LUENTO 13

VUOROTTAMINEN: SMP ja Reaaliaikajärjestelmät Linux, W2000

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

1

Vuorottaminen yhdellä suorittimella

- Milloin vuorotetaan?
  - Short-term, median-term, long-term
- Mitä optimoidaan järjestelmän toiminnassa?
  - Vastausaika, läpimenoaika, ennustettavuus, moniajoaste
- Prioriteetit
- Vuorottamisalgoritmeja,
  - FCFS (FIFO), Round-Robin, Shortest Remaining Time, Multilevel Feedback, …

2

Vuorottaminen moniprosessorijärjestelmässä

3

Moniprosessorijärjestelmä

- Löyhästi kytketyt (loosely coupled)
  - erillisten koneiden ryväs (cluster)
  - hajautettu järjestelmä (distributed system)
- Toiminnollisesti erikoistuneet
  - erilliset I/O-prosessorit
  - matikkaprosessori nykyään osa CPU:ta
- Tiukasti kytketyt (tightly coupled) Tällä luennolla!
  - prosessoreilla yhteinen keskusmuisti
  - isäntä - rengit (host-slaves)
  - SMP, symmetric multiprocessing

4

Rinnakkaisuus & Vuorottaminen

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

5

SMP Prosessin vuorottaminen

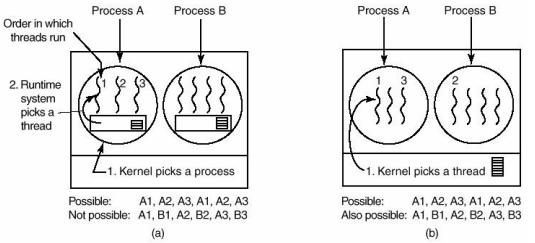
- Isäntä-renki (master-slave)
  - KJ aina isäntäCPU:lla (*master*)
    - vuorottaminen, muistinhallinta, I/O
  - muut CPU:t suorittavat vain sovelluksia (*slaves*)
  - jos isäntä kuukahtaa?
  - suorituskyky?
- SMP (peer architecture)
  - KJ millä tahansa CPU:lla, jopa yhtäaikaa
    - vapaakäyntisyys
    - synkronointi
    - vikasietoisuus
  - kukin CPU vuorottaa itse itseään

6

SMP Prosessien vuorottaminen

- Yhteiset jonot
  - kullakin prioriteetilla oma Ready-jono
  - yhteiset muut resurssit
- Prosessi ei yleensä sidottu tiettyyn CPU:hun
  - vapaa prosessori vuorottaa yhteisestä Ready-jonosta
  - "common CPU-pool"
- Jos sidotaan tiettyyn suorittimeen, tarvitaan myös suoritinkohtaiset Ready-jonot
  - vuorota aina ensin omasta jonosta
  - jos jono tyhjä, vuorota globaalista jonosta
    - onko sellaista?
    - käydään muiden jonot läpi?

Säikeiden vuorottaminen



ULT säikeen CPU-aika 10% aikaviipaleesta

(Fig 2-43 [Tane01])

- **ULT: KJ vuorottaa prosesseja, säiekirjasto säikeitä**
- **KLT: KJ vuorottaa säikeitä**

Säikeiden vuorottaminen

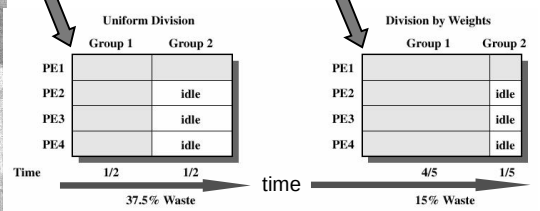
- Suoritus yhteisessä osoiteavaruudessa
  - kullakin säikeellä oma suoritusympäristö
    - pino, tallealue rekistereille
  - muut resurssit yhteisiä
    - koodi, globaali data, avoimet tiedostot
  - kommunikointi yhteisen data-alueen välityksellä
- Rinnakkain suoritettavat osat säikeiksi
- Erilaisia tapoja sijoitella säikeet prosessoreille:
  - kuorman jako (load sharing)
  - kimppavuorottaminen (gang scheduling)
  - suorittimen sitominen (dedicated processor assignment)
  - dynaaminen vuorottaminen

