LUENTO 6

Säietoteutuksia: Windows, UNIX

Samanaikaisuuden hallinta: UNIX, Solaris ja W2K

Ch 4.5 – 4.9 [Stal 05]
Ch 6.7 – 6.13 [Stal 05]
(Ch 11.4 [Tane 01])

1

Windows 2000 (eli W2K)

Säikeet ja SMP
Ch 4.4 [Stal 05]

2

W2K resurssien ja suorituksen hallinta

- Työ (job)
  - prosessien kokoelma, jolla yhteiset rajoitukset
  - max prosessien lkm, max CPU aika per prosessi, ...
- Prosessi (process)
  - omistaa resurssit
  - koostuu useasta säikeestä
- Säie (thread)
  - oma access token (saatu esim. asiakkaalta)
  - suorittava vuorotuksen yksikkö
  - kaksi pinoa, oma user ja kernel tiloissa suoritusta varten
- Kuitu (fiber) – "kevytsäie" (nopea vuorotus)
  - ydin ei tiedä kuiduista mitään Win32 API hoitaa kaiken
    - hyvin nopea vuorotus user tilassa (Win32 API hoitaa)
    - jos kuitu blokkaa, niin toinen saman säikeen kuitu vuorotetaan automaattisesti (säie ei siis blokkaannu)

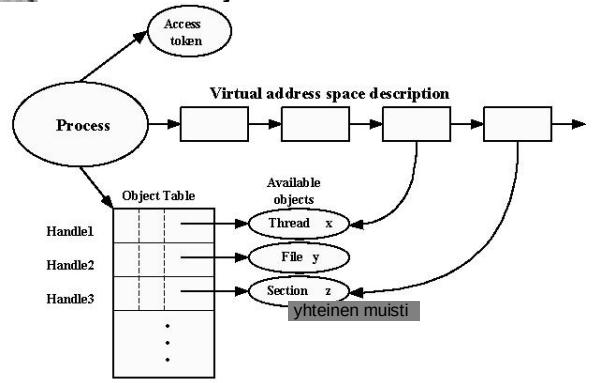
3

W2K: Säikeet

- W2K tukee useita erilaisia ajoympäristöjä (Win16, Win32, Posix?, OS/2?), eroja:
  - prosessien nimeäminen
  - säikeitä vai ei
  - prosessin kuvaaminen
  - prosessin resurssien suojaus
  - prosessien välinen kommunikointi
  - prosessien sukulaisuussuhteet
- Perusta
  - prosessit ja säikeet toteutettu olioina
  - prosessi = yksi tai useampi säie
- Sekä prosessi- että säieoliolla synkronointiprimitiivejä

4

W2K: Prosessi ja sen resurssit



(Fig 4.12 [Stal05])

5

W2K: Prosessi ja sen resurssit

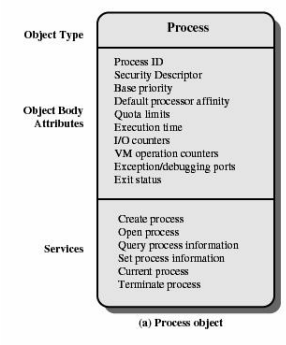
- Oliotaulukko (object table)
  - pääsy olioiden tietorakenteisiin
  - prosessiin liittyvät säikeet — access token
- Pääsypoletti (pääsylippu?) istuntoa käynnistettäessä
  - määrää oletusoikeudet luotaville olioille
    - esim. säikeille
    - myöhemmin säie voi saada esim. asiakkaan pääsypoletin, jolloin lisää oikeuksia
  - rajoittaa / sallii olioiden käyttöä
    - kuka saa käyttää, kuka ei
    - käyttötapa: read, write, change, …
- Osa tietorakenteista vain KJ:n hallussa
  - sovelluksella ei suoraa kahvaa (esim. prosessin oma pääsypoletti)

6

W2K Prosessiolio

- prosessiolio
  - attribuutit: I/O counters, quota limits, debug ports, …
  - palvelut
    - create process
    - terminate process

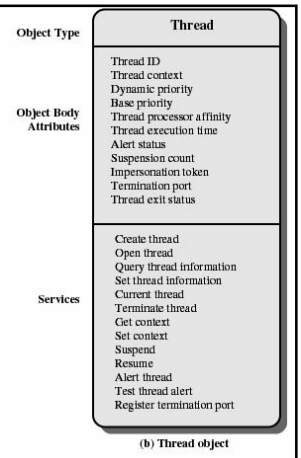
Fig 4.13 [Stal05]



