

LUENTO 13

## VUOROTTAMINEN: SMP ja Reaaliaikajärjestelmät Linux, W2000

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

1

## 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 - renkit (host-slaves)
  - SMP, symmetric multiprocessing

4

## 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

## 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

## Vuorottaminen moniprosessorijärjestelmässä

3

## 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 yhtäaikaan
    - vapaakäyntisyys
    - synkronointi
    - vikasietoisuus
  - kukin CPU vuorottaa itse itsensä

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?

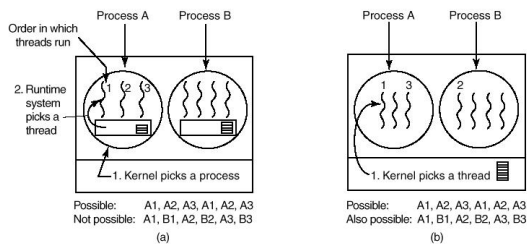
7

## 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)

10

## Säikeiden vuorottaminen



ULT säikeen CPU-aika 10% aikaviipaleesta

(Fig 2-43 [Tane01])

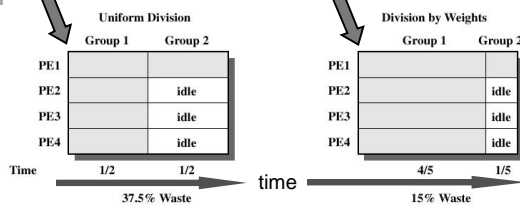
n ULT: KJ vuorottaa prosesseja, säiekirjasto säikeitä -> yksi prosessi per prosessori

n KLT: KJ vuorottaa säikeitä -> yksi säie per prosessori

8

## 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])

## Säikeiden vuorottaminen

- Suoritus yhteisessä osoitevaruudessa
  - 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

9

## 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

12

## 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

## Reaaliaikajärjestelmän vaatimuksia

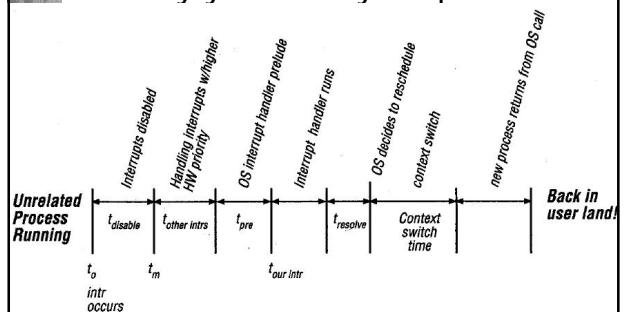
- Ohjelmoijan kontrollointi
  - mitä osia lukitaan muistiin tai KJ ei lainkaan sivuta
  - vuorottamisalgoritmin valinta
  - prioriteetti, aikaviipale, aikarajat
  - asynkroninen siirräntä
- Luotettavuus (reliability)
  - suorituskyvyn muutosvaikutus
  - Hard RT vs. Soft RT
  - virheistä toipuminen (fail-soft operation)
    - nilkuttaa saa, muttei kaatua
- Deterministisyys
  - käsittely kiinteästi ennaltamääritellyinä aikoina tai kiintein aikavalein
  - pakottaa KJ:n reagoimaan keskeytykseen
    - ylaraja olemassa kuitaamiselle (muutama ms)
- Reagointinopeus (responsiveness)
  - kauanko menee keskeytyksen käsittelemiseen?
    - keskeytyksen huomioiminen
    - palvelun suorittaminen
- a Vasteaika (response time R, turnaround time TAT)

16

## REAALIAIKAJÄRJESTELMÄT Yleiskuva

14

## Keskeytyksenkäsittelyn viipeet



lähde?