Kuorman jako (load sharing)

- Säikeet jaettavissa usealle CPU:lle
  - jouten olevat käyttöön
  - parantunut suorituskyky, lyhyemmät läpimenoajat
- Saman prosessin säikeet eri CPU:lle
  - aidosti rinnakkainen suoritus
    - huomioitava sovelluksessa!
  - välimuistin hyödyntäminen?
- Globaali Ready-jono
  - tarvitaan poissulkeminen
  - pullonkaula?
- Kukin CPU vuorottaa itsensä (valitsee säikeen)
  - FCFS
  - (prioriteetti: esim. pienin säikeiden lkm)

Kimppavuorottaminen (gang scheduling)

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



(Fig 10.2 [Stal05])

CPU:n sitominen (Dedicated CPU assignment)

- Osa tai kaikki CPU:t varataan yhdelle sovellukselle
  - varaus kestää, kunnes sovellus päättyy
- Jos säie estynyt (*blocked)*, CPU silti varattuna
- Hyöty?
  - välttää prosessin vaihtoja, prioriteetti
  - ko. sovelluksen lyhentynyt läpimenoaika
  - välimuisti
- Muut saattavat kärsiä
  - jätettävä joku/joitakin CPU:ita varaamatta
- Ero kimppavuorottamiseen?
  - ei vuorotusta, varaus sovelluksen ajaksi
  - säie aina samalla suorittimella

Dynaaminen vuorottaminen (dynamic scheduling)

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

13

REAALIAIKAJÄRJESTELMÄT
Yleiskuva

14

Reaaliaikajärjestelmät

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

15

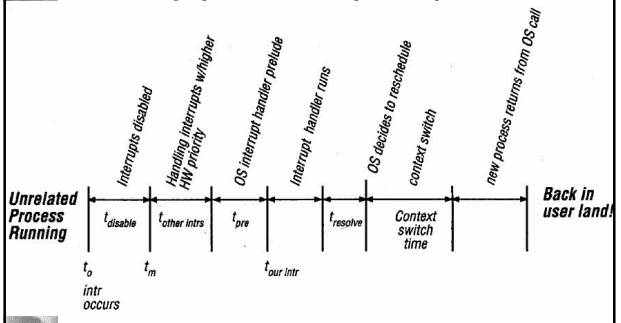
Vaatimuksia

- Deterministisyys
  - käsittely kiinteästi ennaltamääriteltyinä aikoina tai kiintein aikavälein
  - pakottaa KJ:n reagoimaan keskeytykseen
    - yläraja olemassa kuittaamiselle (muutama ms)
- Reagointinopeus (responsiveness)
  - kauanko menee keskeytyksen käsittelemiseen?
    - keskeytyksen huomioiminen
    - palvelun suorittaminen

→ Vasteaika (response time R, turnaround time TAT)

16

Keskeytyskäsittelyn viipeet



lähde?

17

Reaaliaikajärjestelmän vaatimuksia

- Ohjelmoijan kontrollointi
  - mitä osia lukitaan muistiin tai KJ ei lainkaan sivuta
  - vuorottamisalgoritmin valinta
  - prioriteetti, aikaviipaleen pituus, aikarajat, kesto
  - asynkroninen siirräntä
- Luotettavuus (reliability)
  - suorituskyvyn tipahtaminen voi olla katastrofi
  - Hard RT vs. Soft RT
  - virheistä toipuminen (fail-soft operation)
    - nilkuttaa saa, muttei kaatua

18

Peruspiirteitä 1/2

- Minimijoukko toiminnallisuutta
  - pieni ja tehokas toteutus
- Reagoitava nopeasti ulkoisiin keskeytyksiin
  - minimoitava kohdat, joissa keskeytykset estettynä
    - mukaan lukien keskeytysten käsittely!
- Nopea prosessin / säikeen vaihto
- Keskeytyvä vuorottaminen (preemptive)
  - prioriteetit
  - vähemmän kiireellinen homma voidaan hyllyttää
- Moniajo, välineet prosessien kommunikointiin
  - semaforit, signaalit, tapahtumat, ...