W2K Säieolio

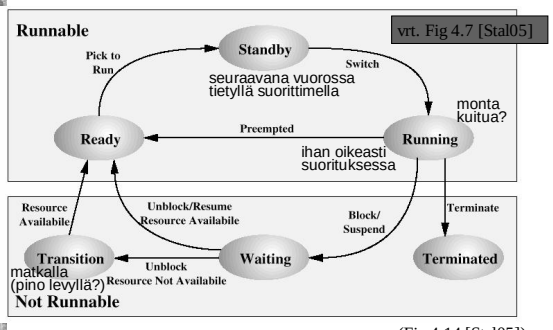
- säieolio
  - attribuutit: processor affinity, base priority, …
  - palvelut
    - create thread
    - suspend, resume
    - get context

Fig 4.13 [Stal05]



W2K Säietilat

vrt. Fig 4.7 [Stal05]



(Fig 4.14 [Stal05])

W2K: SMP

- Prosessin säikeet voivat olla yhtä aikaa eri prosessoreilla
  - myös W2K Executive
- Prosessi/säie voidaan sitoa tietylle prosessorijoukolle (affinity)
  - tieto kirjattu sekä PCB:hen että TCB:hen
  - vuorotetaan aina samoille prosessoreille
  - thread affinity $\subseteq$ process affinity
  - pehmeä sidonta (soft affinity) – yritä edes (välimuistin vuoksi)
  - kova sidonta (hard affinity) – muu ei kelpaa
- Pyritään vuorottamaan uudelleen samalle CPU:lle
  - välimuisti

Affinity: *kinship by marriage or adoption, the force attracting atoms to each other*

[hyperdictionary]

Linux säikeet ja SMP

Ch 4.6 [Stal05]

Linux säikeet

- Ei erota käsitteellisesti prosessia ja säiettä
  - toiminnallisesti ytimen säikeet (KLT, kernel level threads)
  - käyttää yleistermiä task (tehtävä, työ, askare, puuha...)
  - prosessille ja säikeelle ei erillistä kuvaajaa
    - säikeet ovat prosesseja, joilla on sama osoiteavaruus ja thread-group?
    - task_struct
    - linkittää task-kuvaajat myös sukulaissuhteiden mukaan

LINUX säikeet

- POSIX-säiekirjasto pthreads käyttäjätasolle
  - `man -k pthread`
- Task luodaan systeemikutsulla clone()
  - lapselle kopio äidin task_struct:stä
    - yhteiskäytössä olevat (sama groupid) resurssit, avoimet tiedostot sekä virtuaalimuisti (kuten säikeillä kuuluu ollakin)
- fork() perustuu clone()-kutsun käyttöön

13

Linux: Mitä yhteiskäytössä?

/ * cloning flags: include/linux/sched.h */

#define CSIGNAL	0x000000ff	signal mask to be sent at exit
#define CLONE_VM	0x00000100	VM shared between processes
#define CLONE_FS	0x00000200	fs info shared between processes
#define CLONE_FILES	0x00000400	open files shared between processes
#define CLONE_SIGHAND	0x00000800	signal handlers and blocked signals shared
#define CLONE_PID	0x00001000	pid shared
#define CLONE_PTRACE	0x00002000	we want to let tracing continue on the child too
#define CLONE_VFORK	0x00004000	parent wants child to wake it up on mm_release
#define CLONE_PARENT	0x00008000	we want to have the same parent as the cloner
#define CLONE_THREAD	0x00010000	Same thread group?
#define CLONE_SIGNAL	(CLONE_SIGHAND | CLONE_THREAD)	

- Yhteiskäyttöisyys tavallaan määrää luotiinko 'prosessi' vai 'säie'!
  - otettava huomioon prosessin vaihdossa?
  - kuinka paljon ympäristöä pitää vaihtaa?

