Datalogi 1F rapportopgave K1
Implementering af multiprogrammeringskerne med tvungen tidsdeling og lagerallokering

Klaus Hansen

11. marts 2002

1 Administrativ information

Rapportopgave K1 stilles mandag den 4. marts 2002 og skal **afleveres senest mandag den 25. marts kl. 14:00** i DIKU's 1.delsadministration, Universitetsparken 1, 2100 København Ø. Besvarelser, der sendes med posten, skal være DIKU i hænde senest ved afleveringsfristens udløb.

Besvarelser kan udarbejdes individuelt eller i grupper op til tre deltagere. Såfremt den studerende har tilmeldt sig rapporteksamen vil K1 senere blive censureret. Der gives ingen mulighed for at forbedre besvarelsen og der gives ikke reduktion i opgavens omfang ved mindre grupper. Gruppesammensætningen ved aflevering af K1 behøver ikke at være den samme som for G1.

Vigtige meddelelser vedrørende denne opgave vil blive meddelt i kursets nyhedsgruppe og på kursets Internethjemmeside `http://www.diku.dk/teaching/2002f/dat1f/`. Dette inkluderer plan for instruktorvagt og reservationsordning. Ændringer eller præciseringer meddeles i nyhedsgruppen og på opslagstavlen af den ansvarlige lærer.

2 Kerneopgaven

Opgaven består i at udvikle en multiprogrammeringskerne (herunder et kaldbibliotek) og afprøve det udviklede programmel med et sæt multiprogrammer (se afsnit 2.1). Det udførte arbejde skal beskrives i en rapport (se afsnit 2.2). Den udviklede kerne og rapporten vægtes ligeligt ved vurderingen. Den skriftlige fremstilling indgår i bedømmelsen af rapporten.

2.1 Kernen

Multiprogrammeringskernen skal udvikles til DIKU's AlphaPC164SX maskiner, der er beskrevet i Kursusbog 5 ("Programmering på Digital Alpha-arkitekturen"). Kernen skal opfylde følgende:

1. Kernen skal være skrevet i C/C++ og assemblerkode.
2. Det udviklede programmel består af kernen selv og et kaldbibliotek samt tilhørende header-filer (herunder en, der indeholder fælles ting for bibliotek og brugerprogram,

f.eks. med navnet `ekernelib.h`); antallet af filer med kildetekst er uafhængigt af dette, men skal ved oversættelse resultere i en fil med kernen (f.eks. med navnet `ekerne.o`) og en fil med kernekaldbiblioteket (f.eks. med navnet `ekernelib.o`). Det kørende program fremkommer ved at linke brugerprogrammellet med disse to.

3. Kernen skal kunne håndtere indlejrede afbrydelser.

4. Brugerprocesser skal køre i brugertilstand og kernefunktioner i kernetilstand.

5. Hver brugerproces må maksimalt køre i en fast tidsskive (eng. "time slice"). Under et systemkald i afslutningen af en tidsskive kan brugerprocessen dog tillades at fortsætte kortvarigt.

6. Kernen skal mindst implementere funktionerne `malloc`, `free`, `readdata`, `writedata`, `readline`, `print`, `start_proc`, `exit`, `sleep`, `getpid` og `halt` med typer og funktionalitet som beskrevet i filen `ekernelib.h`; herudover skal kernen implementere klasserne `Semaphore` og `BeskedSem`, også defineret i `ekernelib.h`. Resten af filen samt hjælpefilerne `ekerne.h` og `ekernelib.cc` er implementationsafhængige og ment som en hjælp, og kan ændres efter forgodtbefindende. Filerne findes elektronisk (se afsnit 3), og er vedlagt denne opgavetekst (se side 6).