19

Peruspiirteitä 2/2

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

20

Vuorottaminen reaaliaikajärjestelmissä

21

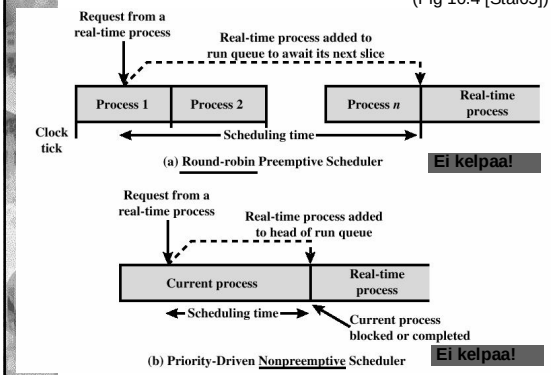
RT: Vuorottaminen

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

22

RT: Vuorottaminen

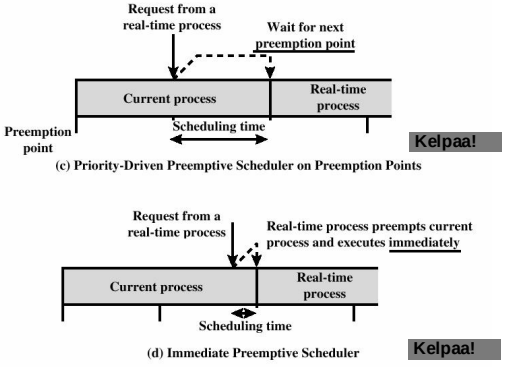
(Fig 10.4 [Stal05])



23

RT: Vuorottaminen

(Fig 10.4 [Stal05])



24

Staattiset lähestymistavat

- Voiko kuormaa analysoida?
  - etukäteen tai ajonaikana; aikataulu
- Staattinen taulukkoperustainen — Tehdään etukäteen!
  - sopii jaksollisille (periodic) töille
  - alkuaika, kesto, päättymisaika, prioriteetti
  - Esim: EDF, Earliest Deadline First -algoritmi
- Staattinen prioriteetteihin perustuva — Analyysi etukäteen!
  - normaali keskeyttävä (preemptive)
  - prioriteetti määräytyy annettujen aikarajojen nojalla
  - Usein RMS, Rate Monotonic Scheduling –algoritmi

25

Dynaamiset lähestymistavat

- Dynaaminen aikatauluttaminen (planning based)
  - aperiodisille töille
  - kun uusi työ käynnistyy, laske prioriteetti ja uusi käytettävä aikataulu
  - jos pystyy takaamaan aikarajat, ota työ suoritukseen
  - käytä esim EDF-vuorottamista
- Dynaaminen, parhaan kyvyn mukaan ponnistelu (best effort)
  - kuten edellä, mutta ota suoritukseen ja rukoile, että onnistuu

26

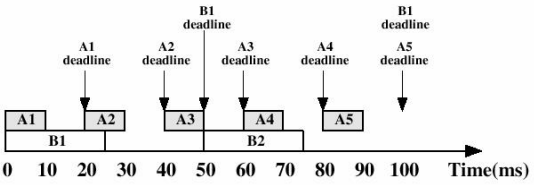
Earliest Deadline First

- Prioriteetti ≠ aikarajat
- Tarvitaan lisätietoa työstä (=ohjelmoijalta)
  - milloin valmis suoritukseen (Ready time)
    - periodinen ⟶ aikajono
    - aperiodinen ⟶ saatetaan tietää tai KJ valpas huomaamaan
  - takaraja aloitukselle, takaraja valmistumiselle
  - prosessointiaika, muiden resurssien tarve
  - prioriteetti
  - onko pakollisia alitöitä?
- KJ voi laatia suoritusaikataulun

27

Esimerkki

Fig 10.5 [Stal05]



Kaksi jaksollista (periodic) 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**

