NTNU
Norges teknisk-vitenskapelige universitet

Institutt for teknisk kybernetikk
Fakultet for informasjonsteknologi,
matematikk og elektroteknikk

Bokmål &
Engelsk

Eksamen
i
TTK4145
Sanntidsprogrammering
17. desember 2004
15.00 -19.00

Sensuren vil bli avsluttet i henhold til gjeldende regelverk.

Oppgavesettet inneholder en engelsk og en norsk versjon. Den norske versjonen finnes fra side 2 til og med side 5. Du kan velge om du vil besvare eksamen på norsk eller engelsk.

This exam contains two versions, one in norwegian and one in english. The english version is found on pages 6 through 9.

Generelt:
Vær konsis og klar. Skriv kort. Sett opp momenter og argumenter punktvis, bruk figurer der det egner seg. Du kan få trekk for «vås».

Generally:
Be short and clear. Enumerate items and arguments, use figures where appropriate. You may be punished for text without substance.

Faglig kontakt under eksamen:
Sverre Hendseth
Telefon 98443807

Oppgave 1: Delt variabel synkronisering

Oppgave 1 a) Semaforer er en standardmekanisme for synkronisering av tilgang til delte resurser. Hvor ligger ulempene/vanskelighetene ved bruk av semaforer ?

Oppgave 1 b) Hvorfor har ikke Occam semaforer eller andre mekanismer for å styre tilgangen på felles variable fra forskjellige prosesser ?

Oppgave 1c) Monitorer anses for å være en bedre/mer høynivå mekanisme enn semaforer. Hvorfor ?

Oppgave 1d) Hvor ligger ulempene/vanskelighetene ved bruk av monitorer ?

Øving 7 gikk ut på å lage et «høytilgjengelig» system med to prosesser hvor den ene – masteren – gjorde fornuftig arbeide, mens den andre – slaven – bare sto klar til å ta over i tilfelle masteren kræsjet.

Sverres løsningsforslag på øvingen var som følger;

- En kommunikasjonsmodul med prosedyrer for å sende og motta data i tillegg til to prosedyrer for å åpne forbindelse (en for «slave modus» som vil mislykkes hvis den ikke får kontakt med den andre prosessen og en i «master modus» som vil legge seg å lytte etter forbindelser fra slaveprosessen) og en for å lukke alle forbindelser.
- Programmet kan være i en av tre moduser: MasterMedSlave, MasterUtenSlave og Slave. Dette bestemmes av den globale heltallsvariabelen g_mode.
- Tre prosesser; En (P1) for å lese og ekspedere meldinger, en (P2) for å gjøre fornuftig arbeide og sende data/log til slaveprosessen hvis det finnes en, og en (P3) for å sende IAmAlive meldinger. P1 svarer på IAmAlive meldinger i MasterMedSlave modus og oppdatere dataene i Slave modus. P2 gjør ingen ting i slave modus, og sender data til slaven bare i MasterMedSlave modus. P3 gjør noe bare i slavemodus, hvor den sender IAmAlive meldinger.
- Ved oppstart prøver programmet å åpne forbindelsen til den andre prosessen; Hvis dette lykkes er vi i slave modus, ellers går vi til MasterUtenSlave modus. Uansett startes de 3 prosessene etter at denne avgjørelsen er tatt.
- Hvis vi mister forbindelsen med den andre prosessen (fra MasterMedSlave eller Slave-modus) kjøres «tidyAndReconnect» som kobler med den eksisterende forbindelsen, prøver å opprette forbindelse til den andre prosessen og tar avgjørelsen MasterUtenSlave/Slave ut ifra om dette lykkes. Dette kan skje etter at et sende eller lese kall mislykkes, eller ved at vi (P1) detekterer at vi ikke har fått IAmAlive meldinger på en stund.
- Virkemåten til prosedyrene for sending og mottak i kommunikasjonsmodulen er avhengige av g_mode også.

Kildekoden til løsningsforslaget er vedlagt for dem som vil ha noe konkret å se på, men det skal være mulig å svare perfekt uten å relatere svarene til denne.