   Udover de i `ekernelib.h` nævnte krav til typer og funktionalitet, stilles yderligere følgende krav til implementeringen af funktionerne:

   - Alle kernefunktionerne skal udnytte AlphaPC164SX's systemkaldsmekanisme.
   - Der skal anvendes en kernestak pr. proces.
   - `readdata()` og `writedata()` skal støtte læsning fra/skrivning til `COM`– og `ETH0`–enheden og `HD` (en simuleret disk), dog kræves det ikke, at `writedata()` på `ETH0` er afprøvet.
   - `readline` og `print` er en alternativ måde at lave I/O til/fra `COM`, idet der anvendes `NUL`-terminerede strenge.
   - Al I/O skal (undtagen den simulerede til/fra `HD`) foregå ved hjælp af afbrydelser og udeleligt, så uddata til/inddata fra to forskellige processer ikke blandes sammen.
   - Funktionerne `malloc` og `free` skal have funktionalitet som beskrevet i G1 (men med en passende stor hob).
   - Funktionen `sleep` skal blokere den kaldende proces i det specificerede tidsrum målt i millisekunder; der tillades en vis tolerance på grund af fx at behandling af afbrydelser skal gøres færdig.
   - Kernen skal indeholde en sporingsmekanisme i form af et internt dataområde (en cyklisk buffer) med mindst 30 sporingsposter. En post skal genereres ved hvert kernekald, afbrydelse, og tilbagehop til brugerprogrammet, samt i det omfang det er hensigtsmæssigt i forbindelse med den aktuelle implementation. Hver post skal mindst indeholde data om tidspunkt (f.eks. i form af et cyklisk sekvensnummer), kørende proces og årsag.

   Bemærk at `getpid` og `sleep` ikke umiddelbart har nogen anvendelse; kaldene er kun med som forberedelse til K2, men skal indgå i besvarelsen af K1.

7. Kernen skal ved opstart kun oprette én brugerproces, der startes ved (i brugertilstand) at overføre kontrollen til brugerfunktionen `void INIT ()`.

8. Kernen skal kunne genkende en tastekombination fra `COM` (fx tegnet ESC, ASCII 0x1B), som stopper udførelsen som om `halt` blev kaldt.

9. Kernen skal korrekt afvikle de otte udleverede multiprogrammer, der forsøger så vidt muligt at afprøve centrale dele af kernens funktionalitet. Multiprogrammerne kan antages at være fejlfrie i den forstand, at det f.eks. ikke er nødvendigt at håndtere undtagelser som division med nul.

10. Til test af læsning fra netkortet vil en maskine("herodot") regelmæssigt sende små pakker ud på nettet.

11. Til test af `HD`-enheden skal der designes og implementeres et simpelt persistent filsystem, der overholder grænsefladen i `filesys.h`, se side 9. Filsystemet skal implementeres kørende i brugermodus, dvs. kalde kernens funktioner gennem `ekernelib`, og skal anvende `HD` til at lagre permanente data, der ellers ville gå tabt ved slukning af maskinen.

    Der stilles ikke krav til filsystemet om effektivitet mht. udnyttelse af pladsen i lager eller på `HD`, det anvendte antal skrive/læse-operationer eller hastigheden.

2.2 Rapporten

Det udførte arbejde skal beskrives og dokumenteres i en rapport. Kravene til rapporten er:

1. Omfanget må maksimalt være 30 sider, eksklusiv bilag. Der lægges vægt på, at rapporten er velstruktureret og præcis. Materiale, der udleveres i forbindelse med opgaven skal ikke afleveres, men eventuelle ændringer skal angives.

2. Rapporten skal indeholde en analyse af de relevante problemstillinger, f.eks:

    - Procesbeskrivelser samt håndtering af afbrydelser og processkift.
    - Kernens håndtering af afbrydelser fra ydre enheder.
    - Kernens håndtering af dynamisk lagerallokering.
    - Oprettelse og afslutning af processer.
    - Brug og implementering af synkronisering.
    - Processkedulering ("short-term scheduling"), specielt med hensyn til behovet for prioriteret skedulering.

3. Rapporten skal indeholde en kort beskrivelse af kernens funktioner og struktur.

4. Rapporten skal kort beskrive forenklende antagelser og begrænsninger fx i forbindelse filsystemet.

5. Rapporten skal dokumentere korrekt udførelse af de udleverede multiprogrammer og øvrige multiprogrammer anvendt ved afprøvningen, og af en fungerende sporingsmekanisme. Brug f.eks. skærmlog i et xterm-vindue til at dokumentere kommunikationen på den serielle linje. Præciser, hvilke krav kernen opfylder og hvilke krav den *ikke* opfylder.

Rapporten behøver ikke at at omfatte en brugsvejledning.