28

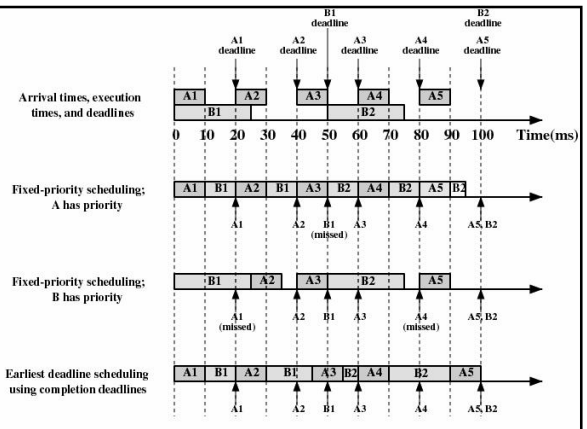


Figure 10.5 Scheduling of Periodic Real-time Tasks with Completion Deadlines (based on Table 10.2)

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

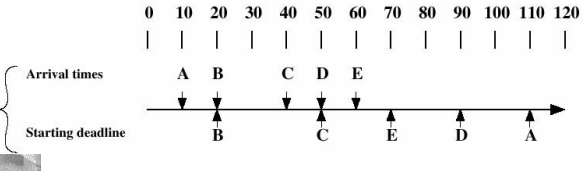


Fig 10.6 [Stal05]

30

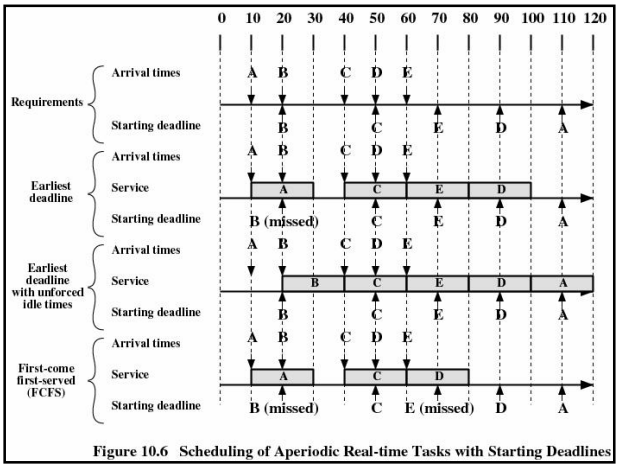
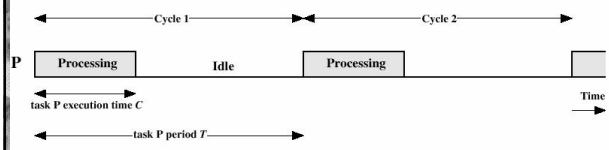


Figure 10.6 Scheduling of Aperiodic Real-time Tasks with Starting Deadlines

Rate Monotonic Scheduling



(Fig 10.7 [Stal05])

- Vain jaksollisille (periodisille) töille
- Sama työ tasaisella tahdilla
  - käyttöaste yhden työn osalta on U = C/T
- Jakson loppu = Hard deadline
- Pienin jakso (T) = suurin prioriteetti

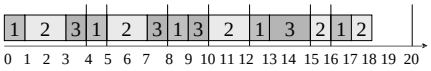
Rate: määrä yksikköä kohden, taajuus

Rate monotonic

- Ajoittaja toimiessaan valitsee kullakin tapahtumahetkellä suoritukseen sen, jonka prioriteetti (eli taajuus) on korkein

	p/d	e
T1	4	1
T2	5	2
T3	20	5

• Esim. ajanhetkellä 16: kun T1 tulee suoritukseen, se keskeyttää T2:n suorituksen.



