LUENTO 4

Käyttöjärjestelmät II

Solaris ja UNIX Säikeet
Ch 4.5 - 6 [Stal 05]

UNIX, Solaris ja W2K
Samanaikaisuuden hallinta
Ch 6.7 - 10 [Stal 05]
(Ch 11.4 [Tane 01])

Käyttöjärjestelmät II

Solaris, UNIX

Säikeet ja SMP

Ch 4.5 - 6 [Stal05]

Solaris: säikeet

Peruskäsitteet

Fig 4.15 [Stal05]

- **Prosessi**
  - PCB, prosessin osoiteavaruus (koodi, data), pino
- **ULT (User Level Thread) - käyttäjätason säikeet**
  - säiekirjasto (pthreads, Solariksen oma)
  - eivät näy KJ:lle, ohjelmoija hoitaa vuorottamisen
- **LWP (Light Weight Processes) - kevytprosessit**
  - kokoelma ULT säikeitä, jotka näkyy ytimelle yhtenä nippuna (yksi tai useampi LWP säie)
- **KLT: Kernel Level Thread – ytimen säie**
  - vuorottaminen KJ:n heiniä
  - KLT = LWP? (ei ihan, koska myös muita KLT säikeitä)

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 3

Solaris: ULT / LWP / KLT

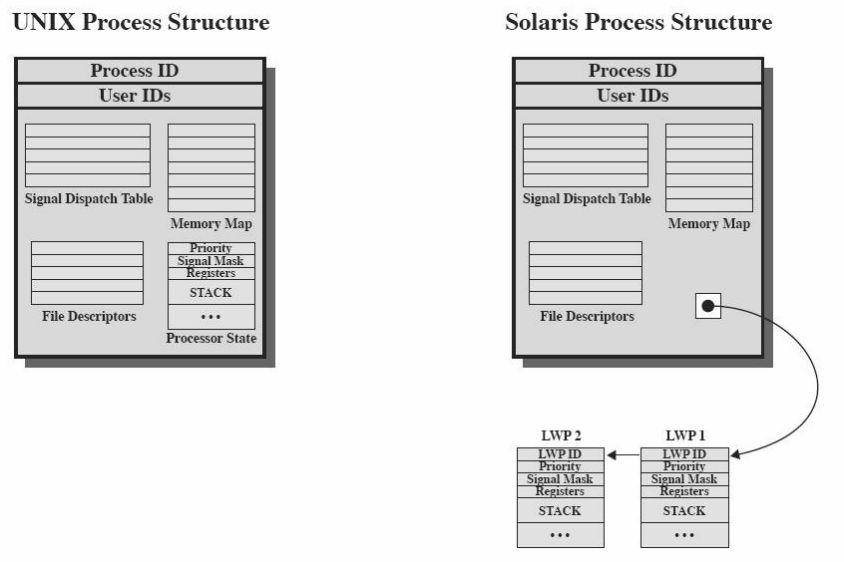
- **Joustavuus**

Fig 4.15 [Stal05]

  - ohjelmoi ‘by-best-possible-taste’
  - laitetason rinnakkaisuus vs. looginen rinnakkaisuus
    - esim. ikkunointi
  - säiekirjaston ei tarvitse aina huolehtia prosessin säikeiden vuorottamisesta
- **Tehokkuus**
  - LWP: ei tarvitse kuormittaa KJ:tä ylim. säikeillä
  - ULT: vuorottaminen nopeaa (kirjasto)
  - KLT: Jos säie blokkaa, muut saman prosessin (tai KJ’n ytimen) KLT-säikeet voivat silti jatkaa

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 4

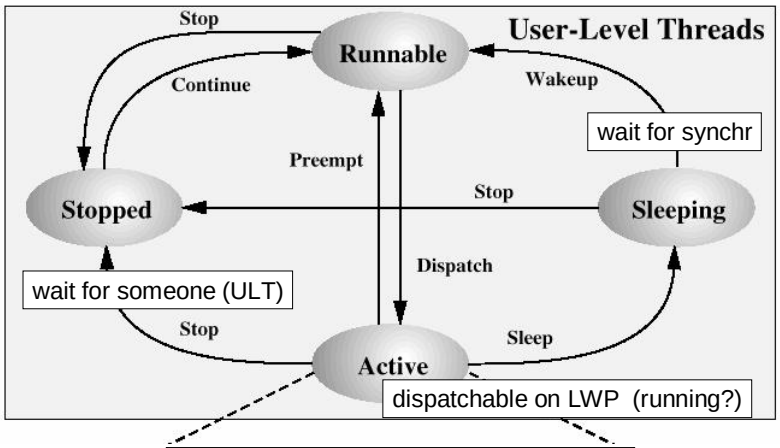
Solaris: prosessi



(Fig 4.16 [Stal05])

