# ĮTERPTINĖS SISTEMOS

PROGRAMINĖ ĮRANGA

Doc. dr. Šarūnas Kilius

## ĮS programinė įranga

- ĮS programinė įranga „paslėpta“ visuose įterptinėse sistemose
  - Buitinėje technikoje, mob. telefonuose, TV
  - Automobilių elektroninėse sistemose
  - Medicinos technikoje
- Kiekvienu atveju skirtingi reikalavimai ir aparatinė įranga

- Programinė įranga sąveikauja su aplinkiniu pasauliu
  - Laikas kiekvienai užduočiai
  - Energijos sąnaudos
  - Nenutrūkstamas veikimas



## Reikalavimai ĮS programinei įranga

- Patikimumas
  - Vartotojo įsikišimas ir klaidų aptarnavimas daugeliu atveju neįmanomas
- Rentabilumas
- Žemas energijos suvartojimas
- Efektyvus atminties išnaudojimas
- Našumas
  - Savalaikiškumas



## ĮS programinės įrangos ypatybės

- Savalaikiškumas (timeliness)
  - Skaičiavimai užima tam tikrą laiką
    - Jei ir būtų be galo greitas procesorius, ĮS programinė įranga turi atsižvelgti į laiko reikalavimus, kadangi fizikiniai procesai realiame pasaulyje vyksta tam tikrą laiką.
    - Turi tenkinti laiko suvaržymus – atlikti užduotis per nustatytą laiką.
    - Skaičiavimų našumui padidinti mikrovaldikliuose bei DSP įtrauktos įvairios aparatinės priemonės (spartinančioji atmintis ir kt.). Tačiau šios aparatinės priemonės didina energijos sąnaudas, todėl idealiu atveju programa turi būti įgyvendinta taip, kad atitiktų laikinius reikalavimus, tačiau vengtų šių aparatinių patobulinimų.

## ĮS programinės įrangos ypatybės (2)

- Konkurentiškumas
  - Įterptinės sistemos yra susietos su aplinkiniu pasauliu, kuriame įvykiai gali įvykti artimu arba vienu metu.
    - ĮS turi reaguoti į įvairius poveikius – priimti duomenis iš tinklo, nuskaityti įvairių jutiklių signalus, suformuoti valdymo signalus. Visos šios užduotys „konkuruoja tarpusavyje“ dėl ĮS resursų.
- PĮ gyvybingumas
  - Programa neturi nutrūkti ar „pakibti“ belaukdama įvykio, kuris gali taip ir nepasirodyti. Programai „pakibus“, sutriks visos ĮS veikla.
    - Šiuo atveju gali būti panaudota viena iš aparatinių priemonių „watchdog“ laikmatis.

5

## ĮS programinės įrangos ypatybės (3)

- Komponentų sąsaja
  - Programinė įranga sudaryta iš atskirų komponentų, kurie aptarnauja atskirus procesus
    - Garso koderis verčia analoginius signalus į skaitmeninius duomenis ir atvirkščiai.
    - Kaip integruoti atskirus programinius paketus, kurie gali būti paimti iš skirtingų gamintojų? Kaip integruoti juos tarpusavyje, kad būtų užtikrintas sistemos saugumas, patikimumas?

6

## ĮS programinės įrangos ypatybės (4)

- Nevienodumas
  - Skirtingų gamintojų komponentai gali turėti skirtingus skaičiavimo stilius ir įgyvendinimo technologijas
    - Mobilus telefonas – duomenų priėmimas, dekompresavimas, kompresavimas, užrašų knygelės naršymas – skirtingos užduotys
  - Skirtingas įvykių periodiškumas
    - Nereguliarūs įvykiai (vartotojo komandos, pranešimai,...)
    - Reguliarūs įvykiai (jutiklių apklausimas, valdymo signalų formavimas, ...)
- Reakcingumas
  - ĮS turi nuolatos reaguoti į aplinkos poveikį ir tas reagavimo greitis turi atitikti aplinkos greitį.
  - Turi būti užtikrintas saugumas.
    - PĮ valdo sudėtingą (ir brangią) įrangą, todėl negali sutrikti arba su sutrikti be atitinkamo pranešimo.

