

TTU1918

Arvutusprotsesside organiseerimine

Sardsüsteem = programm + mikroskeem
[rakendus + OS] + [digitaal + analoog]

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 1

TTU1918

Von Neumanni arhitektuur

- Mälu
- Protsessor
- Juhitseade
- ALU

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 2

TTU1918

Mälu & Protsessor

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 3

TTU1918

Mälu & Protsessor

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 4

TTU1918

Sardtarkvara süntees

- Süsteemi analüüs
- RTOS & rakenduste valimine
- Tükeldamine
- Liideste defineerimine
- Moodulite projekteerimine
- Tarkvara integreerimine
- Tarkvara paigaldamine (ROM)
- Tarkvara kompileerimine & linkimine

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 5

TTU1918

Sardtarkvara

- Üks ja ainuke rakendusprogramm
 - Mälujaotus
 - programm – rakendus + süsteem
 - andmed – staatilised / dünaamilised / pinu
 - Programmi töö
 - alglaadimine
 - protsessori initialiseerimine
 - programmi initialiseerimine
 - programm tsüklis
 - süsteempöördused

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 6

Mälajaotus – üks rakendus

0000 (füüsiline) mälu-aadress FFFF

ROM RAM I/O

ROM
boot .init main() ... sys.f-ns

RAM
.data .heap .stack

füüsiline moodul vallatakse/aktiveeritakse dekodeerimise abil

address-in ROM RAM I/O DEV I/O DEV andme-sin

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 7

Sardtarkvara

☞ Mitu rakendusprogrammi (ülesannet)

- ☒ lisaks üksikule programmile
- ☒ Mälajaotus
 - ☒ eri programmide andmed/programm eri kohtades
 - ☒ andmete kaitsmine
 - MMU – Memory Management Unit
- ☒ Programmide töö
 - ☒ süsteemi initsialiseerimine
 - ☒ ülesannete vahetamine
 - planeerimine, CPU sisu salvestamine, ...

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 8

Mälajaotus – mitu rakendust

☞ Iga rakendusprogramm füüsiliselt erineval aadressil

- ☒ rakenduse sisemised mälupiirkonnad samad (.data, .heap jne.)
- ☒ MMU-d kasutatakse mälu kaitsmiseks valepöörde eest
- ☒ linkimine keerukas – aadressid vaja eraldi ette anda

0000 (füüsiline) mälu-aadress FFFF

ROM RAM I/O

ROM task 1 task 2 task 3

RAM task 1 task 2 task 3

0000 "task 2" mälajaotus (aadressväli) FFFF

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 9

Mälajaotus – mitu rakendust

☞ Kõikide rakendusprogrammide jaoks on mälajaotus ühesugune – nn. virtuaalne aadressväli (virtuaalmälu)

- ☒ rakenduse sisemised mälupiirkonnad samad (.data, .heap jne.)
- ☒ MMU vajalik nii mälupiirkondade kaitsmiseks kui ka virtuaalse aadressi "tõlkimisel" füüsiliseks aadressiks
- ☒ linkimine lihtne – sama kõikidele rakendustele

0000 (füüsiline) mälu-aadress FFFF

ROM RAM I/O

MMU

0000 "task 2" mälajaotus (virtuaalne aadressväli) FFFF

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 10

Mälajaotus

☞ ROM

- ☒ RO segment (mitu programmi/kasutajat)
- ☒ .boot piirkond
 - ☒ CPU initsialiseerimine
 - ☒ katkestusvektorid
 - ☒ pinu jt. mälupiirkondade aadressid
 - ... või teostatakse (RT)OS-i poolt
- ☒ .init piirkond
 - ☒ parameetrid programmile / tulemus OS-le
 - ☒ objektide algväärtustamine (nt. C++)

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 11

Mälajaotus

☞ ROM (jätkub...) .text piirkond

- ☒ main() ...
 - ☒ rakendusprogrammid
 - ☒ kompileeritud või teekidest .o-failidena
- ☒ sys.f-ns
 - ☒ standardised alamprogrammid
 - kompilaatori-spetsiifilised
 - ☒ süsteemsed alamprogrammid
 - OS-spetsiifilised
 - S/V draiverid jms.

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 12

Mälujaotus

- RAM
 - RW segment (mitu programmi/kasutajat)
- .data piirkond
 - staatilised (globaalsed) muutujad/andmed
- .heap piirkond
 - dünaamilised muutujad/andmed
 - kasutatud/vabade plokkide nimekiri
 - prügikogumine (garbage collection) aega-ajalt vajalik
 - vajadusel suurendatakse OS-i poolt
- .stack piirkond
 - lokaalsed muutujad/andmed (pinu)

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 13

Mälukujutis

- a.out / .exe / .com
 - OS-i poolt mällu laetav programmi "toorik"
- Sisaldab
 - programmi initsialiseerimine & sisu
 - algväärtustatud andmed (muutujad & konstandid)
 - osa globaalsetest muutujatest
- OS-i poolt teostatakse
 - mälujaotuse ette valmistamine
 - piirkondade eraldamine (MMU), viidad registritesse, ...
 - programmi juhtimine
 - alustamine, katkestamine, OS-i pöördused, ...