Solaris: ULT tilat

Fig 4.15 [Stal05]



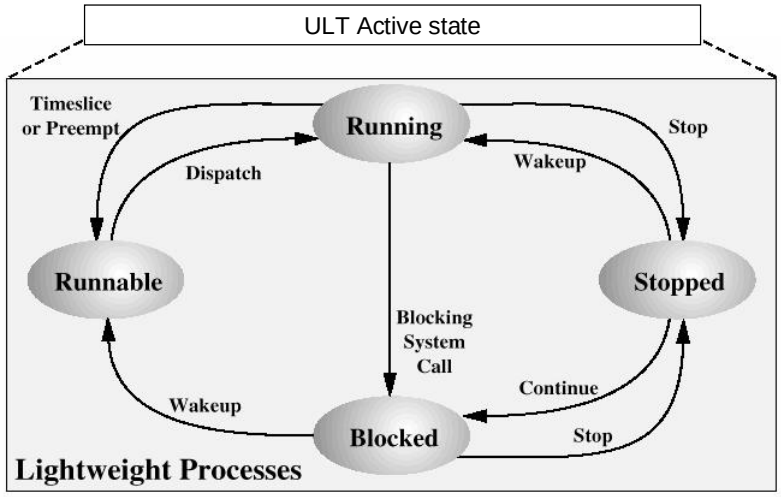
LWP tilat

(Fig 4.17 upper) [Stal05]

- Onko ULT sidottu LWP:hen vai ei? (bound vs. unbound)
  - ULT → sleep => LWP odottaa vai valitsee toisen ULT:n?

Solaris: LWP tilat

Fig 4.15 [Stal05]



(Fig 4.17 lower [Stal05])

Solaris: SMP

- **Normaalit SMP:n käyttötavat**
  - KJ vapaakäyntinen
  - poissulkeminen hoidettu
- **LWP:n voi sitoa (bind) tiettyyn CPU:hun**
  - vrt. W2K “hard affinity”
  - välimuistin hyödyntäminen
    - vanhat tiedot juuri tämän CPU:n välimuistissa
    - vähemmän välimuistihuteja

Solaris: keskeytys

- **Keskeytys voi sattua milloin tahansa**
  - prosessi ei voi varautua
  - yleensä poissulkeminen estämällä keskeytykset käsittelyn ajaksi
- **Käsittely KLT-säikeenä**
  - prioriteetti, oma pino, oma talletusalue rekistereille
  - poissulkeminen kuten muillakin säikeillä
    - lukot, semaforit, ehdolliset muuttujat
    - poissulje vain ne rakenteet, joita käsitellään
    - odota tarvittaessa
- **Keskeytyskäsittelysäikeillä suurin prioriteetti**
  - korkeamman prioriteetin keskeytys voi keskeyttää
- **Keskeytyskäsittelysäikeet olemassa pysyvästi**
  - aikaa ei kulu luontiin / siivoamiseen

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 9

Käyttöjärjestelmät II

Linux säikeet ja SMP

Ch 4.6 [Stal05]

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 10

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
- **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**

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 11

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?

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 12

Linux: task_struct

rivi 528
click!

- **`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

click!
thread_struct

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 13

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

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 14

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')

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 15

POSIX säikeet

```
void *thread_fn(void *arg) {
        int i; for ( i=0; i<30; i++ ) {
                printf("Haloo %d\n", i);
                sleep(1);
                }
        return NULL;
}

int main(void) {
        pthread_t thread_new;
        if ( pthread_create( &thread_new, NULL, thread_fn, NULL) ) {
                printf("error in create thread\n.");
                abort();
                }

        if ( pthread_join ( thread_new, NULL ) ) {
                printf("error join thread.\n");
                abort();
                }

        exit(0);
}
```

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 16

Käyttöjärjestelmät II

Samanaikaisuuden hallinta

Ch 6.7-10 [Stal05]

