

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]

n Prosessi

- u PCB, prosessin osoiteavaruus (koodi, data), pino

n ULT (User Level Thread) - käyttäjätason säikeet

- u säiekirjasto (pthreads, Solariksen oma)
- u eivät näy KJ:lle, ohjelmoija hoitaa vuorottamisen

n LWP (Light Weight Processes) - kevytprosessit

- u kokoelma ULT säikeitä, jotka näkyy ytimelle yhtenä nippuna (yksi tai useampi LWP säie)

n KLT: Kernel Level Thread – ytimen säie

- u vuorottaminen KJ:n heiniä
- u KLT = LWP? (ei ihan, koska myös muita KLT säikeitä)

Solaris: ULT / LWP / KLT

n Joustavuus

- u ohjelmoi 'by-best-possible-taste'
- u laitetason rinnakkaisuus vs. looginen rinnakkaisuus
 - F esim. ikkunointi
- u säiekirjaston ei tarvitse aina huolehtia prosessin säikeiden vuorottamisesta

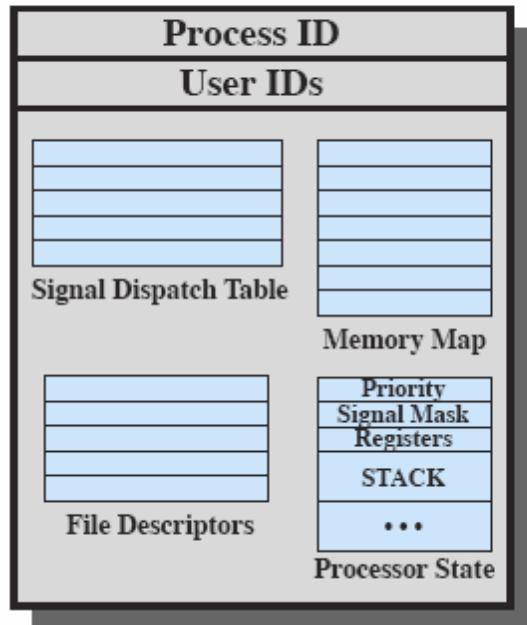
Fig 4.15 [Stal05]

n Tehokkuus

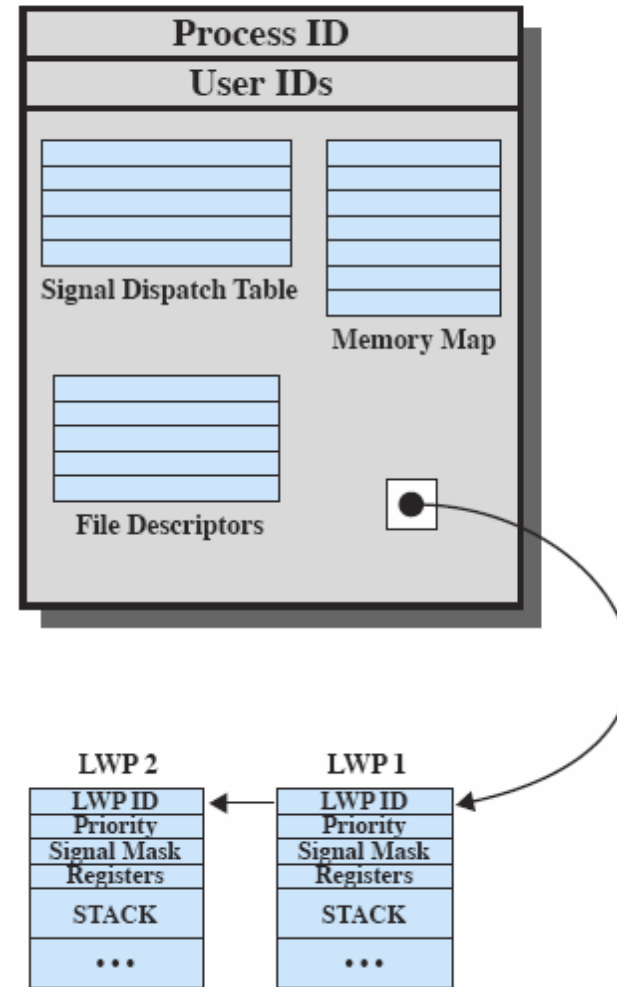
- u LWP: ei tarvitse kuormittaa KJ:tä ylim. säikeillä
- u ULT: vuorottaminen nopeaa (kirjasto)
- u KLT: Jos säie blokkaa, muut saman prosessin (tai KJ'n ytimen) KLT-säikeet voivat silti jatkaa

