Sardtarkvara

Erkki Moorits

Cybernetica AS, Navigatsioonisüsteemide osakond

Loengud

- Programmeerimiskeeled
- Debugimine
- Optimeerimine, ohtlikud koodilõigud
- Operatsioonisüsteemid ja draiverid

Loengu eesmärk

Anda ülevaade sardsüsteemides ja enamlevinud operatsioonisüsteemide kernelites kasutatavatest programmeerimiskeeltest, nende plussidest ja miinustest.

Sardtarkvara erinevus „tavalisest“ tarkvarast

- Sardtarkvara vead kipuvad enamjaolt katastroofilisemalt ilmnema
- Sardtarkvara funktsionaalsus sõltub väga suurel määral riistvarast
- Sardtarkvara kirjutamiseks ja testimiseks kulub reeglina rohkem aega kui „tavalisele“ tarkvarale
- Tarkvara loomiseks kuluva aja ennustamine nõuab väga suurt kogemust sarnaste sardsüsteemide vallas

Sardsüsteemide tarkvara tüüpilised probleemid

- Inimresurss
- Limiteeritud mälu (väikeste süsteemide puhul)
- Protsessori jõudlus
- Süsteemi voolutarve, eriti oluline patareitoite puhul
- Toote hind
- (Sard-)süsteemis kasutatav operatsioonisüsteem koos oma teekidega
- Tarkvara loomisel ei pruugi alati toimida ülalt-alla meetodid

Peamised sardsüsteemides kasutatavad programeerimiskeeled

- C
- C++
- Assembler (ASM)

- Java
- Python
- Ada

- LabVIEW

C

- Loodud 1972 aastal Dennis Ritchie poolt
- Algusest peale mõeldud asenduseks assemblerile
- Lisaks on pandud väga suur rõhk koodi loetavusele

```
#include <stdio.h>

int main (void)
{
    puts ("Hello world!");
    return 0;
};
```

C keele plussid

- Hea kompilaatoriga sama kiire programm kui ASM'is
- Oluliselt lihtsam kirjutada ja debugida kui ASM'is kirjutatud programmi
- Piisavalt palju programmeerijaid kes valdavad C keelt
- Paljud teegid on kirjutatud C keeles
- On võimalik suhteliselt lihtsalt kirjutada porditavat koodi

C keele miinused

- Programmeerimisel on võimalik väga lihtsalt vigu teha
- On võimalik teha vigu mida on väga raske avastada
- Pole korralike UML'i tööriistu (vähemalt vabavara omasid)
- Lõpliku programmi kiirus ja mälu kasutus sõltub täielikult kompilaatorist
- C kompilaatoril kasutatav optimeermine võib mõningad koodilõike välja jätta

C keele kasutuskohad

- Enamusjaolt kasutatakse riistvara lähedaste programmide loomisel:
  - Kernelid ja draiverid
  - Väiksemad ilma kernelita sardsüsteemid (mikrokontrollerite programmid)
- Teegid mis peavad olema kiired:
  - Matemaatika funktsioonid
  - Stringitöötlus
  - Java JNI funktsioonid

Kohad kus C'd ei kasutata

- Suured keerukad programmid mis ei ole otseselt seotud riistvaraga (keerukad graafilised kasutajaliidesed, serverid, andmebaasid, jne)
- Keerukad programmid, mille koodi ei saa kuskilt juba valmiskirjutatuna, kas lähtekoodina või teegina ning on vaja minimaalse ajaga turule saada
- Ohutuskriitilistes kohtades, välja arvatud juhud kus on kasutatud spetsiaalseid koodi korrasoleku kontrolle (IEC 61508 nõue)

Assembler (ASM)

- Loodud 1950'ndate paiku
- On sisuliselt masinkoodi „tõlge“ inimesele arusaadavamale kujule

```
section .text          ;section declaration
                       ;we must export the entry point to the ELF linker
    global _start      ;or loader. They conventionally recognize _start
                       ;as their entry point.

section .data          ;section declaration
    msg db "Hello, world!",0xa  ;our dear string
    len equ $-msg      ;length of our dear string


_start:                ;write our string to stdout
    mov edx,len        ;third argument: message length
    mov ecx,msg        ;second argument: pointer to message to write
    mov ebx,1          ;first argument: file handle (stdout)
    mov eax,4          ;system call number (sys_write)
    int 0x80           ;call kernel
                       ;and exit
    mov ebx,0          ;first syscall argument: exit code
    mov eax,1          ;system call number (sys_exit)
    int 0x80           ;call kernel
```