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 14

Kompileerimine

- Etapid
 - teisendamine abstraktseks operatsioonide jadaks
 - optimeerimine
- Programmi teisendamine operatsioonide jadaks
 - Keelekonstruktsioon → baas-plokkide jada
 - Baas-plokk ~ avaldis

```
for (i=0; i<5; i++) { foo(); }
```

```
<for_cycle> ::= 'for' ' (' (<expr>* ';' (<expr>* ';' (<expr>* ';' [<statement>] ';' ;
```

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 15

Kompileerimine

- Programmi teisendamine operatsioonide jadaks
 - Baas-plokk ~ avaldis
 - Avaldis → operatsioonide jada
 - nn. Poola kuju (Polish notation, prefix notation, Jan Łukasiewicz)
 - operatsioonide prioriteetidid
 - primaarsed avaldised
 - sulud, muutujad, funktsioonid, ...
 - pinu või virtuaal-registrid
 - Ühised alamavaldised

```
x = a + b + c ;
r1 <- + a b
z2 <- + r1 c
x <- r2

y = a + b * c ;
r1 <- * b c
z2 <- + a r1
y <- r2

z = (a + b) * c ;
r1 <- + a b
z2 <- * r1 c
z <- r2
```

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 16

Kompileerimine

- Avaldis VHDL-s (BNF, lihtsustatud)

```
<expression> ::= <relation> [ <logic_operation> <relation> ] *
<logic_operation> ::= 'and' | 'or' | 'xor' | 'nand' | 'nor' | 'xnor'
<relation> ::= <shift_expression> [ <relational_operation>
<relational_operation> ::= '<' | '>' | '<=' | '>=' | '<>' | '<=' | '>='
<shift_expression> ::= <simple_expression> [ <shift_operator> ]
<simple_expression> ::= <simple_expression> [ <simple_expression> ]
<shift_operator> ::= 'sll' | 'srl' | 'sla' | 'sra' | 'rol' | 'ror'
<simple_expression> ::= [ <sign> ] <term> [ <adding_operator> <term> ] *
<sign> ::= '+' | '-'
<adding_operator> ::= '+' | '-' | 'g'
<term> ::= <factor> [ <multiplying_operator> <factor> ] *
<multiplying_operator> ::= '*' | '/' | 'mod' | 'rem'
<factor> ::= <primary> [ '**' <primary> ] |
'abs' <primary> | 'not' <primary>
<primary> ::= <literal> | <identifier> | '(' <expression> ')' |
<function_call>
<function_call> ::= <identifier> '(' [ <expression>
' ' <expression> ] * ')' ;
```

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 17

Kompileerimine

- Avaldis VHDL-s (lihtsustatud)

```
<relation> ::= <shift_expression> [ <relational_operation>
<relational_operation> ::= '<' | '>' | '<=' | '>=' | '<>' | '<=' | '>='
<shift_expression> ::= <simple_expression> [ <shift_operator> ]
<simple_expression> ::= <simple_expression> [ <simple_expression> ]
<shift_operator> ::= 'sll' | 'srl' | 'sla' | 'sra' | 'rol' | 'ror'
<simple_expression> ::= [ <sign> ] <term> [ <adding_operator> <term> ] *
<sign> ::= '+' | '-'
<adding_operator> ::= '+' | '-' | 'g'
<term> ::= <factor> [ <multiplying_operator> <factor> ] *
<multiplying_operator> ::= '*' | '/' | 'mod' | 'rem'
<factor> ::= <primary> [ '**' <primary> ] |
'abs' <primary> | 'not' <primary>
<primary> ::= <literal> | <identifier> | '(' <expression> ')' |
<function_call>
<function_call> ::= <identifier> '(' [ <expression>
' ' <expression> ] * ')' ;
```

- Võrdlustehet (avaldist) analüüsiv lõik parserist (osa kompilaatorist)

```
static boolean ParseRelation(cExpression &expression, cOperand &result)
{
  char const *op=NULL; cOperand operand1,operand2;
  if (ParseShift(expression,operand1)) return TRUE;
  if ( NEW_MATCH (VHDL_EQ) ) op="=";
  else if ( REP_MATCH (VHDL_NE) ) op!="=";
  else if ( REP_MATCH (VHDL_LT) ) op("<";
  else if ( REP_MATCH (VHDL_LE) ) op("<=";
  else if ( REP_MATCH (VHDL_GE) ) op(">=";
  else if ( REP_MATCH (VHDL_GT) ) op(">";
  else { ParsePUTBack(); result=operand1; return FALSE; }
  if (ParseShift(expression,operand2)) return TRUE;
  if (AddExpOp(expression,result,op,BOOL_OPER,operand1,operand2)) return TRUE;
  return FALSE;
}
```

