

Käyttöjärjestelmät II

VUOROTTAMINEN SMP JA REAALIAIKAJÄRJESTELMÄT Linux, W2000

Ch 10 [Stal 05]
(Ch 20 [DDC04], 11.4 [Tane01])

Tämä luento

- n Moniprossorijärjestelmien vuorottaminen
 - n Reaaliaikaskedulointi
 - n Linux: Vuorottaminen
 - n SVR4: Vuorottaminen
 - n W2K: Vuorottaminen
-
- n Linux: Ch 10.3 [Tane01], Ch 20 [DDC04]
 - n W2K: Ch 11.4 [Tane01]

Käyttöjärjestelmät II

VUOROTTAMINEN Moniprosessori- järjestelmässä

Moniprosessorijärjestelmä

n Löyhästi kytketyt (loosely coupled)

- u erillisten koneiden ryväs (cluster)
- u hajautettu järjestelmä (distributed system)

n Toiminnollisesti erikoistuneet

- u erilliset I/O-prosessorit
- u matikkaprosessori nykyään osa CPU:ta

n Tiukasti kytketyt (tightly coupled)

- u prosessoreilla yhteinen keskusmuisti
- u isäntä - rengit (host-slaves)
- u SMP, symmetric multiprocessing

Rinnakkaisuus & Vuorottaminen

- n **Synkronoinnin tarve / helppous?**
- n **Riippumattomat prosessit** (independent)
 - u ei keskinäistä synkronointia
 - u esim. normaali osituskäyttö
- n **Tosi karkea** (very coarse-grained)
 - u hajautettu verkon yli tapahtuva laskenta
 - u itsenäisten koneiden yhteiskäyttö
 - u ei hyvä, jos vaatii paljon kommunikointia
- n **Karkea**
 - u yksi kone - useita prosessoreita
 - u erillisten prosessien synkronointi
- n **Keskikarkea** (medium-grained)
 - u yhden sovelluksen sisäinen
 - u säikeiden käyttö, vuorottaminen ja synkronointi

SMP Prosessin vuorottaminen

n Isäntä-renki (master-slave)

- u KJ aina isäntäCPU:lla

 - F vuorottaminen, muistinhallinta, I/O

- u muut CPU:t suorittavat vain sovelluksia

- u jos isäntä kuukahtaa?

- u suorituskyky?

n SMP (peer architecture)

- u KJ millä tahansa CPU:lla, jopa yhtäaikaan

 - F vapaakäyntisyys

 - F synkronointi

 - F vikasietoisuus

- u kukin CPU vuorottaa itse itsensä

SMP Prosessien vuorottaminen

n Yhteiset jonot

- u kullakin prioriteetilla oma Ready-jono
- u yhteiset muut resurssit

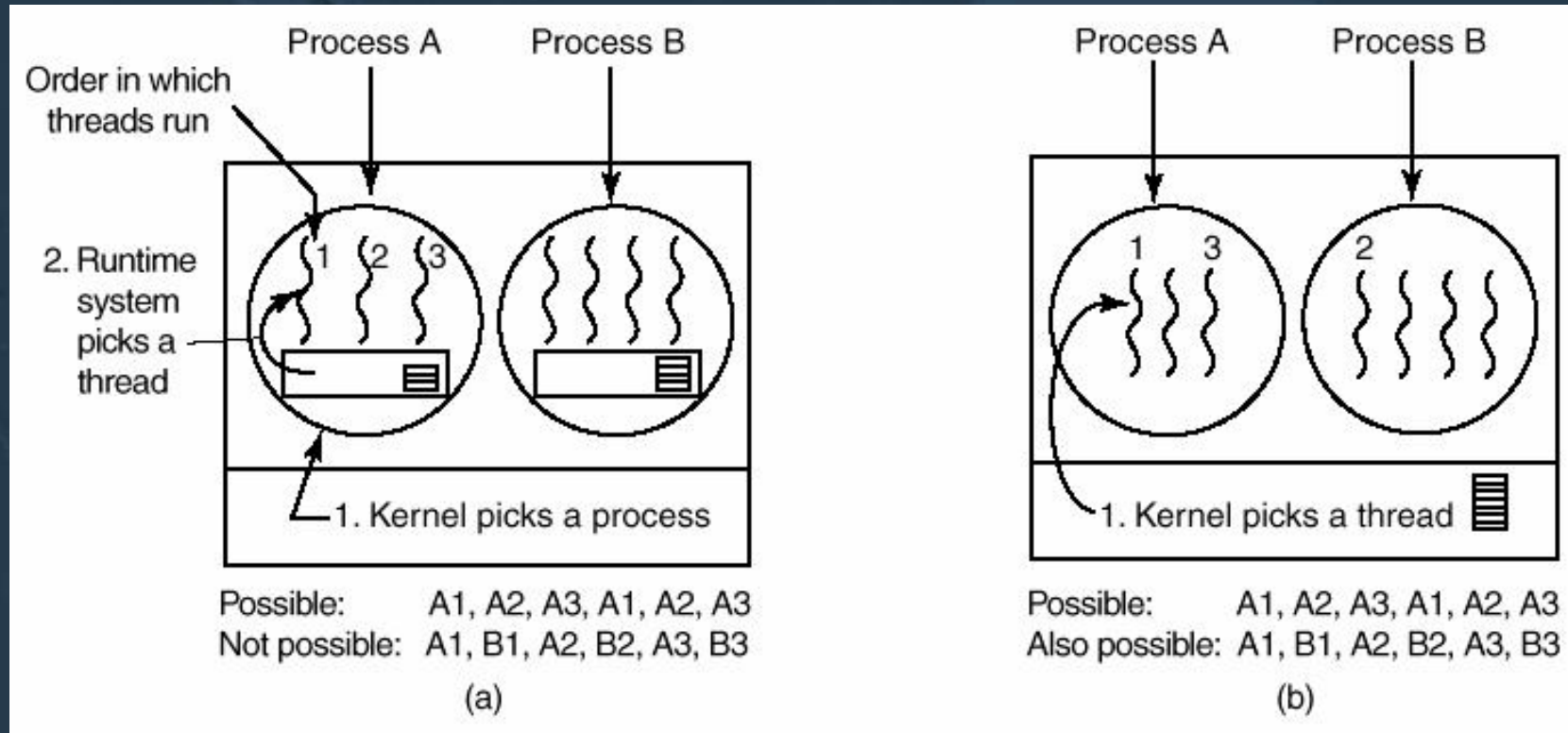
n Prosessi ei yleensä sidottu tiettyyn CPU:hun

- u vapaa prosessori vuorottaa yhteisestä Ready-jonosta
- u “common CPU-pool”

n Jos sidotaan tiettyyn suorittimeen, tarvitaan myös suoritin-kohtaiset Ready-jonot

- u vuorota aina ensin omasta jonosta
 - u jos jono tyhjä, vuorota globaalista jonosta
- F onko sellaista?