Veel üks „Hello world!“ näide

```
; (C) Copyright 2001, 2002 Ian P. Cardenas
; <ian.cardenas@ultraviolent.com>

.data
.cstring
.align 2
msg:
    .asciz "Hello world!\n"
    len = . - msg

.text
.align 2
.globl _start
_start:
    li  r3,1              ; arg1 = stdout = 1
    lis r4,hi16(msg)      ; arg2 = address of string. loaded in two steps
    ori r4,r4,lo16(msg)   ; high halfword and then low
    li  r5,len            ; arg3 = numBytes = 12
    li  r0,4              ; 4 is write syscall
    sc                    ; call write
    nop                   ; need the no-op because sc skips one instr.
    li  r0,1              ; 1 is exit syscall
    li  r3,0              ; exit status 0
    sc                    ; call exit
```

Assembleri plussid

- Saab absolutselt kõiki riistvara võimalusi kasutada, st. on täielik kontroll riistvara üle
- Programmi transleerimisel masinkoodi ei ole mingit optimeerimist vahel
- Võimalik teha kindla suurusega või väga kiireid programme

Assembleri miinused

- Kood on reeglina raskesti loetav
- Suurt programmi väga raske ilma vigadeta kirjutada
- Grupitöö on raskendatud
- Praktiliselt võimatu on teha erinevate arhitektuuride vahel porditavat koodi
- Vägagi keerukas on teha taaskasutatavat koodi

Assembleri kasutuskohad I

- Kohad mida ei ole mingil põhjusel võimalik teha muudes keeltes
  - Mõndadel arhitektuurides schedulerid
  - Programmide osad mis kasutavad kompilaatorile tundmatuid käske
  - Tsüklite „täitmine“. Näiteks, on vaja ühte lihtsat viidet:

```
/* intuitiivne variant */
for (i = 0; i < 0x1ff; i++);

/* korrektne variant */
for (i = 0; i < 0x1ff; i++)
        asm volatile ("nop");
```

Assembleri kasutuskohad II

- Programmid mis peavad olema kindla suurusega
  - Mõningad bootloaderid, eriti nende esimesed astmed
  - Mõned viirused
- Väga suurt kiirust nõudvad programmilõigud (juhul kui ei ole võimalik mõnes kõrgkeeles kirjutada)
  - DSP ja matemaatika funktsioonid
  - Katkestuste teenindamine

Mõningad märkused assembleri kasutamise kohta I

- Assemblerit on mõtet ainult siis kasutada kui on täiesti kindel, et teisiti ei saa
- Kui on vaja assemblerit kasutada koos mõne muu keelega, siis peab olema piiratud assembleri kasutamine ainult kindlate funktsioonidega, st. peab kapseldama assembleri lõigud eraldi funktsioonidesse. Näide järgmisel slaidil.

Assembleri lõikude kapseldamine eraldi funktsioonidesse

```
/* funktsioon 1 */
uint8_t lpm (uint16_t addr16)
{
    uint8_t result;
    asm volatile ("lpm" "\n\t"
                  "mov %0, r0" "\n\t"
                  : "=r" (result)
                  : "z" (addr16)
                  : "r0");
    return result;
};

/* funktsioon 2 */
void delay (uint8_t ms)
{
    uint16_t cnt;
    uint16_t delay_count = 0x200;
    asm volatile ("L_dl1%=:" "\n\t"
                  "mov %A0, %A2" "\n\t"
                  "mov %B0, %B2" "\n"
                  "L_dl2%=:" "\n\t"
                  "sbiw %A0, 1" "\n\t"
                  "brne L_dl2%=" "\n\t"
                  "dec %1" "\n\t"
                  "brne L_dl1%=" "\n\t"
                  : "=&w" (cnt)
                  : "r" (ms), "r" (delay_count));
};

delay (lpm (0x100));
/* lõplik programmilõik */
```

Mõningad märkused assembleri kasutamise kohta II

- Kui kasutatakse assemblerit mõne teise keelega, siis ei tohi ilma **väga mõjuva põhjuseta** reserveerida assemblerile mingeid kindlaid mälualasid või registreid. Näide on järgmisel slaidil.