17

## Reaaliaikajärjestelmät

- Oikeellisuus ei riipu pelkästään tuloksista, vaan myös valmistusajasta
- Tarve reagoida ulkopuolisiin tapahtumiin
  - ohjauksjärjestelmät: laboratoriokeet, 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

## Peruspiirteitä 1/2

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

18

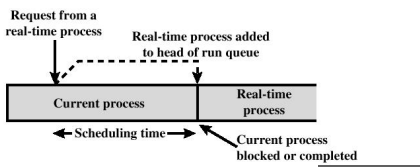
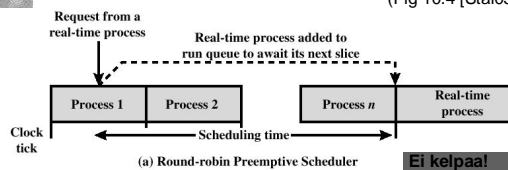
## Peruspiirteitä 2/2

- 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

19

## RT: Vuorottaminen

(Fig 10.4 [Stal05])



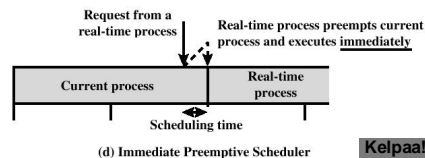
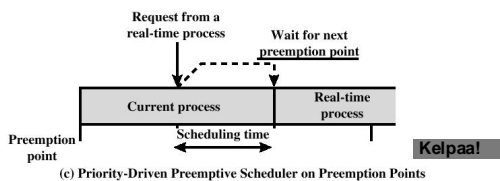
22

## Vuorottaminen reaaliaikajärjestelmissä

20

## RT: Vuorottaminen

(Fig 10.4 [Stal05])



23

## 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-toille
  - ennalta laadittu aikataulu
  - muista ei väliä

21

## Staattiset lähestymistavat

- Voiko kuormaa analysoida?
  - etukäteen tai ajonaikana; aikataulu
- Staattinen taulukkoperustainen Tehdään etukäteen!
  - sopii jaksollisille (periodic) toille
  - alkuaika, kesto, päättymisaika, prioriteetti
  - Useita vaihtoehtoja, vaikka ihan oma
  - Yksi mahdollinen EDF, Earliest Deadline First -algoritmi
- Staattinen prioriteetteihin perustuva Analyysi etukäteen!
  - normaali keskeyttävä (preemptive)
  - prioriteetti määräytyy annettujen aikarajojen nojalla
  - Usein RMS, Rate Monotonic Scheduling -algoritmi

24

## Dynaamiset lähestymistavat

- Dynaaminen aikataulutaminen (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

25

## Rate Monotonic Scheduling

task P execution time C  
task P period T

- Vain jaksollisille (periodisille) töille (Fig 10.7 [Stal05])
- 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

28

## Earliest Deadline First

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

26

## 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.

29

Kaksi jaksollista (periodic) työtä:  
A: saapuu joka 20 ms, suoritus 10 ms,  
B: saapuu joka 30 ms, suoritus 25 ms,  
Vuorottaminen: 10 ms:n välein

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

## Ajoitettavuuden arviointi käyttöasteen avulla

Tb1 10.4 [Stal05]

- RMS: Selvästi

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

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

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

- 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)

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

## 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

31

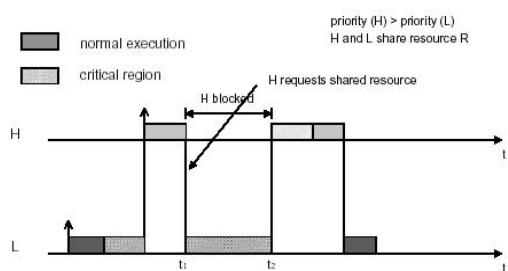
## Mars Pathfinder



- 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](http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html)