Lukkiutumisen välttäminen Ch 6.5

- **Monta resurssiluokkaa**
- **Varaukset aina samassa järjestyksessä:**
  - swap-alueen varaukset
    - varaa kaikki kerralla
  - prosessin resurssit
    - varaa kaikki laitteet etukäteen?
  - keskusmuisti
    - siirrä preempted prosessi levylle (muuten resurssia ei vapaudu)
  - muiden resurssien varaus
    - oikea järjestys?
- **Onnistuuko käytännössä?**

UNIX samanaikaisuuden hallinta

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

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 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		•	

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 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

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
  - käyttö jatkossa kuten tavallisella putkella — read(tdsto) write(tdsto)
  - toimii vain omassa koneessa, ei verkon yli

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 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**
  - viestipuskuri *msqid*'lle täynnä

```
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);
```

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 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?

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 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**

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 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

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
- **Eivät herätä *uninterruptible* tilasta** Fig 4.18 [Stal 05]
  - tärkeä homma kesken, pitää lopettaa kunnolla
  - ei saa tappaa tai häiritä signaaleilla!

Linux ja samanaikaisuuden hallinta

Ch 6.8 [Stal 05]

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

Linux ytimen atomiset operaatiot

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

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 31

Linux ytimen *perus-spinlock*

Tbl 6.4 [Stal 05]

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

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

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 32

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

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 33

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

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 34

Linux ytimen *semafori*

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

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 35

Linux ytimen *barrier*

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;
```

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 36

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

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)

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()*

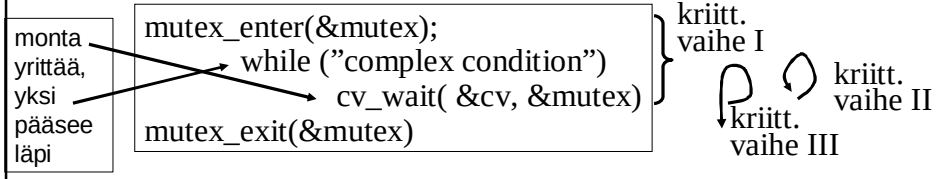
Solaris säikeiden Readers/Writer -lukko

- **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

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 41

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



KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 42

W2K samanaikaisuuden hallinta

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

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**
  - verkon yli — pistoke
- **rpc**
- **yhteiskäyttöiset tiedostot**
  - muistialue mapataan usealle prosessille
  - säikeet hoitavat mutex-ongelman (toivottavasti)
    - eri prosessien säikeillä voi olla yhteinen mustialue

W2K samanaikaisuuden hallinta

- **Normaali toiminto, jota voidaan aina odottaa**
  - prosessi tai säie päättyy
  - tiedosto-operaatio valmis
  - syöte valmis

Tbl 6.7 alaosa [Stal05]

- **Tavalliset synkronointiprimitiivit**
  - tietty muutos tiedostojärjestelmässä

Tbl 6.7 yläosa [Stal05]

FindFirstChangeNotification(“filter”, “directory”)

  - mutex
  - semafori
  - tapahtuma (event)
  - odotettava ajastin (waitable timer)

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 45

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);
```

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 22.3.2006 46

W2K semafori

Tbl 6.7 [Stal05]

- **Tavallinen semafori (counting semaphore)**
  - arvo nolla tai isompi
  - ytimen olio, ytimen olioiden suojaukset
  - timeout optio, jolla vältetään suspend

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

```
WaitForSingleObject( sem )
```

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

```
ReleaseSemaphore( sem )
```


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*

W2K odotettava ajastin (waitable timer)

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

Kertauskysymyksiä

- Miten master/slave ja SMP eroavat toisistaan?
- Miten ne vaikuttavat KJ:n toteutukseen?
- Entä sovellusohjelmoijan työhön?
- Miten välimuistin koherenttius hoituu SMP-ympäristössä?
- Mitkä KJ:n osat sijoitetaan mikroytimeen?
- Mikroytimen hyödyt / haitat?
- Miten käyttäjätason säikeet ja ytimessä toteutetut säikeet eroavat toisistaan?
- Mitä tietoa talletettu PCB:hen / TCB:hen?
- Miten synkronointiongelmat ratkaistaan eri KJ:issä
- Mikä on helppoa kj-X:ssä, mutta vaikeampaa kj-Y:ssä?