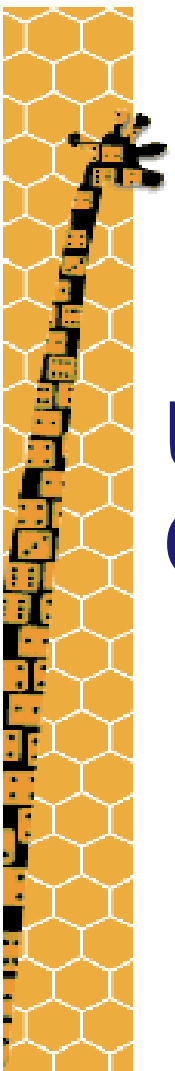


UNIX-sovellusohjelmointi
Osa 2

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Sisältö

- Prosessien välinen kommunikointi: putket
- Prosessien välinen kommunikointi: IPC
- Säikeet ja synkronointi
- Muuta mielenkiintoista

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Prosessien välinen kommunikointi

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Prosessin vaikutuksesta toisiinsa

- Prosessit vaikuttavat toisiinsa suoraan ja epäsuoraan:
  - Kilpailu resursseista (CPU, muisti, siirräntä)
  - fork() / wait()
  - Yhteiset levytiedostot tai niiden olemassaolo
  - Signaalit
  - Tiedostolukot
  - Muistiinkuvattu tiedosto
  - Pseudopääte eli terminaali
  - Putket, FIFOt
  - Sanomanvälitys
  - Yhteiskäytössä oleva muisti
  - Semaforit

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Prosessien välinen kommunikointi: sisältö

- Terminal I/O
- Putket: nimeämättömät, nimetyt
- Inter-process communication (IPC): System V ja POSIX
  - Viestit
  - Semaforit
  - Jaettu muisti

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Prosessien välinen kommunikointi: terminal I/O

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Terminal I/O

- Terminaalilla viitataan erilaisiin laitteisiin, esim.
  - Terminaali-istunto
  - Fyysisesti yhteenliitetyt laitteet (terminaalilaite ja palvelin)
  - Modeemit
  - Tulostin
- Terminaalipohjaisen siirräntä on jokseenkin monimutkaista johtuen juuri laitteiden kirjosta
- Siirräntää voidaan ohjata ja säätää lukuisin eri parametrein
- Seuraavilla kalvoilla käsitellään lähinnä ns. terminaali-istuntoa

Terminal I/O

- Käynnistettävään prosessiin liittyy keskeisesti ns. “ohjaava terminaali”, terminaali-istunto
- Terminaalin ominaisuuksiin voi itse vaikuttaa
- Oletusarvoisesti, terminaali esimerkiksi
  - Toistaa kaiken STDIN:iin kirjoitetun,
  - Palauttaa vain kokonaisia rivejä STDIN:stä
- Kaikilla terminaaleilla on nimi, jonka saa selville kutsulla *char *ctermid*(**char ***s)
- Yleensä terminaalin nimi on “/dev/tty”

- Esimerkki: terminalio.c

Terminal I/O

- Terminaalin ominaisuudet esitetään pääsääntöisesti lipukkeina
- Terminaalin ominaisuudet (lipukkeet) saa kutsulla:
  *int tcgetattr(***int** fd, **struct termios** *termios_p*)*
- Terminaalin ominaisuuksia voi muokata komennolla:
  int tcsetattr(**int** fd, **int** opt_actions, **struct termios** *p)
- Kutsun tietorakenne *struct termios* sisältää kentät
  - `tcflag_t  c_iflag  input modes`
  - `tcflag_t  c_oflag  output modes`
  - `tcflag_t  c_cflag  control modes`
  - `tcflag_t  c_lflag  local modes`
  - `cc_t  c_cc[NCCS]  control chars`
- Asetetut arvot jäävät voimaan ohjelma päätyttyä
- Ominaisuuksien palauttaminen alkuasetuksiin voi olla ongelmallista (tallenna ne siis heti alkuunsa!)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: echo pois

```
#define ECHOFLAGS (ECHO | ECHOE | ECHOK | ECHONL)

int setecho(int fd, int onflag)
{
   int error;
   struct termios term;

   if (tcgetattr(fd, &term) == -1)
      return -1;
   if (onflag)                          /* turn echo on */
      term.c_lflag |= ECHOFLAGS;
   else                                 /* turn echo off */
      term.c_lflag &= ~ECHOFLAGS;

   while (((error = tcsetattr(fd, TCSAFLUSH, &term))
  ==     -1) && (errno == EINTR)) ;

   return error;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: merkki kerrallaan

```
int setnoncanonical(int fd, int vmin, int vtime)
{
   int error;
   struct termios term;

   if (tcgetattr(fd, &term) == -1)/* get its termios
  */
      return -1;
   else
   {
      term.c_lflag &= ~ICANON;
      while (((error = tcsetattr(fd, TCSAFLUSH,
  &term))           == -1) && (errno == EINTR)) ;
      if (error)
         return -1;
   }
   term.c_cc[VMIN] = (cc_t)vmin;
   term.c_cc[VTIME] = (cc_t)vtime;
   while (((error = tcsetattr(fd, TCSAFLUSH, &term))
  ==          -1) && (errno == EINTR)) ;
   return 0;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Prosessien välinen kommunikointi: putket

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Putket

- UNIX/Linux järjestelmien yksinkertaisin prosessien välinen kommunikointimuoto on ns. putket
- Terminaalin komentorivillä voi komentoja putkittaa merkillä '|', esimerkiksi “ls -l | more” tai “ls -l | wc -l”
- Käyttöjärjestelmä pitää huolen siitä, että putkeen kirjoitetut merkit luetaan first-in-first-out järjestyksessä
- Vastaava toiminnallisuus saadaan aikaan omissa ohjelmissa
- Putkia on kahdenlaisia:
  - Nimeämätön: prosessiryhmän sisäiseen käyttöön
  - Nimetty: minkä tahansa prosessien välille

Nimeämättömät putket

- Nimeämätön putki luodaan funktiolla:
  *int pipe(***int fd[2]***)*
- Putki on puskuroitu
- Putki on erikoistiedosto, jolla on kaksi tiedostokuvaajaa:
  - fd[0] täältä voi lukea
  - fd[1] tänne voi kirjoittaa
- Putkella ei ole ulkopuolista nimeä tai paikkaa, ts. siihen voi viitata vain mainituilla tiedostokuvaajilla
- Putket toimivat vain saman prosessiryhmän sisällä
- Putki katoaa, kun prosessiryhmän suoritus päättyy

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Nimeämättömät putket

- Putket on suunniteltu lähinnä äiti ja lapsi prosessien väliseen kommunikointiin
  - Äiti-prosessi luo putken
  - Fork-kutsussa lapsi perii äidin avoimet tiedosto-kuvaajat, jolloin se voi lukea tai kirjoittaa putkeen tarpeen mukaan
- Putkilla ei juuri ole käyttöä yhden prosessin sisällä
- Yhtä putkea järkevää käyttää vain yhteen suuntaan
- Kaksisuuntaiseen kommunikointi: kaksi putkea
- Tarpeettomat putken päät suljetaan sekä äiti- että lapsiprosessissa
- Ainoa varsinainen virhe putkea avattaessa voi olla, että tiedostokuvaajataulu on jo täynnä, eikä uusia tiedostoja voi enää avata

Nimeämättömät putket

- Putkeen mahtuva määrä tavuja on *PIPE_BUF*
- Write on atominen, jos kaikki tavut mahtuvat putkeen
- Toimintoja:
  - Read + tyhjä putki: odottaa kunnes luettavaa
  - Write + täysi putki: odottaa kunnes taas tilaa (joku lukenut pois)
  - Write + osa sopii: kirjoittaa mikä sopii, odottaa kunnes loppukin mahtuu
  - Read, kun write-pää suljettu: lukee tavut ja seuraava read palauttaa *EOF*
  - Write, kun read-pää suljettu: prosessi saa *SIGPIPE*-signaaling ja *errno* = *EPIPE*

Nimeämättömät putket

- Funktio
  *FILE *popen*(const **char** *cmd, const **char** * mode)
  on yhdistelmä fork- ja pipe-kutsuja
- *popen* käynnistää lapsiprosessin, joka suorittaa komennon *cmd*
- Lapsen ja äidin välille luodaan samalla putki
- *mode*-parametri määrittelee, avataanko putki kirjoittamista vai lukemista varten (ei molempia)
  - 'r' tarkoittaa, että putki avataan lukemista varten,
  - 'w' tarkoittaa kirjoittamista
- Putki osoittaa lapsen STDOUT ja STDIN virtoihin
- Tietovirta on täysin puskuroitu
- Putki suljetaan pclose() funktiolla, joka odottaa, kunnes prosessi on päättynyt, ja palauttaa sen paluuarvon

Esimerkki: pipe.c

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <limits.h>

#define BUFSIZE 128

int main(void) {
   char buf[BUFSIZE] = "Hello my child!";
   int bytes;
   pid_t childpid;
   int fd[2];

   bytes = strlen(buf);

   if (pipe(fd) == -1) {
      perror("Failed to create the pipe");
      return 1;
   }
   printf("Size of pipe buffer: %d\n",PIPE_BUF);
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: pipe.c

```
   childpid = fork();
   if (childpid == -1) {
      perror("Failed to fork");
      return 1;
   }

   if (childpid)                                          /* parent code */
   {
        close(fd[0]);
        printf("Parent [%d]: sending \"%s\" to child...",getpid(),buf);
        write(fd[1], buf, bytes);
        printf("Done\n");
        wait(NULL);
        printf("Parent [%d]: child has terminated, exiting\n",getpid());

   }
   else                                                    /* child code */
   {
        close(fd[1]);
      sleep(2);
      bytes = read(fd[0], buf, BUFSIZE);
      buf[bytes]='\0';
      fprintf(stderr, "Child [%ld]: my mamma said \"%s\"\n",
           (long)getpid(), buf);
   }
   return 0;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: popen.c

```
#include<stdio.h>

int main(int argc, char *argv[])
{
    FILE *fp=NULL;

    char buf[1024];

    if(argc != 2)
    {
        printf("Usage: %s command\n",argv[0]);
        return 0;
    }

    printf("**********Suorittamassa popen-kutsun...\n");

    fp = popen(argv[1],"r");

    if(fp == NULL) { perror("popen"); return -1; }

    while(fgets(buf,1024,fp) != NULL)
        printf("popen says: %s",buf);

    printf("**********popen suoritettu\n");

    return 0;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Nimetyt putket (FIFO)

- Kun nimeämättömät putket katoavat prosessien mukana, nimettyä putkea edustaa tiedostojärjestelmässä nimetty fyysinen tiedosto
- FIFO:lla on nimi ja käyttöoikeudet, kuten millä tahansa muullla tiedostolla
- Käyttöoikeuksien rajoissa käytettävissä siis muillakin prosesseilla, esim. 'ei-sukulaisprosessien' välillä
- Kaksisuuntaiseen kommunikointiin tarvitaan kaksi putkea

Nimetyt putket (FIFO)

- Luodaan tavallisen tiedoston luonnin tapaan, kerrotaan tiedostonimi ja annetaan käyttöoikeudet:
  int mkfifo(const **char** *pathname, **mode_t** mode)
- Tämän jälkeen putki avataan funktiolla open()
- Käyttö ei poikkea muiden tiedostojen käytöstä
- FIFO:a voi käyttää esimerkiksi erilaisten koneen sisäisten asiakas-palvelin ohjelmien toteuttamisen osana
- Hävitetään funktioilla
  unlink(const **char** *pathname) tai
  remove(const **char** *pathname)

Nimetyt putket (FIFO)

- Toiminta:
  - Open() + O_WRONLY: odottaa, kunnes toinen prosessi avannut lukemista varten
  - Open() + O_RDONLY: odottaa, kunnes toinen prosessi avaa kirjoittamista varten
  - Read() + tyhjä putki: odottaa kunnes tulee luettavaa
  - Write() + täysi putki: odottaa, kunnes taas tilaa
  - Read(), kun write-pää suljettu: lukee tavut, ja palauttaa seuraavalla kerralla EOF
  - Write(), kun read-pää suljettu: prosessi saa signaalin *SIGPIPE* ja *errno=EPIPE*
- Read palauttaa 0, kun write-pää ei ole enää kenelläkään auki

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Nimetyt putket (FIFO)

- Jos FIFO:a käytetään asiakas-palvelin toteutuksissa, on tärkeä huomioida, että luku-operaatio FIFO:sta ei ole atominen operaatio
- Kaksi kilpailevaa asiakasta voi aiheuttaa toisilleen ongelmia, sillä read(fd,buf,4) FIFO:sta ei välttämättä palauta 4 tavua
- Seuraava skenaario on mahdollinen (ei välttämättä todennäköinen):
  - Palvelin kirjoittaa putkeen kokonaiseksi tarkoitetun viestin “abcdefgh”
  - Ensimmäinen asiakas lukee putkesta “abcd”
  - Toinen lukee loput kirjaimet “efgh”
  - Kumpikaan asiakas ei saanut kokonaista viestiä ulos

Esimerkki: fifo1.c

```
#define BUFSIZE 128
#define FIFO_PERMS (S_IRWXU | S_IWGRP| S_IWOTH)

int main(int argc, char *argv[])
{
   char buf[BUFSIZE];
   int buflen;
   pid_t childpid;
   int fd;

   /* name of server fifo is passed on the command line */

   if (argc != 2)
   {
      fprintf(stderr, "Usage: %s fifoname\n", argv[0]);
      return 1;
   }
      /* create a named pipe to handle incoming requests */
   if ((mkfifo(argv[1], FIFO_PERMS) == -1) && (errno != EEXIST))
   {
       perror("Failed to create a FIFO");
       return 1;
   }
   childpid = fork();
   if (childpid == -1)
   {
      perror("Failed to fork");
      return 1;
   }
```

Esimerkki: fifo1.c

```
   if (childpid)                                /* parent code */
   {
     /* open a write communication endpoint to the pipe */

      if ((fd = open(argv[1], O_WRONLY)) == -1){
        perror("Failed to open FIFO"); return 1; }

      strcpy(buf,"hello my child");
      buflen=strlen(buf);
      printf("Parent [%d]: sending \"%s\" to child...",getpid(),buf);
      write(fd, buf, buflen);
      printf("Done\n");
      wait(NULL);
      printf("Parent [%d]: child has terminated, exiting\n",getpid());
   }
   else                  /* child code */
   {
   /* open a read communication endpoint to the pipe */

      if ((fd = open(argv[1], O_RDONLY)) == -1){
         perror("Failed to open FIFO"); return 1;}

      sleep(2);
      buflen = read(fd, buf, BUFSIZE);
      if(buflen > 0)
        fprintf(stderr, "Child [%d]: parent said \"%s\"\n", getpid(), buf);
   }
   return 0;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: fifo2.c

```
...

if ((mkfifo(argv[1], FIFO_PERMS) == -1) && (errno != EEXIST)) {
  perror("Failed to create a FIFO");
  return 1;
}

if(errno == EEXIST) {
  struct stat stat_buf;

  if(stat(argv[1],&stat_buf) != 0) {
    perror("stat");
    return 1;
  }
  if(S_ISFIFO(stat_buf.st_mode))
    printf("File %s exists, and is a FIFO, not
   problem...\n",argv[1]);
  else {
    printf("File %s exists, and is not a FIFO,
   exiting...\n",argv[1]);
    return 1;
  }
}

...
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: fifo2.c

Äiti-prosessin koodin loppuun:

```
   ...
   wait(NULL);
   printf("Parent [%d]: child has terminated, exiting\n",getpid());

   close(fd);
   unlink(argv[1]);
}
```

Lisäksi voisi vielä ajatella tekevänsä signaalin käsittelijän, joka CTRL-C painamisen (SIG_INT) huomatessaan, poistuu ohjelmasta, ja poistaa samalla FIFO-tiedoston.