7

## Real-time

- Sistemos reakcijos laikas – tai laikas tarp įvykio pasirodymo sistemos įėjime, skaičiavimų atlikimo ir reikiamų signalų suformavimo sistemos išėjime
- Realaus laiko sistema – tai sistema, atliekanti nustatytus veiksmus – suformuojanti atsaką per nustatytą laiko tarpą

8

## Soft and Hard Real time

- Soft real-time system – tokia sistema, kur pavėluotas užduoties atlikimas per nustatytą laiką lemia sistemos našumo sumažėjimą, tačiau nesukelia katastrofiškų padarinių
  - Pavyzdžiui, pardavimo automatai, vaizdo grotuvai,...
- Hard real-time system – tokia sistema, kur pavėluotas užduoties atlikimas lemia pilną sistemos sutrikimą ir gali sukelti katastrofiškus padarinius
  - Pavyzdžiui, valdymo sistemos (lėktuvo, ...)
- Firm real-time system – kelių skaičiavimų vėlavimas rimtų padarinių neturės, tačiau didesnis jų skaičius gali lemti pilną sistemos sutrikimą.
  - Pavyzdžiui, navigacijos sistema

9

## Kaip nustatyti PĮ kokybę?

- Dinaminis efektyvumas
  - Kiek mašininių ciklų reikia programos (užduoties) įvykdymui
- Statinis efektyvumas
  - Kiek baitų reikia:
    - RAM dydis
      - Kintamieji, aparatinis dėklas (stack)
    - ROM dydis
      - Konstantos, programinis kodas
- Energijos sąnaudos

10

## Programinės įrangos kokybė

- Programos teisingumas
  - Loginis
  - Laiko atžvilgiu
- Suprantamumas
- Redagavimo galimybė
  - Priežiūra, lankstumas, atnaujinimas, pakartotinis panaudojimas

11

## Programos optimizavimas

- Tik tinkamai parašius programos kodą, kompiliatorius kompiliuos kaip mums reikia
- Matematinių išraiškų paprastinimas
  - a*b + a*c -> a(b + c)
- Nereikalingo kodo pašalinimas
  - Leistinas kodas (pašalins kompiliatorius, kadangi nėra *else*):

```
#define DEBUG 0
...
if (DEBUG) print_debug_stuff();
```

12

## Programos optimizavimas (2)

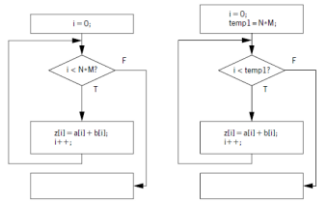
- „inline“ funkcijų naudojimas
  - „inline“ funkcija neturi atskiro „funkcijos kūno“. Kompiliavimo metu funkcijos komandos įterpiamos į jos iškvietimo vietą, taip išvengiant funkcijos iškvietimui reikalingų papildomų veiksmų.
  - Trūkumas – didėja programos apimtis

```
int foo(a,b,c) { return a + b – c ; }
        Function definition

z = foo(w,x,y);
        Function call

z = w + x – y;
        Inlining result
```



## Programos optimizavimas (3)

- Kintamųjų tipo parinkimas
  - Skaičiavimų trukmė priklauso nuo naudojamų kintamųjų tipo
  - char, integer, float...
  - Priklauso nuo ĮS skiltiškumo
- Ciklų optimizavimas
  - Iškelti nereikalingą kodą






## Programos apimties optimizavimas

- Duomenų apimties mažinimas
  - Identifikuoti ir pašalinti duomenų dublikavimą
  - Duomenų buferiai turi būti tinkamo dydžio – identifikuoti aiškiai koks gali būti saugomas maksimalus duomenų kiekis
    - Tas pats buferis gali būti panaudotas skirtingose programos vietose
  - Jei skirtingos konstantos tokio paties dydžio, jos gali būti nukreiptos (mapped) į tą pačią vietą.

