Käyttöjärjestelmät I

Luento 11: SÄIKEET

Stallings, Luku 4.1

Sisältöä

- Prosessi vs. säie
- Miksi säikeitä?
- ULT: Käyttäjätason säikeet
- KLT: Säikeiden toteutus ytimessä
- Säikeen tilat

- Käyttöjärjestelmän suorittamisesta

Käyttöjärjestelmät I

PROSESSI

vs

SÄIE

Prosessi perinteisesti

- **Resurssien omistaja, jolle allokoitu**
  - virtuaaliosoiteavaruus, eli suoritusympäristö
    - prosessin kuva (image): PCB, koodi, data, pino
  - resursseja
    - muistia, tiedostoja, I/O-laitteita ...
- **Vuorottajan hallinnoima kokonaisuus - prosessi on ohjelman suoritus koneessa**
  - suoritus limittäin muiden prosessien kanssa
  - prosessiin liittyy tila (Running, …) sekä prioriteetti

Process, task

Prosessi nykyaikaisesti

- **Resurssien kirjanpidon yksikkö, omistaja**
  - virtuaaliosoiteavaruus, jossa prosessin kuva
  - laitteiden varaus
- **Suojauksen yksikkö**
  - muistinsuojaus
  - prosessien välinen kommunikointi
  - tdstot ja niiden pääsyoikeudet

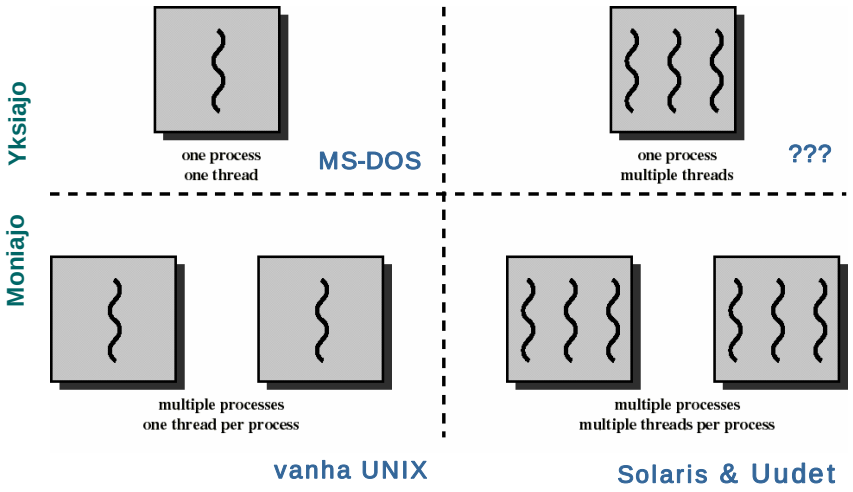
mutta Vuorottamisen yksikkö = Säie

- CPU suorittaa säikeitä, ei prosesseja

- **Yksi koodi + resurssit, monta suoritusta**

Thread, Lightweight process

Prosessi ja säie

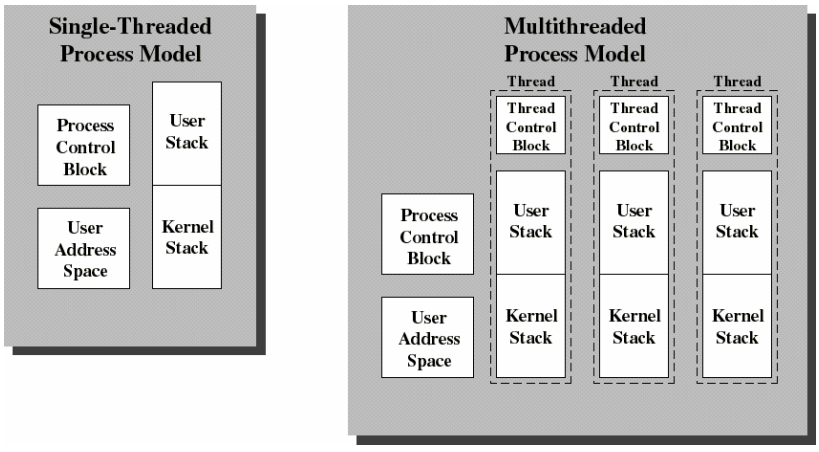


Prosessi voi jakautua säikeisiin

- ... jos KJ:ssä toteutettu säikeet tai = KLT
- ... jos käytetään säiekirjastoa = ULT