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Muita esimerkkejä

- Jatko fifo2:lle eli fifo3: vielä turvallisempi
- pipeserver ja pipeclient:
  - Palvelin luo fifo-putken ja odotteleen asiakkata
  - Palvelimen putkeen voi kirjoittaa pipeclient ohjelmalla tai suoraan: *echo “viestin” > fifo*

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

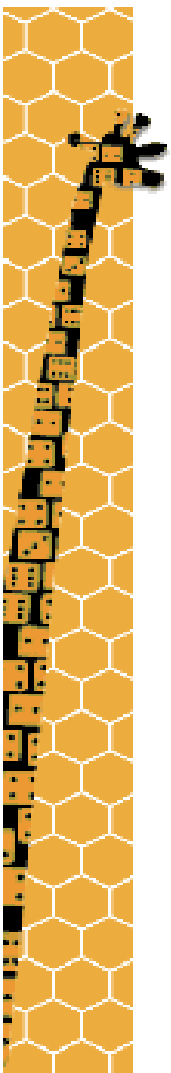
Putkien toteutus Linuxissa

- Puskurille on varattu yksi muistisivu, 4096 tavua, ja on toteutettu renkaana (circular buffer)
- Putken luonti synnyttää i-noden, joka sisältää mm.

```
struct pipe_inode_info
 wait_queue_head_t wait;         /* odotusjono */
 char base;                      /* muistiosoitteen alku */
 unsigned int len;               /* kirjoitetut tavut, lukematta */
 unsigned int start;             /* lukupositio */
 unsigned int readers;           /* lukijoiden lukumäärä */
 unsigned int writers;           /* kirjoittajien lukumäärä */
 unsigned int waiting_readers;   /* nukkuvia lukijoita */
 unsigned int waiting_writers;   /* nukkuvia kirjoittajia */
 unsigned int r_counter;         /* käytetään odottamiseen */
 unsigned int w_counter;         /* käytetään odottamiseen */
```

- Poissulkeminen toteutettu semaforilla
- Toteutus kuten klassinen “readers and writers”-ongelma

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY



Prosessien välinen kommunikointi

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

IPC

- Tässä osassa käsitellään prosessien välistä kommunikointia seuraavilla menetelmillä:
  1. viestit
  2. semaforit
  3. jaettu muisti
- Jokaisesta menetelmästä on olemassa kaksi erilaista rajapintaa.
  1. System V IPC (XSI IPC)
  2. POSIX IPC
- Linux toteuttaa (2.6.x) näistä kaiken:
  - System V IPC: kaikki jo pitkään
  - POSIX IPC: semaforit ja jaettu muisti sekä vasta viime aikoina myös viestit

IPC

- Mikään näistä ei ole tiedosto eikä edes erikoistiedosto, vaan ytimen tietorakenteita
- Jokaisella on oma nimeämismenetelmä, säilymissäännöt ja oikeudet
- Perusominaisuudet:
  - Ovat näkyvissä vain yhden koneen sisällä
  - Elossa kun kernelikin, katoavat rebootissa
  - Käsitellään kokonaislukuavaimella
  - Ei voi käyttää tiedostoissa käytettyjä systeemikutsuja

Avaimet

- System V
  - Avaimet ovat tyyppiä key_t, joka monissa toteutuksissa on käytännössä kokonaisluku
- POSIX
  - Avaimet ovat tiedostonimen näköisiä, näistä muodostetaan eräänlainen tiedostokuvaajanumero
- Oleellista: kommunikoivien ohjelmien pitää tietää, miten viitata oikeaan “paikkaan”, samaan tapaan kuin nimettyjen putkien kanssa

Avaimen sopiminen

- Avaimen voivat asiakas ja palvelin sopia määrittelemällä avaimen, esim.
  1. #define -direktiivillä tai header-tiedostossa
  2. Palvelin kirjoittaa tiedon tunnettuun tiedostoon
  3. Lapsiprosessi saa nimen äidiltään
- Hankaluutena varmistaa että avaimet pysyvät yksikäsitteisinä, ylläpidettävyys kärsii
- Avaimen voi myös generoida käyttäen tiedostonimeä ja lisäosaa funktiolla (System V):
  key_t *ftok*(**const char** *path,**int** id)
  - path parametrissä annetaan tiedostonimi
  - id:stä otetaan 8 vähiten merkitsevää bittiä
  - Palauttaa joko avaimen tai -1 jos avaimen muodostus epäonnistuu

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

System V IPC: Oikeudet

- Oikeudet määritellään aina kun luodaan uusi IPC rakenne
- Tietoja voidaan kysellä tai muuttaa käyttäen Xctl(), jossa X on haluttu IPC rakenne: *sem, shm,* tai *msg*
- Oideudet määritellään struct ipc_perm rakenteen avulla:

```
struct ipc_perm {
  uid_t  uid;  /* owner user-id */
  gid_t  gid;  /* owner group-id */
  uid_t cuid;  /* creator user-id */
  gid_t cgid;  /* creator group-id */
  mode_t mode; /* permission bits */
  };
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

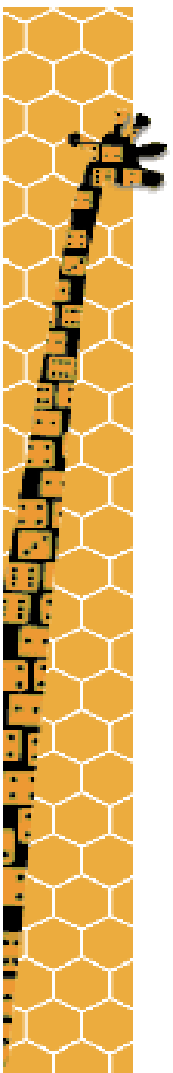
System V IPC rakenteet komentotulkissa

- Voit listata kaikki IPC rakenteet komennolla **ipcs**:

```
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch
   status
0x00000000 2064384    jmanner    600        393216     2          dest
0x00000000 2097153    jmanner    600        393216     2          dest
0x00000000 47185924   jmanner    666        72000      1          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

- Tietyn IPC rakenteen voi poistaa komennolla **ipcrm**
  - Esimerkiksi viestijono avaimella 50 poistettaisiin komennolla:
    - ipcrm -Q 50
  - Vastaavasti jaetun muistin lohko avaimella 2064384 poistettaisiin komennolla
    - ipcrm -M 2064384

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY



IPC: viestijonot

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

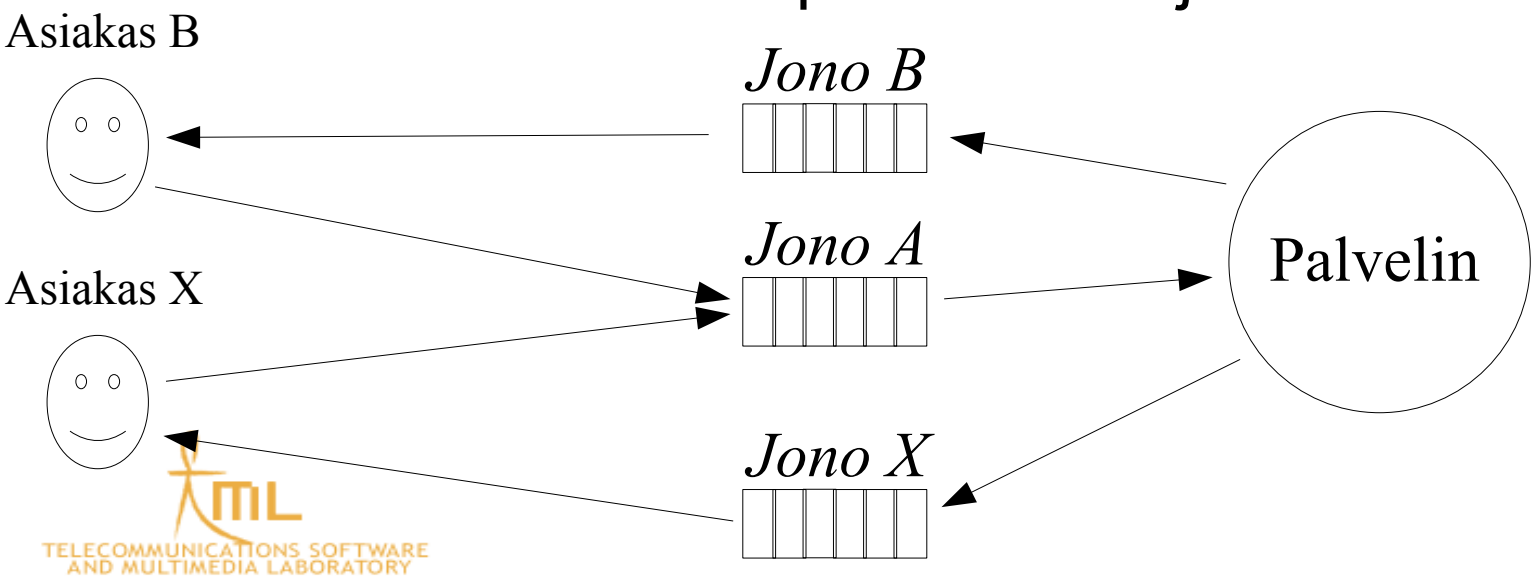
System V IPC viestijonojen systeemikutsut

- #include<sys/msg.h>
- **int** *msgget*(**key_t** key, **int** msgflg);
- **int** *msgsnd*(**int** msqid, **const void** *msgp, **size_t** msgsz, **int** msgflg);
- **ssize_t** *msgrcv*(**int** msqid, **void** *msgp, **size_t** msgsz, **long** msgtyp,**int** msgflg);
- **int** *msgctl*(**int** msqid, **int** cmd, **struct msqid_ds** *buf);

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki viestijonojen käytöstä

- Yleensä palvelin luo viestijonon *A*, josta vastaanottaa palvelupyyntöjä
- Asiakas hakee palvelimen viestijonon *A* sovitulla avaimella
- Tämän jälkeen asiakas luo uuden jonon *B*, johon palvelin lähettää vastauksia
- Lopuksi asiakas välittää uuden viestijonon *B* ja itsensä tunnisteen palvelimelle jonon *A* kautta



TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Viestijonon luonti ja hakeminen

- Yleensä palvelin luo viestijonon, josta vastaanottaa palvelupyyntöjä:

```
key_t msgjono;
key_t key = 12345;
msgjono = msgget(key,IPC_CREAT|0777);
```

- Asiakas hakee palvelimen viestijonon sovitulla avaimella:

```
key_t smsgjono;
key_t key = 12345;
smsgjono = msgget(key,0);
```

- Tämän jälkeen kannattaa asiakkaan luoda uusi viestijono, josta haluaa kuunnella vastauksia:

```
key_t cmsgjono;
key_t ckey = 23456;
cmsgjono = msgget(ckey,IPC_PRIVATE|0777);
```

- Tämän jälkeen asiakas välittää uuden viestijonon tunnisteen palvelimelle

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Viestijonon kontrollointi

- Viestijonon kontrollointiin on funktio **int** *msgctl*(**int** msgid,**int** cmd,**struct msqid_ds** *data)
- cmd argumentilla on kolme mahdollista arvoa:
  - IPC_RMID: poistaa kyseisen viestijonon
  - IPC_STAT: hakee viestijonon tiedot data tietorakenteeseen
  - IPC_SET: asettaa viestijonon tiedoista kentät
    - uid, gid, mode ja msg_qbytes

```
struct msqid_ds {
    struct ipc_perm msg_perm; /* permissions */
    msgqnum_t msg_qnum;  /* number of messages in queue */
    msglen_t msg_qbytes; /* max bytes allowed in queue */
    pid_t msg_lspid;     /* process ID of last msgsnd */
    pid_t msg_lrpid;     /* process ID of last msgrcv */
    time_t msg_stime;    /* time of last msgsnd */
    time_t msg_rtime;    /* time of last msgrcv */
    time_t msg_ctime;    /* time of last msgctl change */
    };
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Viestin lähettäminen

- Viesti lähetetään funktiolla:
  int *msgsnd*(**int** msqid,**const void*** msgp,**size_t** msgsize,**int** flags)
  - Palauttaa 0 onnistuessaan muuten -1
  - jos jono täynnä ja flag == IPC_NOWAIT niin errno = EAGAIN
  - muuten odottaa, että sanomalle on tilaa
  - jos joku poistaa sanomajonon, niin errno = ERMID
  - jos prosessi saa signaalin, niin errno = EINTR

Viestien vastaanottaminen

- Viesti vastaanotetaan funktiolla:
  ssize_t *msgrcv*(**int** msqid,**void** *msgp,**size_t** mbufsize,**long** msgtype,**int** flags)
  - Palauttaa vastaanotettujen tavujen määrän tai -1 virhetilanteessa.
  - Jos jono täynnä ja flag == IPC_NOWAIT niin errno = EAGAIN
  - Muuten odottaa, että sanomalle on tilaa
  - Jos joku poistaa sanomajonon, niin errno = ERMID
  - Jos prosessi saa signaalin, niin errno = EINTR
  - Jos msgtype == 0 palauttaa ensimmäisen viestin jonosta.
  - Jos msgtype > 0 palauttaa ensimmäisen kyseistä tyyppiä olevan viestin jonosta
  - Jos msgtype < 0 palauttaa pienimpityyppisen viestin jossa msgtype <= |msgtype|.
    - eli jos jonossa on viestit 5, 6 ja 17 ja haetaan viestiä -6; palautuu tyypin 5 viesti.

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Viestin rakenne

- Viestin rakenne on täysin ohjelmoijan valittavissa
- Sekä asiakkaan että palvelimen tulee ymmärtää rakenne samoin
- Yleinen viestin rakenne on:

```
struct msg {
  long mtype; /* message type */
  char mtext[MTEXTSIZE]; /* message*/
  };
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Rajoituksia

- Rajoituksia:
  1. Viestien kokonaiskokoraja jonossa. Tämän rajan voi selvittää *msgctl()* systeemikutsulla ja tarkastelemalla *msg_qbytes* kenttää *msqid_ds* rakenteessa.
  2. Viestien määräraja
  3. Viestin kokoraja
  4. Viestijonojen lukumääräraja
- Kaksi ensimmäistä rajoitusta eivät välttämättä ole vakavia. Aina voi määritellä *msgsnd()* kutsun odottavan, jos rajat ovat täysiä.
- Loput ovatkin hankalia, koska niitä ei voi selvittää edes *sysconf()* systeemikutsulla. Ainoa tapa selvittää ne on kokeilemalla.
- Ainoa tapa muuttaa rajoituksia on uudelleenkonfiguroida ydin.