Oppgave 1e) Anta at to prosesser godt kan bruke kommunikasjonsmodulen samtidig, men bare så lenge den ene leser og den andre skriver. Anta også at selve lesingen og skrivingen av g_mode kan sees på som atomisk og at ut over g_mode og nettverksforbindelsen er det ingen delte variable/resurser i systemet. Er det behov for å beskytte tilgangen på g_mode og/eller nettverksforbindelsen med

semaforer el.l. ? Begrunn svaret.

Uten å tenke vidre over om det faktisk er nødvendig eller ikke, bestemmer Sverre seg for å beskytte aksessen til disse to delte resursene. Alle aksesser til g_mode omkranses med reservering og frigiving av en semafor (dette skjer så tett på aksessen av g_mode som mulig, men koden endres ikke for å unngå at switch'er og if-statements havner innenfor de beskyttede regionene), og alle prosedyrene - unntatt meldingslesningen - i kommunikasjonsmodulen reserverer en annen semafor øverst i prosedyren og frigir den i bunnen.

Oppgave 1f) Kritiser denne strategien (konstruktivt :-). Virker programmet nå ?

Oppgave 1g) Hvis vi tar høyde for hardere tidskrav, mulige fremtidige utvidelser/endringer av programmet etc. Hvordan ville du beskytte de to resursene ? (Du har alle mekanismer tilgjengelig, og du står fritt til å foreslå design endringer også om du vil.)

Oppgave 2: Scheduling

Oppgave 2a) Hvordan uttrykkes sanntidskravene til en prosess under Fixed Priority Scheduling (FPS) ?

Oppgave 2b) En enkel "utilization-based schedulability test" for et FPS system av prosesser er gitt ved

$$\sum_{i=0}^{N} (\frac{C_i}{T_i}) \leqslant N * (2^{\frac{1}{N}} - 1\ )$$

hvor N er antall prosesser, C_i er worst-case execution time og T_i er prosessens periode. Under hvilke forutsetninger holder denne ?

Oppgave 2c) Den tilsvarende schedulerbarhetstesten under et Earliest Deadline First (EDF) regime er

$$\sum_{i=0}^{N} (\frac{C_i}{T_i}) \leqslant 1$$

Det ser altså ut som om EDF kan gi bedre garantier/ schedulere flere systemer enn FPS. Hva er årsakene til at FPS er det systemet som er i vanlig bruk ? (mao. hva er ulempene med EDF ?)

Oppgave 2d) En av scheduling-strategiene i boken har vranglåsunngåelse som egenskap. Hvordan virket denne ?

Oppgave 3: Litt av hvert

Oppgave 3a) List opp virkemidler du kan bruke for å unngå problemer med vranglåser.

Oppgave 3b) Hvilke(t) problem er det egentlig Atomic Actions løser og hvordan løses disse/dette ved Atomic Actions ?

Oppgave 3c) "Resurskontroll" er mer enn å oppnå mutual exclusion: List opp flere krav vi kan stille til en resurskontroll modul.

Oppgave 3d) Det ligger implisitt synkronisering i meldingssending... I stort perspektiv; sammenlign delt variabel synkronisering og meldingssynkronisering (konsist og punktvis).

Oppgave 3e) N-Versjons programmering er et konsept for å oppnå feiltoleranse som det er vanskelig å få til å virke etter intensjonene. Hvorfor ?

Oppgave 3f) Vi kan programmere inn eksplisitte feilsjekker i et programvare system, list opp metoder vi kan bruke for å detektere at noe har gått galt.

Oppgave 3g) Microkernel i QNX implementerer et sett med funksjoner for "message passing" mellom prosesser. Forklar semantikken bak denne synkroniseringsmetoden.

Oppgave 3h) Forklar hva som er forskjellen mellom status-drevet input (polling) og interrupt-drevet input.