- Säikeellä oma tila (Running, Ready...)
- Säikeellä oma tallealue rekistereille
  - mm. omat PC:n ja PSW:n arvot
- Säikeellä oma pino
  - aliohjelmakutsuja ja paikallisia muuttujia varten
- Säikeellä oma kuvaaja
  - TCB, Thread Control Block

Yksi säie vs. Monta säiettä



Säikeen kuvaaja TCB

tallealue rekistereille, prioriteetti, tila, ...

Yhteiskäyttöiset resurssit

- **Prosessin säikeet käyttävät yhteistä koodi- ja data-aluetta**
  - viite sivutauluun vain PCB:ssä
  - mutta jokaisella oma suoritusaikainen pino
- **Kun säie muuttaa data-aluetta (muuttujia), muutos näkyy kaikille prosessin säikeille**
- **Säikeen avaama tdsto avoinna myös muille prosessin säikeille**
  - tiedostokuvaajataulu vain PCB:ssä
  - yhteinen luku/kirjoituspositio?

Käyttöjärjestelmät I

MIKSI SÄIKEITÄ?

Miksi säikeitä?

Œ Säikeen luonti (> 10 x) nopeampaa kuin kokonaan uuden prosessin luonti

- Prosessin säikeiden vuorottaminen nopeampaa kuin prosessien vuorottaminen

Ž Kun säie odottaa, voidaan suorittaa jotain muuta saman prosessin säiettä

- Säikeen lopettaminen nopeampaa kuin prosessin lopettaminen

Miksi säikeitä?

- Resurssien jakaminen säikeiden välillä tehokasta
  - u oletus: yhteiskäyttöisiä

‘ Saman prosessin säikeiden välinen kommunikointi helppoa ja nopeaa

- u yhteiskäyttöinen data-alue
  - F kaikilla pääsy globaaleihin muuttujiin
- u säie hoitaa itse tiedon sovittuun paikkaan, toinen noutaa itse
- u ei tarvita ytimen apua, ei siirtymisiä etuoik. tilaan

’ Voi helpottaa ohjelmointityötä

Miksi säikeitä?

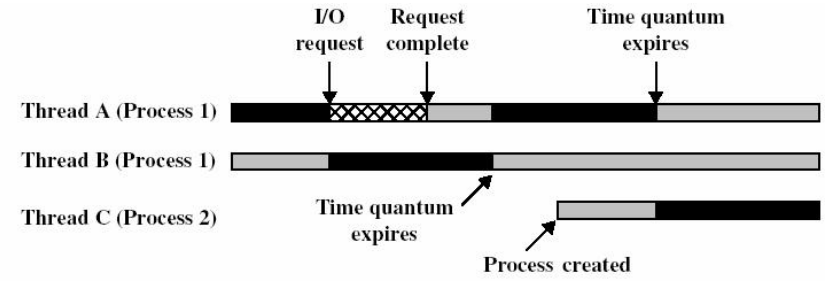
Mutta

- synkronointi ja poissulkeminen kokonaan ohjelmoijan vastuulla
  - **esim. Yhteisen tietorakenteen muuttaminen**
    - jos yksi muuttamassa, muut eivät saa käyttää
    - jos väh. yksi käyttää, kukaan ei saa muuttaa
  - **esim. Tuottaja ja kuluttaja**
    - kuluttaja ei saa edetä ennenkuin tuottaja edennyt tiettyyn vaiheeseen

ð RIO-kurssi

Kuka hyötyy?