Solaris: processi

UNIX Process Structure



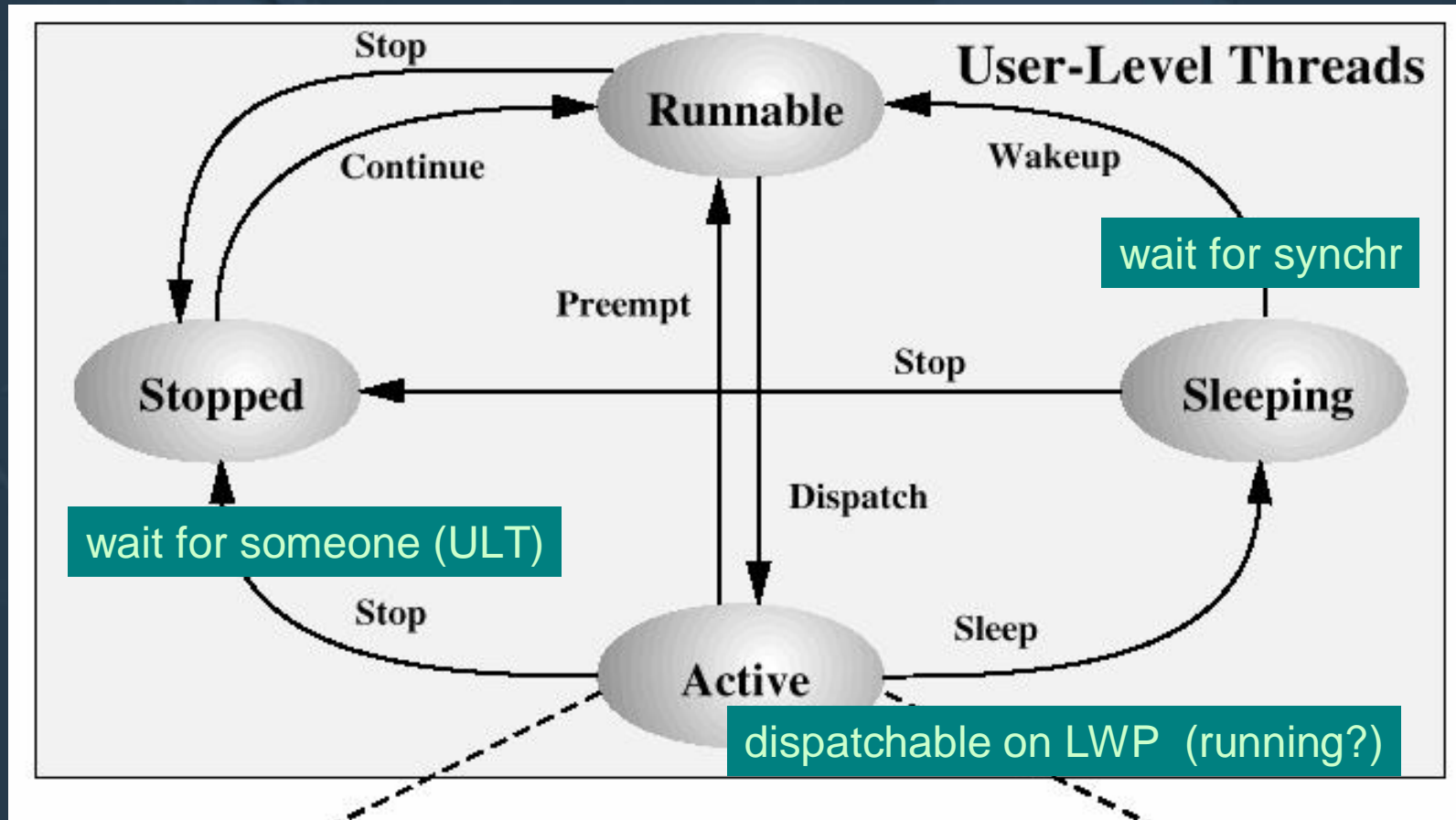
Solaris Process Structure



(Fig 4.16 [Stal05])

Solaris: ULT tilat

Fig 4.15 [Stal05]