Registrite ja mälualade reserveerimine (mittekorrektne)

```
void inc_var (void)
{
    asm volatile ("ld __tmp_reg__, 0x100" "\n\t"
                  "inc __tmp_reg__" "\n\t"
                  "st 0x100, __tmp_reg__" "\n\t"
                  :
                  :);
};

int main (void)
{
    register uint8_t counter asm ("r3");
    /* nullime registri r3 */
    asm volatile ("clr r3");

    while (1)
    {
        counter++;
        inc_var ();
    };
};
```

Eelmise näite korrektne lahendus

```
static uint8_t var;

void inc_var (uint8_t *ptr)
{
    asm volatile ("ld __tmp_reg__, %a0" "\n\t"
                  "inc __tmp_reg__" "\n\t"
                  "st %a0, __tmp_reg__" "\n\t"
                  :
                  : "e" (ptr));
};

int main (void)
{
    uint8_t counter = 0;

    while (1)
    {
        counter++;
        inc_var (var);
    };
};
```

C++

- Loodi vahemikus 1979...1986 Bjarne Stroustrup poolt
- Mõeldud objektorienteeritud täienduseks C keelele

```
#include <iostream>

int main (void)
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
};
```

C++ eelised sardsüsteemides

- On olemas mitmeid UML tööriistu
- Võimaldab mõningatel juhtudel lihtsamalt kirjutada sama kiireid programme kuid C's

C++ pahupooled sardsüsteemides

- Väiksematel sardsüsteemidel annab luua ainult staatilisi klasse
- Väiksematel sardsüsteemidel ei pruugi kompilaatorid korraliku koodi teha
- On mõningal määral keerukam kui C
- Suhteliselt lihtne tekitada mälulekkeid
- 8 ja 16 bitistel kontrolleritel ei anna olulisi eeliseid tarkvara kirjutamisel

C++ eelistatud kasutuskohad

- Keerukamad graafilised kasutajaliidesed
- Sobiv suurematel kontrolleritel (32 bitistel) igasuguste rakendusprogrammide kirjutamiseks

Ada

- Loodud Jean Ichbiah (CII Honeywell Bull) juhtimise all, DoD (United States Department of Defense) tellimusel
- On ettenähtud missiooni- ja turvakriitiliste süsteemide (sard)tarkvara loomiseks

```
with Text_IO;

procedure Hello_World is
begin
    Text_IO.Put_Line("Hello World!");
end Hello_World;
```

Ada plussid ja miinused

Plussid

- Võrreldes teiste keeltega suhteliselt raske teha kriitilisi vigu
- Loodud spetsiaalselt sard- ja reaalaja süsteemidele

Miinused

- Tööajal programmi kontroll võtab palju resurssi
- Väikene kasutajate ring

Ada peamised kasutuskohad

- Lennundus
- Kosmose süsteemides
- Relvasüsteemides

Java

- Loodud James Gosling'i (Sun Microsystems) poolt, esimene versioon 1995 aastal
- Java moto - „Write Once, Run Anywhere“

```
public class hello_world
{
    public static void main (String[] args)
    {
        System.out.println ("Hello world!");
    };
};
```

Java plussid

- Saab ühel arhitektuuril kirjutatud programmi üle viia teise arhitektuuriga seadmesse ilma, et peaks midagi ümber kompileerima
- On võimalik väga kiiresti kirjutada töötavat programmi
- Ei ole vaja eraldi arenduskeskonda

Java miinused

- Reaalselt võiks olla Java moto - „Write Once, Debug Everywhere“
- Kuna Java programm on tarkvaraliselt interpreteeritav, siis on võrreldes mitteinterpreteeritava programmiga tunduvalt aeglasem (näiteks C's kirjutatud)