Kuva 4.4



Sovellus, jossa selkeästi riippumattomia kokonaisuuksia

- ts. suoritusjärjestyksellä ei väliä
- ts. säikeillä ei synkronointitarvetta
- kukin osa toteutettavissa omana säikeenä

Kuka hyötyy?

Esim: lähiverkon tdstopalvelija

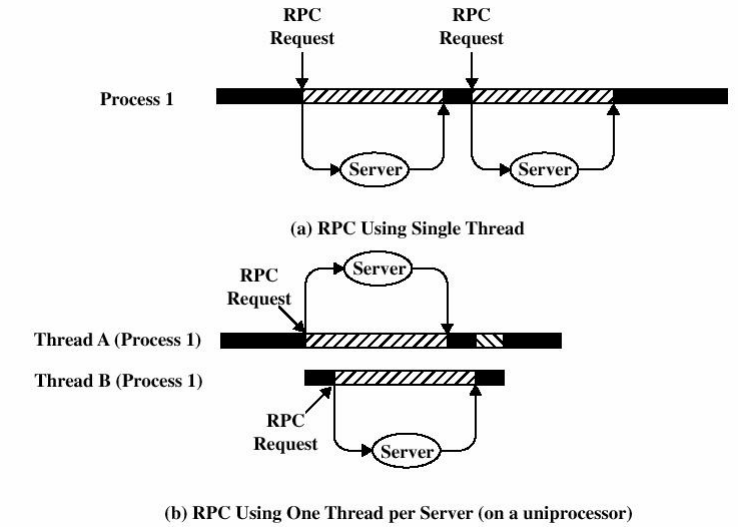
- Käsiteltävä useita pyyntöjä lyhyessä ajassa
- Halvempaa luoda/tappaa kutakin pyyntöä kohden oma säie kuin oma prosessi
- SMP: prosessin säikeet voidaan suorittaa aidosti yhtäaikaa eri prosessoreilla

Esim: komennot valikoista

- Yksi säie näyttää valikon ja lukee syötteen
- Toinen säie suorittaa edellistä komentoa
- Helpompi ohjelmoida

Esimerkki: etäkutsu

Kuva 4.3



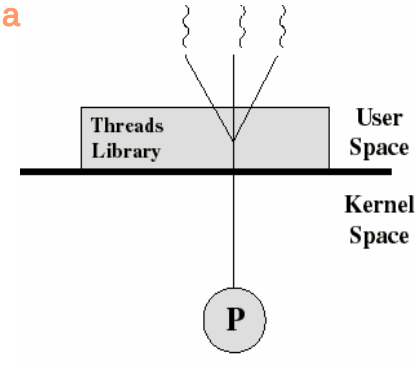
Käyttöjärjestelmät I

KÄYTTÄJÄTASON SÄIKEET

ULT, User Level Threads

Käyttäjätason säikeet

- **Ydin ei tiedä säikeistä**
  - vuorottaa vain prosesseja
- **Sovellus käyttää säikeiden hallintaan säiekirjastoa**
  - kirjanpito, TCB:t
- **Sovellus huolehtii itse säikeidensä vuorottamisen**
  - ohjelmoijan vastuulla
  - ei käytä ytimen koodia
  - ei keskeytysohjattua



Säiekirjaston perusrutiinit

Karkea jako:

- **Säikeen luominen**
  - kirjanpito säikeestä ja sen tilasta
- **Säikeen etenemisen estäminen**
  - rekistereiden tallettaminen
- **Säikeen etenemisen salliminen**
  - rekistereiden palauttaminen
- **Säikeen lopettaminen**

- **Kohdat 2 ja 3 = synkronointi**

POSIX threads (pthreads)