3 Hjælpemateriale

Til løsning af opgaven udleveres følgende materiale:

- Materiale, der skal anvendes eller bør anvendes (driveren til netkortet):
  1. Den grænseflade (`ekernelib.h`), som multiprogrammeringskernen mindst skal implementere.
  2. Et sæt på otte multiprogrammer (`disk_test.cc`, `filesys_test.cc`, `net_test.cc`, `print_test.cc`, `semaphore_test.cc`, `process_test.cc`, `timeout_test.cc` samt `time_test.cc`), der skal bruges til afprøvning af kernen. Yderligere programmer kan blive udleveret undervejs i opgaveperioden, men er ikke obligatoriske.
  3. Driver til netkortet (`edrv.[h,cc], edrvlo.h`)
  4. Driver til disken `hd.[h,cc])`
- Materiale, der kan benyttes frit efter behov, eller slet ikke:
  1. To hjælpefiler, med forslag til implementering. (`ekerne.h`, `ekernelib.cc`).
  2. Programtekst til en kerne med afbrydelser (`dkerne.[h,cc], kblib.[s,h], pal.h, queues.h, multiprogram.cc` samt `regdefs.h`). Bemærk, at der er tale om andre versioner end ved G1.
  3. Kildekode for initialisering af ydre enheder (`kinit.[h,cc]`)
  4. Primitiv dynamisk allokering (`kmalloc.[h,cc]`) og sporing (`rf.[h,cc]`).
  5. Udskriftsprocedurer (`OutputClass.[h,cc]`)
  6. En makefil. (`Makefile`)

  Der vil eventuelt blive stillet andre filer til rådighed, hvis behovet opstår.

Ovennævnte filer kan findes i kataloget: `~dat1f/K1/src` og kan også findes gennem kursets hjemmeside.

4 Råd til løsning af opgaven

Under udvikling af en kerne er det nemt at lave fejl. Selv om mængden af kode er begrænset, kan det være svært at finde fejlene. Derfor kan det være en fordel at starte med noget, der virker, trinvist tilføje ny funktionalitet og for hvert trin sikre sig, at det stadig virker. For arbejdet med rapportopgaven har det yderligere den fordel, at hvis man går i stå undervejs, har man stadig en kørende kerne, selv om den måske ikke helt opfylder kravene i afsnit 2.1, herunder ikke korrekt kan afvikle et eller flere af de udleverede multiprogrammer. Et forslag til udviklingsprocessen kan være:

1. Start med at udvide kaldgrænsefladen for d-kernen med systemkald så brugerprogrammer kan køre i brugertilstand. Implementer sporingsmekanismen. Ignorer afbrydelser fra uret. Brug den udleverede kode til initialiseringen af de ydre enheder.
2. Tilføj funktionerne for dynamisk lageradministration fra G1 opgaven, så de kan kaldes gennem systemkald.

3. Tilføj funktionaliteten for dynamiske processer, herunder systemkaldet `getpid`.
4. Tilføj funktionalitet for kald af `INIT` ved opstart.
5. Udvid kernen så den kan håndtere afbrydelser fra uret mens maskinen er i brugertilstand.
6. Tilføj systemkaldet `sleep`.
7. Udvid kernen så den kan håndtere afbrydelser fra uret mens maskinen er i kernetilstand.
8. Tilføj disk og dernæst et filsystem.

5 Kørselsvejledning

Programudvikling og afprøvning skal foregå på DIKU's udviklings- og kernemaskiner. Der er en reservationsordning for kerne-alpha'erne. Driftsstabiliteten — specielt i opgaveperiodens sidste dage — kan være svingende på grund af stort pres, på trods af edb-afdelingens indsats. Omfanget af instruktorvagt vil fremgå af opslag og kursushjemmesiden.