Java kasutuskohad

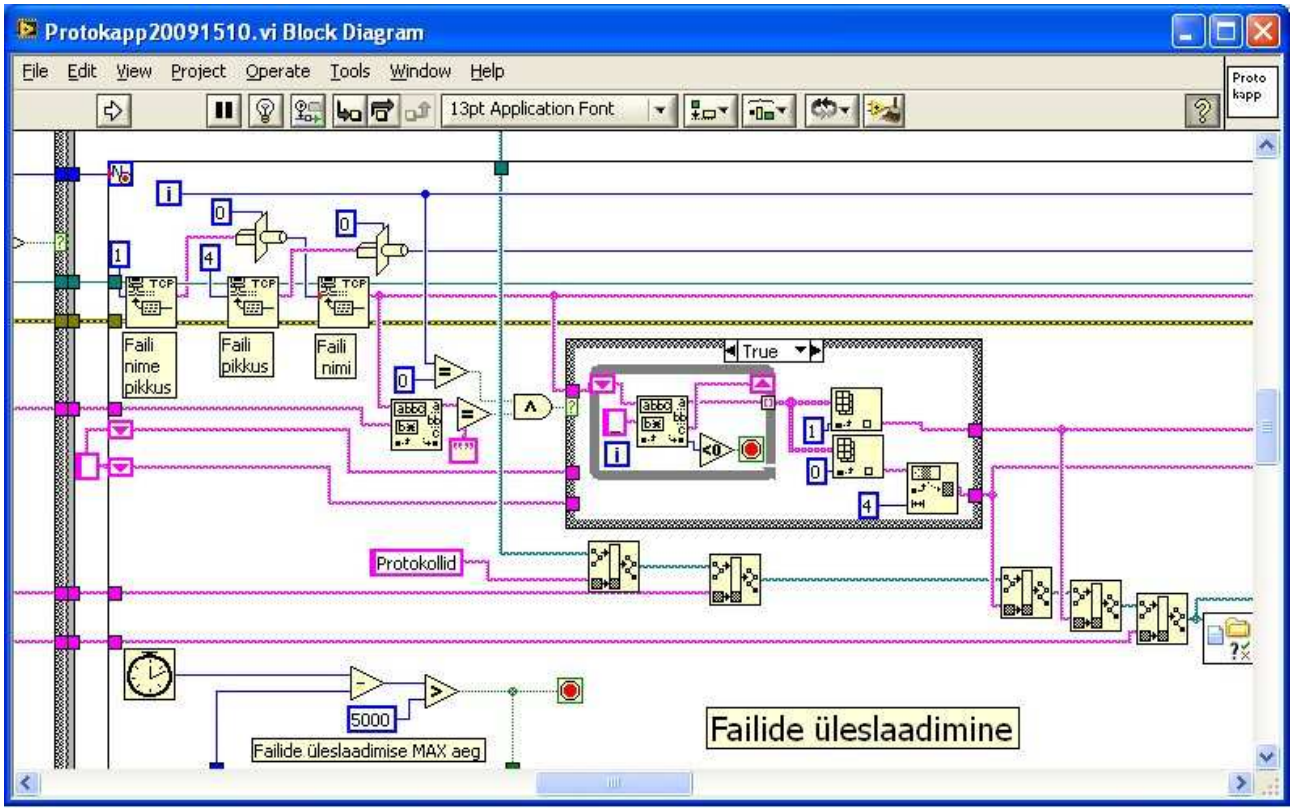
- Kõiksugu multiplatvormsed PC ja serverite programmid

Kohad kus Javat ei ei ole soovitatav kasutada

- Süsteemides millel on väikene arvutusvõimsus võrreldes andmete sisendvooga (väikse arvutusvõimsusega sardsüsteemid ja süsteemid mis nõuavad väga suurt andmetöötluskiirust)
- Otseselt riistvara juhtimiseks
- Range reaalajaga süsteemid, juhul kui ei kasutata spetsiaalset reaalaja Javat

LabVIEW

- Graafiline programmeerimiskeskkond, mis on suhteliselt intuitiivne ja kergesti õpitav



LabVIEW

- Viimastel aastatel on laienenud ka sardsüsteemide valdkonda. Hetkel on toetatud ainult ARM seeria mikrokontrollerid ja Nationali Instruments'i poolt välja töötatud FPGA plaadid.
- Peamine laialdast levi pärssiv faktor on seni toote hind ja platvormide nappus.

- Lingid

  http://www.ni.com/embedded/

  http://www.ni.com/fpga/

Keelte võrdlus

Keel	Koodi kiirus	Mälu nõudlus (RAM ja ROM)	Keele keerukus
C	Suur/väga suur	Vähe	Keerukas
ASM	Suur/väga suur	Vähe	Keerukas
C++	Suur	Vähe	Keerukas
Ada	Keskmine	Keskmiselt	Keskmine
Java	Aeglane	Palju	Lihtne

NB! Koodi kiirus on pöördvõrdelises seoses energia tarbega

Kernelites kasutatavad progarmeerimiskeeled

Keel/ Kernel	C	ASM	C++ või C#
Linux	+	+	
FreeBSD, OpenBSD, NetBSD	+	+	
Darwin	+	+	
Windows	+	+	+
NutOS	+	+	
eCos	+		+
FreeRTOS	+	+	
Contiki	+		

Kokkuvõte

- Progarmeerimiskeeltest on sardsüsteemides kõige paremini kasutatav C ning mõnel juhul ka C++
- Assemblerit on soovitatav kasutada nendes kohtades kus muid keeli ei saa kasutada
- Java kasutamine otse riistvara juhtimiseks on raskendatud, kuigi kasutajaliideseid on väga hea teha
- Missiooni- ja turvakriitilistes süsteemidel on soovitatav kasutada Ada't