- **pthread-kirjastossa yli 60 funktiota**
- **pthread_create()**
  - parametrina funktio, josta suoritus alkaa
- **pthread_exit()**
  - lopeta säikeen suoritus
- **pthread_join()**
  - odota parametrina annetun säikeen loppumista
- **Synkronointi, poissulkeminen (semaforit)**
  - pthread_mutex_init() / _destroy()
  - pthread_mutex_lock() / _trylock() / _unlock()
- **Ynnä muita funktioita**
  - sched_yield(): luovu vapaaehtoisesti CPU:sta

Esimerkki

```
void main() {
   pthread_t thr1, thr2;
   char *msg1 = "Hello";
   char *msg2 = "World";

   pthread_create(&thr1, pthread_attr_default,
                  (void*)&print_message_function,(void*)msg1);
   pthread_create(&thr2, pthread_attr_default,
                  (void*)&print_message_function,(void*)msg2);
   exit(0);
}
void print_message_function(void *ptr){
   char *message;
   message = (char *) ptr;
   printf("%s ", message);
}
```


Käyttäjätason säikeet

Hyötyjä

- **Vuorottaminen nopeaa**
  - ei KJ:n apua
  - ei keskeytystä
  - ei prosessinvaihtoa!
- **Ohjelmoija voi valita sopivan vuorottamistavan**
  - KJ ei tunne sovelluksen tarpeita
- **Sovellus siirrettävissä helposti ympäristöihin, joissa sama säiekirjasto**

Haittoja

- **Kun säikeen palvelupyyntö aiheuttaa odotusta, kaikki muut saman prosessin säikeet odottavat**
- **Saman prosessin säikeet eivät voi olla suorituksessa usealla prosessorilla yhtäaikaa**
  - Ydin vuorottaa vain prosesseja!

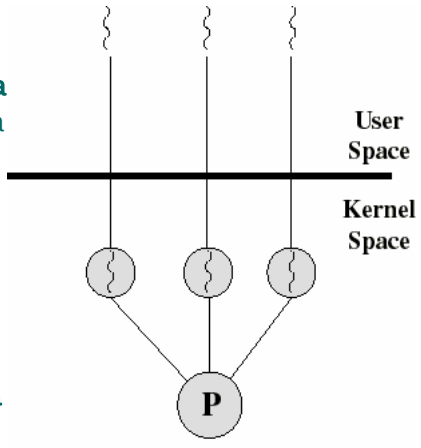
Käyttöjärjestelmät I

YTIMEN SÄIKEET

KLT, Kernel Level Threads

Säikeiden toteutus ytimessä

- **Säikeiden hallinta kokonaan KJ:n ytimessä**
- **Toteutus ei käytä kirjastoa**
  - ohjelmoijalle näkyvä rajapinta voi olla silti sama kuin edellä
  - palvelupyynnöt
- **Ydin tietää säikeistä**
  - prosesseista PCB:t
  - niiden säikeistä TCB:t
- **Vuorottaminen KJ:n heiniä**
  - aikaviipale säikeelle



Säikeiden toteutus ytimessä

Hyötyjä

- Prosessin säikeitä voi olla yhtäaikaa suorituksessa eri prosessoreilla
- Jos säie Blocked- tilaan, muut prosessin säikeet voivat silti jatkaa
- Myös KJ-ytimen toteutus voi käyttää säikeitä

Haittoja

- Säikeen vaihto vaatii aina 2 vaihetta:
  - keskeytys + siirtyminen KJ:hin (-> etuoik. tila)
  - vuorottaminen (->kjätila)
- Vaihto hitaampaa kuin kirjastoa käytettäessä
  - ks. taulukko 4.1

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

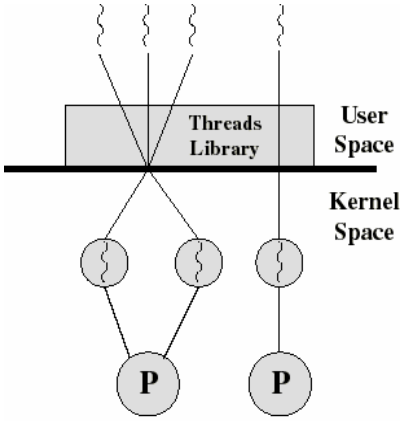
Käyttöjärjestelmät I

KULTAINEN

KESKITIE ?