Säikeiden vuorottaminen



ULT säikeen CPU-aika 10% aikaviipaleesta (Fig 2-43 [Tane01])

n **ULT: KJ** vuorottaa prosesseja, säiekirjasto säikeitä

n **KLT: KJ** vuorottaa säikeitä

Säikeiden vuorottaminen

n Suoritus yhteisessä osoiteavaruudessa

- u kullakin säikeellä oma suoritusympäristö
 - F pino, tallealue rekistereille
- u muut resurssit yhteisiä
 - F koodi, globaali data, avoimet tiedostot
- u kommunikointi yhteisen data-alueen välityksellä

n Rinnakkain suoritettavat osat säikeiksi

- u kuorman jako (load sharing)
- u kimppavuorottaminen (gang scheduling)
- u suorittimen sitominen
(dedicated processor assignment)
- u dynaaminen vuorottaminen

Kuorman jako (load sharing)

n Säikeet jaettavissa usealle CPU:lle

- u jouten olevat käyttöön
- u parantunut suorituskyky, lyhyemmät läpimenoajat

n Saman prosessin säikeet eri CPU:lle

- u aidosti rinnakkainen suoritus
- u välimuistin hyödyntäminen?

n Globaali Ready-jono

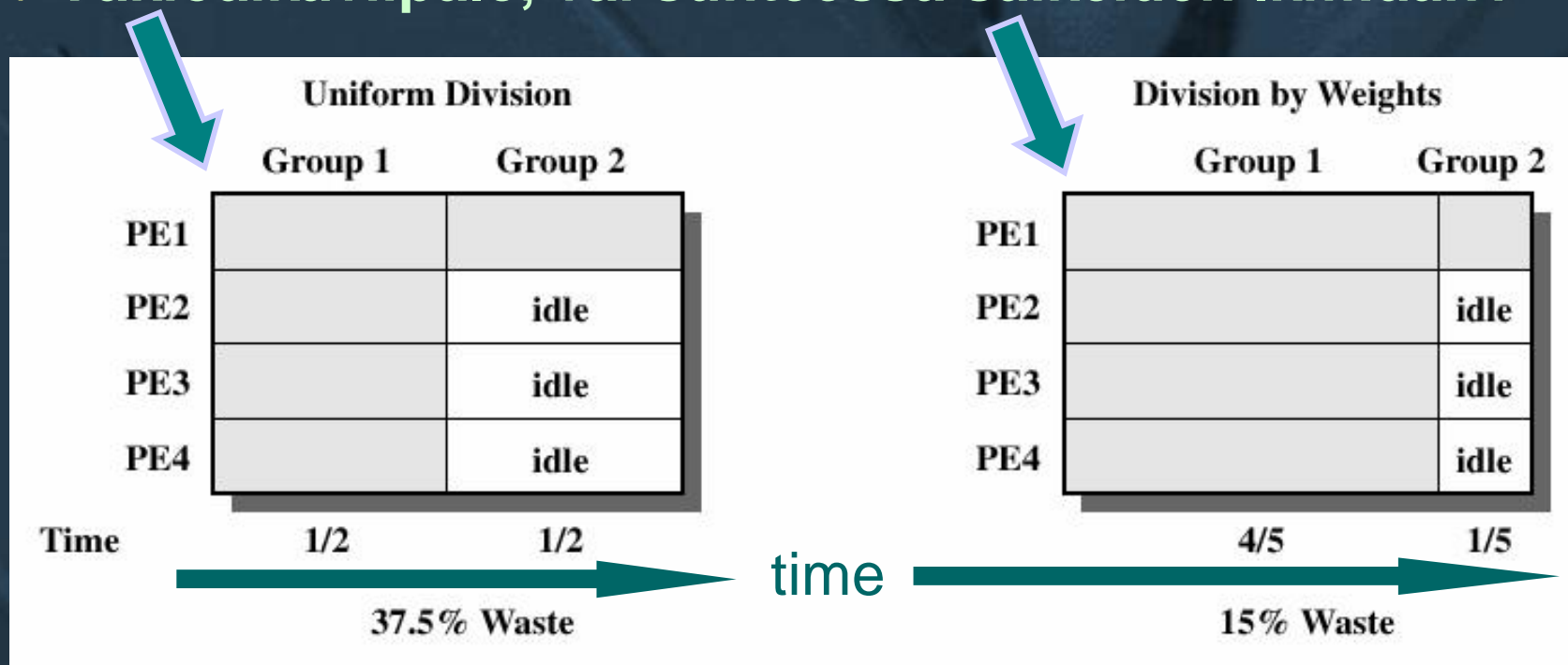
- u tarvitaan poissulkeminen
- u pullonkaula?

n Kukin CPU vuorottaa itsensä (valitsee säikeen)

- u FCFS
- u (prioriteetti: esim. pienin säikeiden lkm)

Kimppavuorottaminen (gang scheduling)

- n Kimpan säikeet yhtä aikaa usealle CPU:lle
- n Vähemmän odottelua synkronoinnin takia
- n Vähemmän prosessin vaihtoja
- n Vakioaikaviipale, vai suhteessa säikeiden lkm:ään?



(Fig 10.2 [Stal05])

CPU:n sitominen (Dedicated CPU assignment)

n Osa tai kaikki CPU:t varataan yhdelle sovellukselle

- u varaus kestää, kunnes sovellus päättyy

n Jos säie *blocked*, CPU silti varattuna

n Hyöty?

- u välttää prosessin vaihtoja, prioriteetti

- u ko. sovelluksen lyhentynyt läpimenoaika

- u välimuisti

n Muut saattavat kärsiä

- u jätettävä joku/joitakin CPU:ita varaamatta

n Ero kimppavuorottamiseen?

- u ei vuorotusta, varaus sovelluksen ajaksi

- u säie aina samalla suorittimella

Dynaaminen vuorottaminen (dynamic scheduling)

- n **Kimppavuorottamisen ja CPU-sitomisen välimuoto**
- n *“Anna ensin mitä on, ja sitten myöhemmin ehkä lisää”*
- n **Uusi prosessi**
 - u jos vapaita CPU:ita tarpeeksi, anna ne
 - u jos ei tarpeeksi CPU:ita vapaana
 - F ota joltain CPU pois tai
 - F sovelluksen odotettava tai
 - F voi peruuttaa pyynnön
- n **CPU vapautuu**
 - u jos joku odottaa ensimmäistä CPU:ta, anna sille
 - u muuten anna ensimmäiselle lisä-CPU:ta odottavalle

Käyttöjärjestelmät II

REAALIAIKA- JÄRJESTELMÄT

Yleiskuva

Reaaliaikajärjestelmät

- n **Oikeellisuus ei riipu pelkästään tuloksista, vaan myös valmistumisajasta**
- n **Tarve reagoida ulkopuolisiin tapahtumiin**
 - u ohjausjärjestelmät: laboratorionkokeet, teollisuus, lentoliikenne, teleliikenne, robotiikka, ...
- n **Hard Real-time vs. Soft Real-time**
 - u hard: ei saa missata aikarajoja (deadline) vs.
 - u soft: yrittää parhaansa, saa joskus myöhästyäkin
- n **Periodinen vs. aperiodinen**
 - u ajallinen tai määrällinen säännöllisyys vs.
 - u alku- ja/tai päättymisajalle aikaraja