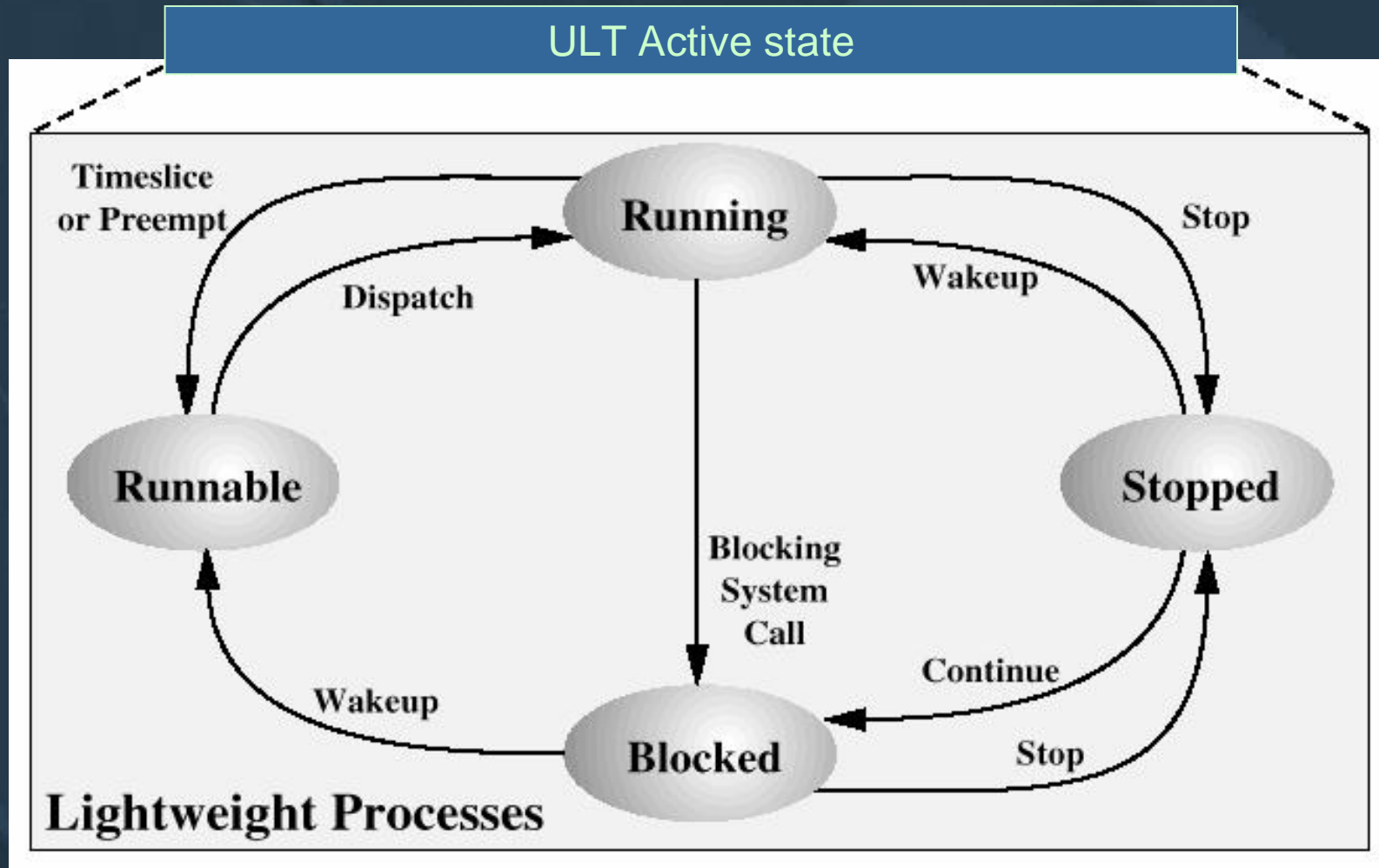
LWP tilat

(Fig 4.17 upper)
[Stal05]

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

Solaris: LWP tilat

Fig 4.15 [Stal05]



(Fig 4.17 lower [Stal05])

Solaris: SMP

n Normaalit SMP:n käyttötavat

- u KJ vapaakäyntinen
- u poissulkeminen hoidettu

n LWP:n voi sitoa (bind) tiettyyn CPU:hun

- u vrt. W2K "hard affinity"
- u välimuistin hyödyntäminen
 - F vanhat tiedot juuri tämän CPU:n välimuistissa
 - F vähemmän välimuistihuteja

Solaris: keskeytys

n Keskeytys voi sattua milloin tahansa

- u prosessi ei voi varautua