14

Linux: task_struct

- `include/linux/sched.h` (http://lxr.linux.no/source)
- Yhteiskäyttöinen tieto erillään varsinaisesta task_structista
  - viitteet useasta task_structista
  - varattujen muistialueiden kuvaus (koodi, data)
  - tiedostokuvaajat (= avoimet tiedostot)
  - muu tiedostoihin liittyvä data (pwd, umask, root)
  - signaalikäsittelijätaulu
- Yksittäiskäyttöisestä datasta omat kopiot
  - oma pinoalue
  - tallealue rekistereille
  - tunnistetiedot, linkkikentät

15

Linux: prosessin tilat

- `TASK_RUNNING`
  - CPU:lla tai odottamassa CPU:ta

Fig 4.18 [Stal05]

- `TASK_INTERRUPTIBLE`
  - odotus, joka voidaan katkaista signaalilla (esim. ajastus)
- `TASK_UNINTERRUPTIBLE`
  - odottaa laitetapahtumaa
  - homma täytyy saada loppuun joskus!
  - ei ota vastaan signaaleja (esim. "kill -9 1234")
- `TASK_STOPPED`
  - pysäytetty, vaatii käyttäjän jatkotoimia, esim. taustaprosessi haluaisi tulostaa näytölle, lukea näppäimistöltä
  - saanut signaalin SIGSTOP, SIGTTIN tai SIGTTOU
  - odottaa nyt signaalia SIGCONT
- `TASK_ZOMBIE`
  - homma valmis, odottaa saattohoitoa

16

Linux: SMP

- Normaalit SMP:n käyttötavat
- Systeemikutsu sysmp()
  - tehtävät (task) voi naittaa jollekin/joillekin prosessorille
  - PSET, processor sets
  - voi määritellä, ettei joku CPU saa suorittaa muita tehtäviä (task) kuin sille määriteltyjä
  - jokin CPU:n voidaan kytkeä pois käytöstä
  - joku CPU:n voidaan määrätä suorittamaan vain yhtä prosessia ja sen säikeitä

('CPU suvun omaisuutena')

17

Samanaikaisuuden hallinta

Ch 6.7-10 [Stal05]

18

UNIX samanaikaisuuden hallinta

- IPC mekanismit
  - kommunikointi
    - putket
    - viestit
    - yhteinen muisti
  - synkronointi
    - semaforit
    - signaalit

19

IPC eri Unix'eissa

IPC Type	POSIX.1	SVR4	4.4BSD
Signals	•	•	•
Pipes (HDX)	•	•	•
FIFOs (named pipes)	•	•	•
Stream pipes (FDX)		•	•
Named stream pipes		•	•
Message queues		•	
Semaphores		•	
Shared memory		•	
Sockets		•	•
Streams		•	

20

UNIX Putki (pipe)

ls sourcedir | grep fileA

- Vuorottaisrutiinien laajennus
  - yksi lukija/yksi kirjoittaja, yhteinen puskuri
  - vuorottelu automaattisesti
  - mutex automaattisesti
  - kiinteä puskurin koko
  - voi olla monta prosessia pareittain
  - end-of-file välitetään sulkemalla putken pää
- esim: parent-child pipe
  - luodaan *int p[2]* taulukkoon putki kutsulla *pipe(p);*
    - kaksi "tiedostoa": *p[0] ja p[1]*
  - *fork()*
  - isäprosessi sulkee toisen pään (esim *p[1]*) ja avaa toisen pään (*p[0]*) kirjoittamista varten
  - lapsiprosessi tekee päinvastoin

esim. seuraavalla sivulla

21

Parent-child pipe

```
#define MSGSIZE 32
char *msg1 = "hi there";
char *msg2 = "terve siellä";
main() {
        char inbuf[MSGSIZE];
        int p[2], j;
        pid_t pid; /* open pipe, both ends now open */
        if( pipe(p) == -1)   { perror("pipe call error"); exit(1); }

        switch(pid = fork()) {
            case 0: /* if child then write down pipe */
                    close(p[0]); /* close the read end of the pipe */
                    write(p[1], msg1, MSGSIZE);
                    write(p[1], msg2, MSGSIZE);
                    break;

            default: /* parent reads pipe */
                    close(p[1]); /* close the write end of the pipe */
                    for(j=0; j<2; j++) {
                            read(p[0], inbuf, MSGSIZE);
                            printf("Parent: %s\n", inbuf);
                            }
                    wait(NULL);
            }
        exit(0);
}
```

UNIX nimetty putki (named pipe)