Ajoitettavuuden arviointi käyttöasteen avulla

Tbl 10.4 [Stal05]

- RMS: Selvästi

$$U_i = C_1/T_1 + C_2/T_2 + .. + C_n/T_n <= 1$$

- Riittävä ehto sille, että työt voidaan ajoittaa RMS-algoritmilla on

$$C_1/T_1 + C_2/T_2 + .. + C_n/T_n <= n(2^{1/n} - 1)$$

- Arvon n kasvaessa, RMS:lle yläraja lähenee

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

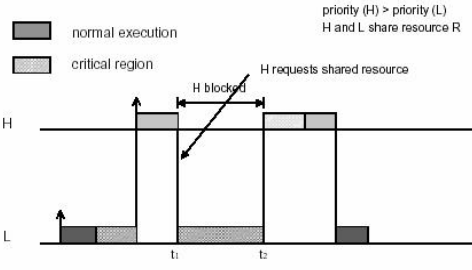
- EDF: tehokkaampi, sille riittää (*riittävä ja välttävä ehto*)

$$U_i = C_1/T_1 + C_2/T_2 + .. + C_n/T_n <= 1$$

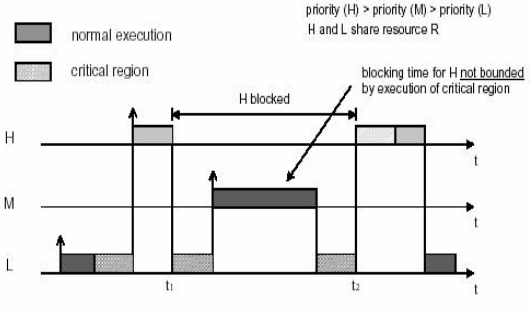
Mitä voi mennä pieleen ?

- Korkean prioriteetin työ joutuu odottamaan, vaikka ei pitäisi!
- Syynä on resurssikilpailu
  - Yhteisiä resursseja, kriittiset alueet, odotus
  - Prioriteetin kääntyminen, lukkiumat
  - Keskeytymättömät kriittiset alueet

Odotusongelma (Blocking problem)



Prioriteetin kääntyminen (Priority inversion)



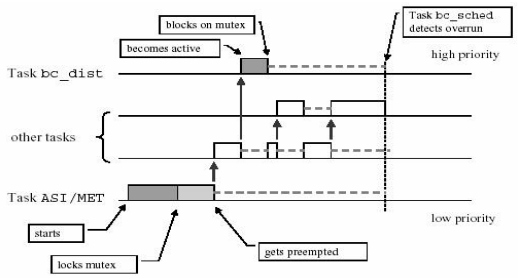
37

Mars Pathfinder

- Laskeutui 4.7.1997
- Kohtaa ohjelmisto-ongelmia (software glitches).
- Pathfinder kokee toistuvia RESET:jä meteorologisen tiedon keruussa.
- Prioriteetin kääntymisestä aiheutuvia ajoitusten ylityksiä.
- http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html

38

Prioriteetin kääntyminen Mars Pathfinder



39

Prioriteetin kääntymisen välttäminen

- Keskeytymättömät kriittiset alueet
  - Luovat tarpeetonta odotusta.
  - Käyttökelpoisia vain lyhyille kriittisille alueille.
- Sisääntuloprotokollia kriittiselle alueelle
  - Prioriteetin perintä (Priority Inheritance Protocol).
  - Prioriteetin kattomenetelmä (Priority Ceiling Protocol).

40

Prioriteetin perintä

- Idea: Jos tapahtuma T estää (blocks) yhden tai useamman korkeampi prioriteettisen tapahtuman etenemisen, tapahtuma T perii väliaikaisesti korkeimman estetyn tapahtuman prioriteetin.
- Hyödyt
  - Estää prioriteetiltaan keskitasoa olevan tapahtuman keskeyttämästä tapahtumaa T.
- Haitat
  - Prioriteetin perintä voi aiheuttaa lukkiuman
  - Linkitetty odotus (chained blocking)

41

Prioriteetin perintä

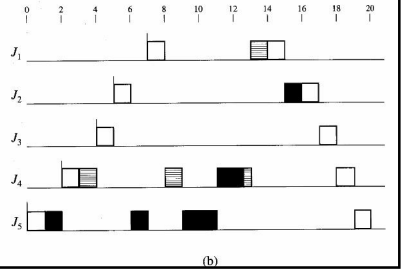
- HUOM: 1 on korkein prioriteetti

Ajanhetkellä

- 6: J5 perii J2:n perioriteetin, koska J2 joutuu odottamaan mustaa resurssia.
- 8: J2 perii J1:n prioriteetin, koska J1 joutuu odottamaan mustaa resurssia.
- 9: J5 perii J1:n prioriteetin J4:ltä, koska J4 haluaa myös mustan resurssin.