- u yleensä poissulkeminen estämällä keskeytykset käsittelyn ajaksi

n Käsittely KLT-säikeenä

- u prioriteetti, oma pino, oma talletusalue rekistereille
- u poissulkeminen kuten muillakin säikeillä
 - F lukot, semaforit, ehdolliset muuttujat
 - F poissulje vain ne rakenteet, joita käsitellään
 - F odota tarvittaessa

n Keskeytyskäsittelysäikeillä suurin prioriteetti

- u korkeamman prioriteetin keskeytys voi keskeyttää

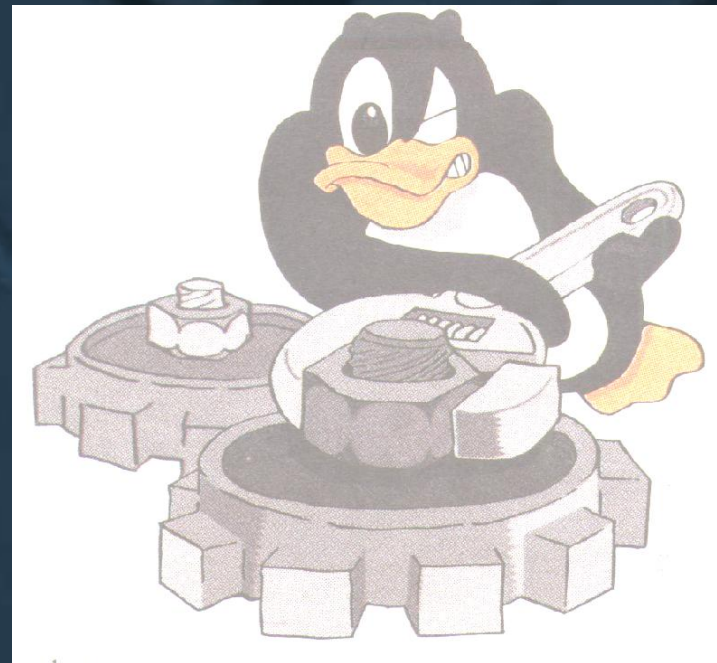
n Keskeytyskäsittelysäikeet olemassa pysyvästi

- u aikaa ei kulu luontiin / siivoamiseen

Käyttöjärjestelmät II

Linux säikeet ja SMP

Ch 4.6 [Stal05]