- Yhdensuuntainen kommunikointi nimetyn tiedoston kautta (FIFO file)
- Ei tarvitse tietää toisen prosessin nimeä!
  - joku luo tiedoston — mkfifo(tdsto)
  - joku prosessi avaa sen lukemiseen — open(tdsto)
  - joku (tai jotkut) avaavat sen kirjoittamiseen
  - lukija/kirjoittajat blokkaantuu kunnes molemmat paikalla — read(tdsto) write(tdsto)
  - käyttö jatkossa kuten tavallisella putkella
  - toimii vain omassa koneessa, ei verkon yli

23

UNIX viestit (messages)

- Joka prosessilla oma viestijono: *msqid* (message queue, mailbox)
- Viesteillä tyyppi (esim. *long int*)
- Voi lukea seuraavan tai seuraavan tietyn tyyppisen viestin
- Myös lähettävä prosessi voi blokkaantua

```
int msgsnd( int msqid, const void *msgp,
            size_t msgsz, int msgflg);

int msgrcv( int msqid, void *msgp,
            size_t msgsz, long msgtyp, int msgflg);
```

24

UNIX yhteinen muisti (shared memory)

- Yhteinen muisti helposti käytössä
  - nopea
- Synkronointiprimitiivit käytettävissä
  - mutex - spin vai wait?
  - semaforit monimutkaisempaan synkronointiin
  - muistetaanko käyttää?
  - luotetaan siihen, että muistetaan
    - käytetäänhän oikein, ettei lukkiudu?

25

UNIX semaforit

- Kuten P/V, mutta monisärmäisempi ratkaisu
- Rakenne
  - arvo
  - pid viime prosessille
  - odottajien lkm raja-arvolle (> arvo)
  - odottajien lkm arvolle nolla (0)
  - odottavien prosessien jono
- Semaforijoukot (sets)
  - operaatio voi kohdistua kaikkiin joukon semaforeihin
- Monimutkaiset operaatiot, joilla toivon mukaan voi toteuttaa monimutkaisetkin synkronoinnit

26

UNIX Semafori-operaatiot

semctl(semid, {sem_num, sem_op, sem_flg}$^{+}$, nsops)

- sem_op > 0 (ALTER)
  - lisää sem_op arvoon
  - vapauta arvon kasvamista odottajat
- sem_op == 0 (READ)
  - jos arvo oli nolla, niin päästä tämä läpi
  - muuten odota, kunnes arvo on nolla
- sem_op < 0 ja |sem_op| ≤ arvo (ALTER)
  - vähennä |sem_op| arvosta
  - jos arvo tuli nollaksi, herätä kaikki sitä odottaneet
- sem_op < 0 ja |sem_op| > arvo (ALTER)
  - odota kunnes arvo kasvaa

27

UNIX signaalit

Tbl 6.2 [Stal 05]

- ”Ohjelmiston antama keskeytys”
  - laitekeskeytyksen käsittelijä voi generoida
- Ei prioriteetteja
- Kaikki käsitellään, jossain järjestyksessä
- PCB:ssä vain yksi bitti per signaalityyppi
  - useasta saman tyyppisestä signaalista jää vain yksi jälki

Fig 4.18 [Stal 05]

- Eivät herätä *uninterruptible* tilasta
  - tärkeä homma kesken, pitää lopettaa kunnolla
  - ei saa tappaa tai häiritä signaaleilla!

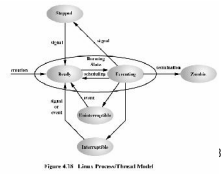


Figure 4.16 Linux Process/Thread Model

28

Table 6.2 UNIX Signals

Value	Name	Description
01	SIGHUP	Hang up; sent to process when kernel assumes that the user of that process is doing no useful work
02	SIGINT	Interrupt
03	SIGQUIT	Quit; sent by user to induce halting of process and production of core dump
04	SIGILL	Illegal instruction
05	SIGTRAP	Trace trap; triggers the execution of code for process tracing
06	SIGIOT	IOT instruction
07	SIGEMT	EMT instruction
08	SIGFPE	Floating-point exception
09	SIGKILL	Kill; terminate process
10	SIGBUS	Bus error
11	SIGSEGV	Segmentation violation; process attempts to access location outside its virtual address space
12	SIGSYS	Bad argument to system call
13	SIGPIPE	Write on a pipe that has no readers attached to it
14	SIGALRM	Alarm clock; issued when a process wishes to receive a signal after a period of time
15	SIGTERM	Software termination
16	SIGUSR1	User-defined signal 1
17	SIGUSR2	User-defined signal 2
18	SIGCHLD	Death of a child
19	SIGPWR	Power failure