Solaris

- Yhdistää molempien parhaat piirteet
- Luonti käyttäjätilassa
- Synkronointi käyttäjätilassa
- Pääosa vuorottamisesta käyttäjätilassa
- Ohjelmoija voi *liittää* käyttäjätason säikeet ytimen säikeiksi haluamallaan tavalla



Solaris

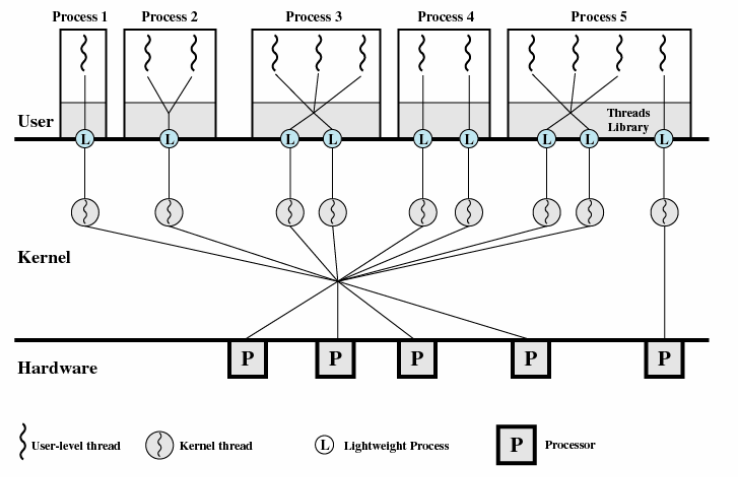


Figure 4.15 Solaris Multithreaded Architecture Example

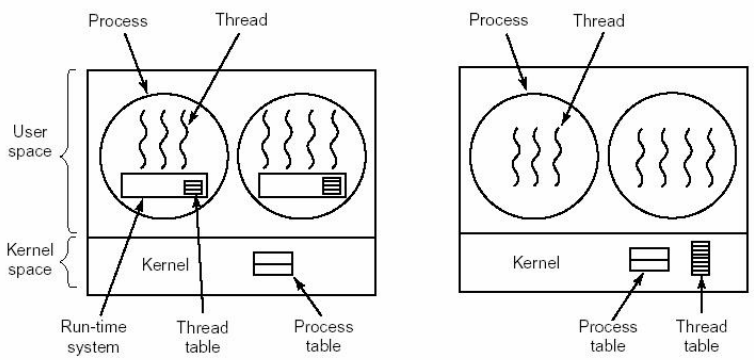
Käyttöjärjestelmät I

SÄIKEEN TILAT

Säikeen tilat

- **Perustilat: Running, Ready, Blocked**
  - kuten prosessilla
- **Suspend-tila koskee aina koko prosessia**
  - liittyy heittovaihtoon
  - osoiteavaruus prosessitason käsite
  - mm. koodialue + globaali data yhteiskäytössä
- **Kun prosessi joutuu Suspend-tilaan, joutuvat kaikki sen säikeet odottamaan**
- **Kun prosessi lopetetaan, poistetaan samalla kaikki sen säikeet**

ULT-säie vs KLT-säie



(a) A user-level threads package.