Vaatimuksia

n Deterministisyys

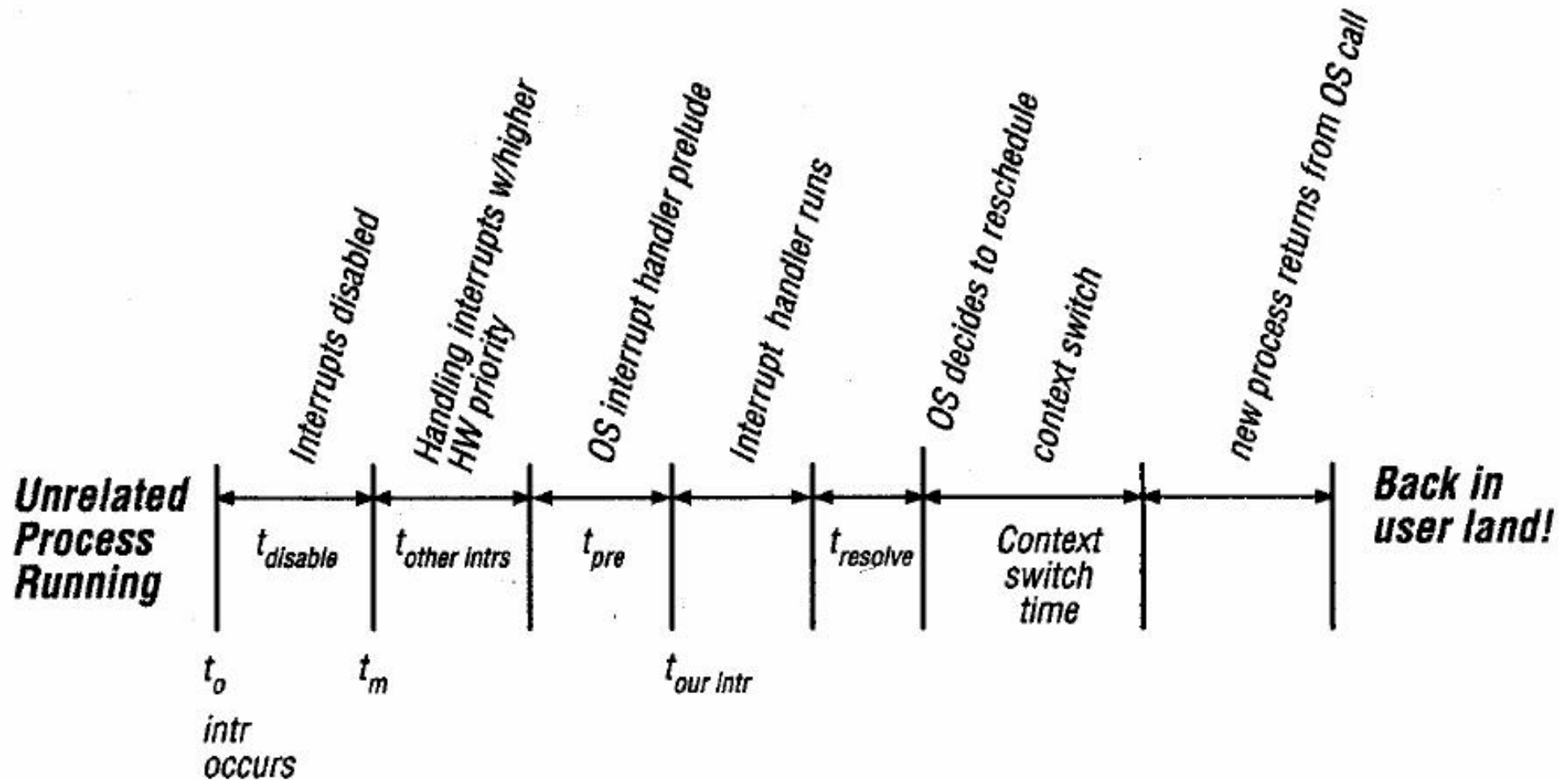
- u käsittely kiinteästi ennaltamääriteltynä aikoina tai kiintein aikaväleinä
- u pakottaa KJ:n reagoimaan keskeytykseen
 - F yläraja olemassa kuittaamiselle (muutama ms)

n Reagointinopeus (responsiveness)

- u kauanko menee keskeytyksen käsittelemiseen?
 - F keskeytyksen huomioiminen
 - F palvelun suorittaminen

â Vasteaika (response time R, turnaround time TAT)

Keskeytyskäsittelyn viipeet



lähde?

Reaaliaikajärjestelmän vaatimuksia

n Ohjelmoijan kontrollointi

- u mitä osia lukitaan muistiin tai KJ ei lainkaan sivuta
- u vuorottamisalgoritmin valinta
- u prioriteetti, aikaviipaleen pituus, aikarajat, kesto
- u asynkroninen siirräntä

n Luotettavuus (reliability)

- u suorituskyvyn tipahtaminen voi olla katastrofi
- u Hard RT vs. Soft RT
- u virheistä toipuminen (fail-soft operation)

F nilkuttaa saa, muttei kaatua

Reaaliaikajärjestelmän peruspiirteitä

- n **Minimijoukko operaationaalisuutta**
 - u pieni ja tehokas toteutus
- n **Reagoitava nopeasti ulkoisiin keskeytyksiin**
 - u minimoitava kohdat, joissa keskeytykset estettyinä
- n **Nopea prosessin / säikeen vaihto**
- n **Keskeytyvä vuorottaminen (preemptive)**
 - u prioriteetit
 - u vähemmän kiireellinen homma voidaan hyllyttää
- n **Moniajo, välineet prosessien kommunikointiin**
 - u semaforit, signaalit, tapahtumat, ...

Reaaliaikajärjestelmän peruspiirteitä

- n **Tehokas peräkkäistiedostojen käsittely**
 - u indeksirakenteet
 - u raw I/O
 - u muistiinkuvatut tiedostot
- n **Hyvät aikaan liittyvät palvelut, ajastukset**
- n **Välineet myöhästyttää työn käynnistämistä**
 - u Hard RT palveltava
 - u Soft RT voi joskus myöhästyä

Käyttöjärjestelmät II

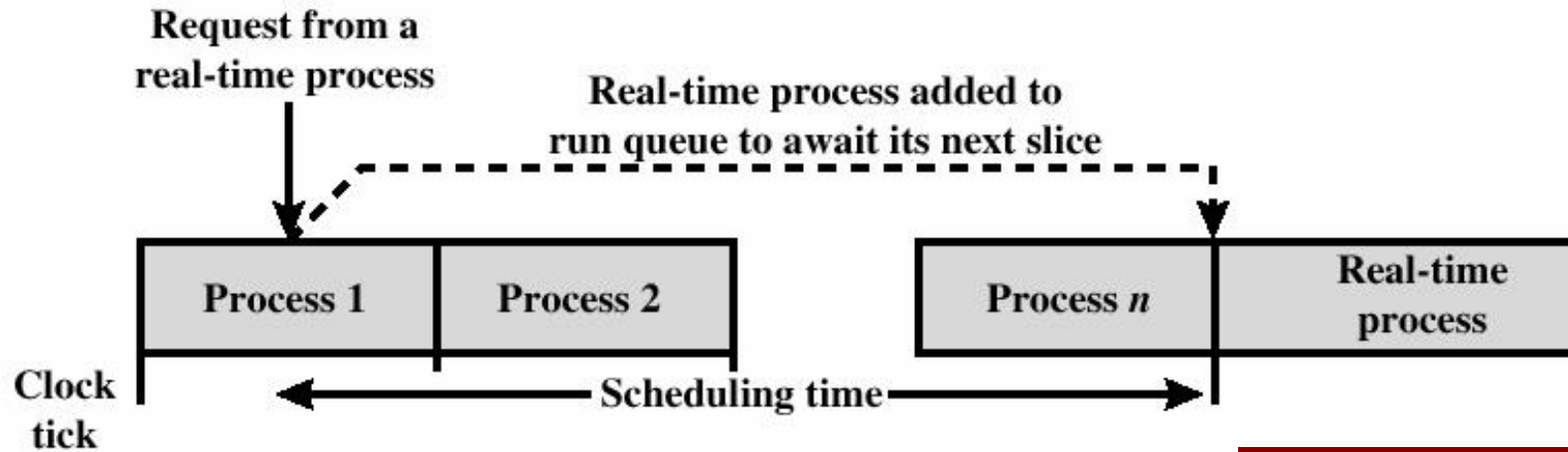
Vuorottaminen reaaliaikajärjestelmissä

RT: Vuorottaminen

- n **Vastausajan minimoiminen usein toissijaista**
- n **Tärkeää suorittaa työt ajallaan**
 - u ei liian aikaisin eikä liian myöhään
 - u hard RT: aloitus- ja/tai valmistumisaika kiinnitetty
- n **Aikarajan varaan rakentaminen vaikeaa**
 - u mistä KJ tietää työn kestoajan?
 - F käyttäjän antama arvio kestosta, periodeista
- n **Pyrkii vain vuorottamaan RT-työn nopeasti**
 - u suuri prioriteetti RT-töille
 - u ennalta laadittu aikataulu
 - u muista ei väliä