29

Linux ja samanaikaisuuden hallinta

Ch 6.8 [Stal 05]

30

Linuxin samanaikaisuuden hallinta

- Kuten muissa Unix'eissa
  - putket, viestit, yhteinen muisti, signaalit, semaforit
- Etuoikeutetussa tilassa myös muita
  - atomiset *int*- ja *bitmap*-operaatiot
  - *spinlock* kriittisten vaiheiden suojaamiseen
  - *spinlock* variantit keskeytyskäsittelijöiden kriittisten vaiheiden suojaamiseen
  - *reader-writer spinlock* reader-writer -ongelmien ratkaisuun
  - *kernel semaphore* ytimen tason semaforiratkaisuihin
  - *barrier* synkronointiprimitiivi käskytason suorituksen kontrollointiin

31

Linux ytimen atomiset operaatiot

Katso: Tbl 6.3 [Stal 05]

- *Atomic int* operaatiot
  - vain erityiselle atomiselle kokonaislukuarvoiselle tietotyypille
  - tietotyyppiä ei voi manipuloida muilla tavoin
    - esim. kääntäjän optimoiman aliaksen kautta
  - atomic int operaatiot eivät voi manipuloida muuta dataa
  - hitaampaa kuin vastaavat normaali operaatiot
- *Atomic bitmap* operaatiot
  - manipuloi bittiä osoittimen osoittamassa bittikartassa

32

Linux ytimen *perus-spinlock*

Katso: Tbl 6.4 [Stal 05]

- Neljä varianttia
  - *plain* on tavallinen spinlock
    - tavalliset poissulkemisongelmat
    - busy-wait kunnes resurssi vapaa

```
spin_lock (&lock);
/* kriittinen vaihe */
spin_unlock (&lock);
```

  - *_irq* keskeytyskäsittelijöille
    - estää keskeytykset odotuksen aikana (kesk. sallittu siis alkuaan)
    - sallii keskeytykset odotuksen jälkeen
  - *_irqsave* keskeytyskäsittelijöille
    - kuten _irc, mutta keskeytystilan talletus/palautus
  - *_bh* keskeytyskäsitteljöiden top half/bottom half -toteutukseen
    - bottom half'ia ei saa suorittaa, kun top half on suorituksessa ja päinvastoin?

33

Linux ytimen *spinlock vs. SMP*

- Spinlock'in toteutus erilainen yhden suorittimen koneessa ja SMP'ssä
- Yksi suoritin
  - jos ytimen koodi on ei-keskeytyvä (non-preemptable) lukkoja ei tarvita ja kääntäjä jättää ne generoimatta
  - jos ytimen koodi keskeytyvä (preemptable), lukot voi käännösvaiheessa toteuttaa keskeytysten estolla
- Monta suoritinta (SMP)
  - spinlock'eista generoidaan busy-wait koodi

34

Linux ytimen *Reader-Writer spinlock*

- Multiple readers- single writer –ongelmaan
  - lukkomuuttujalla 2 kenttää
    - flag = 1, joss lukko vapaa
    - flag = 0, jos lukko varattu
      - counter = 0, jos lukko varattu kirjoittajalle
      - counter = n, jos lukko varattu n:lle lukijalle
  - kirjoittaja saa lukon varattua vain, jos flag = 1
    - odottava kirjoittaja ei estä uusia lukijoita
    - kirjoittaja voi odottaa kauan! epäreilua?
  - *plain*, *_irq* ja *_irqsave* –versiot
- Miksi ei semaforeilla?
  - tehokkuuden vuoksi busy-wait odotus

35

Linux ytimen *semafori*

Katso: Tbl 6.5 [Stal 05]

- Tehokkaampia kuin käyttäjätason semaforit
- Binary semaphore ja Counting semaphore
  - *down()* on tavallinen semaforin wait-operaatio
  - *down_interruptiple()* sallii semaforia odottavan prosessin herättämisen myös ytimen signaalin vuoksi
  - *down-trylock()* on ei blockaava wait-operaatio
- Reader-Writer semaphore
  - multiple readers-single writer -ongelmaan
  - vain perusversio, prosessi odottaa kun saa luvan jatkaa

36

Linux ytimen *barrier*

Katso Tbl 6.6 [Stal 05]