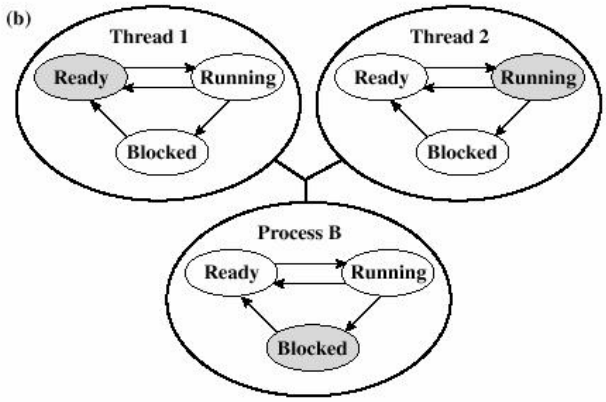
(b) A threads package managed by the kernel.

Tan01 kuva 2-13

ULT-säikeen vs Prosessin tila

- **KJ:n ydin ei tiedä ULT-säikeistä**
  - KJ vuorottaa prosessitasolla
    - Ready-jonossa prosesseja
  - aikaviipale prosessille
  - säiekirjasto huolehtii säikeiden vuorottamisesta prosessin aikaviipaleen sisällä
- **Jos ULT-säie odottaa palvelupyynnössä, prosessi joutuu Blocked-tilaan**
- **mutta säiekirjaston kirjanpidossa säie edelleen Running-tilassa**
  - ð ULT- säikeen ja prosessin tila erillisiä

ULT-säikeen vs Prosessin tila

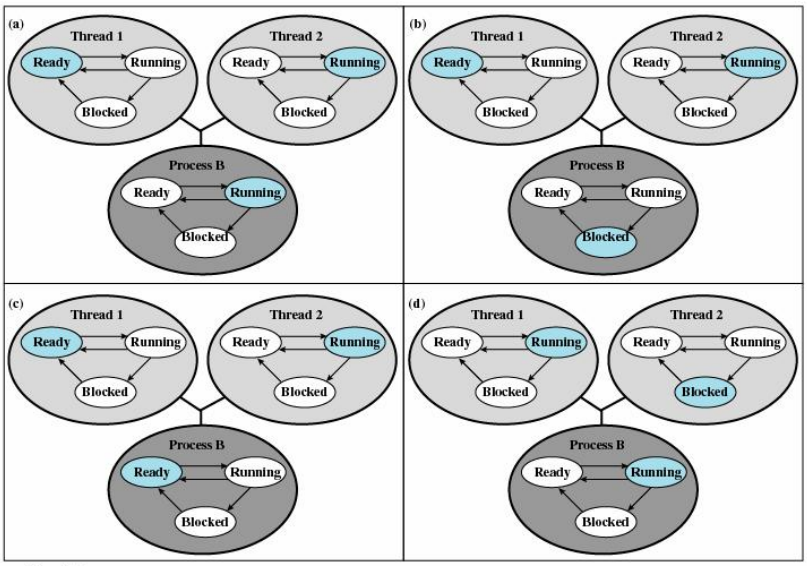


Kuva 4.7

KLT-säikeen vs Prosessin tila

- **KJ tietää säikeiden olemassaolosta**
  - KJ ylläpitää säiekuvaajia
  - KJ vuorottaa säietasolla
    - jos prosessin aikaviipaletta jäljellä, vuorota saman prosessin säikeitä tai
    - koko aikaviipale säikeelle
- **Jos KLT-säie odottaa palvelupyynnössä, prosessin muut säikeet voivat silti jatkaa**

Mahdollisia tilan vaihtoja



Colored state
is current state

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Kertauskysymyksiä

- Miten prosessi ja säie eroavat toisistaan?
- Mitä yhteistä niillä on?
- Mitä hyötyä säikeistä?
- Miten ULT ja KLT eroavat toisistaan?
- Miksi ULT säikeiden vuorottaminen nopeampaa kuin KLT säikeiden?
- Miksi ULT säikeen Blocked-tila vie prosessin Blocked-tilaan, mutta KLT säikeen ei?