A Kaldgrænseflade

```
// ------------------------------------------------------------------------
// $Id: ekernelib.h,v 1.7 2002/03/08 13:46:27 di01i036 Exp $
// ekernelib.h
// Kaldgrænseflade for kernen.
// En besvarelse af K1/2002 skal implementere KØRETIDSBIBLIOTEKET nedenfor.
// ------------------------------------------------------------------------
// $Log: ekernelib.h,v $
// Revision 1.7  2002/03/08 13:46:27  di01i036
// K1 2002 Release 2
//
// Revision 1.7  2002/03/08 13:43:28  di01i036
// Tilrettede tekster - release 2
//
// Revision 1.6  2002/03/01 13:08:42  di01i036
// Justering af kommentarer
//
// Revision 1.5  2002/03/01 13:07:07  di01i036
// K1 2002 release
//
// K1 2001-03-02 Første version
//
// ----------------------
#ifndef EKERNELIB_H
#define EKERNELIB_H

#include <stddef.h>
#include "queues.h"

// ------------------------------------------------------------------------
//                      STANDARDGRÆNSEFLADE TIL
//                KØRETIDSBIBLIOTEKET - SKAL IMPLEMENTERES.
// ------------------------------------------------------------------------
//
// Der må ændres i implementationen, dvs. kroppen, af nedenstående,
// men funktionerne (metoderne) skal implementeres med de navne og typer
// der er angivet her, og de skal have den angivne funktionalitet.
// ------------------------------------------------------------------------

// ---------------------------------------------
// Vigtige konstanter og typer
// ---------------------------------------------

// ---------------------------------------------
// Typer
// ---------------------------------------------

#ifndef EKERNE_H

#include "hd.h"

typedef unsigned long ulong;

enum device {
COM = 0,
ETH0,
HD
};

enum SysErr {
SYS_OK = 0,
SYS_ERR_TIMEOUT = -1,
SYS_ERR_BAD_SYSCALL = -2,
SYS_ERR_DEVICE_BUSY = -3,
SYS_ERR_DEVICE_ERROR = -4
```

```
};

// ------------------------------------------------------------------------
// Interne strukturer
// ------------------------------------------------------------------------

struct Process : public Queueable {
void* sp; // Processens brugerstakpeger
void* storage; // Starten af procens (kerne-) allokerede lager.
ulong pid;
};

struct Buffer : public Queueable {
  char* data; // Peger til data i bufferen
  size_t size; // Størrelse på data
};

struct KSem {
        unsigned int value;
        Queue<Process> queue;
};

struct KBSem {
        Queue<Buffer> buffers;
        KSem counter;
};

#endif

// -------------------------------------------
// Ydre enheder
// -------------------------------------------

#ifndef _NET_H
typedef unsigned char MAC[6];
#endif

void print (char* s);
// Udskriv nultermineret til com port.
//
int readline (char* s,size_t max);
// Læs nultermineret fra com. Returværdien er antallet af tegn læst.
//
extern unsigned short dummyshort;
// Brugt til default værdier til netværket
//

#ifndef EKERNE_H
typedef union ADDR {
  MAC netadr;
  size_t diskadr;
  ADDR(const MAC s) {memcpy(netadr,s,6);} // the size of MAC is 6
  ADDR(const size_t adr) {diskadr=adr;}
};
// Union til addresse håndtering
//
#endif

size_t writedata (char* s,
      size_t size,
      device dev = COM,
      const ADDR* dst=NULL);
// Skriv 'size' enheder data til den ydre enhed (COM,ETH0,HD).