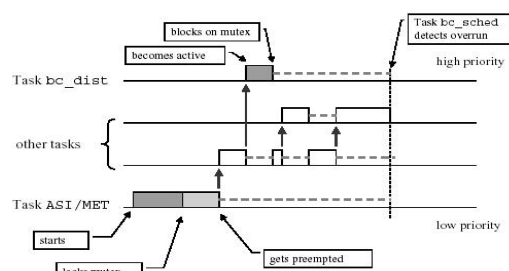
34

## Odotusongelma (Blocking problem)



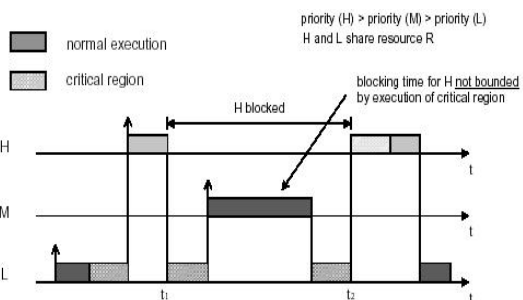
32

## Prioriteetin kääntyminen Mars Pathfinder



35

## Prioriteetin kääntyminen (Priority inversion)



33

## Prioriteetin kääntymisen välttäminen

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

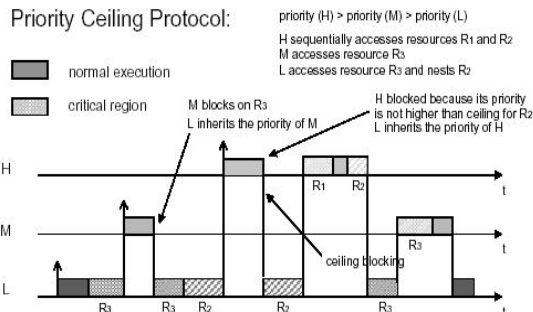
36

## Prioriteetin perintä

- Idea: Jos tapahtuma T estää (blocks) yhden tai useamman korkeampi prioriteetisen 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)

37

## Prioriteettikatto

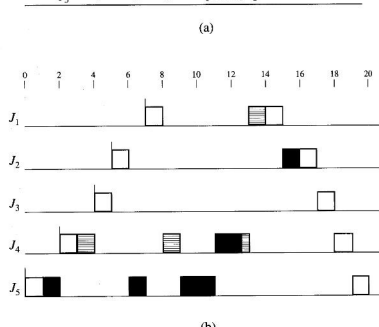


## Prioriteetin perintä

Job	$r_i$	$e_i$	$p_i$	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]

Ajanhetkellä

- 6:  $J_5$  perii  $J_2$ :n prioriteetin, koska  $J_2$  joutuu odottamaan mustaa resurssia.
- 8:  $J_4$  perii  $J_1$ :n prioriteetin, koska  $J_1$  joutuu odottamaan harmaata resurssia.
- 9:  $J_5$  perii  $J_1$ :n prioriteetin  $J_4$ :lta, koska  $J_4$  haluaa myös mustan resurssin.



Liu Kuva 8-8

## Linux VUOROTTAMINEN



41

## 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  $\Pi(\cdot)$ .
  - Jos tapahtuma T estää yhden tai useamman korkeampi prioriteetisen tapahtuman, se väliaikaisesti perii korkeimman estetyn tapahtuman prioriteetin.
- Ongelma:
  - Tiedettävä resurssien käyttö etukäteen!

39

## Linux: Vuorottaminen

POSIX




- Poikkeaa perinteisestä Unixista
  - root: `sched_setscheduler()`
- Soft Real-time työt:
  - SCHED\_FIFO**
    - kiinteät prioriteetit, tapahtumaohjattu (ei aikaviipaleita)
    - 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

42

## Linux: työn prioriteetti

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



43

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 säättö > -1 (aika usein)
  - nice* = -20 → *expired* tilaan, jos säättö > +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

46

kernel 2.6 [DDC04]

## Linux: prioriteetin vaihtelu

- 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 *ilo-sidonnaisia prosesseja*
- Prosessi käyttää aikaviipaleensa loppuun
  - epri*++
  - silti aina *epri* ≤ *nice* +5 ja *epri* ≤ 19
  - rankaisee cpu-sidonnaisia prosesseja

44

## Linux aikaviipale

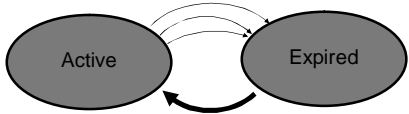
- Ongelma
  - prosessi luo paljon kopiota (clone), ja kahmii kaiken CPU-ajan, koska niitä on monta ja jokaisella on täysi iso *quantum*
- Ratkaisu
  - kloonatun prosessin ensimmäinen *quantum* otetaan isäprosessin *quantumista* (50%)
  - uuden *quantumin* laskennassa (seuraava *epoch*) molemmat saavat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