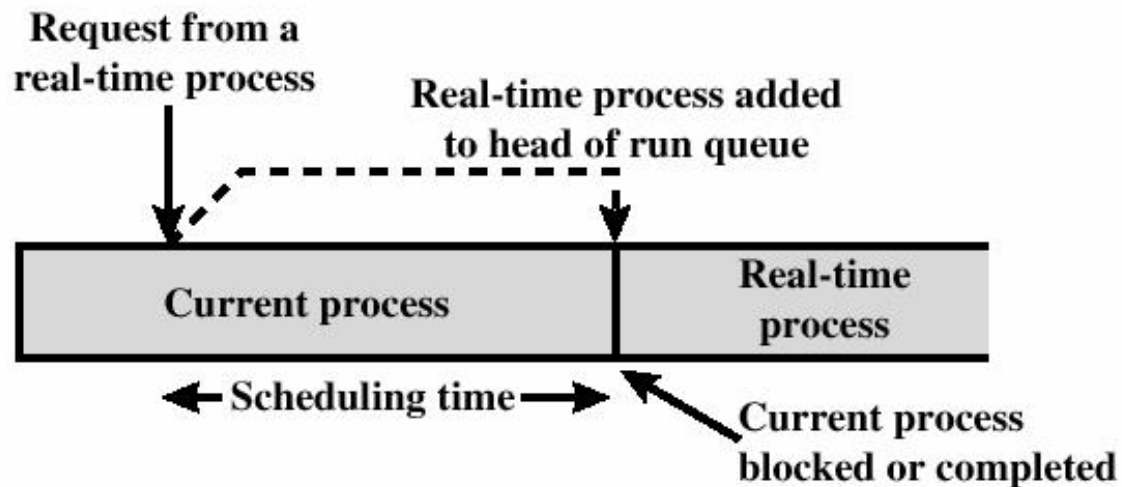
RT: Vuorottaminen

(Fig 10.4 [Stal05])



(a) Round-robin Preemptive Scheduler

Ei kelpaa!

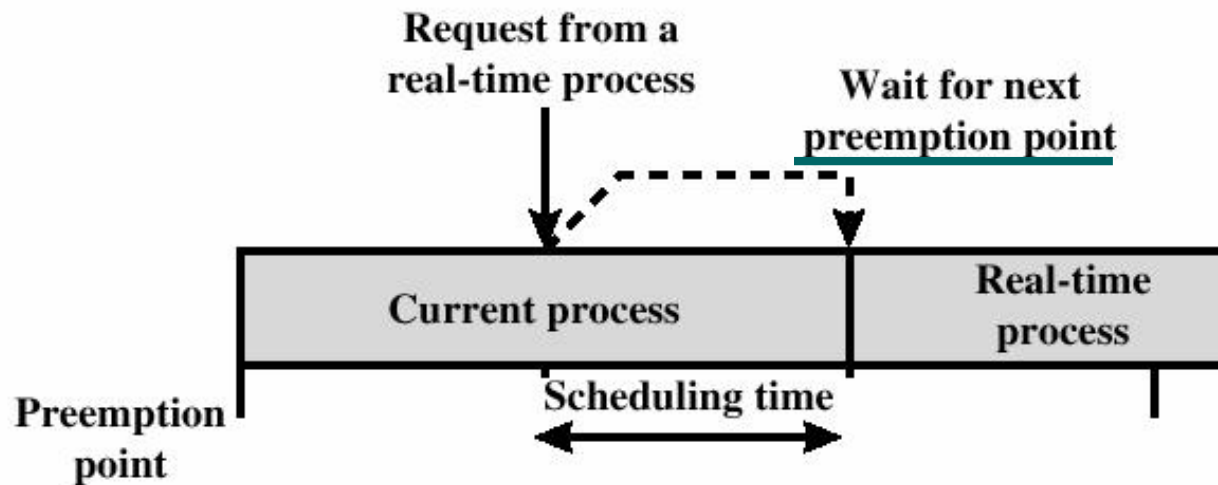


(b) Priority-Driven Nonpreemptive Scheduler

Ei kelpaa!

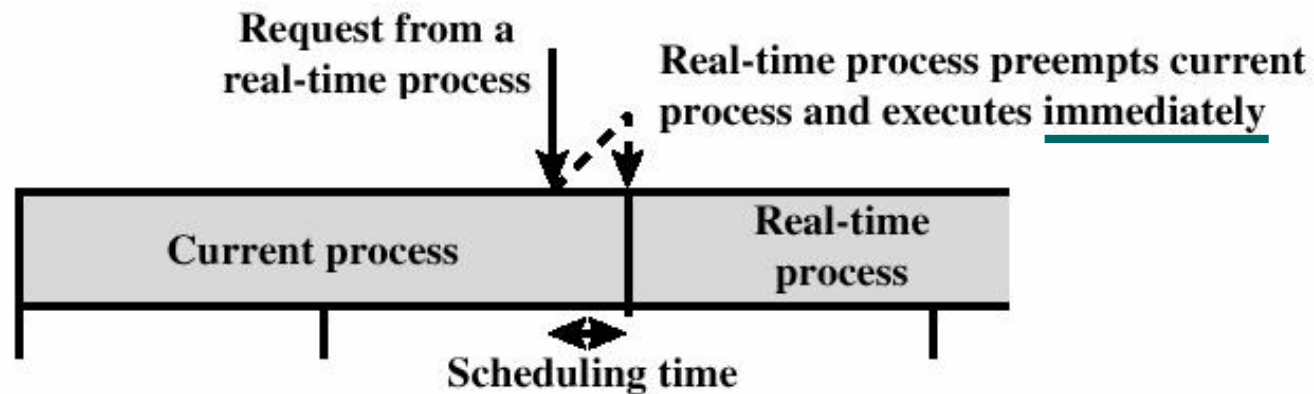
RT: Vuorottaminen

(Fig 10.4 [Stal05])



Kelpaa!

(c) Priority-Driven Preemptive Scheduler on Preemption Points



Kelpaa!

(d) Immediate Preemptive Scheduler

Staattiset lähestymistavat

n Voiko kuormaa analysoida?

- u etukäteen tai ajonaikana
- u aikataulu

n Staattinen taulukko-perustainen

- u sopii periodisille töille
- u alkuaika, kesto, päättymisaika, prioriteetti
- u **EDF, Earliest Deadline First** -algoritmi

n Staattinen prioriteetteihin perustuva

- u normaali keskeyttävä (preemptive)
- u prioriteetti määräytyy annettujen aikarajojen nojalla
- u **RMS, Rate Monotonic Scheduling** -algoritmi

Earliest Deadline First

n **Prioriteetti \neq aikarajat**

n **Tarvitaan lisätietoa työstä (=ohjelmoijalta)**

u milloin valmis suoritukseen (Ready time)