Linux: Säikeet

- n **Ei erota käsitteellisesti prosessia ja säiettä**
 - u toiminnallisesti ytimen säikeet (KLT, kernel level threads)
 - u käyttää yleistermiä task (tehtävä, työ, askare, puuha...)
 - u prosessille ja säikeelle ei erillistä kuvaajaa
 - F säikeet ovat prosesseja, joilla on sama osoiteavaruus ja thread-group?
 - F task_struct
 - F linkittää task-kuvaajat myös sukulaissuhteiden mukaan
- n **POSIX-säiekirjasto pthreads käyttäjätasolle**
 - u `man -k pthread`
- n **Task luodaan systeemikutsulla clone()**
 - u lapselle kopio äidin task_struct:stä
 - F yhteiskäytössä olevat (sama groupid) resurssit, avoimet tiedostot sekä virtuaalimuisti (kuten säikeillä kuuluu ollakin)
- n **fork() perustuu clone()-kutsun käyttöön**



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)
```

n Yhteiskäyttöisyys tavallaan määrää luotiinko 'prosessi' vai 'säie'!

- u otettava huomioon prosessin vaihdossa?
- u kuinka paljon ympäristöä pitää vaihtaa?



Linux: task_struct

rivi 528

[click!](#)

n `include/linux/sched.h` (<http://lxr.linux.no/source>)

n **Yhteiskäyttöinen tieto erillään varsinaisesta task_structista**

- u viitteet useasta task_structista
- u varattujen muistialueiden kuvaus (koodi, data)
- u tiedostokuvaajat (= avoimet tiedostot)
- u muu tiedostoihin liittyvä data (pwd, umask, root)
- u signaalikäsitteilytaulu

[click!](#)

n **Yksittäiskäyttöisestä datasta omat kopiot**

`thread_struct`

- u oma pinoalue
- u tallealue rekistereille
- u tunnistetiedot, linkkikentät



Linux: prosessin tilat

n **TASK_RUNNING**

- u CPU:lla tai odottamassa CPU:ta

Fig 4.18 [Stal05]

n **TASK_INTERRUPTIBLE**

- u odotus, joka voidaan katkaista signaalilla (esim. ajastus)

n **TASK_UNINTERRUPTIBLE**

- u odottaa laitetapahtumaa
- u homma täytyy saada loppuun joskus!
- u ei ota vastaan signaaleja (esim. "kill -9 1234")

n **TASK_STOPPED**

- u pysäytetty, vaatii käyttäjän jatkotoimia, esim. taustaprosessi haluaisi tulostaa näytölle, lukea näppäimistöltä
- u saanut signaalin SIGSTOP, SIGTTIN tai SIGTTOU
- u odottaa nyt signaalia SIGCONT

n **TASK_ZOMBIE**

- u homma valmis, odottaa saattohoitoa



Linux: SMP

n **Normaalit SMP:n käyttötavat**

n **Systemikutsu sysmp()**

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

(‘CPU suvun omaisuutena’)



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);
}
```

Käyttöjärjestelmät II

Samanaikaisuuden hallinta

Ch 6.7-10 [Stal05]

Lukkiutumisen välttäminen ch 6.5

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

UNIX samanaikaisuuden hallinta

n IPC mekanismit

u kommunikointi

- F putket

- F viestit

- F yhteinen muisti

u synkronointi

- F semaforit

- F signaalit

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		•	

UNIX Putki (pipe)

ls sourcedir | grep fileA

n Vuorottaisrutiinien laajennus

- u yksi lukija/yksi kirjoittaja, yhteinen puskuri
- u vuorottelu automaattisesti
- u mutex automaattisesti
- u kiinteä puskurin koko
- u voi olla monta prosessia pareittain
- u end-of-file välitetään sulkemalla putken pää

n esim: parent-child pipe

- u luodaan `int p[2]` taulukkoon putki kutsulla `pipe(p)`;
 - F kaksi "tiedostoa": `p[0]` ja `p[1]`
- u `fork()`
- u isäprosessi sulkee toisen pään (esim `p[1]`) ja avaa toisen pään (`p[0]`) kirjoittamista varten
- u 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)

n Yhdensuuntainen kommunikointi nimetyn tiedoston kautta (FIFO file)

n Ei tarvitse tietää toisen prosessin nimeä!