Esimerkki: msend.c

```
int main(int argc, char *argv[])
{
  key_t key;
  int id;
  long type;
  typedef struct {        /* This is the standard method */
           long mtype;
           char mtext[1]; } msg_t;

  size_t mlen;
  msg_t *mess;

  if ( argc < 4 ) { printf("msend <keyfile> process message\n");
                    exit(1); }

  CHECK((key = ftok(argv[1],'D')) != -1);
  CHECK((id = msgget(key,0)) != -1);
  CHECK((type = atol(argv[2])) > 0); /* message type: proc. id */
  mlen = strlen(argv[3]);
  CHECK((mess = (msg_t *) malloc(sizeof(long) + mlen)));
  mess->mtype = type;
  memcpy(mess->mtext, argv[3], mlen);
  CHECK(msgsnd(id, mess, mlen, 0) == 0); /* send msg to proc. */
  free(mess);
  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: mreceive.c (katso myös mreceive_any)

```
int main(int argc, char *argv[]) /* argumenttina: tiedostonimi */
{
  key_t key;
  int id;
  typedef struct {
      long mtype;
      char mtext[1];
  } msg_t;

  size_t   msize;            /* paljonko tilaa varattu */
  msg_t    *mess;
  int                mlen;             /* todellinen pituus */

  if ( argc < 2 ) { printf("mreceive <keyfile>\n"); exit(1); }

  CHECK((key = ftok(argv[1],'D')) > 0);
  CHECK((id = msgget(key, IPC_CREAT|0777)) != -1);

  /* varaa tilaa sanomalle */
  msize = 20;
  CHECK((mess = (msg_t *)malloc(sizeof(long) + msize + 1)));
  printf("Process %d ready to receive
messages.\n",(int)getpid());
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa...

```
do {
    /* Get messages for this process ! */
      if ((mlen=msgrcv(id, mess, msize, getpid(), 0)) == -1) {
          if (errno == EINTR) continue;

          if (errno == E2BIG) {
           free(mess);
           msize *= 2;
           printf("%d sanoman koko -> %d\n", getpid(), msize);
           CHECK((mess = (msg_t *)malloc(sizeof(long)+msize+1)));
           continue;
          }

          CHECK(errno == 0);
      }

      mess->mtext[mlen] = (char) 0;
      printf("%d '%s'\n", getpid(), mess->mtext);

  } while (mlen != 0);

  free(mess);
  CHECK((msgctl(id,0,IPC_RMID)) != -1);

  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

POSIX viestijonot

- Linux toteuttaa POSIX viestien rajapinnan vasta versiosta 2.6.5 alkaen
- Lisäksi tarvitaan kirjasto, joko erillinen, tai gnu libc-2.3.4:ssa ja uudemmissa oleva
- Kääntämiseen tarvitaan Linuxissa vielä vipu “-lrt”
- Mahdollistaa viestien järjestämisen tärkeysjärjestykseen viestijonossa
- Rajapinta on hieman yksinkertaisempi kuin System V:ssä
- Jonoon viitataan nimellä, joka alkaa “/”, esim. “/jokujono”

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

POSIX viestijonot

- Funktiokutsut ovat:
  - **mqd_t** *mq_open*(**const char** *name, **int** oflag, ...)
  - **int** *mq_close*(**mqd_t** mqdes)
  - **int** *mq_send*(**mqd_t** mqd, **const char** *msg,**size_t** msize,**unsigned** prio)
  - **ssize_t** *mq_receive*(**mqd_t** msg,**char** *msg,**size_t** bufsize,**unsigned** *priority)
- Esimerkit: msend_posix.c ja mreceive_posix.c

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

msend_posix.c

```
#include"mydbg.h"
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <mqueue.h>

#define MQ_NAME       "/testmsg"
#define MSG_SIZE        128
#define MAX_MSG         3

int main(int argc, char *argv[]){
  long type;
  typedef struct {        /* This is standard method */
           long mtype;
           char mtext[128];
  } msg_t;
  mqd_t jono;
  unsigned prio;
  size_t mlen;
  msg_t *mess;

  if ( argc != 4 ) { printf("Usage: msend_posix msg_type msg_priority
   message\n"); exit(1); }
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

msend_posix.c

```
  jono = mq_open(MQ_NAME,O_WRONLY);

  if(jono == -1)
  {
        perror("Failed to open message queue");
        return 0;
  }

  CHECK((type = atol(argv[1])) > 0); /* message type */
  CHECK((prio = atoi(argv[2])) > 0); /* message prio */

  mlen = strlen(argv[3]);

  CHECK((mess = (msg_t *) malloc(sizeof(long) + mlen)));

  mess->mtype = type;
  memcpy(mess->mtext, argv[3], mlen);
  mlen += 4;
  CHECK(mq_send(jono, (char *) mess, mlen, prio) == 0); /* send
   message to process */

  free(mess);
  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

mreceive_posix.c

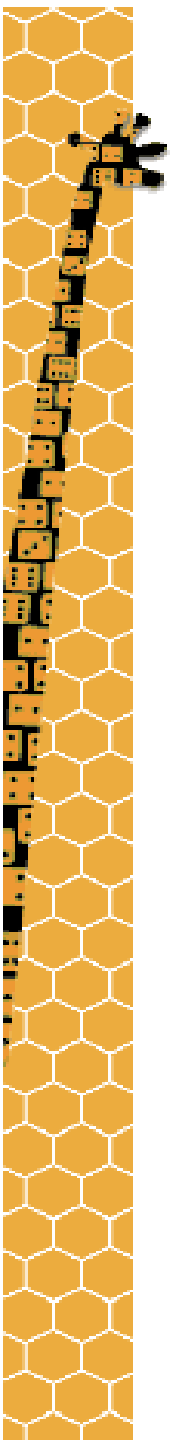
```
#include"mydbg.h"
#include <stdio.h><string.h><sys/types.h><sys/ipc.h><sys/msg.h><errno.h>
   <mqueue.h><fcntl.h><sys/stat.h>
#define MQ_NAME       "/testmsg"
#define MSG_SIZE        128
#define MAX_MSG         3
#define MQ_MODE 0777

int main(int argc, char *argv[]){

  typedef struct {
      long mtype;
      char mtext[1];
  } msg_t;

  size_t   msize;            /* paljonko tilaa varattu */
  msg_t    *mess;
  int               mlen;            /* todellinen pituus */
  mqd_t jono;
  unsigned prio;
  struct mq_attr mqstat;
  if ( argc > 1 ) { printf("Usage: mreceive\n"); exit(1); }

  memset(&mqstat, 0, sizeof(mqstat));
  mqstat.mq_maxmsg = MAX_MSG;
  mqstat.mq_msgsize = MSG_SIZE;
  mqstat.mq_flags = 0;
  jono = mq_open(MQ_NAME,O_CREAT|O_RDWR, MQ_MODE, &mqstat);
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

mreceive_posix.c

```
  if(jono == -1){
    perror("Failed to open message queue");
    return 0; }

  msize = 200; /* varaa tilaa sanomalle */
  CHECK((mess = (msg_t *)malloc(sizeof(long) + msize + 1)));
  printf("Process %d ready to receive messages.\n",(int)getpid());

  do {     /* Get messages for this process ! */
      if ((mlen=mq_receive(jono, (char *) mess, msize, &prio)) == -1) {
          if (errno == EINTR) continue;
          if (errno == EMSGSIZE) {
           free(mess);
           msize *= 2;
           printf("%d sanoman koko -> %d\n", getpid(), msize);
           CHECK((mess = (msg_t *)malloc(sizeof(long)+msize+1)));
           continue;
          }
          CHECK(errno == 0);
      }
      mess->mtext[mlen-sizeof(long)] = (char) 0;
      printf("Msg type %ld prio %u: '%s'\n", mess->mtype, prio, mess-
   >mtext);

  } while (mlen != 0);
  free(mess);
  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY



IPC: semaforit

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Semafori

- Semafori on 'erityistä suojelua' nauttiva ei-negatiivinen kokonaisluku, jolla on alkuarvo (yl. jaossa olevien resurssien määrä) ja johon liittyy odotusjono.
- Tarkoitettu yhteiskäytössä olevien resurssien käyttövuorojen hallintaan eli synkronointiin ja poissulkemiseen.
- Mahdollista jakaa prosessien välillä
- Prosessien ei tarvitse olla sukulaissuhteessa
- Prosessi voi kasvattaa semaforin arvoa milloin vain, mutta vähentää arvoa vain, jos arvo > 0, muuten prosessi odottaa kunnes vähennys on mahdollista.

Semafori

- Odotus päättyy, kun toinen prosessi kasvattaa semaforin arvoa
- KJ huolehtii odottajien jonotuksesta: jos semafori kasvaa yhdellä, vain yksi prosessi vapautetaan odotuksesta ja pääsee eteenpäin
- Odotuksen jälkeen useampikin prosessi on voinut päästä jatkamaan
- Odottaminen on atominen operaatio, saadaan kaikki mitä pyydetään tai sitten ei mitään
- Semaforia voi myös yrittää laskea ilman riskiä siitä, että prosessi jää jumiin, jos semafori on “0”
- Semaforin arvoa voi myös kysäistä, muuttamatta sitä

Semaforien systeemikutsuja

- System V semaforit:
  - semget(): luo semaforijoukko ja nimeä avain tai etsi semaforijoukko avaimen perusteella.
  - semctl(): aseta semaforiin liittyviä parametreja tai semaforien vapautus muistista.
  - semop(): tee semaforiin liittyviä operaatioita eli kasvatus, vähennys.
- POSIX semaforit (nimetyt/nimeämättömät)
  - sem_open() tai sem_init() luo semaforit
  - sem_close() tai sem_destroy() tuhoaa semaforin
  - sem_wait() : vähennä semaforia
  - sem_post() : kasvata semaforia
  - sem_getvalue() : hae semaforin arvo
  - sem_trywait() : yritä vähentää semaforia, älä odota
  - sem_timedwait() : yritä vähentää määrätyn ajan

POSIX semaforin avaaminen

- Nimetty semafori luodaan funktiolla:
  sem_t **sem_open*(**const char** *name,**int** flags, **mode_t** perms,**unsigned** val);
  - flags = O_CREAT
  - perms = normaalit tiedosto-oikeudet
  - val = semaforin alkuarvo
- Nimetty luotu semafori haetaan funktiolla:
  sem_t **sem_open*(**const char** *name,**int** flags)
- Molemmat palauttavat osoittimen sem_t rakenteeseen onnistuessaan tai SEM_FAILED arvon epäonnistuessaan
- Nimeämätön semafori luodaan ja haetaan funktiolla:
  int sem_init(**sem_t** *sem,**int** pshared,**unsigned** value)
  - Palauttaa virhetilanteessa -1

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Semaforin sulkeminen ja kasvattaminen

- Nimetty semafori suljetaan funktiolla
  int sem_close(**sem_t** *sem)
- Nimetyn semaforin voi poistaa funktiolla
  int *sem_unlink*(**const char** *name)
- Nimeämättön semafori suljetaan funktiolla
  int *destroy*(**sem_t** *sem)
- Kaikki palauttavat 0 onnistuessaan tai -1 jos operaatio epäonnistuu jolloin asetetaan myös errno.

- Semaforin kasvatus:
  int *sem_post*(**sem_t** *semd)
  - Samalla vapautuu jokin odottanut prosessi

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Semaforin arvon vähennys (odottaen)

- Semaforin arvon vähentäminen vastaa resurssin varaamista
- Operaatio vastaa operaatiota V (kuuluisia P ja V lyhenteitä käytti semaforien kehittäjä E. W. Dijkstra ja ne tulevat hollanninkielisistä sanoista ”proberen te verlagen” (try to decrease) ja ”verhogen” (increase)).
- Vähennys tapahtuu funktiolla
  int *sem_wait*(**sem_t*** sem)
- Palauttaa onnistuessaan 0
- Virhetilanteesssa palauttaa -1
- Jää odottamaan jos operaatiota ei voida suorittaa, signaali voi katkaista tämän odotuksen jolloin errno = EINTR.

Semaforin arvon vähennys (odottamatta) ja kysely

- Semaforin arvoa voi yrittää vähentää funktiolla
  int *sem_trywait*(**sem_t** *sem)
  - Palauttaa -1 ja errno = EAGAIN jos semaforin arvo oli 0 (ei resursseja).
  - Onnistuessaan palauttaa arvon 0.

- Semaforin arvoa voi kysyä funktiolla
  int *sem_getvalue*(**sem_t** *sem,**int** *value)
  - Palauttaa onnistuessaan 0, jolloin value kentässä on semaforin arvo kutsuhetkellä.
  - Virhetilanteessa palauttaa -1.
  - Et voi luottaa tähän arvoon! Vaikka funktio palauttaisi positiivisen arvon, voi olla että funktion palauduttua jokin muu prosessi on ehtinyt varata kaikki resurssit.

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: posixsema.c

```
#define SEMA "/jepulis"
int semheld = 0;

void release(sem_t *id);
void request(sem_t *id);
void try(sem_t *id);
void getvalue(sem_t *id);

int main(int argc, char *argv[])
{
  sem_t *id;

  /* No arguments: "server". */
  if (argc < 2)
  {
    /* Request a semaphore. */
    CHECK((id = sem_open(SEMA,O_CREAT,0777,2)) != SEM_FAILED);
  }
  else   /* client */
  {
    /* Open up the existing one. */
    CHECK((id = sem_open(SEMA,0)) != SEM_FAILED);
    printf("Using existing semaphore %p.\n", (void *)id);
  }

  printf("Successfully allocated semaphore id %p\n", (void *)id);
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa...

```
printf("Successfully allocated semaphore id %p\n", (void *)id);

 while (1)
 {
   int selection;
   printf("\nStatus: %d resources held by this process.\n",
                semheld);
   printf("Menu:\n");
   printf("1. Release a resource\n");
   printf("2. Request a resource\n");
   printf("3. Try to request\n");
   printf("4. Get value\n");
   printf("5. Exit this process\n");
   printf("6. Remove resource and exit\n");
   printf("Your choice: ");

   scanf("%d", &selection);

   switch(selection) {
     case 1: release(id); break;
     case 2: request(id); break;
     case 3: try(id); break;
     case 4: getvalue(id); break;
     case 5: { sem_close(id);exit(0); break; }
     case 6: {sem_unlink(SEMA); exit(0); break; }
   }
 }
return 0;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa...