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 18

Kompileerimine

- Alamprogrammid
 - parameetrid ja naasmisaadress pinus
- Muutujad
 - lokaalsed – pinu (stack)
 - globaalsed – üldmälu (data)
 - dünaamilised – (muutuv) üldmälu laiendus (heap)
- Kompilaatori väljundiks on nn. objekt-fail (.o)
 - eraldi tabel märgendite jaoks (lahendab linkur)
- Kompilaatori tüüp sõltub protsessorist
 - erinevad OS-d võivad kasutada sama kompilaatorit

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 19

Kompileerimine – näide

☞ C-programm

```
#include <stdio.h>
int main (void) {
    printf ("Hello, world!\n");
    return 0;
}
```

☞ Assembler-kood

```
.file "hello.c"
.section .rodata
.LC0:
.string "Hello, world!"
.text
.globl main
.type main, @function
main:
    leal 4(%esp), %ecx
    andl $-16, %esp
    pushl -4(%ecx)
    pushl %ebp
    movl %esp, %ebp
    pushl %ecx
    subl $4, %esp
    movl $.LC0, (%esp)
    call puts
    movl $0, %eax
    addl $4, %esp
    popl %ecx
    popl %ebp
    leal -4(%ecx), %esp
    ret
.size main, .-main
.ident "GCC: (GNU) 4.1.2 20061115 (pre-release) (SUSE Linux)"
.section .note.GNU-stack,"",@progbits
```

☞ Assembler-koodi saamine

```
> gcc -S hello.c
```

☞ Nimede tabel objekt-failis

```
> nm hello.o
00000000 T main
00000000 U puts
```

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 20

Linkimine

- Linkur seob erinevad objekt-failid, lahendab märgendite asukohad mälus ja loob programmi mälu kujutise (täidetav fail - .com/.exe/a.out/...)
- OS määrab mälu kaardi
 - programm (code) aadressist 0 (ainult loetav)
 - andmed loetavas ja kirjutatavas segmendis
- Staatiline linkimine
 - kõik alamprogrammid paiknevad mälu kujutises
- Dünaamiline linkimine
 - alamprogrammid laetakse vastavalt vajadusele

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 21

Linkimine

☞ Objekt-fail

- märgendid lahendamata
- koosneb kahest osast
 - programmi kood
 - märgendi sisu 0000
 - märgendite tabel
 - nimi & asukohad koodis

☞ Linkur

- kogub kokku globaalsed märgendid
- otsib üles objekt-failid, kus need märgendid on
- staatiliselt linkimisel tühi koht asendatakse konkreetse aadressiga

☞ Mälukujutise linkurile

- lihtsustatud

```
OUTPUT_FORMAT ("elf32-arm",
               "elf32-arm", "elf32-arm")
OUTPUT_ARCH (arm)
ENTRY (_boot)
/* Forcing static linking */
_DYNAMIC = 0;
SECTIONS
{
    /* Commands executed at exceptions */
    . = 0x0;
    .boot : { *(.boot) }
    /* Read-only sections */
    . = 0x040;
    .text : { *(.text) } = 0
    /* Data segment address adjustment */
    . = ALIGN(0x020);
    .data : { *(.data) }
    .dynamic : { *(.dynamic) }
}
```

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 22

Märgendite tabel – näide

```
/* GCD */
#include <stdio.h>

#define MX 8
const int a[MX] = { -27, 33, 245, 121, 52, 333, 125, 422 };
static const int b[MX] = { 18, 256, 45, 11, 452, 121, 625, 312 };
int iter;

static int gcd(int xi, int yi) {
    int x=abs(xi), y=abs(yi); iter=0;
    while (x != y) {
        if (x < y) y-=x; else x-=y;
        iter++;
    }
    return x;
}

int main() {
    int i,x;
    for (i=0; i<MX; i++) {
        x=gcd(a[i],b[i]);
        printf("%4d %4d: %4d %d\n",a[i],b[i],x,iter);
    }
    return 0;
}
```

```
> nm god.o
00000000 R a
00000020 R b
00000000 t gcd
00000004 C iter
00000068 T main
U printf
```

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 23

OS ülesanded

- Rakenduste juhtimine
 - mäluhaldus
 - rakenduste tööaja eraldamine
 - planeerimine, aja jaotus, saalimine (swap), ...
- Teenused
 - kommunikatsioon
 - rakenduste vaheline, internet, ...
 - failisüsteem
 - muud päringud
 - füüsiline s/v, ...

© Peeter Ellervee sardsüsteemid - arvutusprotsessid 24