u joku luo tiedoston

`mkfifo(tdsto)`

u joku prosessi avaa sen lukemiseen

`open(tdsto)`

u joku (tai jotkut) avaavat sen kirjoittamiseen

u lukija/kirjoittajat blokkaantuu kunnes molemmat paikalla

u käyttö jatkossa kuten tavallisella putkella

`read(tdsto)`

u toimii vain omassa koneessa, ei verkon yli

`write(tdsto)`

UNIX viestit (messages)

- n Joka prosessilla oma viestijono: *msqid* (message queue, mailbox)
- n Viesteillä tyyppi (esim. *long int*)
- n Voi lukea seuraavan tai seuraavan tietyn tyyppisen viestin
- n Myös lähetävä prosessi voi blokkautua
 - u 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);
```

UNIX yhteinen muisti (shared memory)

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

UNIX semaforit

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

UNIX Semafori-operaatiot

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

n `sem_op > 0` (ALTER)

- u lisää `sem_op` arvoon
- u vapauta arvon kasvamista odottajat

n `sem_op == 0` (READ)

- u jos arvo oli nolla, niin päästä tämä läpi
- u muuten odota, kunnes arvo on nolla

n `sem_op < 0 ja |sem_op| ≤ arvo` (ALTER)

- u vähennä `|sem_op|` arvosta
- u jos arvo tuli nolaksi, herätä kaikki sitä odottaneet

n `sem_op < 0 ja |sem_op| > arvo` (ALTER)

- u odota kunnes arvo kasvaa

UNIX signaalit

Tbl 6.2 [Stal 05]

- n ”Ohjelmiston antama keskeytys”
 - u laitekeskeytyksen käsittelijä voi generoida
- n Ei prioriteetteja
- n Kaikki käsitellään, jossain järjestyksessä
- n PCB:ssä vain yksi bitti per signaalityyppi
 - u useasta saman tyyppisestä signaalista jää vain yksi jälki
- n Eivät herätä *uninterruptible* tilasta Fig 4.18 [Stal 05]
 - u tärkeä homma kesken, pitää lopettaa kunnolla
 - u ei saa tappaa tai häiritä signaaleilla!

Linux ja samanaikaisuuden hallinta

Ch 6.8 [Stal 05]

Linuxin samanaikaisuuden hallinta

n Kuten muissa Unix'eissa

- u putket, viestit, yhteinen muisti, signaalit, semaforit

n Etuoikeutetussa tilassa myös muita

- u atomiset *int*- ja *bitmap*-operaatiot
- u *spinlock* kriittisten vaiheiden suojaamiseen
- u *spinlock* variantit keskeytyskäsitteijöiden kriittisten vaiheiden suojaamiseen
- u *reader-writer spinlock* reader-writer -ongelmien ratkaisuun
- u *kernel semaphore* ytimen tason semaforiratkaisuihin
- u *barrier* synkronointiprimitiivi käskytason suorituksen kontrollointiin

Linux ytimen atomiset operaatiot

Tbl 6.3 [Stal 05]

n *Atomic int* operaatiot

- u vain erityiselle atomiselle kokonaislukuarvoiselle tietotyypille
- u tietotyyppiä ei voi manipuloida muilla tavoin
 - F esim. kääntäjän optimoiman aliaksen kautta
- u atomic int operaatiot eivät voi manipuloida muuta dataa
- u hitaampaa kuin vastaavat normaali operaatiot

n *Atomic bitmap* operaatiot

- u manipuloi bittiä osoittimen osoittamassa bittikartassa

Linux ytimen *perus-spinlock*

Tbl 6.4 [Stal 05]

n Neljä varianttia

u *plain* on tavallinen spinlock

- F tavalliset poissulkemisiongelmat
- F busy-wait kunnes resurssi vapaa

u *_irq* keskeytyskäsitteijöille

- F estää keskeytykset odotuksen aikana (kesk. sallittu siis alkuaan)
- F sallii keskeytykset odotuksen jälkeen

u *_irqsave* keskeytyskäsitteijöille

- F kuten *_irc*, mutta keskeytystilan talletus/palautus

u *_bh* keskeytyskäsitteijöiden top half/bottom half -toteutukseen

- F bottom half'ia ei saa suorittaa, kun top half on suorituksessa ja päinvastoin?

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

Linux ytimen *spinlock* vs. *SMP*