```
void release(sem_t *id) {
  if (semheld < 1) {
    printf("I don't have any resources; nothing to release.\n");
    return;
  }

  sem_post(id);
  semheld--;

  printf("Resource released.\n");
}

void request(sem_t *id) {
  if (semheld > 0) {
    printf("I already hold the resource; not requesting another
one.\n");
    return;
  }

  printf("Requesting resource...");
  fflush(stdout);

  sem_wait(id);
  semheld++;

  printf(" done.\n");
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin loppu...

```
void try(sem_t *id) {
  if (semheld > 0) {
    printf("I already hold the resource; not requesting another
one.\n"); return;}

  printf("Trying to request resource...");
  fflush(stdout);

  if(sem_trywait(id) == -1) {
        if(errno == EAGAIN)printf("Semaphore was locked\n");
        else perror("Semphore error")}
  else{
        semheld++; printf(" done.\n"); }
}

void getvalue(sem_t *id) {
        int sval;
        if(sem_getvalue(id,&sval) == -1)
                perror("sem_getvalue");
        else {
                if(sval > 0)
                        printf("Semaphore value %d\n",sval);
                else if (sval == 0)
                        printf("Semaphore locked, no processes
                                waiting\n");
                else
                        printf("Semaphore locked, %d processes
                                waiting\n",sval*-1); }
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: sysvsema.c

```
#include"mydbg.h"
#include <semaphore.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <stdio.h>
#include <string.h>

union semun {
  int val;                    /* value for SETVAL */
  struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
  unsigned short int *array;  /* array for GETALL, SETALL */
  struct seminfo *__buf;      /* buffer for IPC_INFO */
};

int semheld = 0;

void release(int id);
void request(int id);

int main(int argc, char *argv[])
{
  int id;
  union semun sunion;
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: sysvsema.c

```
/* No arguments: "server". */
if (argc < 2) {
  /* Request a semaphore. */
  CHECK((id = semget(IPC_PRIVATE,2, SHM_R|SHM_W)) != -1);

  /* Initialize its resource count to 1. */

  memset(&sunion,0,sizeof(sunion));
  sunion.val = 1;
  CHECK((semctl(id, 0, SETVAL, sunion)) != -1);
} else { /* client */
  /* Open up the existing one. */
  id = atoi(argv[1]);
  printf("Using existing semaphore %d.\n", id);
}

printf("Successfully allocated semaphore id %d\n", id);
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: sysvsema.c

```
    while (1) {
      int selection;
      printf("\nStatus: %d resources held by this process.\n",
                    semheld);
      printf("Menu:\n");
      printf("1. Release a resource\n");
      printf("2. Request a resource\n");
      printf("3. Exit this process\n");
      printf("Your choice: ");

      scanf("%d", &selection);

      switch(selection) {
        case 1: release(id); break;
        case 2: request(id); break;
        case 3:
        {
          if ( argc < 2 ) { /* server removes semaphore */
            CHECK((semctl(id,0,IPC_RMID)) != -1);
          }
          exit(0);
          break;
        }
      }
    }
    return 0;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: sysvsema.c

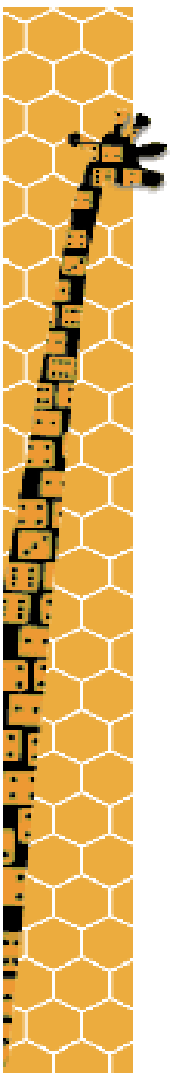
```
void release(int id)
{
  struct sembuf sb;

  if (semheld < 1)
  {
    printf("I don't have any resources; nothing to release.\n");
    return;
  }

  memset(&sb,0,sizeof(sb));
  sb.sem_num = 0;
  sb.sem_op = 1;
  sb.sem_flg = 0;

  CHECK((semop(id, &sb, 1)) != -1);
  semheld--;
  printf("Resource released.\n");
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: sysvsema.c

```
void request(int id)
{
  struct sembuf sb;

  if (semheld > 0)
  {
    printf("I already hold the resource; not requesting another
one.\n");
    return;
  }
  memset(&sb,0,sizeof(sb));
  sb.sem_num = 0;
  sb.sem_op = -1;
  sb.sem_flg = 0;

  printf("Requesting resource...");

  fflush(stdout);

  CHECK((semop(id, &sb, 1)) != -1);

  semheld++;

  printf(" done.\n");
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY



IPC: jaettu muisti

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Jaettu muisti

- Nopein prosessien välinen kommunikointitapa
- Sopii samassa koneessa pyöriviin asiakas-palvelin ratkaisuihin
- Tarvitaan synkronointia ja poissulkemista, eli semaforit, tiedostolukot tai signaalit
- Systeemikutsuja lähinnä vain käsittelyä aloitettaessa, muuten yhteiskäytössä olevan muistin käyttö ei eroa prosessin oman data-alueen käytöstä

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Jaetun muistin systeemikutsut

- System V IPC
  - shmget(): luo tai etsi jaetun muistin lohko
  - shmctl(): kontrolloi jaetun muistin lohkoa
  - shmat(): liitä jaetun muistin lohko prosessiin
  - shmdt(): poista jaetun muistin lohko prosessista
- POSIX IPC
  - shm_open(): luo jaetun muistin lohko
  - shm_unlink(): poista jaetun muistin lohko
  - ftruncate(): aseta koko
  - mmap(): liitä muisti prosessin muistiin
  - munmap(): poista muisti prosessin muistista
- Näistä System V on hieman suorempi tapa, POSIX taasen toimii ikäänkuin muisti olisi tiedosto

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Jaetun muistin lohkon luominen ja hakeminen

- Jaetun muistin lohko luodaan tai haetaan funktiolla:
  int *shmget*(**key_t** key,**size_t** size, **int** flags);
- Palauttaa jaetun muistin avaimen tai -1 virhetilanteessa
- Jaetun muistin lohko luodaan antamalla flags kenttään arvo IPC_CREAT tai IPC_PRIVATE
- Luodun lohkon koko on ylöspäin lähin size-muuttujasta laskettu PAGE_SIZE-muuttujan moninkerta (ylöspäin), ts. muistia varataan lohkoissa, vähintään size-verran, mahdollisesti enemmän.

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Jaetun muistin lohkon kontrollointi

- Jaetun muistin lohkon attribuutteja voi kontrolloida funktiolla:
  int *shmctl*(**int** shmid, **int** cmd, **struct shmid_ds** *data)
- Palauttaa 0 onnistuessaan ja -1 virhetilanteessa.
- *cmd* kentällä samat arvot kuin viestijonoilla: IPC_STAT, IPC_SET, IPC_RMID
- *data*-kenttään tallettuu jaetun muistin tietoja:

```
struct shmid_ds
{
  struct ipc_perm shm_perm; /* operation permission struct */
  size_t shm_segsz;         /* size of segment in bytes */
  __time_t shm_atime;       /* time of last shmat() */
  __time_t shm_dtime;       /* time of last shmdt() */
  __time_t shm_ctime;    /* time of last change by shmctl() */
  __pid_t shm_cpid;         /* pid of creator */
  __pid_t shm_lpid;         /* pid of last shmop */
  shmatt_t shm_nattch;      /* number of current attaches */
};
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: shared.c (katso myös posixshared.c)

```
#include"mydbg.h"

union semun {
  int val;                    /* value for SETVAL */
  struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
  unsigned short int *array;  /* array for GETALL, SETALL */
  struct seminfo *__buf;      /* buffer for IPC_INFO */
};

#define SHMDATASIZE 1000
#define BUFFERSIZE (SHMDATASIZE - sizeof(int))

#define SM_EMPTY 0
#define SM_FULL  1

void server(void);
void client(int shmid);
void delete(void);
void sigdelete(int signum);
void locksem(int semid, int semnum);
void unlocksem(int semid, int semnum);
void clientwrite(int shmid, int semid, char *buffer);

int DeleteSemid = 0;
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Pääohjelma

```
int main(int argc, char *argv[]) {

  /* No arguments: "server". */
  if (argc < 2) {
    server();
  } else {
    client(atoi(argv[1]));
  }
  return 0;
}
```

server funktio:

```
void server(void) {
  union semun sunion;
  int semid, shmid;
  void *shmdata;
  char *buffer;

  /* First thing: generate the semaphore. */
  CHECK((semid = semget(IPC_PRIVATE, 2, SHM_R | SHM_W)) != -1);
  DeleteSemid = semid;

  /* Delete the semaphore when exiting. */
  atexit(&delete);
  CHECK((signal(SIGINT, &sigdelete)) != SIG_ERR);

  /* Initially empty should be available and full should not be.
*/
  memset(&sunion,0,sizeof(sunion));
  sunion.val = 1;
  CHECK((semctl(semid, SM_EMPTY, SETVAL, sunion)) != -1);
  sunion.val = 0;
  CHECK((semctl(semid, SM_FULL, SETVAL, sunion)) != -1);
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

...

```
/* Now allocate a shared memory segment. */
  CHECK((shmid = shmget(IPC_PRIVATE, SHMDATASIZE, IPC_CREAT |
SHM_R | SHM_W)) != -1);
/* Map it into memory. */
  CHECK((shmdata = shmat(shmid, 0, 0)) != NULL);
/* Delete it when the last holding process exits. */
  CHECK((shmctl(shmid, IPC_RMID, NULL)) != -1);
/* Write the semaphore id to its beginning. */
  *(int *)shmdata = semid;
  buffer = (char *)((int *)shmdata + sizeof(int));
  printf("Server is running with SHM id ** %d **\n",shmid);

/*************************************************************
*
   MAIN SERVER LOOP

*************************************************************
/
  while (1) {
    printf("Waiting until full...");
    fflush(stdout);
    locksem(semid, SM_FULL);
    printf(" done.\n");

    printf("Message received: %s\n", buffer);
    unlocksem(semid, SM_EMPTY);
  }
}
```

Asiakas

```
void client(int shmid) {
  int semid;
  void *shmdata;
  char *buffer;

  CHECK((shmdata = shmat(shmid, 0, 0)) != NULL);
  semid = *(int *)shmdata;
  buffer = (char *)((int *)shmdata + sizeof(int));
  printf("Client operational: shm id is %d, sem id is %d\n",
         shmid,semid);

  while (1) {
    char input[3];

    printf("\n\nMenu\n1. Send a message\n");
    printf("2. Exit\n");
    fgets(input, sizeof(input), stdin);

    switch(input[0]) {
      case '1': clientwrite(shmid, semid, buffer); break;
      case '2': { exit(0); break; }
    }
  }
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Siivous

```
void delete(void) {
  printf("\nMaster exiting; deleting semaphore %d.\n",
DeleteSemid);
  CHECK((semctl(DeleteSemid, 0, IPC_RMID, 0)) != -1);
}

void sigdelete(int signum) {
  /* Calling exit will conveniently trigger the normal
     delete item. */
  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Apufunktiot

```
void locksem(int semid, int semnum) {
  struct sembuf sb;

  memset(&sb,0,sizeof(sb));
  sb.sem_num = semnum;
  sb.sem_op = -1;
  sb.sem_flg = SEM_UNDO;

  CHECK((semop(semid, &sb, 1)) != -1);
}

void unlocksem(int semid, int semnum) {
  struct sembuf sb;

  memset(&sb,0,sizeof(sb));
  sb.sem_num = semnum;
  sb.sem_op = 1;
  sb.sem_flg = SEM_UNDO;

  CHECK((semop(semid, &sb, 1)) != -1);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

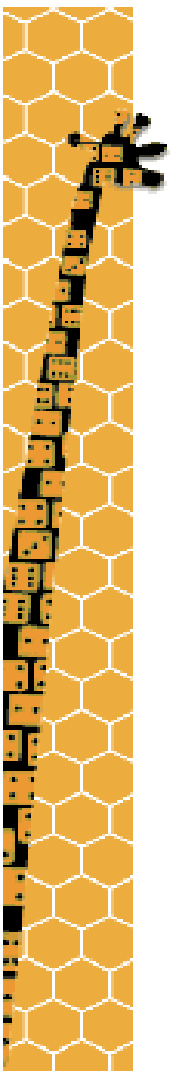
...

```
void clientwrite(int shmid, int semid, char *buffer) {
  printf("Waiting until empty...");
  fflush(stdout);
  locksem(semid, SM_EMPTY);
  printf(" done.\n");

  printf("Enter message: ");
  fgets(buffer, BUFFERSIZE, stdin);
  unlocksem(semid, SM_FULL);
}
```

IPC: Summa summarum

	System V	*POSIX*
Viestijonot	**Käskyt:** msgget, msgsnd, msgrcv,msgctl, ftok **Tunnus:** key_t (globaali), int (paikallinen)	**Käskyt:** mq_open, mq_close, mq_send, mq_receive **Tunnus:** char * (globaali), mqd_t (paikallinen)
Semaforit	**Käskyt:** semget, semctl, semop, ftok **Tunnus:** key_t (globaali), int (paikallinen)	**Käskyt:** sem_open, sem_close, sem_init, sem_destroy, sem_wait, sem_post, sem_getvalue, sem_trywait, sem_timedwait, **Tunnus:** char * (globaali), sem_t (paikallinen)
Jaettu muisti	**Käskyt**: shmget, shmctl, shmat, shmdt, ftok **Tunnus:** key_t (globaali), int (paikallinen)	**Käskyt:** shm_open, shm_unlink, ftruncate, mmap, munmap **Tunnus:** char * (globaali), int (paikallinen)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY



Säikeet: perusteet

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Säikeiden hyödyt

- Säikeiden avulla on mahdollista hyödyntää ohjelmassa rinnakkaisuutta moniajojärjestelmissä
- Mahdollista hyödyntää ohjelman normaalia rinnakkaisuutta tehokkaammin sallimalla ohjelman toiminnan jatkumisen muualla samalla, kun esimerkiksi odotetaan hidasta levyoperaatiota
- Modulaarinen ohjelmointimalli, joka selkästi kuvaa itsenäisten 'tapahtumien' yhteyttä ohjelmaan

Säikeiden haitat

- Säikeiden vaihtamiseen ja säikeiden väliseen synkronointiin kuluu aikaa
- Ohjelmointi monimutkaisempaa:
  - Kaikki kirjastofunktiot eivät ole säieturvallisia
  - Säikeet jakavat saman muistiavaruuden, joka ei ole suojattu virheelliseltä toiminnalta
  - Hyvä peräkkäisratkaisu voi olla ratkaisu rinnakkaiseen laskentaan
- Virheenjäljitys vaikeata:
  - Virheiden jäljittäminen vaikeutuu, koska virheenjäljitys muuttaa ajoituksia
  - Ajoitusongelmista johtuvat virheet hyvin vaikeita löytää

Säieturvalliset kirjastot

- Funktioiden tulee olla ns. *re-entrant* ja *säieturvallisia*
- *Re-entrant* funktio ei talleta staattista tietoa, joka muuttuu joka kutsukerralla
- Vaarallisia funktioita ovat mm. *strtok* ja *ctime*
- Vertaa esim.
  - *ctime* ja *ctime_r*
  - *asctime* ja *asctime_r*
  - *gmtime* ja *gmtime_r*
  - *localtime* ja *localtime_r*
- Jos manuaalisivu puhuu “statically allocated memory”, niin on syytä varoa
- Samoin, jos funktio palauttaa muistiosoitteen (jota kutsuja ei siis ole itse antanut ja alustanut)

Säieturvalliset kirjastot

- *Säieturvallinen* funktio on toteutettu siten, että globaalien muuttujien käsittely on synkronoitu
- Esim. *errno* arvon käsittely voi olla turvatonta (jos *errno* on KJ:ssa yhteinen kaikille säikeille)
- Jonkinlaista listaa mahd. vaarallisista funktioista saa hakemalla esim.
  - Google: “reentrant site:opengroup.org”
  - http://www.opengroup.org/cgi-bin/susv3search.pl ja hakusana “reentrant”
- Yksi tapa korjata olemassaolevan funktion säieturvallisuus on luoda itse semafori vaarallisten funktioiden ympärille
- Säieturvallisuus on tärkeää myös omissa signaalinkäsittelijöissä

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Milloin tulisi käyttää säikeitä ?