Job	r_i	e_i	π_i	Critical Sections
J_1	7	3	1	[*Shaded*; 1]
J_2	5	3	2	[*Black*; 1]
J_3	4	2	3	
J_4	2	6	4	[*Shaded*; 4 [*Black*; 1.5]]
J_5	0	6	5	[*Black*; 4]

(a)



(b)

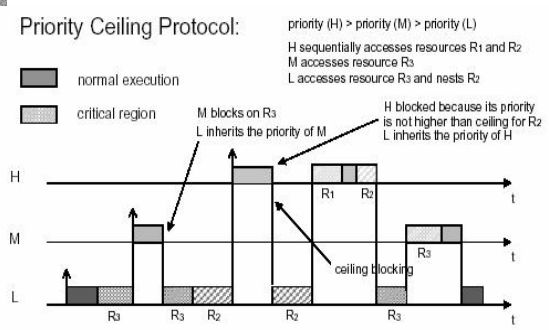
Liu Kuva 8-8

Prioriteettikatto

- Idea
  - Jokaiselle resurssille asetetaan prioriteettikatto $\Pi(R_i)$ yhtä suureksi kuin sen korkeimman tapahtuman prioriteetti, joka tarvitsee tätä resurssia ja siis saa lukita resurssin.
  - Tapahtuma T saa siirtyä kriittiselle alueelle ja varata resurssin vain, jos sen prioriteetti on korkeampi kuin kaikkien muiden samanaikaisten tapahtumien sillä hetkellä varaamien resurssien prioriteettikatot $\Pi(t)$.
  - Jos tapahtuma T estää yhden tai useamman korkeampi prioriteettisen tapahtuman, se väliaikaisesti perii korkeimman estetyn tapahtuman prioriteetin.
- Ongelma:
  - Tiedettävä resurssien käyttö etukäteen!

43

Prioriteettikatto



Esimerkki: prioriteetin katto

Job	ri	ei	πi
J1	7	3	1
J2	5	3	2
J3	4	2	3
J4	2	6	4
J5	0	6	5

Job	kriitt. alueet
J1	[Sh;1]
J2	[Bl;1]
J3	
J4	[Sh;4 [Bl;1.5]]
J5	[Bl;4]

Liu kuva 8-10

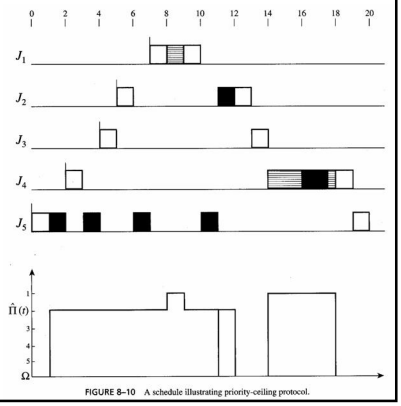


FIGURE 8–10 A schedule illustrating priority-ceiling protocol.

Linux
VUOROTTAMINEN

46

Linux: Vuorottaminen

POSIX

- Poikkeaa perinteisestä Unixista
  - root: **`sched_setscheduler()`**
- Soft Real-time työt:
  - **`SCHED_FIFO`**
    - kiinteät prioriteetit, tapahtumaohjattu (ei aikaviipaletta)
    - suoritus keskeytyy vain jos
      - suuremman prioriteetin työ (task) Ready-tilaan
      - työ joutuu Blocked-tilaan
      - työ itse luovuttaa CPU:n (*sched_yield()*)
  - **`SCHED_RR`**
    - kuten edellinen, mutta lisäksi aikaviipale
- Muut, osituskäytön työt, vaihtelevan prioriteetin työt:
  - **`SCHED_OTHER`**
  - käytä tätä, jos edellisissä ei töitä Runnable-tilassa