F periodinen \rightarrow aikajono

F aperiodinen \rightarrow saatetaan tietää tai

KJ valpas huomaamaan

u takaraja aloitukselle, takaraja valmistumiselle

u prosessointiaika, muiden resurssien tarve

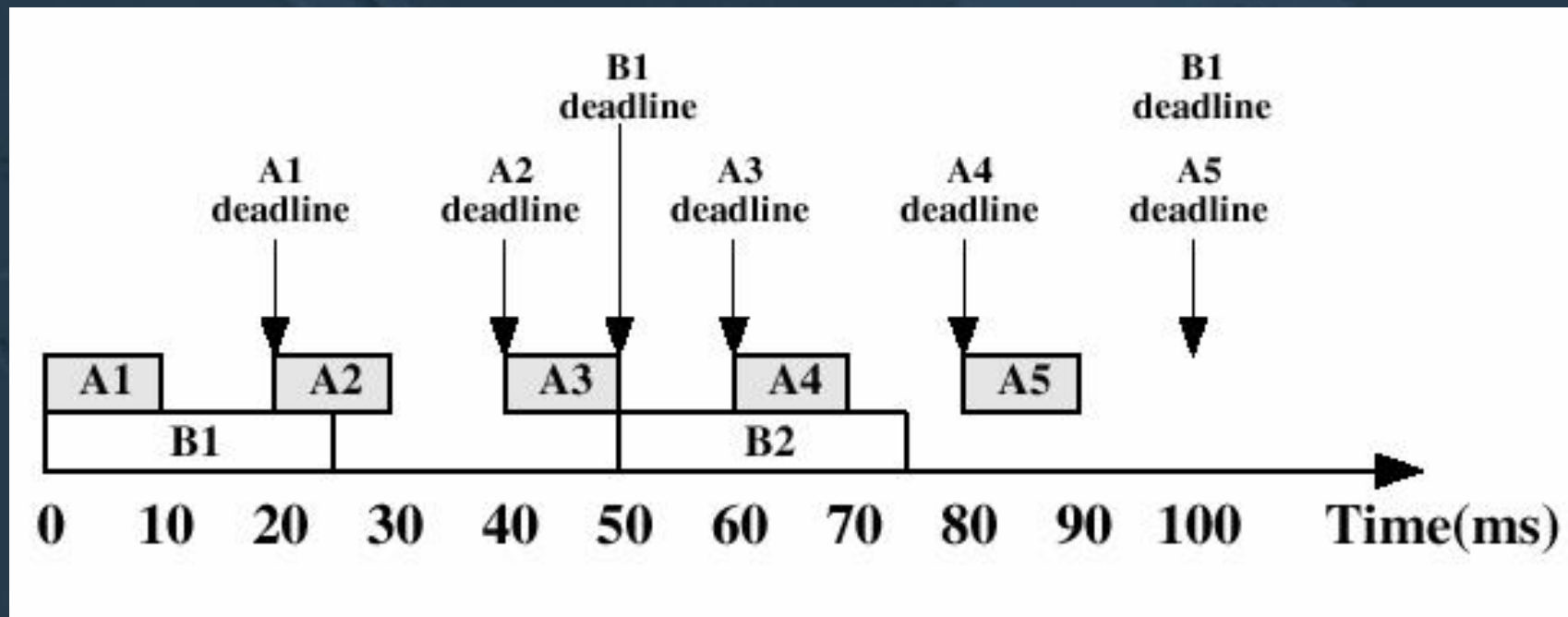
u prioriteetti

u onko pakollisia alitöitä?

n **KJ voi laatia suoritusaikataulun**

Esimerkki

(Fig 10.5 [Stal05])



Kaksi periodista työtä:

- Saapumisajat A 20 ms, B 50 ms välein
- Suoritusajat A 10 ms, B 25 ms
- Vuorottaminen 10 ms:n välein
- Valm. takaraja A 20 ms, B 50 ms saapumisesta

Fig 10.5 [Stal05]

Esimerkki 2

Viisi aperiodista työtä, aloitukselle aikaraja

(Tbl 10.3 [Stal05])

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

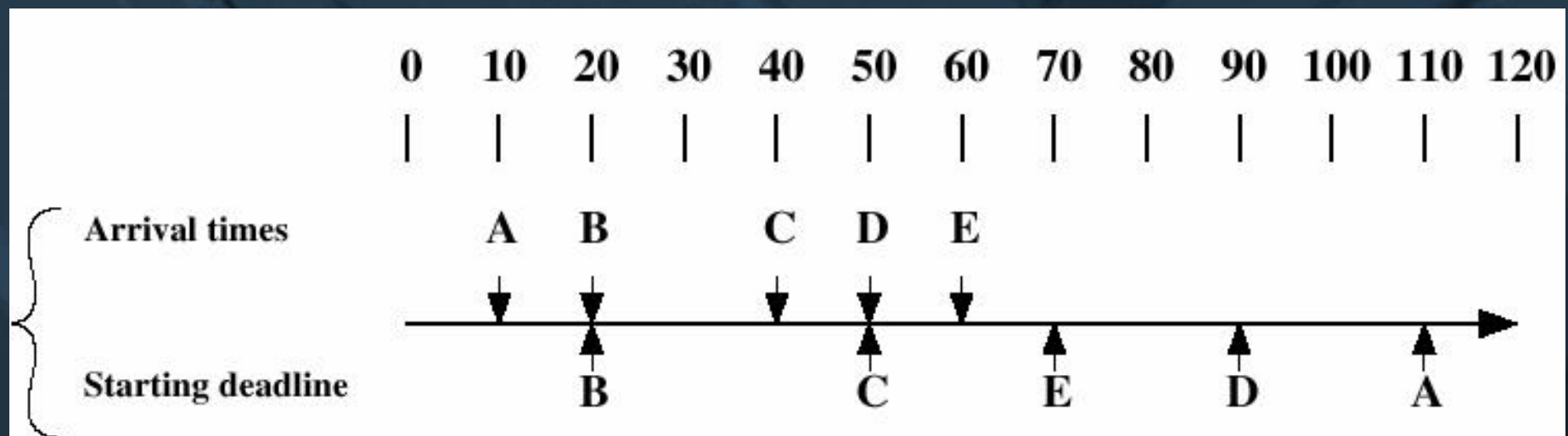
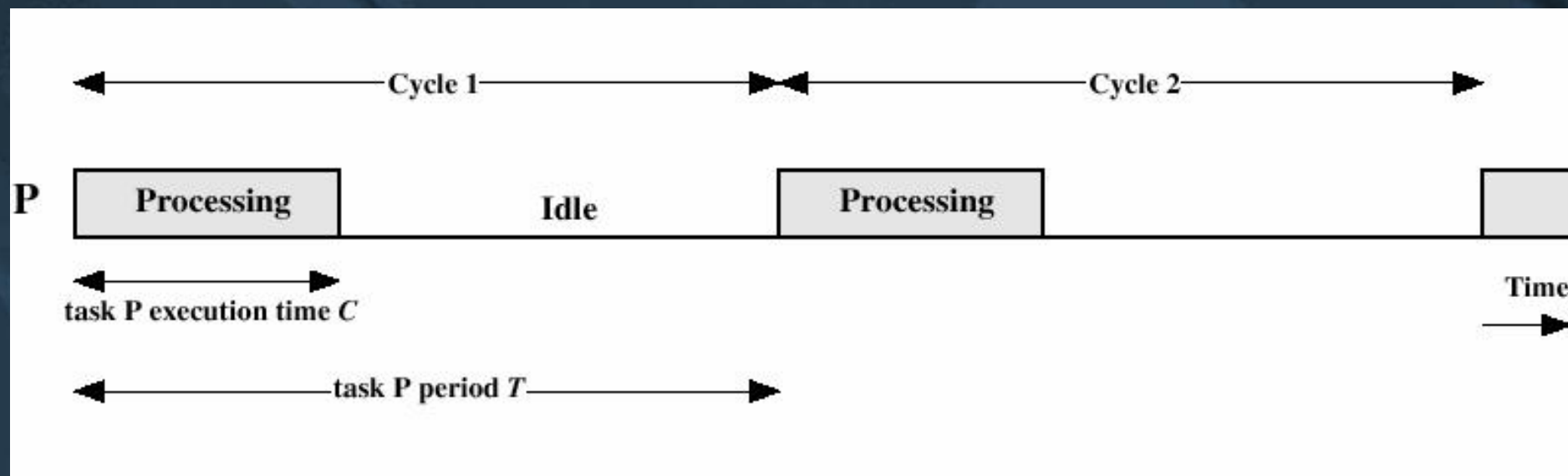


Fig 10.6 [Stal05]

Rate Monotonic Scheduling



n **Vain periodisille töille**

(Fig 10.7 [Stal05])

n **Sama työ tasaisella tahdilla**

u käyttöaste yhden työn osalta on $U = C/T$

n **Jakson loppu = Hard deadline**

n **Pienin jakso (T) = suurin prioriteetti**

Rate: määrä yksikköä kohden, taajuus

RMS - Rate Monotonic Scheduling

n **Selvästi**

$$DU_i = C_1/T_1 + C_2/T_2 + \dots + C_n/T_n \leq 1$$

n **Riittävä ehto** sille, että työt voidaan ajoittaa RMS-algoritmillä on

$$C_1/T_1 + C_2/T_2 + \dots + C_n/T_n \leq n(2^{1/n} - 1)$$

n Arvon n kasvaessa, RMS:lle yläraja lähenee

$$\ln 2 \sim 0.693 \text{ eli rajatapaus } DU_i < 0.693 \text{ !}$$

n **EDF tehokkaampi, sille riittää**

Tbl 10.4 [Stal05]