- Ohjelma suorittaa massiivista laskentaa, joka voidaan rinnakkaistaa tai hajottaa useampaan säikeeseen ja ohjelmaa on tarkoitus ajaa moniprosessorikoneella
- Ohjelma suorittaa hyvin paljon levyoperaatioita, joita voi limittää tehokkuuden lisäämiseksi: Useampi säie voi odottaa eri levyoperaatioita samanaikaisesti.
- Hajautetut palvelimet ovat hyvia kandidaatteja, koska niissä pitää varautua useisiin eri nopeuksilla toimiviin asiakkaisiin kohtuullisen hitaan verkkoliikenteen yli

Säikeiden ominaisuuksia

- Säikeella omaa:
  - Suorituksen "lanka" eli kutsupino
  - Kutsujen parametrit
  - Kutsujen paikalliset muuttujat
  - Errno (pitäisi olla nykyisin kaikkialla)
  - Signaalipeite
- Säikeillä yhteistä
  - Ohjelman argc, argv ympäristö
  - Globaalit muuttujat
  - Tiedostokuvaajat
  - Fcntl-lukot
  - Resurssirajat
  - Pääsyoikeudet

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Prosessit vs Säikeet

- Koska säikeet jakavat muistiavaruuden, voivat säikeet kommunikoida suoraan. Tämä kommunikointi vaatii luonnollisesti synkronointia.
- Prosesseilla oma muistiavaruus ja siksi kommunikointiin tarvitaan jokin IPC menetelmistä ja synkronointia
- Prosessien vaihtaminen on raskaampaa kuin säikeiden
- Säikeitä kannattaa käyttää, jos laskennassa on hyvin paljon yhteisiä osia (esim. jokainen laskennan osa tarvitsee samaa muistia)
- Säikeiden toteutus vaihtelee alustojen välillä, esim. ovatko säikeet omia prosessejaan

Systeemikutsut (POSIX pthread kirjasto)

#include <pthread.h>

- **int** *pthread_create*(**pthread_t** * thread, **pthread_attr_t** * attr, **void** * (**start_routine*)(**void** *), **void** * arg)
- **void** *pthread_exit*(**void** *retval)
- **int** *pthread_join*(**pthread_t** th, **void** **thread_return)
- **int** *pthread_detach*(**pthread_t** th)
- **int** *pthread_equal*(**pthread_t** thread1, **pthread_t** thread2)
- **pthread_t** *pthread_self*(**void**)
- **int** *pthread_cancel*(**pthread_t** th)
- **void** *pthread_cleanup_push*(**void** (*routine) (void *), **void** *arg)
- **void** *pthread_cleanup_pop*(**int** execute)
- **int** *sched_yield*(**void**)

- Funktiot ei virhetilanteessa aseta *errno* arvoa
- Tyyppi *pthread_t* on alustariippuvainen

Säikeen luominen

- Uusi säie luodaan funktiolla:
  int *pthread_create*(**pthread_t** * thread, **pthread_attr_t** * attr, **void** * (**start_routine*)(**void** *), **void** * arg)
  - *thread* parametriin talletetaan säikeen tunnus
  - *attr* parametrissa voi muokata säikeen ominaisuuksia (pitää rakentaa eri funktioilla)
  - *start_routine* on funktio, josta säikeen suoritus alkaa ja sen prototyyppi tulee olla muotoa: **void** **f*(**void** *)
  - *arg* parametri välitetään funktiolle start_routine parametriksi
  - Palauttaa 0 onnistuessaan, virhetilanteessa palautetaan virhenumero

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Säikeen suorituksen päättyminen / päättymisen odotus

- Säikeen suoritus päättyy joko:
  - return lauseeseen funktiossa, josta suoritus alkoi,
  - kutsumalla funktiota *pthread_exit*
  - Kutsumalla funktiota *pthread_cancel*
  - Kutsumalla funktiota *pthread_cleanup_pop(1)*
- Jos säie kutsuu exit-funktiota koko prosessi päättyy
- Säikeen päättymistä voi odottaa funktiolla:**int** *pthread_join*(**pthread_t** th, **void** **thread_return)
  - Onnistuessaan funktio palauttaa 0 ja thread_return kentässä on säikeen palauttama arvo. Muuten funktio palauttaa 0:sta poikkeavan arvon.
  - Funktion yhteydessä vapautetaan säikeen varaamat resurssit.

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki (thread_simple.c):

```
#include"mydbg.h"

void *thread_routine(void *arg) {
  printf(“Olen säie. Moi, moi...\n”);
  printf("void *arg oli: %s\n",(char *)arg);
  pthread_exit(arg);
}

int main(void) {
  pthread_t thread_id;
  void *thread_result;

  CHECK((pthread_create(&thread_id,NULL,thread_routine,
        (void *)"kukkuu")) == 0);

  CHECK((pthread_join(thread_id,&thread_result)) == 0);

  if ( thread_result == NULL)
    printf("Error on threads\n");
  else
    printf("Results was %s\n",(char *)thread_result);

  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Säikeen irroitus emostaan

- Säikeen voi irroittaa erilleen funktiolla:
  int *pthread_detach*(**pthread_t** th)
- Kutsulla voi varmistua että säikeen varaamat resurssit vapautetaan heti säikeen päätyttyä (eikä vasta prosessin päätyttyä!)
- Kutsun jälkeen ei ole mahdollista odottaa säikeen päättymistä *pthread_join*() kutsulla
- Jos ennen kutsun suoritusta jokin säie on jo kutsunut *pthread_join()* funktiota kyseiselle säikeelle, *pthread_detach*() ei tee mitään
- Kutsusta on hyötyä erityisesti silloin, kun halutaan ettei säikeet palauta päättyessään arvoa ja siten tilatieto jää turhaan talteen

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Säietunnusten vertailu

- POSIX määrittelee vain säikeiden tunnuksen tyypin nimen, ei sen rakennetta
- Eri alustoilla tyyppi vaihtelee: ulong, uint, osoitin
- Kahta säietunnusta voidaan verrata toisiinsa funktiolla:

  int *pthread_equal*(**pthread_t** thread1, **pthread_t** thread2)
  - Palauttaa 0 jos tunnukset viittaavat eri säikeisiin
  - Palauttaa 0:sta poikkeavan arvon, jos tunnukset viittaavat samaan säikeeseen
- Säikeen oman tunnuksen voi selvittää funktiolla:

  pthread_t *pthread_self*(**void**)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Säikeen exit-funktiot

- Säie voi määritellä funktioita, joita kutsutaan, kun ollaan poistumassa:
  - *pthread_exit* kutsun johdosta tai
  - *pthread_cancel* kutsun johdosta
- Funktiot talletaan pinoon kutsulla:
  void *pthread_cleanup_push*(void (*routine) (void *), void *arg)
- Kutsuja voi poistaa funktiolla:
  void *pthread_cleanup_pop*(int execute)
  - Jos parametri on 0, funktiokutsu poistetaan pinosta, mutta sitä ei kutsuja
  - Muilla parametreilla funktio myös suoritetaan
- *Push*- ja *pop*-kutsuja pitää asettaa pareittain

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

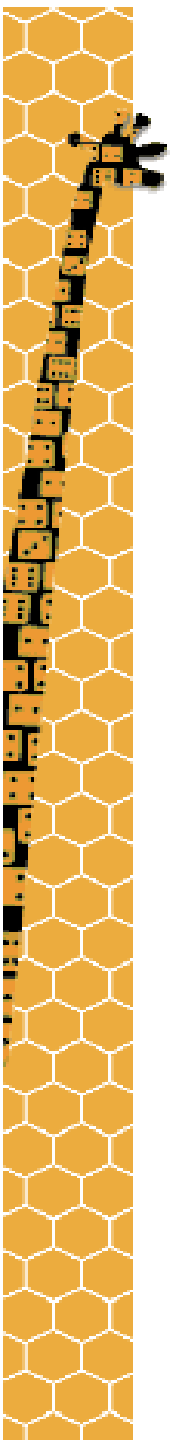
Säiekeen lopettaminen muualta

- Säikeen voi pyytää päättymään funktiolla:
  int *pthread_cancel*(**pthread_t** th)
- Tämä on siis vain pyyntö
- Kutsu palaa heti
- Säie voi itse hallita omaa lopettamistaan, esim. säikeellä voi olla useampi exit-funktio allokoituna

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Säikeen palautusarvot

- Säie voi palauttaa lähes mitä tahansa *pthread_exit* funktion avulla
- Funktio palauttaa osoittimen muistialueeseen, josta säikeen käynnistänyt prosessi voi sitten lukea palautusarvon tms.
- Tämän muistialueen allokointi pitää tehdä oikein
- Huomaa, että tavalliset muuttujat talletetaan säikeen pinoon, mikä häviää säikeestä poistuttaessa
- Jos haluat, että paluuarvo jää oikeasti kutsujalle, muistialue, johon paluutieto kirjoitetaan pitää
  - Alustaa malloc-funktiolla säikeessä
  - Alustaa malloc-funktiolla kutsujassa
  - Olla globaali muuttuja
- Katso esimerkki: thread_return.c



Säikeiden synkronointi

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Säikeiden synkronoinnin välineet

- Mutex-lukot
- Ehtomuuttujat
- Luku/kirjoituslukot
- Aidat

Mutexit

- Mutex (Mutual Exclusion) on tarkoitettu säikeiden väliseen synkronointiin ja poissulkemiseen
- Mutex:n avulla suojataan kriittinen alue, esim.
  - Laskuri, jota säikeet voivat kasvattaa ja kysyä
  - Funktioiden suojaus
  - Tietorakenteiden suojaaminen, esim. linkattu lista useiden säikeiden käyttöön
- Samantyyppisiä kuin semaforit

Mutex systeemikutsut

- **pthread_mutex_t** mutex = PTHREAD_MUTEX_INITIALIZER;
- **int** *pthread_mutex_init*(**pthread_mutex_t** *mutex, **pthread_mutexattr_t** *attr)
- **int** *pthread_mutex_destroy*(**pthread_mutex_t** *mutex)
- **int** *pthread_mutex_lock*(**pthread_mutex_t** *mutex)
- **int** *pthread_mutex_trylock*(**pthread_mutex_t** *mutex)
- **int** *pthread_mutex_unlock*(**pthread_mutex_t** *mutex)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki (thread_array.c)

```
#include"mydbg.h"

#define MAXTHREADS 10

static unsigned long global_seed = 1751;

typedef struct {
  pthread_mutex_t       mutex;
  int id;
  unsigned long         array[MAXTHREADS+1];
} my_array_t;

my_array_t *thread_array;
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa: funktio thread_routine

```
void *thread_routine(void *arg) {
  register unsigned long tmp_seed;
  int id;

  printf("Thread %lu in thread_routine, request mutex\n", pthread_self());
  fflush(stdout);
  CHECK((pthread_mutex_lock(&thread_array->mutex)) == 0);
  printf("Thread %lu got mutex\n", pthread_self());
  fflush(stdout);

  tmp_seed = 16807 * ( global_seed % 127773) - 2836 *
                     ( global_seed / 127773);

  if ( tmp_seed > 0)    global_seed = tmp_seed;
  else                  global_seed = tmp_seed + 2147483647;

  CHECK((printf("Thread %lu storing random value %lu to array pos %d\n",
                pthread_self(),global_seed,id)) > 0);
  fflush(stdout);
  id = thread_array->id++;
  thread_array->array[id] = global_seed;
  thread_array->array[MAXTHREADS] += global_seed;

  CHECK((pthread_mutex_unlock(&thread_array->mutex)) == 0);
  printf("Thread %lu unlock mutex\n", pthread_self());
  fflush(stdout);

  pthread_exit(arg);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa: funktio main

```
int main(void) {
  pthread_t thread_id[MAXTHREADS];
  void *thread_result;
  int i;

  CHECK((thread_array = (my_array_t *)malloc(sizeof(my_array_t))) !=
    NULL);
  memset(thread_array,0,sizeof(my_array_t));
  CHECK((pthread_mutex_init(&thread_array->mutex,NULL)) == 0);

  for(i=0; i < MAXTHREADS;i++) {
        int id = i;
        printf("Mama ready to create thread\n");
        fflush(stdout);

        CHECK((pthread_create(&(thread_id[i]),NULL,
                              thread_routine,(void *)&id)) == 0);
        printf("Mama created thread %lu\n", (thread_id[i]));
        fflush(stdout);
  }
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa: main loppuu...

```
    for(i=0; i < MAXTHREADS;i++) {
          printf("Mama waiting for thread %lu to end\n",
                    thread_id[i]);
          fflush(stdout);
          CHECK((pthread_join(thread_id[i],&thread_result)) == 0);
    }

    CHECK((printf("Sum of random variables was %lu\n",
                  thread_array->array[MAXTHREADS])) > 0);
    fflush(stdout);
    CHECK((pthread_mutex_destroy(&thread_array->mutex)) == 0);
    free(thread_array);

    exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Ehtomuuttujat

- Ehtomuuttujia voidaan käyttää jonkin ehdon täyttymisen odottamiseen ”busy-loopin” sijasta
- Ehtomuuttujia käytetään yleensä aina mutexin kanssa yhdessä
- Tärkeimmät ehtomuuttujien käsittelyfunktiot:
  - **pthread_cond_t** my_cond = PTHREAD_COND_INITIALIZER;
  - **int** *pthread_cond_init*(**pthread_cond_t** *cond,**pthread_conattr_t** *attr);
  - **int** *pthread_cond_destroy*(**pthread_cond_t** *cond);
  - **int** *pthread_cond_wait*(**pthread_cond_t** *cond,**pthread_mutex_t** *mutex);
  - **int** *pthread_cond_signal*(**pthread_cond_t** *cond);

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Ehtomuuttujan idea

- Funktion *pthread_cond_wait()* perusidea on siinä, että funktio avaa säikeen hallussa olleen mutex-lukon ja jää odotustilaan atomisesti
- Odotustila laukeaa, kun odotettava ehtomuuttuja laitetaan signaloituun tilaan funktiolla *pthread_cond_signal()* (jostakin toisesta säikeestä)
- Samalla kun funktio *pthread_cond_wait()* palaa, säie saa taas haltuunsa mutex-lukon
- *pthread_cond_signal*() kutsu herättää yhden säikeen
- Kaikki odottavat säikeet voi herättää funktiolla: **int** *pthread_cond_broadcast(***pthread_cond_t** *cond)

Esimerkki (thread_cond_array.c)