Oppgave 4: Design

Du skal gjøre grovdesignet av programvaren i følgende system: En mekanisk port skal styres/overvåkes. Folk kan gå inn eller ut av porten og formålet er at vi skal holde rede på hvor mange mennesker som er inne i området bak porten (Dette er et absolutt krav – det skal ikke under *noen* omstendighet være mulig å komme ut av tellingen). Noen kompliserende elementer er følgende:

- Vi skal kunne ha flere porter inn til samme område. Avlesing av hvor mange mennesker som er inne i området skal kunne skje ved alle portene. Anta at alle portene kjenner til hverandre, og kan sende meldinger til hverandre.
- Vi skal håndtere at programmet vårt kræsjer/restarter (men vi antar at bare en port er nede i slengen). Vi kan imidlertid stole på kommunikasjonen og selve porten.

"Grovdesign" innebærer her at du skal overbevise om at du er på riktig spor når det gjelder designet, dvs. at du skal ta de design-avgjørelsene som har størst konsekvenser for resten av designet og de "vanskelige" design-avgjørelsene, hvor det ikke er åpenbart om det finnes en løsning eller hva som er løsningen.

Du trenger ikke beskrive selve den fysiske portens virkemåte – gjør antagelser som du synes passer.

Vær oppmerksom på at presentasjonen i seg selv er en viktig del av svaret på denne oppgaven.

Noen velmente hint:
- Ta eksplisitt stilling til hvilke problemstillinger som du sikter på å løse i dette grovdesignet.
- Kladd designet ditt først – og så presenter det i besvarelsen. Ikke lever sider med "betraktninger underveis".

Oppgave 5: Implementasjon

I begge disse oppgavene er pseudokode ok.

Oppgave 5a) Ligningen under gir en iterativ metode for å beregne roten av x.

$$e_{i+1} = (e_i + \frac{x}{e_i})/2$$

Skriv et occam program som beregner roten av tall med 20 iterasjoner, hvor hver iterasjon er en egen prosess. (Hint; Bruk replicated PAR)

Oppgave 5b) Skriv en resursmanager (grensesnitt allocate(...) & free(...)) for en resurs som det er en av i systemet. Vi ønsker imidlertid full og eksplisitt kontroll over hvem som skal bli tildelt resursen – anta for dette eksemplet at allocate kallet tar (bl.a. ?) en parameter inn som representerer "viktigheten" av at tråden får resursen. Bruk det språket eller de synkroniseringsmekanismene du vil. Du kan anta generiske datastrukturer og algoritmer om du trenger det – fokus er på synkroniseringen.

1. Shared Variable-based Synchronization

1a) Semaphores is a standard mechanism for synchronization of access to shared resources. What are the downsides/difficulties by using semaphores ?

1b) Why does not OCCAM have semaphores or other mechanisms for controlling access to shared variables between processes ?

1c) Monitors are accepted as being a better/higherlevel mechanism than semaphores Why ?

1d) What are the downsides/difficulties by using monitors ?

The last exersice this autumn was to make a "high availability" system with two processes where one – the master – did useful work and the other – the slave – only was ready to take over in case the master crashed.

Sverres solution to this was as follows:

- A communication module with procedures for sending and receiving data, in addition to two procedures for opening connections (one for "slave mode" that will fail if it cannot establish contact with the master, and one for "master mode" that will hang, waiting for a connection from a slave) and one for closing all connections.
- The program can be in one of three states: Master, Slave and Single (A master without connection to a slave). The mode is kept in the global variable g_mode.
- Three processes: P1 reads and handles incomming messages. – It answers IAmAlive messages in Single mode, and updates the data in Slave mode. P2 does "the useful work" (in Single and Master mode) and sends in addition updates/log to the slave process in Master mode. P3 sends IAmAlive messages in Slave mode.
- At startup, the program tries to open the connection to the other process; If this succeeds we are in Slave mode, if not we are in Single mode. In any case the three processes are started after this decition is taken.
- If we loose connection to the other process (from either Master or Slave mode) the procedure "tidyAndReconnect" is run that closes all connections, tries to reconnect to the other process, and sets mode to Single or Slave depending on the outcome. Loosing connection can be detected by either a failing read or write call, or by detecting that we have not received IAmAlive messages in a while.
- The send and receive procedures in the communication module is dependent on the g_mode variable also.