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

Linux ytimen *Reader-Writer spinlock*

n Multiple readers- single writer –ongelmaan

- u lukkokuuntujalla 2 kenttää

 - F flag = 1, joss lukko vapaa

 - F flag = 0, jos lukko varattu

 - counter = 0, jos lukko varattu kirjoittajalle

 - counter = n, jos lukko varattu n:lle lukijalle

- u kirjoittaja saa lukon varattua vain, jos flag = 1

 - F odottava kirjoittaja ei estä uusia lukijoita

 - F kirjoittaja voi odottaa kauan! epäreilua?

- u *plain*, *_irq* ja *_irqsave* –versiot

n Miksi ei semaforeilla?

- u tehokkuuden vuoksi busy-wait odotus

Linux ytimen *semafori*

Tbl 6.5 [Stal 05]

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

Linux ytimen *barrier*

Tbl 6.6 [Stal 05]

n Käskytason suorituksen kontrollointiin

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

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




Sun Solaris, säikeiden hallinta

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

Solaris säikeiden mutex

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

Solaris säikeiden semaforit

n *sema_p()* ja *sema_v()* normaalit P ja V

n myös *sema_try_p()*

- u toimii kuten P, mutta palauttaa vain tiedon epäonnistumisesta, jos olisi blokkaantunut

- u kuten *mutex_tryenter()*

Solaris säikeiden Readers/Writer -lukko

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

Solaris säikeiden ehtomuuttajat

- n yleisiin barrier-synkronointeihin
 - u monta säiettä odottaa
 - u yksi tai kaikki säikeet saa jatkaa
- n voidaan käyttää minkä tahansa ehdon yhteydessä
 - u käyttö pitää suojata dedikoidulla mutexilla
 - u säie voisi muuten vaihtua ehdon laskemisen aikana
 - F ehto voi olla mikä tahansa lauseke
 - u `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

W2K samanaikaisuuden hallinta

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



W2K IPC

[Ch 11.4.2 Tane01]

n pipes, named pipes

- u tavu- ja viestiperustaiset
- u nimetyt putket verkon yli, tavalliset ei

putki, nimetty putki

n mailslots

- u W2K:ssa, mutta ei UNIXeissa
- u one-to-one, one-to-many, verkon yli

"postilaatikko"

n sockets

- u verkon yli

pistoke

n rpc

n yhteiskäyttöiset tiedostot

- u muistialue mapataan usealle prosessille
- u säikeet hoitavat mutex-ongelman (toivottavasti)
 - F eri prosessien säikeillä voi olla yhteinen muistialue

W2K samanaikaisuuden hallinta

n Normaali toiminto, jota voidaan aina odottaa

- u prosessi tai säie päättyy
- u tiedosto-operaatio valmis
- u syöte valmis

Tbl 6.7 alaosa [Stal05]

n Tavalliset synkronointiprimitiivit

- u tietty muutos tiedostojärjestelmässä

```
FindFirstChangeNotification( "filter", "directory")
```

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

Tbl 6.7 yläosa [Stal05]

W2K mutex

Tbl 6.7 [Stal05]

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

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

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

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

W2K semafori

Tbl 6.7 [Stal05]

n Tavallinen semafori (counting semaphore)

- u arvo nolla tai isompi
- u ytimen olio, ytimen olioiden suojaukset
- u 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]

n Manual-reset event (& auto-reset event)

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

W2K odotettava ajastin (waitable timer)

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

Kertauskysymyksiä

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