47

Linux: suorittimen prioriteetti

- Vuorottaminen säikeen tasolla (työ ~ task)
  - ytimen säikeet
  - joka suorittimella omat jonot
- Luokat: -20 ... +19
  - perusprioriteettiarvo: 0 (paras user-taso)
  - joka luokalla oma run-jono, negatiiviset KJ:lle
- Joka prosessilla staattinen prioriteetti: *nice* [-20,+19]
  - pieni nice-arvo ⟶ pieni prioriteettiarvo ⟶ hyvä juttu
  - user prosesseilla nice välillä [0, +20]
- Lähtökohtaprioriteetti = *nice (-19 ... +20)*
- Dynaaminen vaihtelu, vaihteluväli: *±5 (säätö)*
- Tehokas prioriteettiluku: *epri = nice +säätö*
  - *aina rajoissa [-19, +20]*

48

Linux: prioriteetin vaihtelu

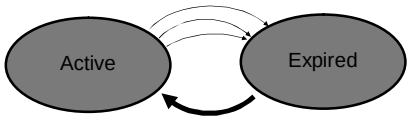
kernel 2.6 [DDC04]

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

kernel 2.6 [DDC04]

Linux – nälkiintymisen esto

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



kernel 2.6 [DDC04]

Linux – nälkiintymisen esto

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

Linux: aikaviipale

kernel 2.4? [Tane01]

- *quantum*
  - montako 10 ms jaksoa (jiffy) voi olla suorituksessa
  - vähenee jokaisen käytetyn *jiffy*n jälkeen
    - jos *quantum* == 0, niin vuoro menee
    - suspend tilan jälkeen jatketaan jäljellä olevalla *quantum*illa
- Kun kaikilla *quantum* == 0, priority: 1-40
  - kaikille (myös odottaville) lasketaan uusi quantum:
    - $quantum_{uusi} = quantum_{jäljellä}/2 + priority$
    - *prio = 20,* $quantum_{jäljellä} = 0 \longrightarrow quantum_{uusi} = 20$ jiffies = 200 ms
    - *prior = 20,* $quantum_{jäljellä} = 10 \longrightarrow quantum_{uusi} = 25$ jiffies = 250 ms
    - *prior = 30,* $quantum_{jäljellä} = 18 \longrightarrow quantum_{uusi} = 39$ jiffies = 390 ms
  - Aina kun i/o-sidonnaiselta työltä loppu *quantum*, myös CPU-sidonnaiset saavat takaisin alkuperäisen ison *quantum*in
    - niillä on huono cpu prioritetti, mutta iso *nice* arvo

Linux aikaviipale

- Ongelma
  - prosessi luo paljon kopiota (clone), ja kahmii kaiken CPU-ajan, koska niitä on monta ja jokaisella on täysi iso *quantum*
- Ratkaisu
  - kloonatun prosessin ensimmäinen *quantum* otetaan isäprosessin *quantum*ista (50%)
  - uuden *quantum*in laskennassa (seuraava *epoch*) molemmat saavat prioriteettinsa mukaisen arvon

Linux: Vuorottaminen

kernel 2.4? [Tane01]

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

```
if (policy == SCHED_FIFO or SCHED_RR)
   goodness = 1000 + priority
if (policy == SCHED_OTHER && quantum > 0)
   goodness = quantum + priority
if (policy == SCHED_OTHER && quantum ==
0)
   goodness = 0
```

reaaliaika

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

Linux SMP tuki

(kernel 2.6 , [DDC04])

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

55

Windows 2000 VUOROTTAMINEN

Ch 10.5 [Stal 05]
Ch 11.4 [Tane01]

56

W2K vuoronanto

- säie itse suorittaa schedulerin, kun
  - blokkaa resurssiin
  - signaloi toiselle (V-operaatio)
  - aikaviipale päättyy
- scheduler käynnistyy DPC:nä (delayed proc call)
  - I/O-keskeytyskäsittelyn jälkeen
  - semaforin tms timeout'in jälkeen
  - alkuper. kesk. käsittelyn aika on minimoitu
  - DCP:llä on oma suoritusympäristö (säie)
- vuoronanto ytimen säikeiden pohjalta
  - monta säiettä – enemmän suoritinaikaa!