// Se om dataenheder og addressering nedenfor under readdata(...).
//
size_t readdata (char* s,
```

```
     size_t max,
     device dev = COM,
     ADDR *src = NULL,
     ADDR *dst = NULL);
// Læs 'max' enheder data fra den ydre enhed (COM,ETH0,HD).
// Returværdien er antallet faktisk læste dataenheder.
// En dataenhed er:
//    COM    1 byte
//    ETH0   1 byte
//    HD     en blok af størrelse HD_BLOCKSIZE (defineret i hd.h)
// Læsning kan returnere færre enheder hvis den termineres:
//   COM henter linietermineret, dvs. indtil end-of-line (\n)
//   ETH0 henter pakketermineret, dvs. en hel IEEE 802 ramme.
//   HD henter indtil slut på disk.
// src og dst er adresser med følgende betydning, afhengigt af den
// ydre enhed:
//    COM    src og dst benyttes ikke
//    ETH0   src og dst er 48 bit MAC adresser fra IEEE 802 rammen
//           Se mere i net.h
//    HD     src (read) og dst (write) angiver bloknummeret >=0
//
// ---------------------------------------------
// Lageradministration
// ---------------------------------------------

// Reserver et s byte stort lagerområde, og returner en pointer
// til den første adresse i dette lagerområde. Hvis der ikke kan
// findes et lagerområde, der er stort nok, returneres 0.
//
void* malloc (long s);

// Frigiv et tidligere reserveret lagerområde, hvis første adresse er p.
// Hvis p ikke er første adresse i et tidligere reserveret lagerområde, er
// resultatet ikke defineret; det er f.eks. tilladt kernen at gå ned i denne
// situation. (Men det ville selvfølgelig være bedst, hvis den lod være).
//
void free (void* p);

// ---------------------------------------------
// Procesadministration
// ---------------------------------------------

// Start en ny proces med hovedfunktion startAddr og
// stakstørrelse stack_size quadword.
// Returværdien er process id'et på barneprocessen
//

ulong start_proc (void (*startAddr) (),
 unsigned long stack_size);

ulong getpid(); // Få process id'et.
void sleep(long n);   // Bloker i n millisekunder
void exit();    // Terminer den kørende proces.

// ---------------------------------------------
// Overordnede funktioner
// ---------------------------------------------


// Stands kernen.
//
void halt();

// ---------------------------------------------
// Semaforer
// ---------------------------------------------
```

```
//
// Klasisk tællesemafor
//
class Semaphore {
        KSem theSem;
        friend int main ();
 public:
        void Init (int i);
        void Wait ();
        void Signal ();
};

//
// Beskedsemafor til data.
// Semaforen modtager ved signal netop det antal bytes der angives og
// ved vent returneres antallet af bytes op til max der er kopieret til s.
// Er beskeden i semaforen længere end anmodet slettes resten
// af beskeden fra semaforen.
//
// Semaforen skal garantere dataintegritet og skal derfor selv sørge for
// allokering og deallokering af buffere.
// Brugerprocesserne skal tilsvarende selv sørge for allokering og deallokering
//
// Semaforen skal kunne indeholde mindst 5 beskeder.
class BeskedSem {
 private:
        KBSem theSem;
 public:
void Init();
size_t Wait(char *s,size_t max);
void Signal(char *s,size_t size);
};

#endif /* EKERNELIB_H */
```