- Käskytason suorituksen kontrollointiin
  - Suojaa koodia kääntäjän tai suorittimen tekemistä käskyjen suoritusjärjestyksen vaihtamisista aiheuttaneilta ongelmilta
    - tärkeä esim. spinlock-operaatioiden toteutuksessa
    - esim. I/O-kontrollialueita ei ladata välimuistiin, joten välimuisti ei huolehdi yhteneväisyysongelman ratkaisusta
  - Hidastavat suoritusta!
  - Varmista, että load/store operaatiot tietyssä järjestyksessä
    - järjestysvaatimus ei ilmene koodinpätkästä sellaisenaan, vaan isommasta kokonaisuudesta
  - Esim. varmista, että muutettu arvo näkyy myös muilla suorittimilla
  - SMP-operaatioiden toteutus erilainen yhden suorittimen koneissa ja SMP'ssä
    - vain SMP'ssä tarvitaan suoritusaikaista suojaa

```
a = 1;
wmb();
b = 1;
```

37

Solaris

38

Sun Solaris, säikeiden hallinta

- Omat primitiivit ytimessä ytimen säikeille
- Käytettävissä myös user-tason säikeille kirjaston avulla
  - mutex poissulkemisongelmaan
  - semaforit yleiseen synkronointiin
  - omat lukot multiple readers/single writer –ongelmaan
  - ehtomuuttujat (condition variables) barrier-tyyppiseen synkronointiin
    - eri "barrier" kuin Linux ytimen *käskytason* "barrier" käsite

39

Solaris säikeiden mutex

- Odotus spinnaamalla busy-wait –loopissa oletusarvoisesti
- Odotus blocked (tai sleeping) –tilassa, jos halutaan
- Normaali *mutex_enter()* (eli wait)
  - voi blokata myös muut ryhmään kuuluvat ULT:t
  - voi valita odotustyypin: busy wait tai sleep
- Normaali *mutex_exit()* (eli signal)
- *mutex_tryenter()* yrittää lukkoa (kuten mutex_enter)
  - jos lukko vapaa, niin ota se (kuten mutex_enter)
  - jos lukko varattu, niin palauta tieto siitä
    - säie voi nyt suosiolla antaa vuoron toiselle säikeelle tai ULT:lle (vältä block tai busy wait)
    - yritä uudelleen (kun saat suoritusvuoron takaisin)

40

Solaris säikeiden semaforit

- *sema_p()* ja *sema_v()* normaalit P ja V
- myös *sema_tryp()*
  - toimii kuten P, mutta palauttaa vain tiedon epäonnistumisesta, jos olisi blokkaantunut
  - kuten *mutex_tryenter()*

41

Solaris säikeiden Readers/Writer -lukko

- selkeä ratkaisu yleiseen ongelmaan
  - yhteinen data, yhteinen tiedosto
- *rw_enter(lock, access_type), rw_exit()*
- *rw_tryenter()*
  - kuten *sema_tryp()* tai *mutex_tryenter()*
  - yritä, mutta älä blokkaa
- *rw_downgrade()*
  - palaa kirjoituksen jälkeen vain lukijaksi
- *rw_tryupgrade()*
  - lukija yrittää päästä kirjoittajaksi
  - yritä, mutta älä blokkaa

42

Solaris säikeiden ehtomuuttujat

- yleisiin barrier-synkronointeihin
  - monta säiettä odottaa
  - yksi tai kaikki säikeet saa jatkaa
- voidaan käyttää minkä tahansa ehdon yhteydessä
  - käyttö pitää suojata dedikoidulla mutexilla
  - säie voisi muuten vaihtua ehdon laskemisen aikana
    - ehto voi olla mikä tahansa lauseke
  - *cv_wait(&cv, &mutex)* vapauttaa mutexin mahdollisen odottamisen ajaksi

monta yrittää, yksi pääsee läpi

```
mutex_enter(&mutex);
        while (”complex condition”)
                cv_wait( &cv, &mutex)
mutex_exit(&mutex)
```

kriitt. vaihe I

kriitt. vaihe II

kriitt. vaihe III

43

W2K samanaikaisuuden hallinta

Ch 6.10 [Stal05]
(+ Sect 11.4 [Tane91])

Microsoft Windows 2000

44

W2K IPC [Ch 11.4.2 Tane01]