57

Fig 10.14 [Stal 05]

W2K prioriteettiluokat

- Keskeytyvä, aikaviipaleet
- Luokat 0-31, jokaiselle oma RR-jono
  - iso numero, iso prioriteetti
- "Reaaliaikatyöt" - ei takuita vasteajasta
  - 16 kiinteää prioriteettia
  - myös KJ:n säikeet
- Muut työt (variable)
  - 16 vaihtelevaa prioriteettia
    - käytti koko viipaleen ⟶ pienennä
    - odottaa I/O:ta ⟶ kasvata

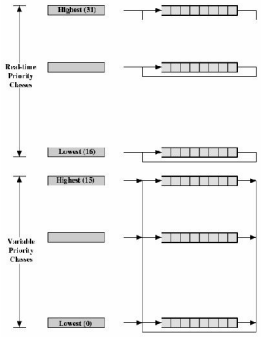


Figure 10.14 Windows Thread Dispatching Priorities

W2K prioriteetin määräytyminen

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

Fig 11-18 [Tane01]

Fig 11-19 [Tane01]

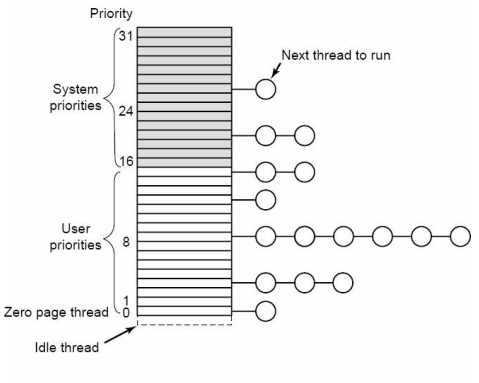
59

		Win32 process class priorities					
		Realtime	High	Above Normal	Normal	Below Normal	Idle
Win32 thread priorities	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])

60



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

W2K prioriteetin vaihdot ([Tane01])

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

W2K priority inversion Fig 11-20 [Tane01]

- Ongelma
  - korkean prioriteetin (esim. 12) työ odottaa resurssia, joka on matalan prioriteetin (4) työllä
  - matalan prioriteetin (4) työ odottaa suoritinta, joka on vähän korkeamman prioriteetin (8) työllä
  - sitä vielä korkeimman prioriteetin (12) työ ei etene
- Ratkaisu
  - jos työ (prior. 4) odottaa kauan aikaa, niin
    - vähäksi aikaa sille paljon plussaa: +15
    - kahden aikaviipaleen jälkeen kaikki plussa pois
  - vain KJ säikeille? (jotka odottivat semaforia?)

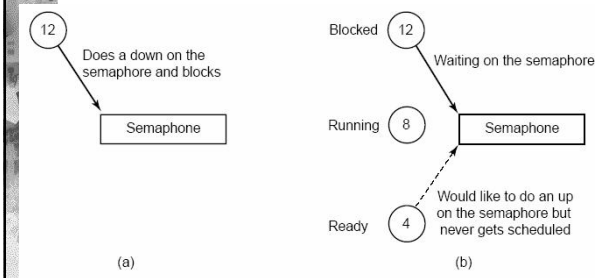


Fig. 11-20. An example of priority inversion.

([Tane01])

W2K aikaviipale

- yleensä vakio
  - 20 ms (W2K Professional)
  - 120 ms (W2K uniprocessor server)

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

- voi säätää monikerroiksi 2x, 4x, 6x
- ikkunasäie saa aktivoituessaan pidemmän aikaviipaleen

W2K: SMP tuki

- Suurimpien prioriteettien säikeille kullekin luokalle omat CPUt (affinity)
  - N suoritinta ⟶ N-1 korkeimman prioriteetin säikeellä dedikoitu suoritin
    - aina KJ-säikeitä????
  - muille säikeille vain yksi suoritin
- Yhteinen ready-jono taulukko (32 kpl)
  - spin-lock mutex
  - voi olla pullonkaula?
- Eri politiikka Professional ja Server konfiguraatioille?

([Stall05])
??
([Tane01])

Fig 11-19 [Tane01]