B Filsystem

```
// $Id: filesys.h,v 1.3 2002/03/08 13:46:27 di01i036 Exp $
// filesys.h
// K1 2002
// header file for a simple file system
// $Log: filesys.h,v $
// Revision 1.3  2002/03/08 13:46:27  di01i036
// K1 2002 Release 2
//
// Revision 1.5  2002/03/08 13:43:28  di01i036
// Tilrettede tekster - release 2
//
// Revision 1.4  2002/03/08 09:17:04  di01i036
// Forbedret version
//
// Revision 1.3  2002/03/01 12:26:52  di01i036
// Kommentar om parallel adgang til en fil
//
// Revision 1.2  2002/03/01 12:15:04  di01i036
// Justeret headerfilerne (ekerne.h ekernelib.h filesys.h) indbyrdes
//
//
// If the implementation sets limits on the maximum size of a file
// (other than the capacity of the disk) the limit may be set in
// the definition of MAXFILESIZE, measured in bytes. This name must
// have a value of at least 4096 bytes.
//
// The following names may be used in conditional compilations
```

```
// to include calls of extra facilities from a user multiprogram
//
// XTRA_FUCTIONS   Non-mandatory functions below are present
// VTOC_FUNCTIONS  Directory functions are present
//
// Uncomment depending of the actual implementation
// #define XTRA_FUCTIONS
// #define VTOC_FUNCTIONS
// #define MAXFILESIZE 4096

#define MAXFILENAME 6
#define ACTUALFILENAMELEN 8
// ACTUALFILENAMELEN = MAXFILENAME + 2 (room for \0)
// An implementation may limit the character set allowed, but
// must include the digits 0 1 2 3 4 5 6 7 8 9. Unprintable
// characters and characters with a special meaning should be excluded.

// results from the file system
enum FSResultater {
  FS_OK      =-1,
  FS_EOF     =-2,  // End-Of-File met
  FS_NOTFOUND=-3,  // File not present in file system
  FS_NOROOM  =-4,  // File system is full (number of files)
  FS_FULL    =-5,  // File system is full (data)
  FS_EXISTS  =-6,  // File does already exist
  FS_NOMEM   =-7,  // Cannot allocate buffers etc.
  FS_ERROR   =-8   // Bad call
};

enum FPositions {
  POS_AT_EOF=-1,
  POS_AT_START=0,
};


// Mandatory
  void InitFileSystem(bool formatdisk);
// init must be called once to initialize the file system and
// will format the disk device if formatdisk is true. If XTRA_FUCTIONS
// is not defined, the call may format the disk even if formatdisk is
// false. This is implementation dependent.

// name may be truncated to MAXFILENAME if longer.
  int CreateFile(char *name);
// create empty file
  int FileRead(char *name, char *buf, size_t maxlength, size_t position);
  int FileWrite(char *name, char *buf, size_t length, size_t position);
// read and write returns number of bytes>0 or (negative) resultcode
// The operation is taking place at the specified byte position in the file.
// position<file length; 0 (POS_AT_START) start of file, -1 (POS_AT_EOF)
// at end of file. A read at EOF returns FS_EOF.
// If FileWrite returns less than 'length', a problem occurred, and
// a FileWrite with the remaining data could be used to either repair
// or get a result code with the reason.
// Simultaneous access from concurrent processes to the same file may result
// in inconsistent or unpredictable results. Serialization or locks is
// the reponsibility of the application processes.

#ifdef XTRA_FUCTIONS
  int FileGetLength(char * name); // size of file or negative result code
  int FormatDisk(int maxfiles,
 int clustersize, int firstfree, int numclusters);
// If relevant in a given implementation:
//   maxfiles is the maximum number of files
//   clustersize is the size of an allocation unit in diskblocks
//   firstfree points to the start of the data area (forst block in cluster)
```

```
//   numclusters is the the total length of the data area (in clusters)
// Returns a result code
#endif

#ifdef VTOC_FUNCTIONS
  int FileDelete(char *name);
// Delete the file and free the data area.

// this printing routine needs an output system
  void PrintVTOC();
// Print information about all files, e.g. name and length
  void PrintInodes();
#endif
```