Käyttöjärjestelmät I

KJ:N

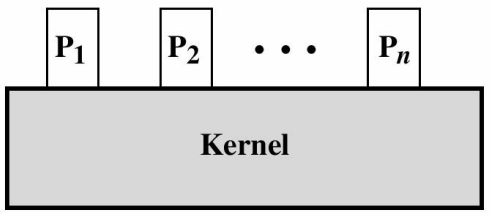
SUORITTAMISESTA

(jäi käsittelemättä luennolla 3 aikapulan vuoksi)

KJ:n suorittamisesta

- Myös KJ eräs CPU:n suorittamista käskykokoelmista
- Käyttäjätilassa / etuoikeutetussa tilassa
- KJ:n osat käsittelevät yhteisiä data-alueita
  - melkein kaikki käyttävät PCB:tä
- Onko KJ myös prosessi?

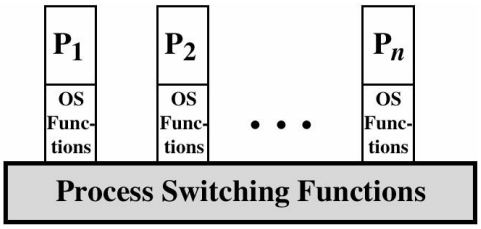
j KJ etuoikeutetussa tilassa



- **Prosessi vain käyttäjätilan käsite**
  - KJ:n osat eivät jonota
- **KJ:llä omat muistialueensa: koodi, data, pino**
- **KJ:n osat suoritetaan omillaan etuoik. tilassa**
  - oikeus tehdä kaikkia KJ:n toimintoja kaikissa osissa

~ **vanha monoliittinen KJ**

k KJ prosessin ympäristössä

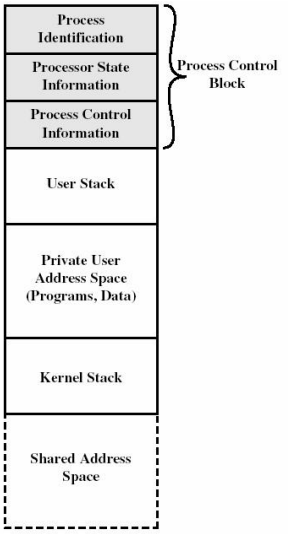


- **KJ yhteiskäyttöisellä muistialueella**
  - kaikkien prosessien osoiteavaruudessa
- **Prosessi itse suorittaa KJ:n rutiineja**
  - hallittu siirtyminen keskeytyksellä, etuoikeutettu tila
- **Kontrolli prosesseilta poissa vain, kun synkronointi tai vuorottaminen vaatii**

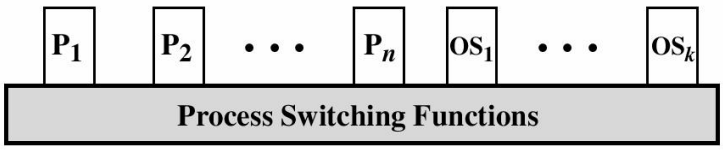
~ **uudempi monoliittinen KJ**

KJ prosessin ympäristössä

- KJ:n koodi ja data yhteisellä muistialueella
- Prosessi käyttää kernel-pinoa, kun suorittaa KJ:n koodia, muuloin normaalia pinoaan
- Prosessi voi odottaa KJ:n koodissa
- Useita KJ:n osia voi olla yhtäaikaa kesken eri prosessien ympäristöissä
  - suoritukseen vuorottajan kautta



KJ = joukko palveluprosesseja



- Monet KJ:n palveluista erillisiä prosesseja
  - odottavat Blocked/Ready-jonossa
  - kullakin oma osoiteavaruus
  - tarvittaessa etuoikeutetussa tilassa, erilaisia oikeuksia
- Vuorottaminen prosessien ulkopuolella
- Sanomanvälitys: pyyntö-vastaus mekanismi
  - palvelupyyntö: lähetä / vastaanota sanoma
  - sopii myös moniprosessori / hajautettuihin järjestelmiin
- Jos ytimessä vain laiteriippuvat toiminnot
  = mikrokernel