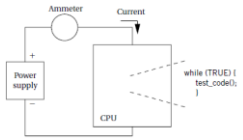


## Programos apimties optimizavimas (2)

- Programos apimties mažinimas
  - Tinkamų komandų parinkimas
    - Procesoriuose gali būti įdiegtos skirtingo ilgio komandos.
  - Paprogramių panaudojimas
    - Jei programoje kartojamos identiškos operacijos, joms atlikti tikslinga panaudoti paprogrames
    - Jei operacijos nedaug skiriasi tarpusavyje, galima naudoti parametrizuotas paprogrames.
    - Tačiau naudojant paprogrames prisideda papildomų instrukcijų paprogramių iškvietimui bei registrų turinio išsaugojimui
  - Kai kurios procesorių architektūros turi papildomus komandų rinkinius, sudarytus iš trumpesnio komandų formato. Programos apimtis gali būti sumažinta 40%
    - ARM Thumb
    - MIPS-16

## Energijos sąnaudų optimizavimas

- Ypatingai svarbu baterijoms maitinamiems įtaisams
- Galimybės optimizuoti energijos sąnaudas:
  - Pakeisti algoritmus į vartojančius mažiau energijos
  - Ypatingai daug energijos suvartojama kreipiantis į atmintį – optimizuoti šį procesą
    - Jei įmanoma, sumažinti „cache" atminties apimtį – SRAM atmintis vartoja daugiau energijos
  - Išjungti nenaudojamas sistemos dalis





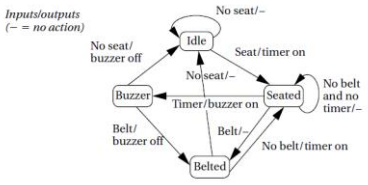

## PĮ tipai

- State machine
  - Tinkamiausia, kuomet ĮS turi reaguoti į įvykius
    - Įvykiai pasirodo aperiodiškai
    - Vartotojo sąsaja ir pan.
- Circular buffers
  - Orientuota į sistema, kur duomenys priimami reguliariai ir iškart turi būti apdoroti
    - Skaitmeninis signalų apdorojimas ir pan.
- Queues (*elastic buffers*)
  - Duomenys gaunami neapibrėžtu dažnumu ir kintamos apimties
    - Skaitmeninis signalų apdorojimas, įvykių apdorojimas



## State machine

- Baigtinių būsenų mašina (*finite-state machine*)
  - Įvykiai pasirodo aperiodiškai ir ĮS turi reaguoti į šiuos įvykius
- Pavyzdys: automobilio saugos diržų valdiklis
  - 3 įėjimai – svorio jutiklis, saugos diržo jutiklis, laikmatis
  - 1 išėjimas - buzeris






## State machine

```
#define IDLE 0
#define SEATED 1
#define BELTED 2
#define BUZZER 3

switch (state) { /* check the current state */
        case IDLE:
                if (seat) { state = SEATED; timer_on = TRUE; }
                /* default case is self-loop */
                break;
        case SEATED:
                if (belt) state = BELTED; /* won't hear the
                                        buzzer */
                else if (timer) state = BUZZER; /* didn't put on
                                        belt in time */
                /* default is self-loop */
                break;
        case BELTED:
                if (!seat) state = IDLE; /* person left */
                else if (!belt) state = SEATED; /* person still
                                        in seat */
                break;
        case BUZZER:
                if (belt) state = BELTED; /* belt is on–turn off
                                        buzzer */
                else if (!seat) state = IDLE; /* no one in
                                        seat–turn off buzzer */
                break;
}
```

## PĮ projektavimo procesas

- Reikalavimų nustatymas
- Sistemos architektūros sudarymas
- Operacinės sistemos parinkimas (jei naudojama)
- Platformos vystymui parinkimas
- Programavimas
- Programinio kodo optimizavimas pagal reikalavimus
- Programos testavimas (programavimo aplinkoje)
- Programos testavimas tikslinėje sistemoje (prototipe)