The source code for the program is in appendix. Look at it if you need to, but it should be possible to answer perfectly without relating your answers to it.

1e) Assume that two processes can use the communication socket at the same time, but only if one reads and the other writes. Assume also that reading and setting g_mode is atomic, and that there are no more shared variables/resources in the system. Is it necessary in this program to protect g_mode and/or the socket ? Explain why/why not.

Without reasoning any further on whether it is necessary, Sverre decides to protect both resources; All accesses to g_mode is enclosed with reserving and releasing a semaphore (as close to each access to the variable as possible, though the code is not changed to avoid that ifs and switches are contained in the critical regions), and all procedures in the communication module, except the one receiving data, reserves another semaphore first in execution, and releases it just before returning.

1f) Give a (constructive) critisism of this strategy. Does the program work now ?

1g) If we prepare for future changes of the program (other timing requirements, changes and improvements...); How would you protect the two resources ? (You have all mechanisms available, and you may also suggest design changes if you find this appropriate.)

2. Scheduling

2a) How do you express the real time demands on a process under Fixed Priority Scheduling (FPS) ?

2b) A simple "utilization-based schedulability test" for a FPS system of processes is given by

$$\sum_{i=0}^{N} (\frac{C_i}{T_i}) \leqslant N * (2^{\frac{1}{N}} - 1\ )$$

where N is the number of processes, C_i is the worst-case execution time and T_i is the process' period. Under which assumption is this valid ?

2c) The corresponding schedulability test under "Earliest Deadline First" (EDF) is

$$\sum_{i=0}^{N} (\frac{C_i}{T_i}) \leqslant 1$$

It look like EDF can guarantie schedulability for a larger set of processes than FPS. What are the reasons why FPS is the system that is in common use ? (In other words; What are the downsides to EDF ?)

2d) One of the scheduling strategies in the book has deadlock avoidance as one of its features. How does this work ?

3. Misc.

3a) List techniques that can be used to avoid problems with deadlocks.

3b) Which problem(s) did Atomic Actions solve, and how does Atomic Actions solve this/these problem(s) ?

3c) Resource Control is more than mutual exclusion: List more demands we can place on a resource

control module.

3d) There are some implicit synchronization in message passing... In a big perspective; Compare shared variable synchronization with message-based synchronization (short and enumerated).

3e) N-Version programming is a concept for fault tolerance that is difficult to make work as intended. Why ?

3f) Explicit error checks can be programmed into a software system. List principles/methods that can be used to detect that something has gone wrong.

3g) The Microkernel in QNX gives a set of functions for message passing between processes. Explain the semantics behind this synchronization method.

3h) Explain the difference between status-driven input (polling) and interrupt-driven input.

4. Design

You are to do the high-level design of the following system: A mechanical gate shall be controlled/monitored. People can walk in and out of the gate, and the purpose is to keep track of how many people are in the area behind the gate. This is an absolute requirement – under no circumstances should it be possible to loose track of the exact number of people. Some complicating elements:

- There may be more gates into the same area. Reading out how many people are in the area should be possible at any gate. Assume that all gates know of each other and can send messages to each other.
- We should handle that our program crashes/restarts (but assume that only one program is down at a time). We can, however, trust the gate and the communication links.

"High-level design" means that you should convince that you are on the right track of designing the system, You should decide on the "large" design decisions – that has consequences for large parts of the system -, and the difficult decitions – where it is not obvious that there is a solution, or what the solution is.

You need not describe the mechanical gate – make assumptions you see fit.

Be aware that *presentation* is an important part of your answer here!

Some hints:
- Be explicit on what design decitions you choose to include in your answer.
- Make a draft version of your design first – then *present it* in your answer. Do not turn in pages with "reflections along the way".

5. Implementation

Pseudocode is OK in both these assignments.

5a) The equation under gives an iterative method for calculating the square root of x.

$$e_{i+1} = (e_i + \frac{x}{e_i})/2$$

Write an OCCAM program that calcuates the square root through 20 iterations, where each iteration is a separate process. (Hint: Use replicated PAR).