$$DU_i = C_1/T_1 + C_2/T_2 + \dots + C_n/T_n \leq 1$$

Dynaamiset lähestymistavat

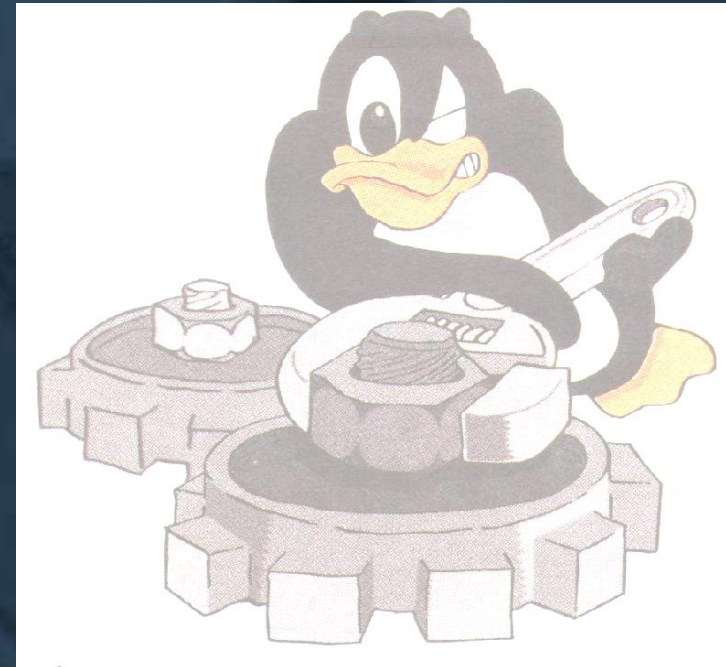
n Dynaaminen aikatauluttaminen (planning based)

- u aperiodisille töille
- u kun uusi työ käynnistyy, laske prioriteetti ja uusi käytettävä aikataulu
- u jos pystyy takaamaan aikarajat, ota työ suoritukseen
- u käytä esim EDF-vuorottamista

n Dynaaminen, parhaan kyvyn mukaan ponnistelu (best effort)

- u kuten edellä, mutta ota suoritukseen ja rukoile, että onnistuu

Käyttöjärjestelmät II



Linux

VUOROTTAMINEN

Linux: Vuorottaminen



n Poikkeaa perinteisestä Unixista

- u root: *sched_setscheduler()*

POSIX

n Soft Real-time työt:

- u **SCHED_FIFO**

- F kiinteät prioriteetit, tapahtumaohjattu (ei aikaviipaletta)

- F suoritus keskeytyy vain jos

- suuremman prioriteetin työ (task) Ready-tilaan
- työ joutuu Blocked-tilaan
- työ itse luovuttaa CPU:n (*sched_yield()*)

- u **SCHED_RR**

- F kuten edellinen, mutta lisäksi aikaviipaletta

n Muut, osituskäytön työt, vaihtelevan prioriteetin työt:

- u **SCHED_OTHER**

- u käytä tätä, jos edellisissä ei töitä Runnable-tilassa

Linux: suorittimen prioriteetti

n Vuorottaminen säikeen tasolla (työ ~ task)

- u ytimen säikeet
- u joka suorittimella omat jonot



n Luokat: -20 ... +19

- u perusprioriteettiarvo: 0 (paras user-taso)
- u joka luokalla oma run-jono, negatiiviset KJ:lle

n Joka prosessilla staattinen prioriteetti: *nice* [-20,+19]

- u pieni nice-arvo → pieni prioriteettiarvo → hyvä juttu
- u user prosesseilla nice välillä [0, +20]

n Lähtökohtaprioriteetti = *nice* (-19 ... +20)

n Dynaaminen vaihtelu, vaihteluväli: ±5 (säätö)

n Tehokas prioriteettiluku: *epri* = *nice* +säätö

- u aina rajoissa [-19, +20]

Linux: prioriteetin vaihtelu

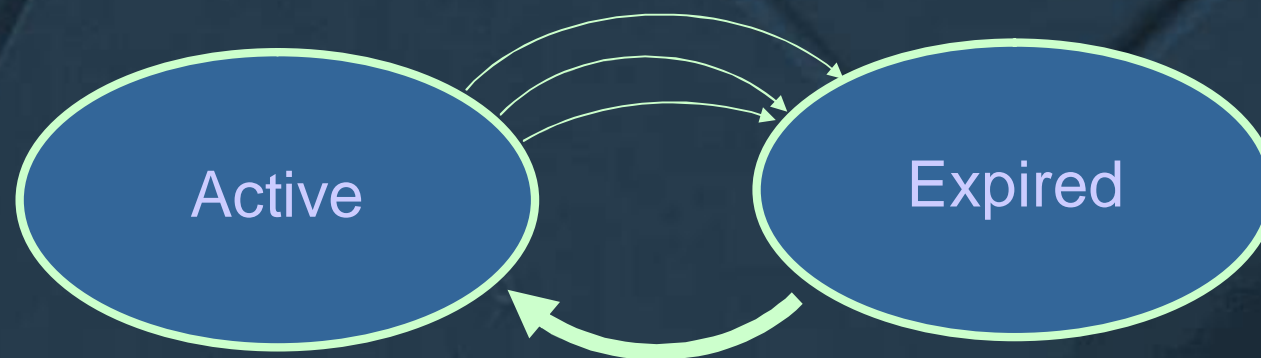
kernel 2.6
[DDC04]

- n **Tehokas prioriteetti: *epri*, alkuaan *nice***
 - u prosessin prioriteetti (20-nice): 1...40 (huonosta hyvään)
 - u sekava termistö: prioriteetti, prioriteettiluku, tehokas prioriteetti, nice
- n **Prosessi luopuu suorittimesta ennen aikaviipaleen päättymistä**
 - u *epri--*
 - u silti aina $epri \geq nice - 5$ ja $epri \geq -20$
 - u suosii i/o-sidonnaisia prosesseja
- n **Prosessi käyttää aikaviipaleensa loppuun**
 - u *epri++*
 - u silti aina $epri \leq nice + 5$ ja $epri \leq 19$
 - u rankaisee cpu-sidonnaisia prosesseja

Linux – nälkiintymisen esto

kernel 2.6 [DDC04]

- n Jos paljon korkean prioriteetin töitä, niin miten käy matalan prioriteetin töille (joilla iso *nice* arvo)?
- n epoch – aika, jonka kuluessa vuoro pitäisi saada
 - u kaksi tilaa *ready* prosesseille: *active*, *expired*
- n vain *active*-tilassa olevia vuorotetaan
- n molemmilla tiloilla omat run-jononsa



Linux – nälkiintymisen esto

kernel 2.6 [DDC04]

- n jos jonkun odotus ”liian kauan”, niin aikaviipaleen jälkeen prosessi ehkä *expired*-tilaan
 - u *nice* = 19 → aina expired tilaan (aina)
 - u *nice* = 0 → expired tilaan, jos säätö > -1 (aika usein)
 - u *nice* = -20 → expired tilaan, jos säätö > +3 (aika harvoin)
 - u jne lineaarisesti
- n kun kaikki *expired*-tilassa siirrä kaikki taas *active*-tilaan
 - u nopea operaatio osoittimia vaihtamalla
- n reaaliaikaprosessit aina heti takaisin *active*-tilaan
- n ”liian kauan” = $10n$ sekuntia, kun n prosessia jonoissa

Linux: aikaviipale

kernel 2.4?
[Tane01]

n *quantum*

- u montako 10 ms jaksoa (*jiffy*) voi olla suorituksessa
- u vähenee jokaisen käytetyn *jiffyn* jälkeen
 - F jos *quantum* == 0, niin vuoro menee
 - F suspend tilan jälkeen jatketaan jäljellä olevalla *quantumilla*

n Kun kaikilla *quantum* == 0,

- u kaikille (myös odottaville) lasketaan uusi quantum:

$$F \text{ quantum}_{\text{uusi}} = \text{quantum}_{\text{jäljellä}} / 2 + \text{priority}$$

priority: 1-40

$$F \text{ prio} = 20, \text{ quantum}_{\text{jäljellä}} = 0 \rightarrow \text{quantum}_{\text{uusi}} = 20 \text{ jiffies} = 200 \text{ ms}$$

$$F \text{ prior} = 20, \text{ quantum}_{\text{jäljellä}} = 10 \rightarrow \text{quantum}_{\text{uusi}} = 25 \text{ jiffies} = 250 \text{ ms}$$

$$F \text{ prior} = 30, \text{ quantum}_{\text{jäljellä}} = 18 \rightarrow \text{quantum}_{\text{uusi}} = 39 \text{ jiffies} = 390 \text{ ms}$$