C Implementationsforslag

C.1 Fælles header for kerne og bibliotek

```
// ------------------------------------------------------------------------
// $Id: ekerne.h,v 1.5 2002/03/08 13:46:27 di01i036 Exp $
// ekerne.h
// Fælles strukturer for kerne og kaldbibliotek.
// Implementationsafhængig.
// ------------------------------------------------------------------------
// $Log: ekerne.h,v $
// Revision 1.5  2002/03/08 13:46:27  di01i036
// K1 2002 Release 2
//
// Revision 1.7  2002/03/08 13:43:27  di01i036
// Tilrettede tekster - release 2
//
// Revision 1.6  2002/03/07 20:40:39  di01i009
// sleep implementeret og testet
//
// Revision 1.5  2002/03/01 16:15:11  di01i036
// Genindsætning af YIELD
//
// Revision 1.4  2002/03/01 12:15:04  di01i036
// Justeret headerfilerne (ekerne.h ekernelib.h filesys.h) indbyrdes
//
// Revision 1.3  2002/02/21 22:28:33  di01i009
// usr8 kører
//
// K1 2001-03-02 Første version
//
// ----------
// Compile flags:
//  NOCOMECHO defined if kernel should not echo on COM input
//

#ifndef EKERNE_H
#define EKERNE_H

#include <stddef.h>

// ------------------------------------------------------------------------
//                      Fælles typer mv.
// ------------------------------------------------------------------------

#include "kblib.h"
#include "queues.h"
#include "net.h"
#include "hd.h"
```

```
typedef unsigned long ulong;

// ------------------------------------------------------------------------
//                      Implementationsafhængige ting
// ------------------------------------------------------------------------

// ------------------------------------------------------------------------
// Interne strukturer
// ------------------------------------------------------------------------


struct KernelStack {
unsigned long r9,r10,r11,r12,r13,r14,r15,r26,r28;
unsigned long stackFrameSize;
};

struct Process : public Queueable {
void* sp; // Processens brugerstakpeger
void* storage; // Starten af procens (kerne-) allokerede lager.
ulong pid;
  // Antallet af millisekunder den skal vente endnu
  long sleep;
};

struct Buffer : public Queueable {
  char* data; // Peger til data i bufferen
  size_t size; // Størrelse på data
};

struct KSem {
        unsigned int value;
        Queue<Process> queue;
};

struct KBSem {
        Queue<Buffer> buffers;
        KSem counter;
};

// tracebuffer - skal mindst indeholde

typedef struct tinfo {
  unsigned int seqno;   // tidsstempel, cyklisk
  int          event;// hvad skete
  ulong        procid;  // kørende proces
  long         param1;  // eventuelle parametre
  long         param2;
} traceinfo;

enum traceID {
  syscall_INITSEM,
  syscall_WAIT,
  syscall_SIGNAL,
  syscall_INITBSEM,
  syscall_BESKEDWAIT,
  syscall_BESKEDSIGNAL,
  syscall_READLINE,
  syscall_WRITELINE,
  syscall_READDATA,
  syscall_WRITEDATA,
  syscall_START_PROC,
  syscall_SLEEP,
  syscall_EXIT,
  syscall_MALLOC,
  syscall_FREE,
  syscall_HALT,
```

```
  syscall_GETPID,
  interrupt_COM,
  interrupt_ETH,
  interrupt_RTC,
  return_call,
  return_int,
  k_initializing
};


// event              param1     param2
// -----------------  --------   --------
// syscall_INITSEM    sem        i
// syscall_WAIT       sem
// syscall_SIGNAL     sem
// syscall_INITBSEM   sem        i
// syscall_WAITBESKED sem
// syscall_SIGNALBESK.sem
// syscall_READLINE   dev        bufaddr
// syscall_WRITELINE  dev        bufaddr
// syscall_START_PROC start      stacksize
// syscall_SLEEP      n
// syscall_EXIT
// syscall_MALLOC     s
// syscall_FREE       p
// syscall_HALT
// syscall_GETPID
// interrupt_COM      in/out
// interrupt_ETH      in/out
// interrupt_RTC
// return_call        result
// return_int
// user_XX            user1      user2
// k_initializing
// ---------------------------------------
//
// ------------------------------------------------------------------------
// Lav-niveau kaldgrænseflade
// ------------------------------------------------------------------------
enum SysCallId {
        SYS_INITSEM,
        SYS_WAIT,
        SYS_SIGNAL,
        SYS_READLINE,
        SYS_WRITELINE,
        SYS_FORK,
        SYS_EXIT = 7,
        SYS_MALLOC,
        SYS_FREE,
        SYS_HALT,
        SYS_YIELD,
SYS_GETPID,
SYS_SLEEP,
        SYS_WRITEDATA,
SYS_INITBSEM,
SYS_WAITBESKED,
SYS_SIGNALBESKED,
SYS_READDISK,
SYS_WRITEDISK,
SYS_READPACKET,
SYS_WRITEPACKET
};

// ---------------------------------------------
// Køretidsbibliotek - hjælpefunktioner etc.
// ---------------------------------------------
```

```
extern "C" unsigned long callsys (SysCallId ...);

inline unsigned strlen (char* s)
{
        unsigned n;
        for (n = 0; *s; ++s, ++n)
                /* NOP */ ;
        return n;
}


// -----------------------------------------
// Vigtige konstanter og typer
// -----------------------------------------

enum device {
COM = 0,
ETH0,
HD
};

enum SysErr {
SYS_OK = 0,
SYS_ERR_TIMEOUT = -1,
SYS_ERR_BAD_SYSCALL = -2,
SYS_ERR_DEVICE_BUSY = -3,
SYS_ERR_DEVICE_ERROR = -4
};

// Union til addresse håndtering
//
typedef union ADDR {
  MAC netadr;
  size_t diskadr;
  ADDR(const MAC s) {memcpy(netadr,s,6);} // the size of MAC is 6
  ADDR(const size_t adr) {diskadr=adr;}
};

#endif /* EKERNE_H */
```