```
#include"mydbg.h"

#define MAXTHREADS 10

static unsigned long global_seed = 0;

typedef struct {
  pthread_mutex_t       mutex;
  pthread_cond_t        cond;
  pthread_t             thd_id;
  int                   id;
  unsigned long         array[MAXTHREADS+1];
} my_array_t;

my_array_t *thread_array;
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa: funktio thread_routine

```
void *thread_routine(void *arg) {
  register unsigned long tmp_seed;
  int id,res;

  printf("Thread %lu in thread_routine, request mutex\n",
         pthread_self());
  fflush(stdout);
  CHECK((pthread_mutex_lock(&thread_array->mutex)) == 0);
  printf("Thread %lu got mutex\n",pthread_self());
  fflush(stdout);

  while( pthread_equal(thread_array->thd_id, pthread_self()) == 0 ) {
    printf("Thread %lu waiting for its turn, turn = %lu\n",
           pthread_self(), thread_array->thd_id);
    fflush(stdout);

    res = pthread_cond_wait(&thread_array->cond,
                  &thread_array->mutex);
    printf("Thread %lu waken\n", pthread_self());
         fflush(stdout);
  }
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa: thread_routine loppuosa

```
  tmp_seed = 16807 * ( global_seed % 127773) - 2836 *
                     ( global_seed / 127773);

  if ( tmp_seed > 0)    global_seed = tmp_seed;
  else                  global_seed = tmp_seed + 2147483647;

  id = thread_array->id;
  CHECK((printf("Thread %lu storing random value %lu to array pos %d\n",
                pthread_self(),global_seed,id)) > 0);
  fflush(stdout);
  thread_array->array[id] = global_seed;
  thread_array->array[MAXTHREADS] += global_seed;

  CHECK((pthread_mutex_unlock(&thread_array->mutex)) == 0);
  printf("Thread %lu unlock mutex\n", pthread_self());
  fflush(stdout);

  pthread_exit(arg);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa: main funktio

```
int main(void) {
  pthread_t thread_id[MAXTHREADS];
  void *thread_result;
  int i;

  global_seed = (unsigned long)time(NULL);
  CHECK((thread_array = (my_array_t *)malloc(sizeof(my_array_t)))
!= NULL);
  memset(thread_array,0,sizeof(my_array_t));
  CHECK((pthread_mutex_init(&thread_array->mutex,NULL)) == 0);
  CHECK((pthread_cond_init(&thread_array->cond,NULL)) == 0);

  for(i=0; i < MAXTHREADS;i++) {
        CHECK((pthread_create(&(thread_id[i]),NULL,
                              thread_routine,NULL)) == 0);
        printf("Mama created thread %lu\n", (thread_id[i]));
        fflush(stdout);
  }
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa: main loppuosa

```
for(i=MAXTHREADS-1;i > -1;i--) {
        printf("Mama setting array id %d\n",i);
        fflush(stdout);
        CHECK((pthread_mutex_lock(&thread_array->mutex)) == 0);
        thread_array->id = i;
        thread_array->thd_id = thread_id[i];
        printf("Mama assigned work to thread %lu\n", thread_id[i]);
        fflush(stdout);
        CHECK((pthread_cond_broadcast(&thread_array->cond)) == 0);
        CHECK((pthread_mutex_unlock(&thread_array->mutex)) == 0);
        printf("Mama waiting for thread %lu to end\n",thread_id[i]);
        fflush(stdout);
        CHECK((pthread_join(thread_id[i],&thread_result)) == 0);
  }

  CHECK((printf("Sum of random variables was %lu\n",
                thread_array->array[MAXTHREADS])) > 0);
  fflush(stdout);
  CHECK((pthread_cond_destroy(&thread->array->cond)) == 0);
  CHECK((pthread_mutex_destroy(&thread_array->mutex)) == 0);
  free(thread_array);

  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Luku/kirjoituslukot (read/write locks)

- Erottelee tehdäänkö jaettuun dataan lukuoperaatio vai kirjoitusoperaatio:
  - Sallivat useamman prosessin rinnakkaisesti lukea jaettua dataa
  - Kirjoittamaan pääsee vain yksi säie kerrallaan, muut lukijat ja kirjoittajat odottavat vuoroaan
- Pyydettäessa kirjoituslukkoa jää pyytäjä odottamaan vuoroaan, esim. lukijoiden tai kirjoittajan poistumista
- Kirjoittajien ja lukijoiden vuorottelu riippuu käyttö-järjestelmästä, esim. jos dataa on lukemassa n säiettä ja säie X pyytää kirjoitus-lupaa, pääseekö X kirjoittamaan n:n lopetettua, vai voiko paikalle tulla uusia lukijoita, jotka ohittavat X:n

Tärkeimmät systeemikutsut

- **int** *pthread_rwlock_init*(**pthread_rwlock_t *restrict** rwlock, **const pthread_rwlockattr_t *restrict** attr)
- **int** *pthread_rwlock_destroy*(**pthread_rwlock_t** *rwlock)
- **int** *pthread_rwlock_rdlock*(**pthread_rwlock_t** *rwlock)
- **int** *pthread_rwlock_tryrdlock*(**pthread_rwlock_t** *rwlock)
- **int** *pthread_rwlock_wrlock*(**pthread_rwlock_t** *rwlock)
- **int** *pthread_rwlock_trywrlock*(**pthread_rwlock_t** *rwlock)
- **int** *pthread_rwlock_unlock*(**pthread_rwlock_t** *rwlock)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: thread_rwlock_array.c

```
#include"mydbg.h"

#define MAXTHREADS 10

typedef struct {
  pthread_rwlock_t      rwlock;
  unsigned long int     sum;
  unsigned long         array[MAXTHREADS];
  int                   readers;
  int                   writers;
} my_array_t;

my_array_t *thread_array;

void arr_fill(void);
void arr_read(void);
void arr_write(void);
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: funktio arr_fill ja thread_routine

```
/* Fill up the array with random values */
void arr_fill(void) {
  int i;

  for(i=0;i < MAXTHREADS;i++) {
        thread_array->array[i] = random();
  }
}
void *thread_routine(void *arg) {
  int i;
  printf("Thread %lu in thread_routine, request mutex\n",
                pthread_self());
  fflush(stdout);

  for(i=0;i < 1000;i++) {
    if ( random() % 2) arr_read();
    else        arr_write();

   sched_yield(); /* Give up the prosessor */
  }

  pthread_exit(arg);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: funktio read()

```
/* Read the array and print information to stdout */
void arr_read(void) {
  int i;

  printf("Thread %lu getting the read lock to array\n",
                 pthread_self());
  fflush(stdout);
  CHECK((pthread_rwlock_rdlock(&thread_array->rwlock)) == 0);
  printf("Thread %lu joining on readers\n", pthread_self());
  fflush(stdout);
  thread_array->readers++;
  CHECK(thread_array->writers == 0);
  printf("Now %d readers %d writers\n",thread_array->readers,
         thread_array->writers);
  fflush(stdout);

  for(i = 0; i < MAXTHREADS; i++) {
    printf("Value of array[%d] = %lu\n",i,thread_array->array[i]);
    fflush(stdout);
    sched_yield(); /* Give up prosessor */
  }

  printf("Sum of array is %lu\n",thread_array->sum);
  fflush(stdout);
  thread_array->readers--;
  CHECK((pthread_rwlock_unlock(&thread_array->rwlock)) == 0);
  printf("Thread %lu exits from read\n", pthread_self());
  fflush(stdout);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: funktio write()

```
void arr_write(void) {
  int i;

  printf("Thread %lu getting the write lock to array\n",
         pthread_self());
  fflush(stdout);
  CHECK((pthread_rwlock_wrlock(&thread_array->rwlock)) == 0);
  printf("Thread %lu joining on writers\n", pthread_self());
  fflush(stdout);
  thread_array->writers++;
  CHECK(thread_array->writers == 1);
  CHECK(thread_array->readers == 0);
  printf("Now %d writers %d readers\n",thread_array->writers,
         thread_array->readers);
  fflush(stdout);

  for(i = 0; i < MAXTHREADS; i++) {
    printf("Value of array[%d] = %lu\n",i,thread_array->array[i]);
    thread_array->array[i]++;
    thread_array->sum+=thread_array->array[i];
    fflush(stdout);
    sched_yield(); /* Give up prosessor */
  }
  printf("Sum of array is now %lu\n",thread_array->sum);
  thread_array->writers--;
  CHECK((pthread_rwlock_unlock(&thread_array->rwlock)) == 0);
  printf("Thread %lu exits from write\n", pthread_self());
  fflush(stdout);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: funktio main()

```
int main(void) {
  pthread_t thread_id[MAXTHREADS];
  void *thread_result;
  int i;

  CHECK((thread_array = (my_array_t *)malloc(sizeof(my_array_t))) !=
NULL);
  memset(thread_array,0,sizeof(my_array_t));
  arr_fill();
  CHECK((pthread_rwlock_init(&thread_array->rwlock,NULL)) == 0);

  for(i=0; i < MAXTHREADS;i++) {
        CHECK((pthread_create(&(thread_id[i]),NULL,
                              thread_routine,NULL)) == 0);
        printf("Mama created thread %lu\n", (thread_id[i]));
        fflush(stdout);
  }

  for(i=0; i < MAXTHREADS;i++) {
        printf("Mama waiting for thread %lu to end\n", thread_id[i]);
        fflush(stdout);
        CHECK((pthread_join(thread_id[i],&thread_result)) == 0);
  }
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: main loppu

```
  CHECK((printf("Sum of random variables was %lu\n",
                thread_array->sum)) > 0);
  fflush(stdout);
  CHECK((pthread_rwlock_destroy(&thread_array->rwlock)) == 0);
  free(thread_array);

  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Aidat (barriers)

- Aidan avulla on mahdollista synkronoida useita säikeitä tekemään ensin jokin tietty operaatio ja vasta kun kaikki ovat valmiina, suoritetaan jokin toinen operaatio, jne.
- Operaatio voi olla mikä tahansa ohjelman osa, esimerkiksi funktio
- Esimerkki: halutaan että kaikki säikeet suorittavat ensin fuktion a() ja kun kaikki ovat sen suorittaneet alkaa funktion b() suoritus

Tärkeimmät systeemikutsut

- **int** *pthread_barrier_destroy*(**pthread_barrier_t** *barrier)
- **int** *pthread_barrier_init*(**pthread_barrier_t** ***restrict** barrier, **const pthread_barrierattr_t** ***restrict** attr, unsigned count)
- **int** *pthread_barrier_wait*(**pthread_barrier_t** *barrier)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Aidan alustaminen

- Aita alustetaan funktiolla:
  int *pthread_barrier_init*(**pthread_barrier_t** ***restrict** barrier, **const pthread_barrierattr_t** ***restrict** attr, **unsigned** count)
  - count parametrillä kerrotaan kuinka monta säiettä pitää kutsun *pthread_barrier_wait()* tehdä ennenkuin yksikään pääsee jatkamaan aidasta eteenpäin

Aidassa odottaminen

- Odotus tapahtuu kutsumalla funktiota:
  int *pthread_barrier_wait*(**pthread_barrier_t** *barrier)
- Kun riittävä määrä säikeitä on kutsunut *pthread_barrier_wait*() funktiota:
  - Yksi säie (määrittelemätön) saa paluuarvonaan PTHREAD_BARRIER_SERIAL_THREAD
  - Muut saavat paluuarvon 0
  - Virhetilanteessa palautuu 0:sta poikkeava arvo

Esimerkki: thread_barrier.c

```
#include"mydbg.h"

#define MAXTHREADS 10

pthread_barrier_t my_barrier;

void* b(void *arg) {
  printf("Thread %lu executing function b\n", pthread_self());
  fflush(stdout);
  pthread_exit(arg);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: funktio a()

```
void *a(void *arg) {
  int res;
  printf("Thread %lu executing function a\n", pthread_self());
  fflush(stdout);
  res = pthread_barrier_wait(&my_barrier);

  if ( res != 0) {
    if ( res == PTHREAD_BARRIER_SERIAL_THREAD ) {
      printf("Thread %lu got PTHREAD_BARRIER_SERIAL_THREAD\n",
             pthread_self());
      fflush(stdout);
    }
    else {
      perror(“pthread_barrier_wait”);
      exit(-1);
    }
  }

  printf("Thread %lu has executed wait on barrier\n",
         pthread_self());
  fflush(stdout);
  b(arg);
  return(arg); /* not reached */
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki: funktio main()

```
int main(void) {
  pthread_t thread_id[MAXTHREADS] = {0};
  int i;
  void *res;

  CHECK((pthread_barrier_init(&my_barrier,NULL,MAXTHREADS)) == 0);
  printf("Mama created barrier\n");
  fflush(stdout);

  for(i=0;i < MAXTHREADS; i++) {
    CHECK((pthread_create(&(thread_id[i]),NULL,a,NULL)) == 0);
    printf("Mama created thread %lu\n", thread_id[i]);
    fflush(stdout);
  }

  for(i=0;i < MAXTHREADS; i++) {
    printf("Mama waits for thread %lu to end\n", thread_id[i]);
    fflush(stdout);
    CHECK((pthread_join(thread_id[i],&res)) == 0);
    printf("Thread %lu ended\n", thread_id[i]);
    fflush(stdout);
  }

  CHECK((pthread_barrier_destroy(&my_barrier)) == 0);
  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

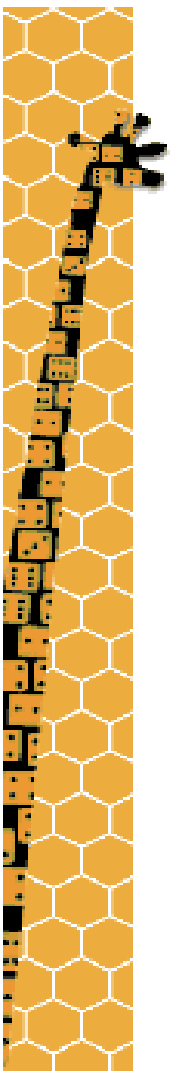
Muuta hyödyllistä

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Sisältö

- POSIX apufunktiot
  - Listat
  - Taulukot
  - Hajautustaulu
  - Binääripuut
  - Satunnaisluvut
- Realiaikaohjelmointi
  - Muistiinlukitus
  - Skedulointi
- Vielä kelloista ja ajastimista

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY



POSIX apufunktiot

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Listojen käsittely

- #include<search.h>
- Tietorakenteena käytetään:

```
struct qelem {
        struct  qelem *q_forw;
        struct  qelem *q_back;
        char    q_data[1];
};
```

- Uusi alkio lisätään listaan funktiolla:
  void *insque*(**void** *elem, **void** *prev)
- Alkio sijoittuu alkion prev jälkeen listaan
- Alkio poistetaan listasta funktiolla:
  void *remque*(**void** *elem)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki (list.c):

```
#include"mydbg.h"
#include<search.h>

typedef struct list {
  struct list *next;
  struct list *prev;
  char mjono[1];
} my_list_t;

int main(void) {
  my_list_t *my_head,*my_tail,*my_item;
  char *input;
  char buf[256+1];
  unsigned long isize;
  my_head = my_tail = my_item = NULL;
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa:

```
while((input = fgets(buf,256,stdin))) {
  isize = strlen(input) + 1;
  CHECK((my_item = (my_list_t *)calloc(1,
       sizeof(struct list *) * 2 + isize)));
  strcpy(my_item->mjono,input);

  if ( my_tail == NULL ) {
    my_head = my_tail = my_item;
  } else {
    insque(my_item,my_tail);
    my_tail = my_item;
  }
}

my_item = my_head;
isize = 0;
while(my_item) {
  printf("Item %lu = %s\n",isize++,my_item->mjono);
  my_head = my_item;
  my_item = my_head->next;
  remque(my_head);
  free(my_head);
}

return(0);
}
```

Taulukosta etsiminen (lineaarinen)

- Taulukosta avainta voi etsiä funktiolla:
  void **lfind*(**const void** *key, **const void** *base, **size_t** *nmemb, **size_t** size, **int**(**compar*)(**const void** *, **const void** *))
  - key parametrissa annetaan etsittävän alkion avain.
  - base parametrissa annetaan osoitin taulukkoon.
  - nmemb parametri kertoo taulukon alkioiden lukumäärän.
  - size parametri kertoo yhden alkion koon.
  - compar parametri on osoitin vertailufunktioon, jossa ensimmäinen parametri on osoitin alkion avain osaan ja toinen parametri taulukon alkioon.
- Palauttaa NULL jos etsittävää alkiota ei löydy taulukosta.

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Taulukkoon lisääminen (lineaarinen)

- Taulukkoon voi lisätä uusia alkioita funktiolla: **void** **lsearch*(**const void** *key, **void** *base, **size_t** *nmemb, **size_t** size, **int**(**compar*)(**const void** *, **const void** *));
  - key parametrissä annetaan lisättävä avain. Lisäys tehdään vain jos avainta ei löydy taulukosta.
  - base on osoitin taulukkoon.
  - nmemb on taulukon alkioiden lukumäärä
  - size on taulukon alkion koko
  - compar on käytettävä vertailufunktio.
- Palauttaa osoittimen löydettyyn alkioon tai osoittimen lisättyyn alkioon jos ei löytynyt.

Taulukon järjestäminen

- **void** *qsort*(**void** *base, **size_t** nmemb, **size_t** size, **int**(**compar*)(**const void** *, **const void** *));
  - base on osoitin taulukkoon
  - nmemb on alkioiden lukumäärä taulukossa
  - size on yhden alkion koko
  - compar on vertailufunktio, joka saa parametrikseen ensin avaimen ja sitten taulukon alkion
    - palauttaa 0 jos avaimet ovat samat ,
    - < 0 jos avain on pienempi kuin alkio
    - > 1 jos avain suurempi kuin alkio

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Binäärihaku lajitellusta taulukosta

- **void** **bsearch*(**const void** *key, **const void** *base, **size_t** nmemb, **size_t** size, **int** (**compar*)(**const void** *, **const void** *));
  - key on osoitin avaimeen
  - base on osoitin taulukkoon
  - nmemb on alkioiden lukumäärä taulukossa
  - size on yhden alkion koko
  - compar on vertailufunktio, joka saa parametrikseen ensin avaimen ja sitten taulukon alkion
    - palauttaa 0 jos avaimet ovat samat ,
    - < 0 jos avain on pienempi kuin alkio
    - > 1 jos avain suurempi kuin alkio

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki (table.c):

```
typedef struct {
  int nr;
  char *name;
} my_month_t;

#define MONTHS 12

my_month_t months[MONTHS];

static int comp(const void *m1, const void *m2) {
  my_month_t *mi1 = (my_month_t *) m1;
  my_month_t *mi2 = (my_month_t *) m2;

  if ( mi1->nr < mi2->nr) return -1;
  else if ( mi1->nr > mi2->nr) return 1;
  else return 0;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa...funktio fill()

```
void fill(void) {
  my_month_t m;
  size_t nmemb=0;
  int j;

  for(j = 12; j >= 1; j--) {
    m.nr = j;
    CHECK((m.name = (char *) malloc(256)));

    switch (j) {
        case 1: strcpy(m.name,"Tammikuu"); break;
        case 2: strcpy(m.name,"Helmikuu"); break;
        case 3: strcpy(m.name,"Maaliskuu"); break;
        case 4: strcpy(m.name,"Huhtikuu"); break;
        case 5: strcpy(m.name,"Toukokuu"); break;
        case 6: strcpy(m.name,"Kesakuu"); break;
        case 7: strcpy(m.name,"Heinakuu"); break;
        case 8: strcpy(m.name,"Elokuu"); break;
        case 9: strcpy(m.name,"Syyskuu"); break;
        case 10: strcpy(m.name,"Lokakuu"); break;
        case 11: strcpy(m.name,"Marraskuu"); break;
        case 12: strcpy(m.name,"Joulukuu"); break;
    }
    lsearch((void *)&(m.nr), (void *) &months, &nmemb,
          sizeof(my_month_t), comp);
  }
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa...main()

```
int main(int argc, char **argv) {
  int i;

  fill();

  qsort(months, MONTHS, sizeof(my_month_t), comp);

  for (i = 1; i < argc; i++) {
    my_month_t key, *res;
    key.nr = atoi(argv[i]);

    CHECK((res = bsearch(&key, months, MONTHS,
                         sizeof(my_month_t), comp)));

    printf("%d. kuukausi on %s\n", res->nr, res->name);
  }
  return 0;
}
```

Hajautustaulujen systeemikutsut ja tietorakenteet

- **int** *hcreate*(*size_t* nel)
- **ENTRY** **hsearch*(**ENTRY** item, **ACTION** action)
- **void** *hdestroy*(**void**)
- Tarvittavat tietorakenteet (<search.h>):

```
typedef struct entry {
  char *key;
  void *data;
} ENTRY;

typedef enum { FIND, ENTER }
 ACTION;
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Hajautustaulun luonti/tuhoaminen

- Hajautustaulu luodaan funktiolla:
  int *hcreate*(**size_t** nel)
  - nel parametrillä **arvioidaan** kuinka monta alkiota enimmillään taulussa on
  - palauttaa 0 onnistuessaan ja jotain muuta virhetilanteessa
- Hajautustaulu tuhotaan funktiolla:
  void *hdestroy*(**void**)
- Ohjelmassasi voi siis olla vain yksi hajautustaulu tällä rajapinnalla
- GNU laajennoksessa voi olla useita hajautustauluja (man hcreate_r)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Hajautustaulusta etsiminen/lisääminen

- Hajautustaulusta voidaan etsiä alkiota avaimen perusteella funktiolla:
  ENTRY **hsearch*(**ENTRY** item, FIND)
  - item.key kentässä tulee olla haettava avain
  - Palauttaa osoittimen löydettyyn alkioon tai NULL osoittimen jos alkiota ei löydy.
- Hajautustauluun voidaan lisätä uusi alkio funktiolla:
  ENTRY **hsearch*(**ENTRY** item, ENTER)
  - item parametrissä on uusi alkio.
  - Palauttaa osoittimen lisättyyn alkioon tai virhetilanteessa NULL osoittimen.

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki (hash.c):

```
int main(void) {
  ENTRY i, *ip;
  char *input;
  char buf[256+1];
  unsigned long isize,id=0,j;

  CHECK((hcreate(256)));

  while((input = fgets(buf,256,stdin))) {
    isize = strlen(input) + 1;
    CHECK((i.key = (char *)malloc(isize)));
    CHECK((i.data = malloc(isize)));
    sprintf(i.key,"%lu",id++);
    strcpy((char *)i.data,input);
    CHECK((hsearch(i, ENTER)));
  }

  for(j=0;j < id; j++) {
    sprintf(i.key,"%lu",j);
    CHECK((ip = hsearch(i,FIND)));
    printf("id %s data %s\n",ip->key,(char *)ip->data);
  }

  hdestroy();
  return(0);
}
```

Binääripuiden systeemikutsut

- Etsi ja lisää, jos ei löydy:
  void **tsearch*(**const void** *key, **void** **rootp,
  int(**compar*)(**const void** *, **const void** *))
- Etsi (ei lisää, jos ei löydy):
  void **tfind*(**const void** *key, **const void** **rootp, **int**(**compar*)(**const void** *, **const void** *))
- Poista:
  void **tdelete*(**const void** *key, **void** **rootp,
  int(**compar*)(**const void** *, **const void** *))
- Käy läpi koko puu ja tee solmuille operaatio:
  void *twalk*(**const void** *root,
  void(**action*)(**const void** *nodep, **const VISIT** which, **const int** depth))

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki (bintree.c)

```
void *root = NULL;

int compare(const void *pa, const void *pb) {
  if (*(int *)pa < *(int *)pb) return -1;
  if (*(int *)pa > *(int *)pb) return 1;
  return 0;
}

void action(const void *nodep, const VISIT which, const int
depth)
{
  int *datap;

  switch(which) {
  case preorder:
    break;
  case postorder:
    datap = *(int **)nodep;
    printf("%6d\n", *datap);
    break;
  case endorder:
    break;
  case leaf:
    datap = *(int **)nodep;
    printf("%6d\n", *datap);
    break;
  }
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkin jatkoa:

```
int main(int argc, char *argv[])
{
  int i, *ptr;
  void *val;
  int num=0;

  if(argc != 2)
  {
        printf("Usage: %s range_of_values\n",argv[0]);
        return 0;
  }

  num = atoi(argv[1]);

  srand(time(NULL));
  for (i = 0; i < 20; i++) {
    CHECK((ptr = (int *)malloc(sizeof(int))));
    *ptr = random()%num;
    CHECK((val = tsearch((void *)ptr, &root, compare)) !=
NULL);
  }
  twalk(root, action);
  return 0;
}
```

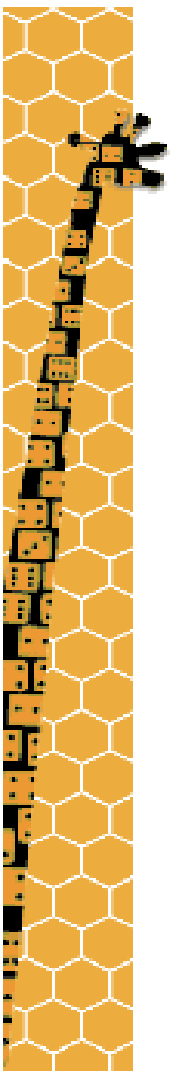
TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Satunnaisluvuista

- Tietokone ei sinänsä sisällä mitään todellista satunnaislukugeneraattoria
- Satunnaislukuja luodaan erilaisilla monimutkaisilla funktioilla
- Idea on, että funktiolle annetaan aloitusluku, josta sitten generoidaan uusia “satunnaisia” lukuja
- Samalla aloitusluvulla saadaan aina sama sarja lukuja, siksi aloitusluvun valinnalla on väliä
- Yksi tapa valita aloitusluku mahdollisimman satunnaisesti on käyttää esim. tietokoneen kelloa
- Jos ohjelmassa on säikeitä, pitäisi käyttää funktioita *erand48()*, *nrand48()* tai *jrand48()*

Satunnaislukujen systeemikutsut

- Satunnaislukuja voi myös generoida funktioilla:
  - **double** *drand48*(**void**); palauttaa satunnaisluvun väliltä [0,1[
  - **long int** *lrand48*(**void**); palauttaa satunnaisluvun väliltä 0 -- 2^31
  - **long int** mrand48(void); palauttaa satunnaisluvun väliltä -2^31 -- 2^31
- Satunnaislukugeneraattori tulee ennen käyttöä alustaa, muuten generaattori palauttaa aina samoja arvoja:
  void *srand48*(**long int** seedval)
- Ilmeisesti random() on myös varteenotettava funktio, ja rand() lienee sama kuin random()?
- Lisää tietoa manuaalisivuilla



Realiaikaohjelmointi

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Linux

- Linux ei ole varsinainen tosiaikakäyttöjärjestelmä
- Tästä huolimatta Linuxia voi hyvin käyttää ainakin pehmeissä (soft) ja tietyissä tilanteissa myös tiukissa (firm) tosiaikajärjestelmissä käyttöjärjestelmänä
- Kaikkissä esimerkeissä oletetaan että käytössä on Linuxin ydin 2.4.x tai uudempi ja ytimeen on asennettu low latency patch
- Lisäksi oletetaan että käytössä on root-tunnus

Reaaliaikaohjelmointi Linuxissa

- Ohjelmoija voi periaatteessa tehdä kaksi asiaa, joiden avulla prosessille voi pyrkiä takaamaan suoritusaika:

1. Sidotaan prosessin käyttämä muisti fyysiseen keskusmuistiin ja estetään sen mahdollinen siirtäminen virtuaalimuistin avulla levylle
2. Säädetään prosessien vuorottajaksi tosiaikaskeduleri ja annetaan prosesille korkea prioriteetti

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Muistiavaruuden sitominen

- Tyypillisesti tietokoneen ja käyttöjärjestelmän toteutukseen kuuluu oleellisena osana virtuaalimuisti
- Kun fyysinen keskusmuisti loppuu, virtuaalista muistiavaruutta jatketaan levylle
- Muistin käsittely levyltä, esimerkiksi sivun nouto muistiin ja vieminen takaisin levylle, on suhteessa hyvin hidasta
- Fyysisen muistin loppuessa, levylle joutuvat tyypillisesti vähiten käytetyt muistialueet
- Silti, esimerkiksi kriittiset ja keskeiset KJ:n prosessit eivät saisi joutua sivutuksen kohteiksi ja niiden muistiavaruus ei saisi joutua levylle, vaikka ne välillä olisivatkin “hiljaa”
- Tavallisten käyttäjien prosesseilla ei ole niin väliä...