- u Aina kun i/o-sidonnaiselta työltä loppu *quantum*, myös CPU-sidonnaiset saavat takaisin alkuperäisen ison *quantumin*
 - F niillä on huono cpu prioriteti, mutta iso *nice* arvo

Linux aikaviipale

n Ongelma

- u prosessi luo paljon kopiota (clone), ja kahmii kaiken CPU-ajan, koska niitä on monta ja jokaisella on täysi iso *quantum*

n Ratkaisu

- u kloonatun prosessin ensimmäinen *quantum* otetaan isäprosessin *quantumista* (50%)
- u uuden *quantumin* laskennassa (seuraava *epoch*) molemmat saavat prioriteettinsa mukaisen arvon

Linux: Vuorottaminen

- n Tarkista jokaisella `schedule()`:n heräämisellä (1 ms välein?), onko syytä vaihtaa suorituksessa olevaa prosessia (preemption points)
 - u perustuu sekä *quantumiin* että prioriteettiin
- n Jos jonkun prosessin *hyvyys* parempi kuin suorituksessa olevan, vaihda
- n Prosessin *hyvyys* (*goodness*):

```
if (policy == SCHED_FIFO or SCHED_RR)  
    goodness = 1000 + priority
```

reaaliaika

```
if (policy == SCHED_OTHER && quantum > 0)  
    goodness = quantum + priority
```

```
if (policy == SCHED_OTHER && quantum == 0)  
    goodness = 0
```

(/usr/src/linux/kernel/sched.c)

Linux SMP tuki

(kernel 2.6 , [DDC04])

n Kuorman tasapainotus (load balancing)

- u joka suorittimella oma ready-jono
- u `schedule()` herää 1 ms välein (jollekin suorittimelle)
- u jos `schedule:n` suorittava suoritin itse *idle*, niin ota (viime aikoina) pisimmän ready jonon omaavalta suorittimelta 1/4 prosesseista
 - F lkm-erotus puolittunut
- u jos joku muu suoritin on *idle*, niin siirrä ruuhkaisimmalta suorittimelta töitä tälle suorittimelle, kunnes jonottavien töiden lkm-erotus on puolittunut
 - F ei aina, mutta 200 ms välein
 - F ei aina, mutta jos ruuhkaisella ready-jonon pituus on 25% isompi kuin tällä `schedule:n` suorittavalla suorittimella
- u siirrettävä prosessi valitaan siten, että kauiten aikaa odottanut valitaan ensin
 - F sen tietoja luultavasti ei ole enää välimuisteissa

Käyttöjärjestelmät II

SVR4

VUOROTTAMINEN

SVR4: Vuorottaminen

- n Kokonaan uusittu Unix-vuorottaminen
- n Ydin itse ei-keskeytyvä (nonpreemptable), mutta vuorottaa tarkistuspisteissä (preemption points)
- n Kolme prioriteettiluokkaa
 - u 159-100: reaaliaika
 - u 99-60: ydin
 - u 59-0: osituskäyttö
- n RT- ja ydin-luokassa kiinteät prioriteetit
- n Osituskäytössä vaihtuva prioriteetti
- n RT-töillä aina samanpituisen aikaviipale
- n Osituskäytössä eripituisia aikaviipaleita
 - u Prioriteetti 0: 10 ms, prioriteetti 59: 100 ms

Käyttöjärjestelmät II

Windows 2000

VUOROTTAMINEN

Ch 10.5 [Stal 05]

Ch 11.4 [Tane01]



W2K vuoronanto

n säie itse suorittaa schedulerin, kun

- u blokkaa resurssiin
- u signaloi toiselle (V-operaatio)
- u aikaviipale päättyy

n scheduler käynnistyy DPC:nä (delayed proc call)

- u I/O-keskeytykskäsittelyn jälkeen
- u semaforin tms timeout'in jälkeen
- u alkuper. kesk. käsittelyn aika on minimoitu
- u DCP:llä on oma suoritusympäristö (säie)

n vuoronanto ytimen säikeiden pohjalta

- u monta säiettä – enemmän suoritinaikaa!

W2K prioriteettiluokat

Fig 10.14 [Stal 05]

- n Keskeytyvä, aikaviipaleet
- n Luokat 0-31, jokaiselle oma RR-jono
- n “Reaaliaikatyöt” - ei takuita vasteajasta
 - u 16 kiinteää prioriteettia
 - u myös KJ:n säikeet
- n Muut työt (variable)
 - u 16 vaihtelevaa prioriteettia
 - F käytti koko viipaleen → pienennä
 - F odottaa I/O:ta → kasvata

W2K prioriteetin määräytyminen

- n **Prosessilla 6 luokkaa, samat kaikille sen säikeille**
 - u realtime, high, above normal, normal, below normal, idle
- n **Säikeellä 7 luokkaa**
 - u time critical, highest, above normal, normal, below normal, lowest, idle
- n **42 kombinaatiota mapataan 32 todelliseen luokkaan**
 - u *base priority*
 - u luokat 17-21 KJ:lle?
 - u user prosessilla luokat 1-15
 - u luokka 0 on Zero Page –säikeelle
 - u luokka -1 todelliselle idle säikeelle
 - F idle looppin skedulerin sisällä?

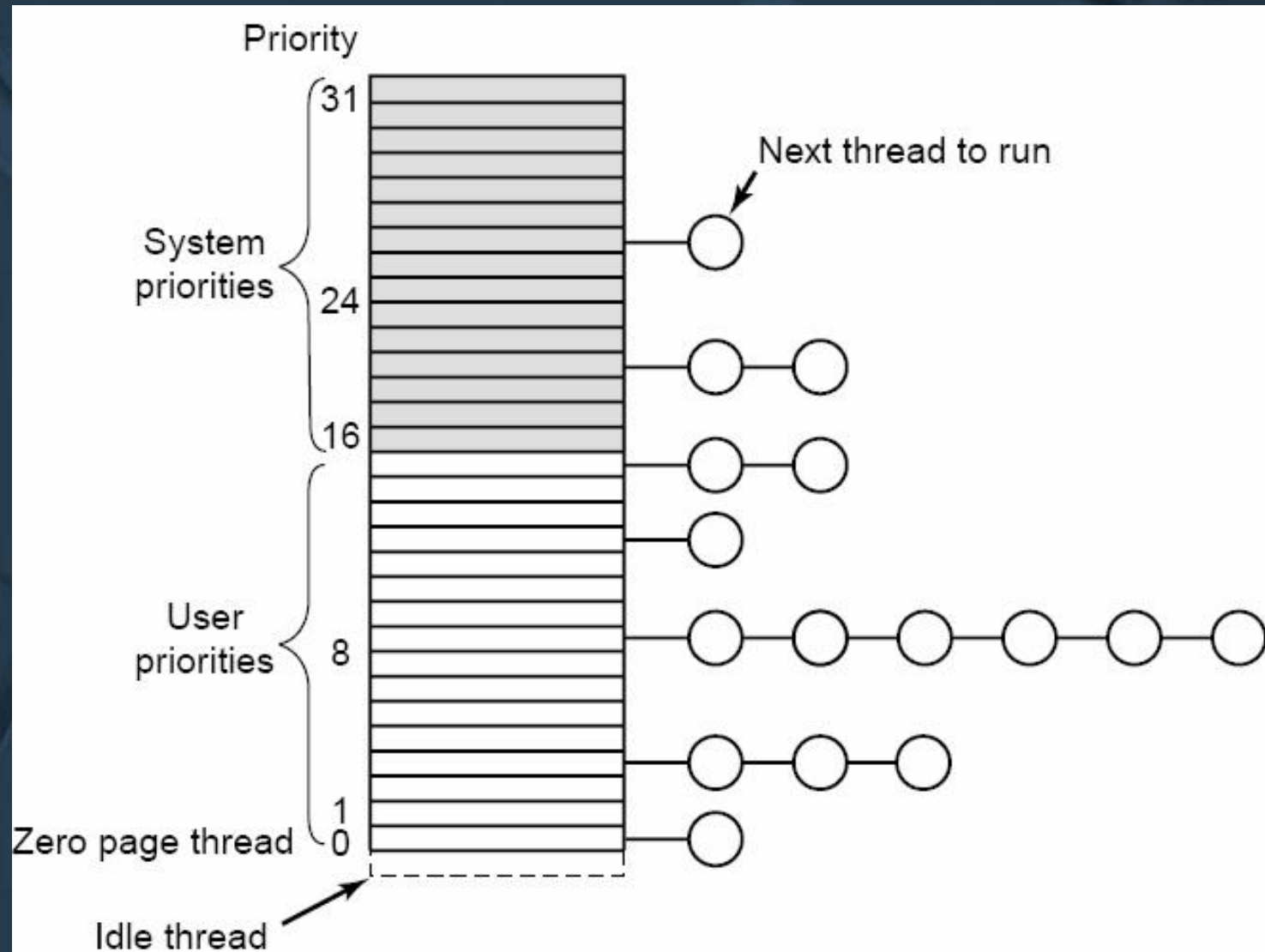
Fig 11-18 [Tane01]