- pipes, named pipes — putki, nimetty putki
  - tavu- ja viestiperustaiset
  - nimetyt putket verkon yli, tavalliset ei
- mailslots — "postilaatikko"
  - W2K:ssa, mutta ei UNIXeissa
  - one-to-one, one-to-many, verkon yli
- sockets — pistoke
  - verkon yli
- rpc
- yhteiskäyttöiset tiedostot
  - muistialue mapataan usealle prosessille
  - säikeet hoitavat mutex-ongelman (toivottavasti)
    - eri prosessien säikeillä voi olla yhteinen mustialue

45

W2K samanaikaisuuden hallinta

- Normaali toiminto, jota voidaan aina odottaa
  - prosessi tai säie päättyy — Tbl 6.7 alaosa [Stal05]
  - tiedosto-operaatio valmis
  - syöte valmis — Tbl 6.7 yläosa [Stal05]
- Tavalliset synkronointiprimitiivit
  - tietty muutos tiedostojärjestelmässä
    `FindFirstChangeNotification( “filter”, “directory”)`
  - mutex
  - semafori
  - tapahtuma (event)
  - odotettava ajastin (waitable timer)

46

Table 6.7 Windows Synchronization Objects

Object Type	Definition	Set to Signaled State When	Effect on Waiting Threads
Event	An announcement that a system event has occurred	Thread sets the event	All released
Mutex	A mechanism that provides mutual exclusion capabilities; equivalent to a binary semaphore	Owning thread or other thread releases the mutex	One thread released
Semaphore	A counter that regulates the number of threads that can use a resource	Semaphore count drops to zero	All released
Waitable timer	A counter that records the passage of time	Set time arrives or time interval expires	All released
File change notification	A notification of any file system changes.	Change occurs in file system that matches filter criteria of this object	One thread released
Console input	A text window screen buffer (e.g., used to handle screen I/O for an MS-DOS application)	Input is available for processing	One thread released
Job	An instance of an opened file or I/O device	I/O operation completes	All released
Memory resource notification	A notification of change to a memory resource	Specified type of change occurs within physical memory	All released
Process	A program invocation, including the address space and resources required to run the program	Last thread terminates	All released
Thread	An executable entity within a process	Thread terminates	All released

Uncolored (ylemmät rivit)
Note: Colored rows correspond to objects that exist for the sole purpose of synchronization.

W2K mutex

Tbl 6.7 [Stal05]

- Keskeytysten esto, spin-lock SMP:lle
- Binääriarvoinen semafori, odotus suspend-tilassa
  - mutex *eri prosessien* säikeiden välillä
  - ytimen olio
  - timeout optio

```
WaitForSingleObject(lock);
        “critical section”
ReleaseMutex(lock);
```

- Critical Section *yhden prosessin* säikeiden välillä
  - toteutus user-tasolla (paitsi suspend)
  - nopea

```
LPCriticalSection CritSect=1;
….
EnterCriticalSection(CritSect);
        “critical section”
LeaveCriticalSection(CritSect);
```

48

W2K semafori

Tbl 6.7 [Stal05]

- Tavallinen semafori (counting semaphore)
  - arvo nolla tai isompi
  - ytimen olio, ytimen olioiden suojaukset

```
sem=CreateSemaphore( init_val, max_val )
```

```
WaitForSingleObject( sem )
```

```
status = WaitForSingleObject( sem, ”timeout” )
```

```
ReleaseSemaphore( sem )
```

49

W2K tapahtuma (event)

Tbl 6.7 [Stal05]

- Manual-reset event (& auto-reset event)
  - tilat: *set, cleared*
  - ResetEvent (ev)
    - aseta tilaksi *cleared*
  - WaitForSingleObject (ev)
    - suspend, jos tila *cleared*
  - SetEvent (ev)
    Aseta tila *set*. Jos odottajia,
    - vapauta kaikki, tilaksi jää *set*
    - vapauta yksi ja aseta tilaksi *cleared* (auto-reset event)
  - PulseEvent (ev)
    - kuten SetEvent, mutta ei muistia
    - jos ei odottajia, niin tilaksi jää *cleared* (eli ”unohda”)
    - jos odottajia, niin vapauttaa yhden, tilaksi jää *cleared*

50

W2K odotettava ajastin (waitable timer)

- Ytimen olio
- Herättää tiettyyn aikaan tai tietyn ajan päästä
- Herättää aina tietyn aikaintervallin välein

51