47

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



45

kernel 2.4? [Tane01]

## Linux: Vuorottaminen

- Tarkista jokaisella *schedule()*:n heräämisellä (1 ms välein?), onko syytä vaihtaa suorituksessa olevaa prosessia (preemption points)
  - perustuu sekä *quantumiin* 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
```

(/usr/src/linux/kernel/sched.c) 48

## 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 puolituntunut
  - jos joku muu suoritin on *idle*, niin siirrä ruuhkaisimmalta suorittimelta toita tälle suorittimelle, kunnes jonottavien töiden lkm-erotus on puolituntunut
    - 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 kautta aikaa odottanut valitaan ensin
  - sen tietoja luultavasti ei ole enää valimuisteissa

49

## W2K prioriteettiluokat Fig 10.14 [Stal 05]

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

Figure 10.14 Windows Thread Dispatching Priorities

## Windows 2000 VUOROTTAMINEN

Ch 10.5 [Stal 05]  
Ch 11.4 [Tane01]

50

## 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 kuvataan 32 todelliseen luokkaan
  - base priority* Fig 11-18 [Tane01]
  - luokat 17-21 KJ:lle? Fig 11-19 [Tane01]
  - luokka 0 on Zero Page -säikeelle
  - luokka -1 todelliselle idle säikeelle
    - idle looppi skedulerin sisällä?

53

## W2K vuoronanto

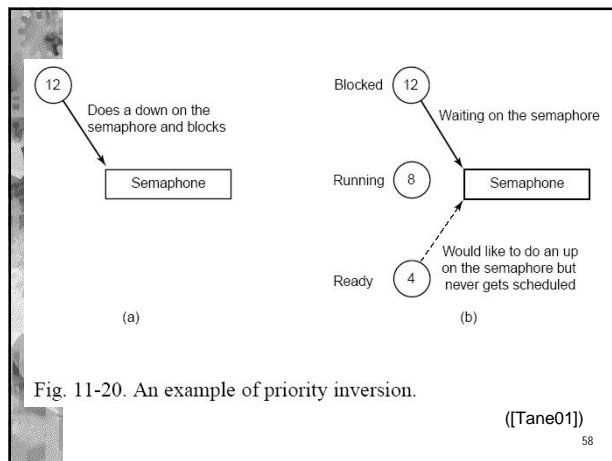
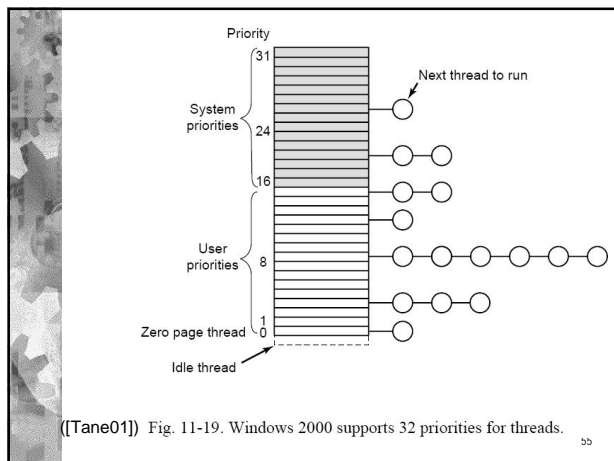
- säie itse suorittaa schedulerin, kun
  - blokkaa resurssiin
  - signaloi toiselle (V-operaatio)
  - aikaviipale päättyy
- scheduler käynnistyy DPC:nä (delayed proc call)
  - I/O-keskeytyksäsittelyn jälkeen
  - semaforin tms timeout'in jälkeen
  - alkuper. kesk. käsittelyn aika on minimoitu
  - DCP:llä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

51

		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])

54



### 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)
- synkronointidotuksen (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 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 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?)

### W2K: SMP tuki

- Suurimpien prioriteettien säikeille kullekin luokalle omat CPU:t (*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?