Fig 11-19 [Tane01]

		Win32 process class priorities					
Win32 thread priorities		Realtime	High	Above Normal	Normal	Below Normal	Idle
	Time critical	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

Fig. 11-18. Mapping of Win32 priorities to Windows 2000 priorities.

([Tane01])



([Tane01]) Fig. 11-19. Windows 2000 supports 32 priorities for threads.

W2K prioriteetin vaihdot

([Tane01])

- n **Prioriteettitasoilla 16-31 ei muutoksia (yleensä)**
- n **I/O-keskeytyksen jälkeen säikeelle plussaa**
 - u +1 jos levy I/O, +6 jos KBD, +8 jos äänikortti (rajojen puitteissa)
- n **synkronointiodotuksen (esim. semafori) jälkeen plussaa**
 - u +2 jos foreground, +1 jos background
- n **GUI säie saa herätettäessä plussaa**
 - u +2 jos foreground, +1 jos background
- n **aina kun aikaviipale päättyy, miinusta (-1)**
 - u mutta ei huonommaksi kuin *base priority*
- n *"odottaminen kannattaa, erityisesti musiikin"*

W2K priority inversion

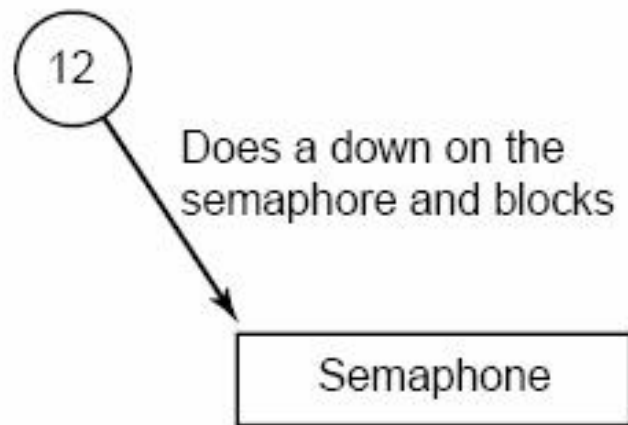
Fig 11-20 [Tane01]

n Ongelma

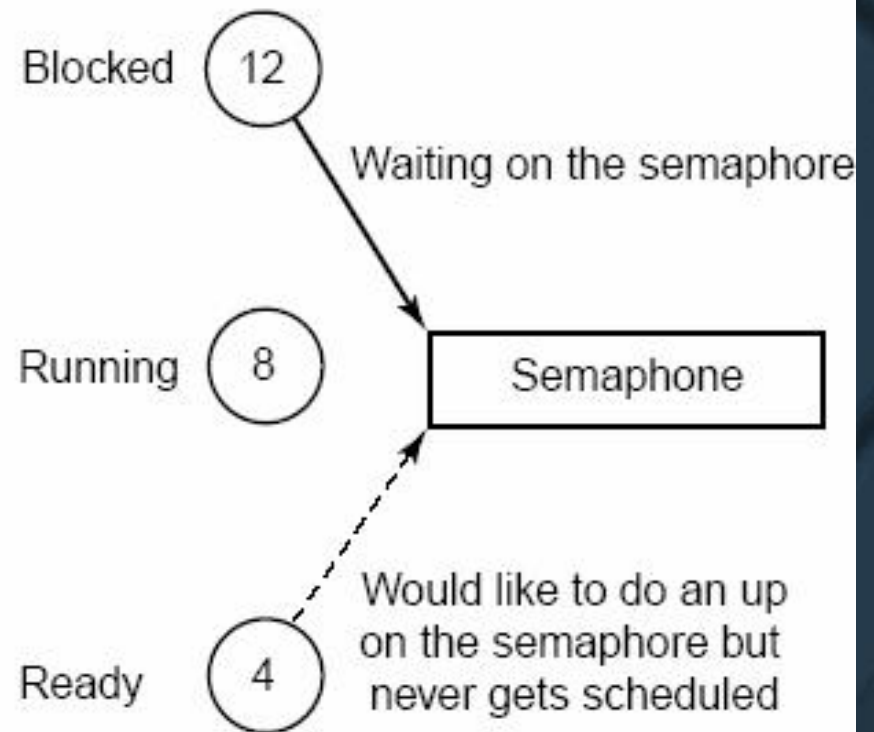
- u korkean prioriteetin (esim. 12) työ odottaa resurssia, joka on matalan prioriteetin (4) työllä
- u matalan prioriteetin (4) työ odottaa suoritinta, joka on vähän korkeamman prioriteetin (8) työllä
- u korkeimman prioriteetin (12) työ ei etene

n Ratkaisu

- u jos työ (prior. 4) odottaa kauan aikaa, niin
 - F vähäksi aikaa sille paljon plussaa: +15
 - F kahden aikaviipaleen jälkeen kaikki plussa pois
- u vain KJ säikeille? (jotka odottivat semaforia?)



(a)



(b)

Fig. 11-20. An example of priority inversion.

([Tane01])

W2K aikaviipale

n yleensä vakio

- u 20 ms (W2K Professional)
- u 120 ms (W2K uniprocessor server)

Ajat yli 10 vuotta sitten päätetty.
Edelleen OK?
Olisiko 2 ja 12 ms parempi?

n voi säätää monikerroiksi 2x, 4x, 6x

n ikkunasäie saa aktivoituessaan pidemmän aikaviipaleen

W2K: SMP tuki

n **Suurimpien prioriteettien säikeille kullekin luokalle omat CPUt (affinity)**

u N suoritinta → N-1 korkeimman prioriteetin säikeellä dedikoitu suoritin

F aina KJ-säikeitä????

u muille säikeille vain yksi suoritin

n **Yhteinen ready-jono taulukko (32 kpl)**

u spin-lock mutex

u voi olla pullonkaula?

n **Eri politiikka Professional ja Server konfiguraatioille?**

([Stall01])

??

([Tane01])

Kertauskysymyksiä

- n Löyhästi kytketty vs. tiukasti kytketty moniprosessorijärjestelmä
- n Miten rinnakkaisuutta on tapana luokitella?
- n Säikeiden vuorottamiseksi on esitetty 4 perusmallia, mitkä?

- n Hard RT vs. Soft RT
- n RT-järjestelmien peruspiirteet?
- n RT-vuorottamisalgoritmien luokittelu
- n Mitä tietoja RT-vuorottaja tarvitsee?
- n Linux ja W2K vuorotuksen isot erot?