5b) Write a resource manager (interface allocate(...) and free(...)) for a resource of which there is one in the system. We want full and explicit control over which thread gets assigned the resource – assume that for this example the allocate call takes (among others ?) a parameter decribing the *importance* of this thread getting the resource. Use the language constructs and the synchronization mechanisms you want. You can assume that you have generic data structures and algorithms if you need so – focus is on synchronization.

APPENDIX

Dette er et løsningsforslag til siste øving. Vær oppmerksom på at det ikke skal være **nødvendig** å forholde seg til dette for å svare på oppgavene.

This is a solution to the last excersice. Be aware that it **should** not be necessary to relate to this to answer the questions in the exam.

```
/*** #file "test4.c" ***/
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include "common.h"
#include "comm2.h"

Mode g_mode;
int g_currentNumber = 0;
char g_machineName[256];
char g_portNumber;

void
tidyAndReconnect(){
  int res;
  static working = 0;
  if(working == 1) return;
  working = 1;
  printf("A communication error - tidying and setting state.\n");
  tidy();

  /* res = openConnection(g_machineName,g_portNumber); */
  res = -1;

  if(res == 0){
    /* We managed to open the connection - we are slave! */
    g_mode = Mode_Slave;
    printf("OK, opened connection, We are slave!\n");
  }else{
    /* We failed - assume we are single (so far...) */
    g_mode = Mode_Single;
  }
  working = 0;
}

/* Wrappers for readMessage and writeMessage */
void
sendMessage_wrapper(int message){
  int res;
  res = sendMessage(message);
  if(res == -1){
    /* write failed - is the other process gone ? */
    tidyAndReconnect();
  }
}

int
readMessage_wrapper(int * message){
  int res;
  res = readMessage(message);
  if(res == -1){
    /* write failed - is the other process gone ? */
    tidyAndReconnect();
    return -1;
  }
  return 0;
}

void *
work(void *threadData) {
  while(1){
    /* printf("\nHello World from work - mode = %d!\n",g_mode); */
    switch(g_mode){
      case Mode_Single: {
        printf("Number is now %d\n",g_currentNumber);
```

```
        g_currentNumber++;
        break;
      }
      case Mode_Master: {
        printf("Number is now %d\n",g_currentNumber);
        g_currentNumber++;
        sendMessage_wrapper(g_currentNumber);
        break;
      }
      case Mode_Slave: {
        break;
      }
    }
    sleep(1);
  }
}

void *
iAmAlive(void *threadData) {
  while(1){
    /* printf("\nHello World from iAmAlive - mode = %d!\n",g_mode); */
    switch(g_mode){
      case Mode_Single: {
        break;
      }
      case Mode_Master: {
        break;
      }
      case Mode_Slave: {
        sendMessage_wrapper(-1);
        break;
      }
    }
    usleep(100000);
  }
}

void *
communication(void *threadData) {
  int number;
  int res;

  printf("\nHello World from communication - mode = %d - %d!\n",g_mode,Mode_Single);

  while(1){
    switch(g_mode){
      case Mode_Single: {
        printf("OK, opened connection, starting listening\n");
        startListening(g_machineName,g_portNumber);
        printf("Listen returned - setting mode to master and trying to read!\n");
        g_mode = Mode_Master;
        break;
      }
      case Mode_Master: {
        res = readMessage_wrapper(&number);
        if(res == 0){
          if(number == -1){
            /* IAmAlive message - just reply */
            sendMessage_wrapper(-1);
          }else{
            printf("Error: Master got log\n");
          }
        }
        break;
      }
      case Mode_Slave: {
        res = readMessage_wrapper(&number);
        if(res == 0){
          if(number == -1){
            /* IAmAlive message */
          }else{
            printf("Got a log record - number is now %d\n",number);
            g_currentNumber = number;
          }
        }
        break;
```