C.2 Skitse til implementation af bibliotek

```
// ------------------------------------------------------------------------
// $Id: ekernelib.cc,v 1.6 2002/03/01 12:31:17 di01i036 Exp $
//
// Skitse til implementation af kaldbibliotek.
// ------------------------------------------------------------------------
// $Log: ekernelib.cc,v $
// Revision 1.6  2002/03/01 12:31:17  di01i036
// Justeret kommentaren til sleep()
//
// Revision 1.5  2002/02/25 14:38:40  di01i009
// all compiles and links
// --------------------------

#include "ekerne.h"
#include "ekernelib.h"

// -----------------------------------------
// Ydre enheder
// -----------------------------------------

// Pass MAC_broadcast in dst for broadcast.
//int writepacket(void  *buf,
```

```
//                      short length,
//                      const MAC  dst = MAC_broadcast) {
//  return callsys(SYS_WRITEPACKET,
//                 buf,
//                 length,
//                dst);     // pass ptr to the dst MAC field. (yes! no '&')
//} /* WritePacket */
//
//
//int readpacket(void * buf, short max,
//               MAC &src = MAC_dummy, MAC &dst = MAC_dummy,
//        unsigned short &status = dummyshort)
//{
//  return callsys(SYS_READPACKET, buf, max, &src, &dst, &status);
//}

// Udskriver nul-termineret streng.
//

void print(char * s){
  callsys (SYS_WRITELINE, s);
}
int readline (char * s, size_t max){
  return (int) callsys (SYS_READLINE, s, max);
}
// Læs max size bytes fra dev, eller timeout.
// Returner antal byte der blev læst, eller SYS_ERR_TIMEOUT,
// (negativt).
//

size_t readdata (char* s,
      size_t max,
      device dev = COM,
      ADDR *src = NULL,
      ADDR *dst = NULL)
{
  switch (dev) {
  case COM:
    return callsys(SYS_READLINE,s,max);
    break;
  case ETH0:
    return callsys(SYS_READPACKET, s, max, src, dst, dummyshort);
    break;
  case HD:
    return callsys (SYS_READDISK, s, max, src);
    break;
  }
  return SYS_ERR_DEVICE_ERROR;
}
// Udskriv size bytes til dev.
// Returnerer antal byte, der blev skrevet.
//
size_t writedata (char* s,
       size_t max,
       device dev = COM,
       const ADDR *dst = NULL)
{
  switch (dev) {
  case COM:
    return callsys(SYS_WRITEDATA,s,max);
    break;
  case ETH0:
    return callsys(SYS_WRITEPACKET, s, max, dst);
    break;
  case HD:
    return callsys (SYS_WRITEDISK, s, max, dst);
```

```
    break;
  }
  return SYS_ERR_DEVICE_ERROR;
}

// ------------------------------------------
// Lageradministration
// ------------------------------------------

// Reserver et s byte stort lagerområde, og returner en pointer til den første
// adresse i dette lagerområde. Hvis der ikke kan findes et lagerområde, der
// er stort nok, returneres 0.
//
void* malloc (long s)
{
return (void*) callsys (SYS_MALLOC, s);
}

// Frigiv et tidligere reserveret lagerområde, hvis første adresse er p.
// Hvis p ikke er første adresse i et tidligere reserveret lagerområde, er
// resultatet ikke defineret; det er f.eks. tilladt kernen at gå ned i denne
// situation. (Men det ville selvfølgelig være bedst, hvis den lod være).
//
void free (void* p)
{
callsys (SYS_FREE, p);
}

// ------------------------------------------
// Procesadministration
// ------------------------------------------

// Start en ny proces med hovedfunktion startAddr og
// stakstørrelse stack_size quadword.
// Returværdien er process id'et på barneprocessen
//
ulong start_proc (void (*startAddr) (),
 unsigned long stack_size)
{
return callsys (SYS_FORK, startAddr, stack_size);
}


// Få process id'et
//
ulong getpid()
{
return callsys (SYS_GETPID);}

// Bloker i n millisekunder
//
void sleep(long n)
{
callsys (SYS_SLEEP,n);
}

// Terminer den kørende proces.
//
void exit()
{
callsys (SYS_EXIT);
}

// ------------------------------------------
// Overordnede funktioner
// ------------------------------------------
```

```
// Stands kernen.
//
void halt()
{
callsys (SYS_HALT);
}

// ------------------------------------------
// Semaforer
// ------------------------------------------

/* Offentlige medlemmers implementering (og KUN implementering)
   er også overladt til implementøren. Igen giver vi
   et forslag: */

void Semaphore::Init (int i) { callsys (SYS_INITSEM, &theSem, i); }
void Semaphore::Wait ()      { callsys (SYS_WAIT, &theSem);       }
void Semaphore::Signal ()    { callsys (SYS_SIGNAL, &theSem);     }

void BeskedSem::Init() { callsys (SYS_INITBSEM, &theSem);}
size_t BeskedSem::Wait(char *s,size_t max) {
  return callsys (SYS_WAITBESKED,&theSem,s,max);}
void BeskedSem::Signal(char *s,size_t size) {
  callsys (SYS_SIGNALBESKED,&theSem,s,size);}
```