21

## Programavimo kalbos parinkimas

- Asembleris
  - Priklauso nuo procesoriaus ir architektūros
    - Keičiant procesorių tenka visiškai keisti programą
  - Didelis efektyvumas
    - Galima maksimaliai išnaudoti architektūros privalumus
  - Sunku kurti ir redaguoti didelės apimties programas
- Aukšto lygio programavimo kalbos
  - Didesnis portabilumas
  - Lengviau ir greičiau kurti programas – mažesnis programos eilučių skaičius
  - Mažesnis efektyvumas
    - Kompiliatorius ne visuomet geba pilnai optimizuoti kodą

22

## Programavimo kalbos

- Šiuo metu daugelis programų parašytos struktūrinėmis programavimo kalbomis
  - Tačiau asemblerio kalbos vis tiek atskirais atvejais reikia
  - Draiveriai (tvarkyklės?)
    - Dažnai turi užtikrinti griežtas laikines formuojamų signalų charakteristikas, atskirų bitų valdymą.
    - Tam geriausiai tiktų asembleris

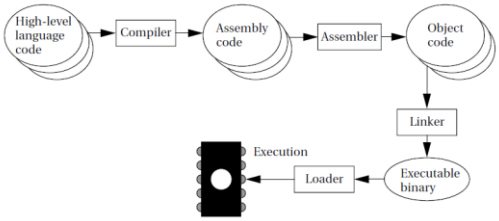
23

## Programos generavimas (Built process)

- Šio proceso metu iš programinio kodo, parašyto asembleriu ar aukštesnio lygio programavimo kalba, sugeneruojamas atminties atvaizdas, kuris gali būti įkeltas į atmintį.
- Naudojamos programavimui priemonės (programavimo aplinkos) nežino visos informacijos apie aparatinę įrangą (atminties apimties, adresų ir kt.)
- Šia informacija turi pasirūpinti vartotojas

24

## Programos generavimas




25

## Kompiliavimas

- Programinio kodo apdorojimas, semantinė analizė ir objektinio kodo generavimas
- Objektinis kodas optimizuojamas atsižvelgiant į procesoriaus architektūrą ir specifines charakteristikas
- Naudojami kros-kompiliatoriai ir kros-asembleriai
  - Programinis kodas generuojamas ir kompiliuojamas vienoje aplinkoje (personaliniame kompiuteryje), o vykdomas kitoje.
- *Rezultatas – objektinis failas*

26

## Objektiniai failai

- Kompiliavimo metu kompiliatoriai generuoja objektinius failus – palyginti didelės apimties, lanksčios struktūros duomenų struktūras, susidedančias iš komandų ir duomenų.
  - Standartizuotas formatas
    - COFF – Common Object File Format
    - ELF – Extended Linker Format
- Objektiniai failai susideda iš:
  - Programos kodo blokų – teksto blokai
  - Inicializuotų globalių kintamųjų – duomenų blokai
  - Simbolių lentelių
- Šie failai nėra tiesiogiai vykdomi failai, skirti įkelti į ĮS. Tačiau jie naudojami programos testavimui

27

## Saistyklės (Linkers)

- Sujungia tarpusavyje atskirus objektinius failus
- Abstrakčias simbolių nuorodas pakeičia nuorodomis į konkrečius kintamuosius ar funkcijų iškvietimą
- Prijungia specialų objektinį failą su Start-up kodu.
  - Nedidelės apimties programinis kodas, kuris paruošia procesorių programos vykdymui

28

## Start-up kodas

- Start-up kodas paprastai susideda iš šių veiksmų
  - Pertraukčių draudimas
  - Kintamųjų perkopijavimas iš ROM į RAM atmintį
  - Neinicializuotos duomenų vietos išvalymas
  - Aparatinio dėklo inicializavimas
  - Heap atminties inicializavimas
  - (konstruktorių inicializavimas objektinėse kalbose)
  - Pertraukčių leidimas
  - main() funkcijos iškvietimas
- Kam reikalingas Start-up kodas?
  - Paruošia įterptinę sistemą programos vykdymui
  - Bendros paskirties kompiuteriuose tai atlieka OS „loader'iai"

29

## Locator

- Atskiras arba integruotas kartu su saistykle
- Pagal programuotojo pateiktą informaciją apie ĮS atmintį, priskiria konkrečius adresus programos kodo ir duomenų segmentams
  - Informacija gali būti perduodama „linker scriptų" pagalba
- Sugeneruojamas atminties atvaizdas – dvejetainis kodas, kuris skirtas įkelti į konkrečią įterptinę sistemą.