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Muistiavaruuden sitominen

- Funktioilla mlock and munlock voidaan lukita muistialueita fyysiseen muistiin ja vapauttaa niitä
  - *int mlock*(const **void** * addr, **size_t** len)
  - *int munlock*(const **void** * addr, **size_t** len)
- Kutsujen onnistuminen vaatii root-käyttäjän oikeudet
- Prosessien koko nykyinen ja tulevaisuuden muistinkäyttö voidaan lukita ja vapauttaa kutsuilla:
  - *int mlockall*(**int** flags) ja
  - *int munlockall*(**void**)
- *Mlockall*-kutsun lipukeella kerrotaan, halutaanko pelkästään nykyinen (MCL_CURRENT) vai myös tulevat (MCL_FUTURE) muistialueet lukita keskusmuistiin
- Jos lukittavat alueet ylittävät keskusmuistin määrän, virhetilanne on alustariippuvainen

Esimerkki: mlok.c

```
int main(int argc, char *argv[])
{
  char buf[128];
  int next=0, amount=0;
  unsigned long total=0;
  char *muistivaraukset[1024];

  if(mlockall(MCL_FUTURE) != 0) {
        perror("mlockall");
        printf("varataan sitten ilman mlockall-käskyä...\n");
  }

  while(1) {
        printf("Anna muistivarauksen määrä: ");
        fflush(stdout);
        if(fgets(buf,128,stdin) == NULL) break;
        amount=atoi(buf);
        if(amount==0) break;
        muistivaraukset[next]=(char *) malloc(amount);
        if(muistivaraukset[next]==NULL) {
                perror("Muistivaraus epäonnistui"); break;
        }
        else  {
          total+=amount;  next++;
          printf("Muistia nyt varattu %lu tavua\n",total);
        }
  }
  munlockall(); return 0;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Ajoittajan vaihtaminen

- Ajoittajan vaihtaminen tapahtuu muuttamalla Linuxin oletusskeduleri toiseksi ja antamalla prosessille prioriteetti
- Tämä tapahtuu funktiolla:

  int *sched_setscheduler*(**pid_t** pid, **int** policy, **const struct sched_param** *param)
  - pid on prosessin tunnus, 0 = tämä prosessi
  - policy on käytettävä ajoitusmenetelmä:
    - SCHED_OTHER
    - SCHED_FIFO
    - SCHED_RR
  - param on rakenne, jossa on:

```
struct sched_param {
    int __sched_priority;
    };
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Ajoittajista

- SCHED_FIFO: preemptiivinen, prioriteettiin perustuva ajoitus
- SCHED_RR: preemptiivinen, prioriteettiin perustuvat ajoitus aikarajalla
- SCHED_OTHER: toteutusriippuvainen ajoitus

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Kelloista ja ajastimista

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Kelloista

- Kellonaikaa voi kysellä funktiolla:
  int *clock_gettime*(**clockid_t** clock_id, **struct timespec** *tp)
- Root voi asettaa kellonajan funktiolla:
  int *clock_settime*(**clockid_t** clock_id, **const struct timespec** *tp)
- Tietyn kellon tarkkuuden voi selvittää:
  int *clock_getres*(**clockid_t** clock_id, **struct timespec** *res)

Kellojen rakenteet

- Mahdolliset kellot:
  - CLOCK_REALTIME : systeemin realikello
  - CLOCK_MONOTONIC : monotonisesti kasvava kello
  - CLOCK_PROCESS_CPUTIME_ID : prosessikohtainen CPU-aika
  - CLOCK_THREAD_CPUTIME_ID : säiekohtainen CPU-aika
- Rakenne timespec

```
struct timespec {
  time_t     tv_sec;
  long       tv_nsec;
  };
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki (rtclock.c):

```
* Calculate difference between two timespec times */
struct timespec time_diff(struct timespec t1,struct timespec t2)
{
  struct timespec res;

  if ( t2.tv_nsec < t1.tv_nsec ) {
    t2.tv_sec--;
    t2.tv_nsec += 1000000000;
  }

  res.tv_sec = t2.tv_sec - t1.tv_sec;
  res.tv_nsec = t2.tv_nsec - t1.tv_nsec;
  return res;
}

/* Calculate something which takes some time */
double b(void) {
  double a = 5;
  int i;

  for(i=0;i < 10000000;i++)
    a = sqrt(a * 1.2);

  return a;
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

esimerkin jatkoa...main()

```
int main(void) {
  struct timespec begin,end,reso,diff;
  double res;

  CHECK((clock_gettime(CLOCK_REALTIME,&begin)) != -1);
  res = b();
  CHECK((clock_gettime(CLOCK_REALTIME,&end)) != -1);
  CHECK((clock_getres(CLOCK_REALTIME,&reso)) != -1);
  diff = time_diff(begin,end);

  printf("CLOCK_REALTIME: Result is %f and time was %ds:%luns and
clock resolution is %ds:%luns\n",
res,(int)diff.tv_sec,diff.tv_nsec,(int)reso.tv_sec,reso.tv_nsec);

  CHECK((clock_gettime(CLOCK_MONOTONIC,&begin)) != -1);
  res = b();
  CHECK((clock_gettime(CLOCK_MONOTONIC,&end)) != -1);
  CHECK((clock_getres(CLOCK_MONOTONIC,&reso)) != -1);
  diff = time_diff(begin,end);

  printf("CLOCK_MONOTONIC: Result is %f and time was %ds:%luns and
clock resolution is %ds:%luns\n",
res,(int)diff.tv_sec,diff.tv_nsec,(int)reso.tv_sec,reso.tv_nsec);
```

esimerkin loppu...

```
  CHECK((clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&begin)) != -1);
  res = b();
  CHECK((clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&end)) != -1);
  CHECK((clock_getres(CLOCK_PROCESS_CPUTIME_ID,&reso)) != -1);
  diff = time_diff(begin,end);

  printf("CLOCK_PROCESS_CPUTIME_ID: Result is %f and time was
%ds:%luns and clock resolution is %ds:%luns\n",
res,(int)diff.tv_sec,diff.tv_nsec,(int)reso.tv_sec,reso.tv_nsec);

  exit(EXIT_SUCCESS);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Nanosekunnin nukkuminen

- **int** *nanosleep*(**const struct timespec** *rqtp, **struct timespec** *rmtp)
  - rqtp parametrillä kerrotaan kuinka kauan halutaan nukkua
  - rmtp parametriin palautuu kuinka paljon jäi aikaa jäljelle kutsun päätyttyä esimerkiksi signaaliin
  - Palauttaa arvon 0 jos nukkuminen on päättynyt pyydetyn ajan jälkeen
  - Muuten palauttaa -1 ja asettaa nukkumatta jääneen ajan. errno kertoo tapahtuiko virhe vai tuliko signaali (EINTR).

Esimerkki (nsleep.c):

```
int main(int argc, char **argv) {
  struct timespec lyhyt_uni, nukkumatta;
  struct timespec begin, end;

  if ( argc < 2 ) {
       printf("usage: nsleep <nsecs>\n"); exit(1);
  }
  lyhyt_uni.tv_sec  = 0;
  lyhyt_uni.tv_nsec = atoi(argv[1]);
  nukkumatta.tv_sec = 0;
  nukkumatta.tv_nsec = 0;

  CHECK((clock_gettime(CLOCK_REALTIME,&begin)) != -1);
  CHECK((nanosleep(&lyhyt_uni,&nukkumatta)) != -1);
  CHECK((clock_gettime(CLOCK_REALTIME,&end)) != -1);

  printf("Nukkumatta %lu s %lu ns\n",
          nukkumatta.tv_sec,nukkumatta.tv_nsec);
  printf("Aikaa kului %lu ns\n",
          end.tv_nsec - begin.tv_nsec);
  exit(0);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Ajastimet (POSIX Interval Timers)

- POSIX ajastimet mahdollistavat nanosekunnin tarkkuuden ajastuksiin
- Ajastin luodaan dynaamisesti halutulle kellolle
- Standardin mukaan ajastimia pitää olla mahdollista luoda vähintään 32/prosessi
- Ajastimet perustuvat rakenteeseen:

```
struct itimerspec {
  struct timespec it_value;
  struct timespec it_interval;
  };
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Ajastimen luonti/tuhoaminen

- Ajastin luodaan funktiolla:
  int *timer_create*(**clockid_t** clockid, **struct sigevent *restrict** evp, **timer_t *restrict** timerid)
  - Onnistuessaan palauttaa arvon 0, muuten -1
  - clockid parametrillä kerrotaan haluttu kello
  - evp parametrillä määritellään mikä signaali halutaan ajastimen lauettua
  - timerid parametriin palautuu itse ajastimen tunniste
- Ajastin tuhotaan funktiolla:
  int *timer_delete*(**timer_t** timerid)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Ajastimen asettaminen

- Ajastin voidaan asettaa funktiolla:
  int *timer_settime*(**timer_t** timerid, **int** flags, **const struct itimerspec *restrict** value, **struct itimerspec *restrict** ovalue)
  - timerid on ajastimen tunnus
  - flags kentässä kerrotaan annetun ajan muoto:
    - 0 : ajastin herätetään annetun ajan kuluttua
    - TIMER_ABSTIME : ajastin laukeaa vain tänä tiettynä aikana
  - value parametrissä annetaan itse aika
  - ovalue kenttään tallettuu edellisen ajastimen asetukset
  - palauttaa 0 jos onnistuu muuten -1

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Ajastimen ylivuoto

- POSIX standardin mukaan ei tarvitse asettaa signaaleja, jotka johtuvat ajastimen laukeamisesta jonoon vaikka käyttäisi POSIX RTS rajapintaa
- Siis ei tiedetä kuinka monta signaalia ajastimen laukeamisesta on tullut
- Siksi on erikseen funktio, jolla voi selvittää kuinka monta kertaa ajastin on lauennut tähän mennessä
- Funktio on:

  int *timer_getoverrun*(**timer_t** timer)

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Esimerkki (timer_signal.c):

```
volatile sig_atomic_t nyt_loppu = 0;

void timer_handler(int signum, siginfo_t *info, void *context)
{
    struct timespec now;
    int overrun;

    CHECK((clock_gettime(CLOCK_REALTIME, &now)) == 0);

    if ( signum == SIGINT ) {
      printf("Trying to kill this program\n");
      nyt_loppu = 1;
      return;
    }
    printf("signal # = %d\n", info->si_signo);
    printf("sending process ID: %lu\n", (unsigned long)info-
>si_pid);
    printf("real user ID of sending process: %lu\n",
           (unsigned long)info->si_uid);
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Funktion timer_handler() loppuosa:

```
/* Find out the reason of the signal */
    switch (info->si_code) {
      case SI_USER:
          printf("Signal from user\n"); break;
      case SI_TIMER: {
          printf("Signal from timer expiration\n");
          printf("Time is NOW about %d sec %lu nsec\n",
                 (int)now.tv_sec, (unsigned long)now.tv_nsec);

          CHECK((overrun = timer_getoverrun(*(timer_t *)
                 info->si_value.sival_ptr)) != -1);

          if ( overrun ) {
            printf("Timer has overflowed %d times -- error\n",
                           overrun);
          }

          break;
      }
    }
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Funktio create_signal_handler()

```
void create_signal_handler(
        int signal,
        void (*sig_handler)(int, siginfo_t *, void *))
{
        struct sigaction act;

        CHECK(sig_handler);

        memset(&act, 0, sizeof(act));
        act.sa_flags = SA_SIGINFO;
        act.sa_sigaction = sig_handler;
        CHECK((sigaction(signal, &act, NULL)) == 0 );
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Funktio create_timer() (create_timer.c):

```
void create_timer(int signal, timer_t *my_timer) {
  struct sigevent mysignal;

  memset(&mysignal,0,sizeof(mysignal));
  memset(my_timer,0,sizeof(timer_t));

  mysignal.sigev_notify = SIGEV_SIGNAL;
  mysignal.sigev_signo = signal;
  mysignal.sigev_value.sival_ptr = (void *) my_timer;

  CHECK((timer_create(CLOCK_REALTIME, &mysignal, my_timer)) !=
-1);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Funktio set_timer() (set_timer.c):

```
void set_timer(
        timer_t my_timer,
        time_t  first_sec,
        long    first_nsec,
        time_t  then_sec,
        long    then_nsec)
{
  struct itimerspec new_timer, old_timer;

  CHECK(my_timer);

  memset(&new_timer, 0, sizeof(new_timer));
  memset(&old_timer, 0, sizeof(old_timer));

  new_timer.it_value.tv_sec = first_sec;
  new_timer.it_value.tv_nsec = first_nsec;
  new_timer.it_interval.tv_sec = then_sec;
  new_timer.it_interval.tv_nsec = then_nsec;

  CHECK((timer_settime(my_timer, 0, &new_timer, &old_timer)) !=
-1);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Funktio set_timer_to (set_timer.c):

```
void set_timer_to(
        timer_t my_timer,
        struct timespec when)
{
  struct timespec now;
  struct itimerspec exp_time;

  CHECK(my_timer);
  memset(&now, 0, sizeof(now));
  memset(&exp_time, 0, sizeof(exp_time));

  /* What time is NOW */
  CHECK((clock_gettime(CLOCK_REALTIME, &now)) == -1);

  exp_time.it_value.tv_sec = when.tv_sec - now.tv_sec;
  exp_time.it_value.tv_nsec = when.tv_nsec - now.tv_nsec;

  if ( when.tv_nsec < now.tv_nsec ) {
    exp_time.it_value.tv_sec--;
    exp_time.it_value.tv_nsec += 1000000000;
  }
  /* Time is NOW + i */
  CHECK((timer_settime(my_timer, 0, &exp_time, NULL)) != -1);
  /* Time is NOW + i + j */
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

Lopulta funktio main() (timer.c):

```
extern volatile sig_atomic_t nyt_loppu;

int main(int argc, char **argv) {
  struct timespec resolution;
  timer_t my_timer;
  sigset_t allsigs;

  create_signal_handler(SIGRTMIN, &timer_handler);
  create_signal_handler(SIGINT, &timer_handler);
  /* Get clock resolution */
  memset(&resolution, 0, sizeof(resolution));
  CHECK((clock_getres(CLOCK_REALTIME, &resolution)) != -1);
  printf("Clock resolution is %d sec %lu nsec\n",
         (int)resolution.tv_sec,(unsigned
long)resolution.tv_nsec);
  /* Create a new timer */
  create_timer(SIGRTMIN, &my_timer);
  printf("Timer %lu created\n",my_timer);
  /* Set timer to 10 * resolution */
  set_timer(my_timer, 0, resolution.tv_nsec * 10,
            0, resolution.tv_nsec * 10);
  /* Wait for signals...*/
  sigemptyset(&allsigs);
  while(nyt_loppu == 0) {
    CHECK((sigsuspend(&allsigs)) == -1);
  }
  timer_delete(my_timer);
  exit(1);
}
```

TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY