LUENTO 8

Käyttöjärjestelmät II

VUOROTTAMINEN
SMP JA REAALIAIKAJÄRJESTELMÄT
Linux, W2000

Ch 10 [Stal 05]

(Ch 20 [DDC04], 11.4 [Tane01])

Tämä luento

- **Moniprossorijärjestelmien vuorottaminen**
- **Reaaliaikaskedulointi**
- **Linux: Vuorottaminen**
- **SVR4: Vuorottaminen**
- **W2K: Vuorottaminen**

- **Linux:** Ch 10.3 [Tane01], Ch 20 [DDC04]
- **W2K**: Ch 11.4 [Tane01]

Käyttöjärjestelmät II

VUOROTTAMINEN
Moniprosessori-
järjestelmässä

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

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

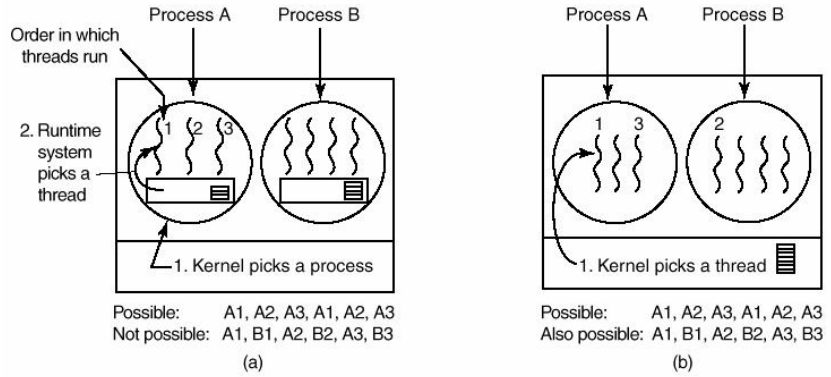
SMP Prosessin vuorottaminen

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

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 suoritin-kohtaiset Ready-jonot**
  - vuorota aina ensin omasta jonosta
  - jos jono tyhjä, vuorota globaalista jonosta
    - onko sellaista?

Säikeiden vuorottaminen



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

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

Säikeiden vuorottaminen

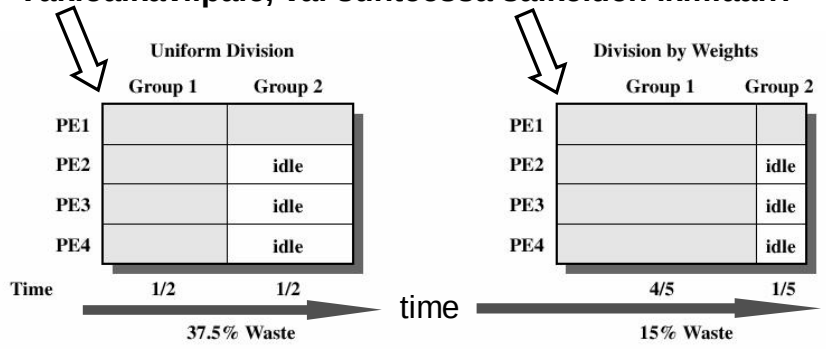
- **Suoritus yhteisessä osoiteavaruudessa**
  - kullakin säikeellä oma suoritusympäristö
    - pino, tallealue rekistereille
  - muut resurssit yhteisiä
    - koodi, globaali data, avoimet tiedostot
  - kommunikointi yhteisen data-alueen välityksellä
- **Rinnakkain suoritettavat osat säikeiksi**
  - 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
  - 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 *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
- **CPU vapautuu**
  - jos joku odottaa ensimmäistä CPU:ta, anna sille
  - muuten anna ensimmäiselle lisä-CPU:ta odottavalle

Käyttöjärjestelmät II

REAALIAIKA-JÄRJESTELMÄT

Yleiskuva

Reaaliaikajärjestelmät

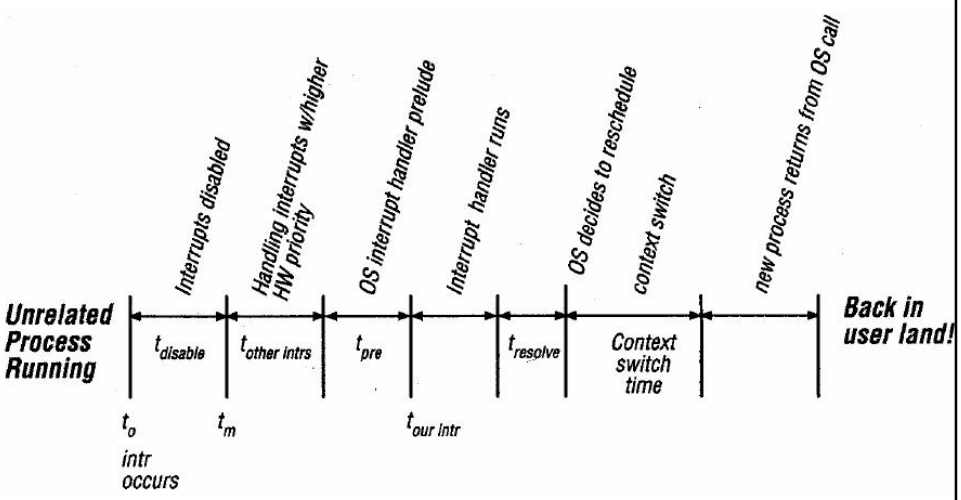
- **Oikeellisuus ei riipu pelkästään tuloksista, vaan myös valmistumisajasta**
- **Tarve reagoida ulkopuolisiin tapahtumiin**
  - ohjausjärjestelmät: laboratoriokokeet, teollisuus, lentoliikenne, teleliikenne, robotiikka, …
- **Hard Real-time vs. Soft Real-time**
  - hard: ei saa missata aikarajoja (deadline)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

Vaatimuksia

- **Deterministisyys**
  - käsittely kiinteästi ennaltamääriteltynä 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)**

Keskeytyskäsittelyn viipeet



lähde?

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

Reaaliaikajärjestelmän peruspiirteitä

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

Reaaliaikajärjestelmän peruspiirteitä

- **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

Käyttöjärjestelmät II

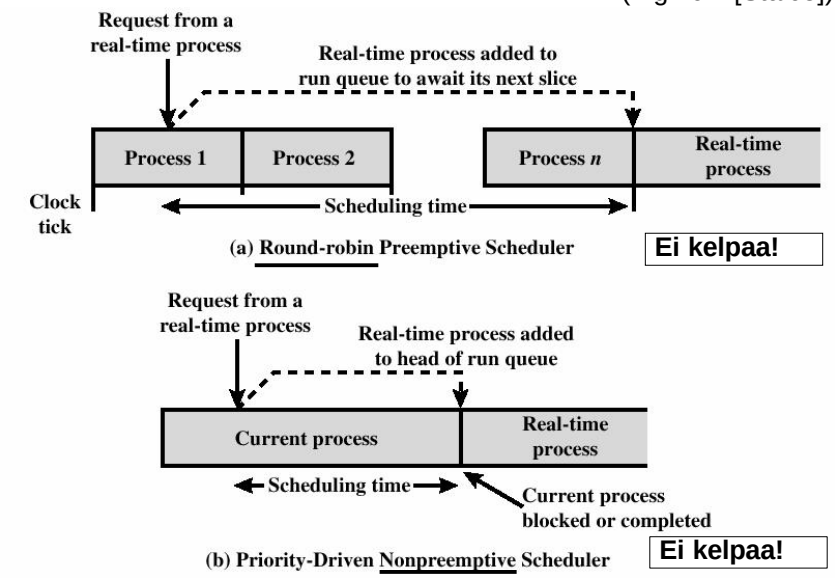
Vuorottaminen reaaliaikajärjestelmissä

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

RT: Vuorottaminen

(Fig 10.4 [Stal05])

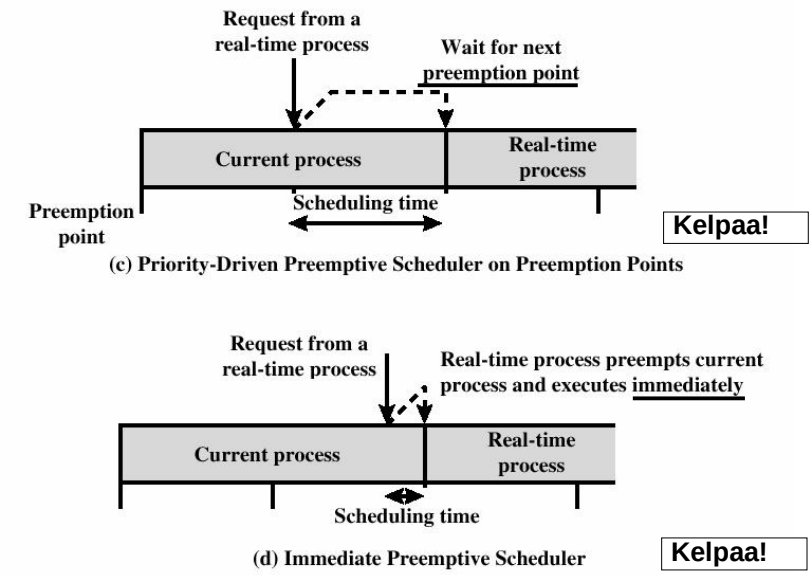


Ei kelpaa!

Ei kelpaa!

RT: Vuorottaminen

(Fig 10.4 [Stal05])



Kelpaa!

Kelpaa!

Staattiset lähestymistavat

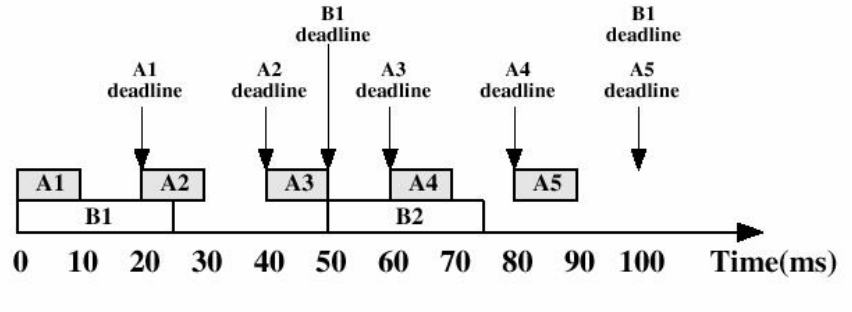
- **Voiko kuormaa analysoida?**
  - etukäteen tai ajonaikana
  - aikataulu
- **Staattinen taulukko-perustainen**
  - sopii periodisille töille
  - alkuaika, kesto, päättymisaika, prioriteetti
  - **EDF, Earliest Deadline First** -algoritmi
- **Staattinen prioriteetteihin perustuva**
  - normaali keskeyttävä (preemptive)
  - prioriteetti määräytyy annettujen aikarajojen nojalla
  - **RMS, Rate Monotonic Scheduling** -algoritmi

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**

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

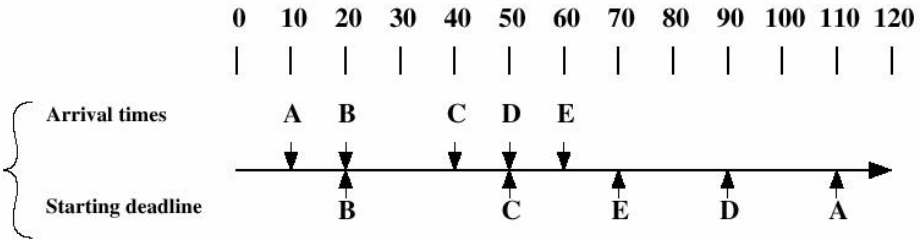
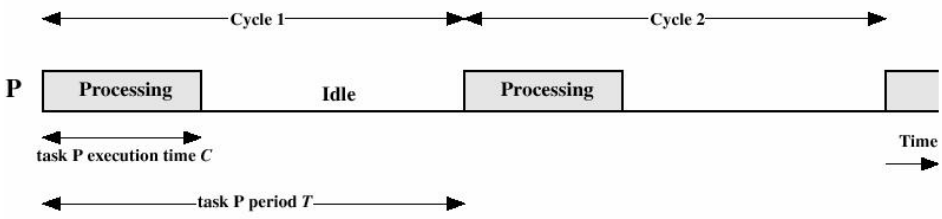


Fig 10.6 [Stal05]

Rate Monotonic Scheduling



(Fig 10.7 [Stal05])

- **Vain 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

RMS - Rate Monotonic Scheduling

- **Selvästi**

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

- **<u>Riittävä ehto</u> 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**

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

Tbl 10.4 [Stal05]

- **EDF tehokkaampi, sille riittää**

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

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

Käyttöjärjestelmät II

Linux

VUOROTTAMINEN

Linux: Vuorottaminen

- **Poikkeaa perinteisestä Unixista**
  - root: ***sched_setscheduler()*** POSIX
- **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

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]*

Linux: prioriteetin vaihtelu

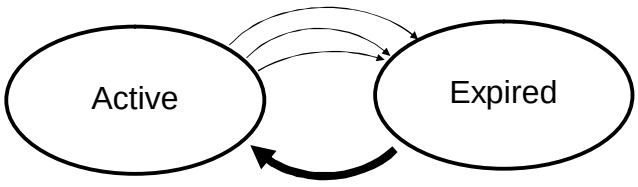
kernel 2.6 [DDC04]

- **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

Linux – nälkiintymisen esto

kernel 2.6 [DDC04]

- **Jos paljon korkean prioriteeetin 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**



Linux – nälkiintymisen esto

kernel 2.6 [DDC04]

- **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,**
  - kaikille (myös odottaville) lasketaan uusi quantum:
    - $quantum_{uusi} = quantum_{jäljellä}/2 + priority$ priority: 1-40
    - *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-jono
  - **`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

Käyttöjärjestelmät II

SVR4

VUOROTTAMINEN

SVR4: Vuorottaminen

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

Käyttöjärjestelmät II

Windows 2000

VUOROTTAMINEN

Ch 10.5 [Stal 05]

Ch 11.4 [Tane01]

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!

W2K prioriteettiluokat

Fig 10.14 [Stal 05]

- **Keskeytyvä, aikaviipaleet**
- **Luokat 0-31, jokaiselle oma RR-jono**
- **"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

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?

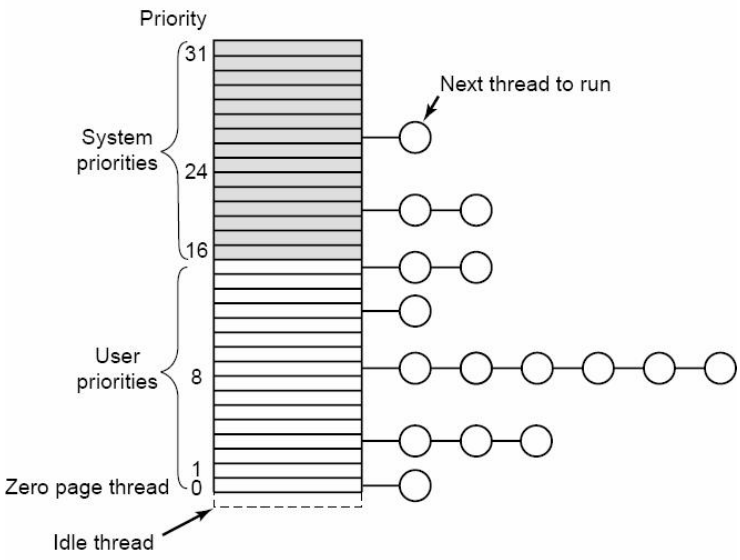
Fig 11-18 [Tane01]

Fig 11-19 [Tane01]

		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])



([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*
- *”odottaminen kannattaa, erityisesti musiikin”*

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
  - 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?)

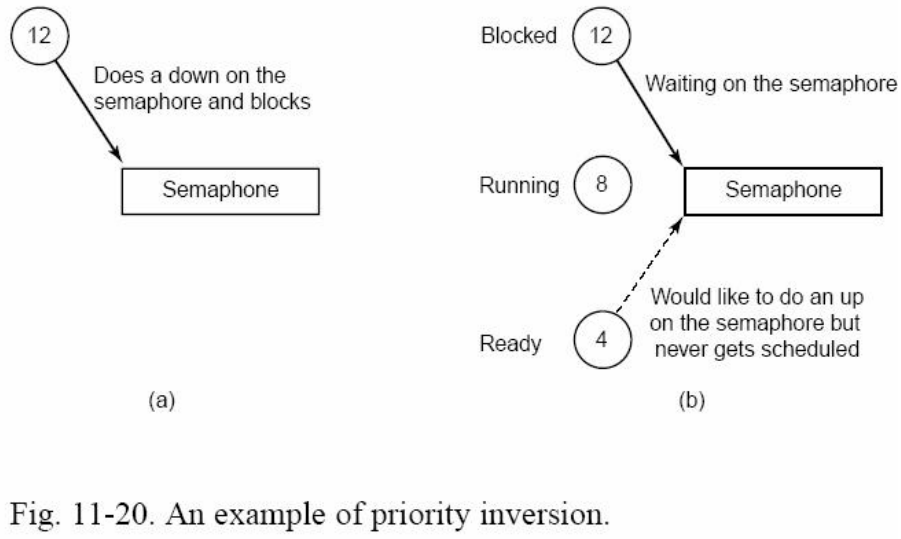


Fig. 11-20. An example of priority inversion.

([Tane01])

W2K aikaviipale

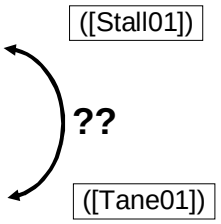
- **yleensä vakio**
  - 20 ms (W2K Professional)
  - 120 ms (W2K uniprocessor server)

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?**



Kertauskysymyksiä

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

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