30

## Atminties paskirstymas

- RAM
  - Globalūs kintamieji
  - Heap atmintis
  - Aparatinis dėklas
    - Kintamieji, laikini duomenys,...
  - Duomenys kinta programos vykdymo metu
- ROM
  - Instrukcijos
  - Konstantos
  - Lėtesnė nei RAM atmintis
- EPROM

31

## Programos testavimas ir vykdymas

- Jei programavimo aplinkos (personalinio kompiuterio) procesorius skiriasi nuo procesoriaus sistemos, kuriai programa rašoma, kokiu būdu galima programą testuoti?
  - Simuliuoti
  - Įrašyti į tikslinę sistemą
- Testavimas programavimo aplinkoje
  - Simuliavimas pažingsniui programiniame simuliatoriuje
  - Simuliavimas simuliatoriuje etapais (breakpoint įterpimas)
  - Vykdyti aparatiniame emuliatoriuje su spec. programine aplinka
- Vykdyti/testuoti tikslinėje sistemoje (prototipe)

32

## Programavimas aukšto lygio kalbomis

- Procedūrines kalbos
  - C – aukšto lygio programavimo kalba, turinti asemblerio kalbos funkcionalumą
- Objektinės kalbos
  - C++
    - eMbedded Visual – programavimui Visual C++ ir Visual Basic
  - Java – didelis aplikacijų mobilumas
    - Tuo atveju, jei JVM palaikoma OS
    - Tinklinės aplikacijos

33

## C kalba

- Patogi ĮS programavimui kalba, kadangi įvairūs objektai, kaip adresai, baitai, bitai ir kt., kurie yra unikalūs kiekvienam procesoriui, gali būti pasiekiami tiesiogiai...
  - Leidžia valdyti pertrauktis, procesoriaus registrus ir kt.
- C kalboje numatyti specialūs kintamųjų specifikatoriai, naudingi programos valdymui ir optimizavimui
  - register
  - volatile
  - static
  - constant

34

## Kintamųjų specifikatoriai

- Register – skirtas dažnai naudojamiems kintamiesiems. Nurodo kompiliatoriui, kad šio tipo kintamasis turėtų būti saugomas taip, kad būtų kuo greičiau pasiekiamas. Gali būti talpinamas procesoriaus registruose ar „cashe“ atmintyje.
- Static – nurodo pastovius kintamuosius tam tikroje funkcijoje ar faile. Išsaugo reikšmę tarp funkcijų iškvietimų, tačiau skirtingai nuo globalių kintamųjų, static kintamasis žinomas tik toje funkcijoje ar faile.

35

## Kintamųjų specifikatoriai

- Volatile – kompiliatoriui nurodo, kad kintamojo reikšmė gali kisti ne tik programoje aiškiai apibrėžtu būdu. Būtina naudoti programose su pertrauktimis, nes kitu atveju gali kilti problemų, jei kompiliavimo metu bus optimizuota programa.
- Extern – naudojamas, kuomet didesnės apimties programa skaidoma į kelis atskirus failus ir naudoja tuos pačius kintamuosius. Jei atskiruose failuose bus bandoma deklaruoti kintamuosius su tuo pačiu pavadinimu, kils klaida dėl dubliavimosi. Dėl extern specifikatoriaus kintamieji tam žinomi atitinkamai programos daliai, tačiau nėra dubliuojami

36

## Apibendrinimas

- Programinė įranga yra „paslėpta“ kiekvienoje įterptinėje sistemoje
- Kiekviena įterptinė sistema yra unikali savo paskirtimi, aparatine įranga, atitinkamai programinė įranga kuriama kiekvienam individualiam atvejui.
- Įterptinių sistemų programavimui galima rinktis asemblerį arba aukšto lygio programavimo kalbas. Aukšto lygio programavimo kalbos leidžia lengviau ir greičiau kurti programas, tačiau jų efektyvumas gali būti mažesnis.

37