lõimeline :)
- LWP (*lightweight process*) – kergprotsess (lõimede liides kasutaja ja tuuma tasemete vahel)
- Igale LWP-le vastab tuuma lõim, tuumas võib lõimesid veel olla
- Kasutajataseme lõimesid saab vajadusel LWP-dega siduda (LWP-sid läheb vaja siis, kui on vaja tuuma teenuseid kasutada)
- Sidumata lõimed jagavad omavahel mingit hulka LWP-sid
- Protsessi poolt kasutatavate LWP-de arvu reguleerib lõimede teek automaatselt (saab ka käsitsi)

Solarise lõimed



Teised

- Windows 2000 lõimed
  - Üks-ühele mudel, *ThreadFiberi* teegi abil ka mitu-mitmele
  - Igal lõimel on ka privaatne andmeala (data, DLL-id jms)
  - Kaks magasini (kasutajataseme ja tuuma taseme jaoks)
- Linuxi lõimed
  - Tuum toetab lõimede tegemist clone() abil – luuakse protsessid, mis jagavad rohkem kui fork() abil tehtud protsessid (aadressiruum, failipidemed, nimeruumid, . . .)
  - clone() abil saadud lõimed on nagu protsessid, kontekstivahetus sama, igal oma PID jne.
  - Ei eristata protsessi ja lõime, kõik on üks *task*
  - Eksisteerib Pthreads'i realisatsioon LinuxThreads (üks-ühele), signaalide töötlemiseks oma lõim

Java lõimed

- Mitte tuumas ega kasutajatasemel, vaid keele tasemel
- JVM reliseerib kuidas tahab (*green threads* – kasutaja tasemel, *native threads* – kasutab tuuma lõimesid)
- Lõimed päritakse klassist Thread (või realiseerivad liidese Runnable), defineeritakse oma meetod `run` ja käivitatakse meetodiga `start`
- Funktsiooni `main` võib vaadata kui ühelõimelise programmi `run`-meetodit
- Massiline lõimede kasutamine – ainult blokeeruva I/O olemasolu tõttu tuleb enamasti iga postkast (soklid jne) oma lõime panna
- Java 2 versioon 1.4 on esimene Java versioon, mis sisaldab korralikku asünkroonset I/O-d ja enam pole nii massiliselt lõimesid vaja