```
      }
    }
  }
}

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

  pthread_t w,a,c;
  int rc, t;

  int res;
  int number;

  if(argc < 3){
    fprintf(stderr,"usage %s hostname port\n", argv[0]);
    exit(0);
  }

  g_portNumber = atoi(argv[2]);

  strncpy(g_machineName,argv[1],strlen(argv[1]) > 255 ? 255:strlen(argv[1]));
  res = openConnection(g_machineName,g_portNumber);

  if(res == 0){
    /* We managed to open the connection - we are slave! */
    g_mode = Mode_Slave;
    printf("OK, opened connection, We are slave!\n");
  }else{
    /* We failed - assume we are single (so far...) */
    g_mode = Mode_Single;
  }

  rc = pthread_create(&w, NULL, work, (void *)t);
  if(rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
  }
  rc = pthread_create(&a, NULL, iAmAlive, (void *)t);
  if(rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
  }
  rc = pthread_create(&c, NULL, communication, (void *)t);
  if(rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
  }
  pthread_exit(NULL);
}
/*** End of File ***/
```

```
/*** #file "comm2.c" ***/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <assert.h>
#include "common.h"

/* Module variables */
int sockfd = -1;
int newsockfd = -1;

void
error(char *msg)
{
  perror(msg);
  exit(0);
}

void
tidy(){
  if(newsockfd != -1){
    shutdown(newsockfd,SHUT_RDWR);
    close(newsockfd);
    newsockfd = -1;
  }

  if(sockfd != -1){
    shutdown(sockfd,SHUT_RDWR);
    close(sockfd);
    sockfd = -1;
  }
}

/* Returns 0 for OK */
int
openConnection(char * machineName,int portNumber){
  struct hostent *server;
  struct sockaddr_in serv_addr;
  int res;

  assert(sockfd == -1);

  server = gethostbyname(machineName);
  sockfd = socket(AF_INET, SOCK_STREAM, 0);

  /* res = setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,-1,0); */

  if (sockfd < 0)
      error("ERROR opening socket");
  server = gethostbyname(machineName);
  if (server == NULL) {
      fprintf(stderr,"ERROR, no such host: %s\n",machineName);
      exit(0);
  }
  bzero((char *) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  bcopy((char *)server->h_addr,
        (char *)&serv_addr.sin_addr.s_addr,
        server->h_length);
  serv_addr.sin_port = htons(portNumber);
  if(connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0){
    printf("ERROR connecting - no partner - we are master ?\n");
    shutdown(sockfd,SHUT_RDWR);
    sockfd = -1;
    return -1;
  }else{
    return 0;
  }
}

void
startListening(char * machineName,int portNumber){
  int clilen;
  struct sockaddr_in serv_addr, cli_addr;
```

```
  int n;
  int res;

  assert(sockfd == -1 && newsockfd == -1);

  sockfd = socket(AF_INET, SOCK_STREAM, 0);

  /* res = setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,-1,0); */

  if(sockfd < 0)
    error("ERROR opening socket");
  bzero((char *) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(portNumber);
  if (bind(sockfd, (struct sockaddr *) &serv_addr,
           sizeof(serv_addr)) < 0)
           error("ERROR on binding");
  listen(sockfd,5);
  clilen = sizeof(cli_addr);
  newsockfd = accept(sockfd,
              (struct sockaddr *) &cli_addr,
              &clilen);
  if(newsockfd < 0)
     error("ERROR on accept");
}

int
sendMessage(int message){
  int n;
  int fd;

  assert(g_mode != Mode_Single);

  if(g_mode == Mode_Slave){
    /* Use the sockfd socket */
    fd = sockfd;
  }else{
    /* Mode_Master */
    fd = newsockfd;
  }
  n = write(fd,(char *) &message,sizeof(int));
  if(n < 0){
    printf("ERROR writing to socket\n");
    return -1;
  }
  return 0;
}

int
readMessage(int * res){
  int n;
  int fd;

  assert(g_mode != Mode_Single);

  if(g_mode == Mode_Slave){
    /* Use the sockfd socket */
    fd = sockfd;
  }else{
    /* Mode_Master */
    fd = newsockfd;
  }
  n = read(fd,(char *) res,sizeof(int));
  if(n != sizeof(int)){
    printf("ERROR reading from socket\n");
    return -1;
  }
  return 0;
}

/*** End of File ***/
```