PROGRAMIRANJE VGRAJENIH SISTEMOV V REALNEM ČASU IN ANALIZA ČASA IZVAJANJA OPRAVIL

- Posebnosti programskih jezikov v sistemih z realnim časom
- Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času
- Analiza najneugodnejšega časa izvajanja opravil (WCET)

Posebnosti programskih jezikov v sistemih z realnim časom

- SRČ so običajno sestavljeni iz večjega števila sočasno delujočih komponent - potrebujemo sočasno izvajanje več opravil:
  - (eksplicitna) deklaracija opravil
  - definiranje pogojev začetka izvajanja (aktivacije) opravila
  - možnost začasnega prenehanja izvajanja opravil, nadaljevanj, terminacija izvajanja, …
  - podpora za medsebojno sinhronizacijo opravil

Posebnosti programskih jezikov v sistemih z realnim časom

- SRČ morajo biti za programe, ki tečejo v realnem času časovno napovedljivi in predvidljivi – potrebno je vključiti časovno komponento:
  - dostop do trenutne (globalne) ure,
  - specifikacijo časa, v katerem naj se akcije sprožijo, frekvenco proženja, specifikacijo časa, v katerem naj bodo akcije končane, ipd.
  - časovni nadzor nad izvajanjem vhodno/izhodnih operacij, dostopa do skupnih virov, ipd.,
  - reagiranje na situacije, ko niso zadovoljene vse časovne zahteve (“timeout”),
  - Podpora za reagiranje na situacije, v katerih se časovne zahteve dinamično spreminjajo (rekonfiguracija).

Posebnosti programskih jezikov v sistemih z realnim časom - nadaljevanje

- Sodobni SRČ običajno spadajo med varnostno-kritične – zahteva se robustnost in varnost, zaznavanje napak, odpornost na napake, ipd:
  - strukturna obravnava izjem v programu
  - možnost opisa in ugotavljanja nepravilnih stanj v delovanju sistema (napačne vrednosti spremenljivk, neupoštevanje časovnih omejitev, …)
  - možnost deklaracije in izvedbe alternativnih delov kode (opravil) glede na trenutno stanje sistema

Posebnosti programskih jezikov v sistemih z realnim časom - nadaljevanje

- Za SRČ je značilna komunikacija z okolico preko vhodni/izhodnih naprav:
  - podpora za neposredni dostop do vmesnikov vhodno/izhodnih naprav (preko registrov ali preko posebnih gonilnikov),
  - podpora za deklaracijo in izdelavo rutin za strežbo prekinitev.

Posebnosti programskih jezikov v sistemih z realnim časom - nadaljevanje

- Uporabniki SRČ so običajno aplikacijski inženirji in tehniki in ne računalniški strokovnjaki – programski jeziki morajo odražati uporabnikov način mišljenja.
- Omogočati morajo podporo za analizo časa izvajanja opravil in analizo razvrstljivosti.
- Biti morajo modularni, podpirati morajo programiranje obsežnejših in kompleksnejših aplikacij in biti morajo enostavni za vzdrževanje, saj imajo sistemi v realnem času pričakovano dolgo življenjsko dobo.

Druge posebnosti razvojnega okolja za SRČ

- Preprečevanje smrtnih objemov,
- Časovno napovedljivi algoritmi razvrščanja,
- Zgodnje odkrivanje in obravnava občasnih preobremenitev sistemov,
- Določanje celotnega in preostalega časa izvajanja opravil,
- Zagotavljanje natančne ura realnega časa,
- Možnost natančne definicije trajanja vseh operacij,

Druge posebnosti razvojnega okolja za SRČ

- Dinamična rekonfiguracija distribuiranih sistemov ob napaki,
- Simulacija in zapisovanje prekinitev,
- Možnost analize razvrstljivosti,
- …

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času

- **Zbirni jezik**
  - v splošnem med najmanj primernimi jeziki za programiranje - pri SRČ se uporablja v polovici primerov
  - ne obstajajo nikakršni konstrukti za analizo razvrstljivosti
  - omogoča direktni dostopa do strojne opreme
  - zanesljivost in varnost programov je minimalna
  - ne obstajajo eksplicitne definicije procesov
  - pisanje večjih programov in vzdrževanje je zelo težko.

```
digits      equ     (maxTerms*209+1673)/1000
cntDigits   equ     6    ; number of digits for counter

            org     100h          ; this is a DOS com file
;*****************************************************************
;*****************************************************************
main:
; initializes the two numbers and the counter.  Note that this assumes
; that the counter and num1 and num2 areas are contiguous!
;
            mov     ax,'00'                 ; initialize to all ASCII zeroes
            mov     di,counter              ; including the counter
            mov     cx,digits+cntDigits/2   ; two bytes at a time
            cld                             ; initialize from low to high memory
            rep     stosw                   ; write the data
            inc     ax                      ; make sure ASCII zero is in al
            mov     [num1 + digits - 1],al ; last digit is one
            mov     [num2 + digits - 1],al ;
            mov     [counter + cntDigits - 1],al

            jmp     .bottom                 ; done with initialization, so begin


.
```

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **FORTRAN**
  - prvi višjenivojski programskih jezik
  - razumljiva struktura, možnost ločenega prevajanja po delih in enostavnost
  - enostaven za vzdrževanje
  - omogoča generiranje optimalne (hitre) kodo
  - nima podpore za analizo razvrstljivosti
  - ne podpira obravnave napak in hkratnega izvajanja opravil
  - ne omogoča neposrednega dostopa do perifernih vmesnikov
  - ni modularen

```
PROGRAM MAIN
INTEGER N, X
EXTERNAL SUB1
COMMON /GLOBALS/ N
X = 0
PRINT *, 'Enter number of repeats'
READ (*,*) N
CALL SUB1(X,SUB1)
END

SUBROUTINE SUB1(X,DUMSUB)
INTEGER N, X
EXTERNAL DUMSUB
COMMON /GLOBALS/ N
IF(X .LT. N)THEN
  X = X + 1
  PRINT *, 'x = ', X
  CALL DUMSUB(X,DUMSUB)
END IF
END
```

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **JOVIAL**
  - razvit pri ameriški aviaciji (USAF) za potrebe programiranja krmilnih procesov
  - temelji na Algolu
  - omogoča vključevanje segmentov v zbirnem jeziku
  - podpira osnovne podatkovne tipe, enonivojske zapise in polja zapisov
  - omogoča ločeno prevajanje delov programa, direktni dostop do strojne opreme, je čitljiv in enostaven

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **JOVIAL – nad.**
  - generira hitro in učinkovito kodo
  - nima podpore za analizo razvrstljivosti, ne podpira obravnave izjem in hkratnega izvajanja
  - danes ga v aplikacijah zamenjuje programski jezik ADA

```
START

    %access members of records%
    DEFINE get'A(G) "itm'!G'A";
    DEFINE get'B(G) "itm'!G'B";

    %test procedure%
    PROC test(:param) S;
      BEGIN
        ITEM param S;
        ITEM sumA S = get'A(one) + get'A(two);
        ITEM sumB S = get'B(one) + get'B(two);
        test = sumA + param + sumB;
        %perform sumA times%
        FOR I: 1 BY 1 WHILE I<= sumA;
          IF I < sumB;
            sumB = sumB + test(I);
          ELSE
            RETURN;
        param = param + sumB;
      END
  TERM
```

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **RTL/1 in RTL/2**
  - razvita za nadzor industrijskih procesov
  - izhajata iz Algola, sta dobro strukturirana, omogočata prevajanje po delih
  - vgrajene imata osnovne ukaze za neposredni dostop do vhodno/izhodnih vmesnikov in hkratno procesiranje
  - omogočata nizkonivojsko obravnavo izjem
  - iz jezika so odstranjeni nekateri časovno nepredvidljivi konstrukti (dinamično dodeljevanje pomnilnika)

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **RTL/1 in RTL/2 – nad.**
  - sta enostavno čitljiva in vzdrževanje je enostavno
  - vsebujeta majhnem naboru osnovnih tipov
  - ni medmodulskega preverjanja tipov
  - Ni podpore za analizo razvrstljivosti
  - RTL/2 se uporablja še danes za manjše SRČ v Veliki Britaniji

```
TITLE Goodbye World;

LET NL=10;

EXT PROC(REF ARRAY BYTE) TWRT;

ENT PROC INT RRJOB();

        TWRT("Goodbye, World!#NL#");
        RETURN(1);

ENDPROC;
```

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **ILIAD**
  - razvit pri General Motors-u na osnovi jezika PL/I
  - namenjen zajemanje podatkov in nadzor montažnih linij
  - dobro strukturiran, ima bogat nabor osnovnih podatkovnih tipov in vključuje opravila
  - vgrajeno ima obravnavo izjem
  - za sinhronizacijo uporablja tehniko kritičnih področij
  - procese je možno zakasniti glede na nek dogodek in/ali časovni pogoj

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **ILIAD – nad.**
  - omogoča direktni dostop do perifernih vmesnikov
  - možno ga je prevajati po delih in je relativno enostaven za vzdrževanje.
  - nima podpore za analizo razvrstljivosti
  - ni modularen in nima strukturno izvedenih obravnave izjem in sinhronizacije procesov

```
while done = no   /* Loop until done */
    begin
        print 'READY FOR COMMAND' at tty;
        read command from tty;
        select case( command )   /* Distinguished commands */
            case('START'):
                if started = no then /* This is first START */
                    begin /* Initiate reading and logging */
                        started = yes;
                        activate reader, logger;
                    end;
                else /* Any START after the first is ignored */
                    print 'PROCESSING ALREADY UNDERWAY' at tty;
            case('LOG'):
                if started = yes then /* Set the log switch */
                    locking log_req
                        log_req = yes;
                else /* LOG before START is meaningless */
                    print 'PROCESSING NOT YET UNDERWAY' at tty;
            case('STOP'): /* Turn all tasks off after log */
                locking ( log_req, done )
                    begin
                        log_req = yes;
                        done = yes;
                    end;
```

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **MODULA in MODULA-2**
  - Modula je namenjena izvajanju na mikroprocesorjih
  - zelo dobro strukturiran, je enostaven in zagotavlja strogo preverjanje tipov
  - modularen
  - vključuje podporo za dostop do perifernih enot tako na visokem kot na nizkem nivoju
  - sinhronizacija med procesi je realizirana s semaforji preko WAIT in SIGNAL ukazov

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **MODULA in MODULA-2 – nad.**
  - Modula 2 je izpeljanka Module, ki za sinhronizacijo procesov uporablja tehniko korutin (ni primerna za velike programe)
  - noben nima podpore za analizo razvrstljivosti in obravnave izjem

```
PROCEDURE Write(x,n:CARDINAL);
  VAR i:CARDINAL;
      buf:ARRAY [1..10] OF CARDINAL;
BEGIN i:=0;
  REPEAT INC(i); buf[i]:=x MOD 10; x:=x DIV 10
  UNTIL x=0
  WHILE n>i DO
    WriteChar(" "); DEC(n)
  END;
  REPEAT WriteChar(CHR(buf[i]+ORD("0")));
    DEC(i)
  UNTIL i=0;
END Write
```

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **PORTAL**
  - razvit v Švici za potrebe sistemskega programiranja in za kontrolo procesov v realnem času
  - uporablja ga le ozek krog uporabnikov
  - podoben je Moduli, sinhronizacija procesov poteka preko semaforjev in monitorjev
  - omogoča direktni dostop do perifernih vmesnikov in prevajanje po modulih
  - Wait ukaz z pridruženim maksimalnim čas čakanja na sprostitev semaforja

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **PORTAL – nad.**
  - omogoča nestrukturirano obravnavo izjem (preko ON ERROR ukaza)
  - premajhna podpora analizi razvrstljivosti

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **FORTH**
  - uporablja se v Združenih državah v aplikacijah za kontrolo procesov
  - je interpreter, programi se izvajajo v obliki skladovnega avtomata
  - nima podpore za analizo razvrstljivosti
  - omogoča samo primitivno obravnavo izjem (preko ABORT ukaza)
  - ni modularen

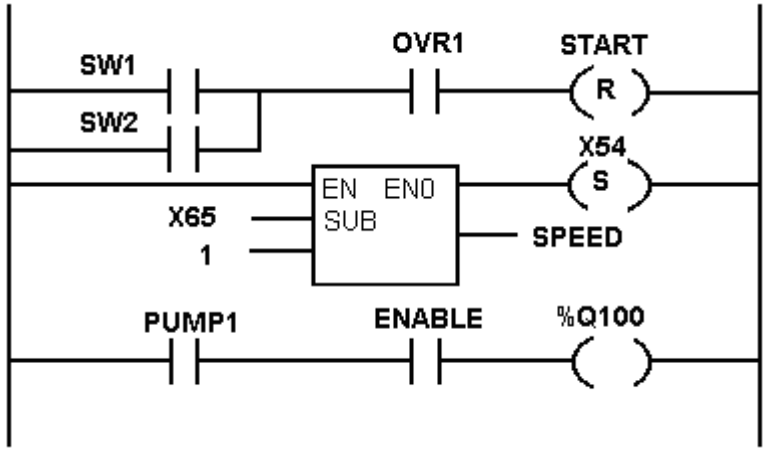
```
: swap_s_ij
    jj SArray get_byte
       ii SArray get_byte  jj SArray set_byte
    ii SArray set_byte
;

: rc4_init ( KeyAddr KeyLen -- )
    256 min TO KeyLen   TO KeyAddr
    256 0 DO   i i SArray set_byte   LOOP
    reset_ij
    BEGIN
        ii KeyArray get_byte   jj +  j_update
        swap_s_ij
        ii 255 < WHILE
        ii i_update
    REPEAT
    reset_ij
;
```

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

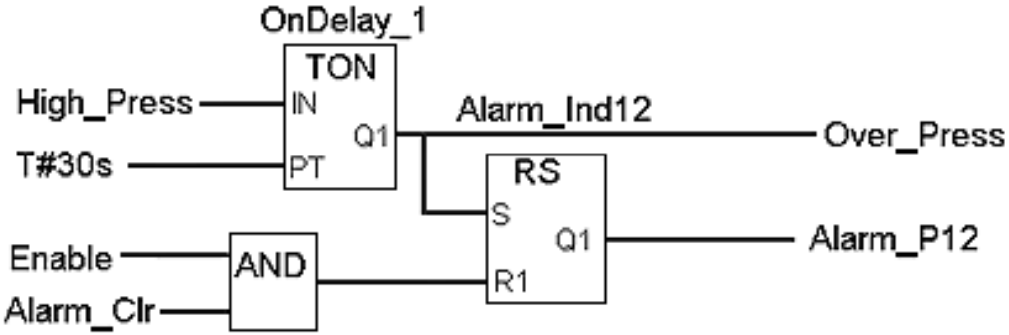
- **Jeziki za programabilne logične krmilnike (Programable Logic Controllers - PLC)**
  - programabilni logični krmilniki se uporabljajo v industriji za vodenje različnih procesov
  - v splošnem jeziki zanje nudijo le osnovno podporo analizi razvrstljivosti
  - ker niso standardizirani, je mednarodna organizacija za standardizacijo IEC naredila predlog standarda za te sisteme
    - IL nabor ukazov (instruction list)
    - LD lestvični diagram (ladder diagram)

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

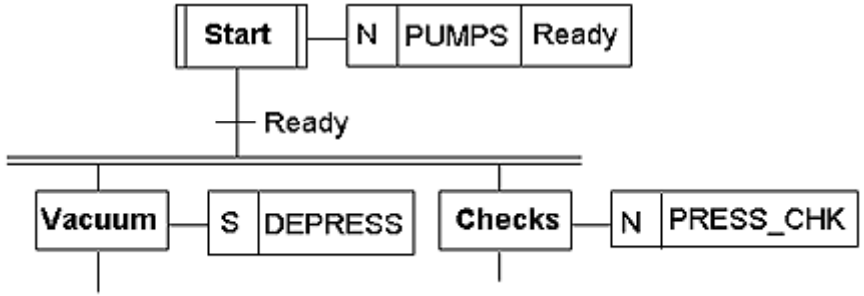
- **Jeziki za programabilne logične krmilnike (Programable Logic Controllers - PLC)**
  - FBD/SFC diagram funkcijskih blokov in sekvenčni funkcijski diagram
  - ST strukturirani tekst (structured text)
  - jeziki naj bi bili medsebojno ekvivalentni (možnost pretvorbe enega v drugega)
  - uporaba eksplicitnih strojnih naslovov perifernih vmesnikov onemogoča prenosljivost
  - časovne lastnosti izvajanja akcij in trajanje posameznih procesov niso pod programsko kontrolo
  - edina časovna kontrola so zakasnitve in ciklično razmeščanje opravil

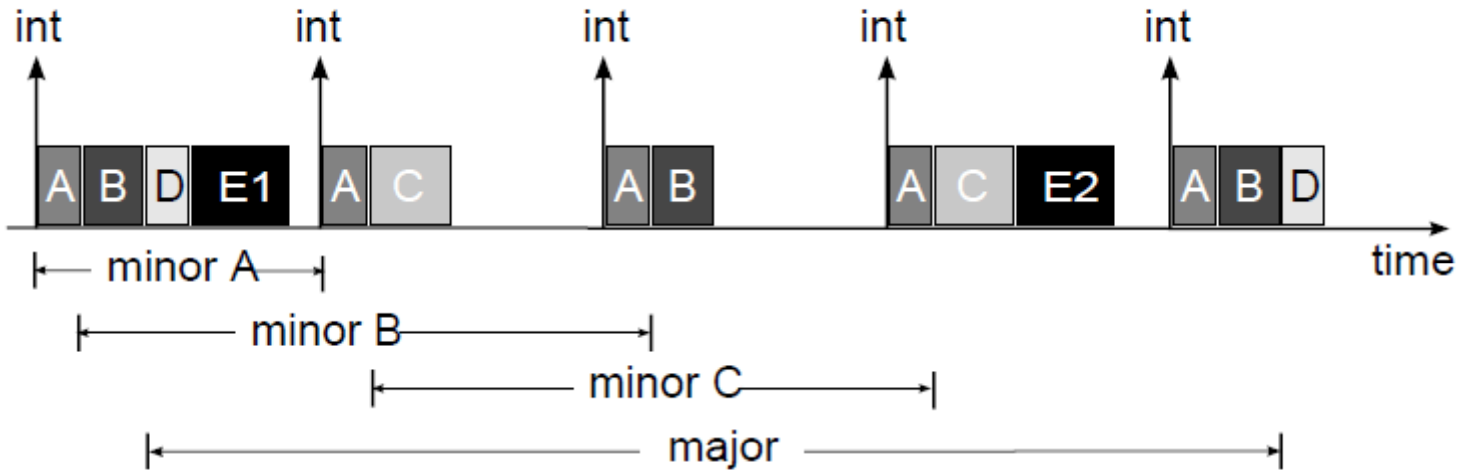



```
If Speed1 > 100.0 then
   Flow_Rate: = 50.0 + Offset_A1;
Else
   Flow_Rate: = 100.0; Steam: = ON
End_If;
```




```
      LD   R1
     MPC   RESET
      LD   PRESS_1
      ST   MAX_PRESS
RESET:  LD   0
      ST   A_X43
```





E1, E2: sporadic tasks

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **Real-Time Euclid**
  - zadošča vsem zahtevam sistemov v realnem času in omogoča analizo razvrstljivosti
  - RTEuclid je proceduralni jezik, namensko grajen za SRČ aplikacije
  - strukturiran in ima strogo preverjanje tipov
  - za konkurenčno izvajanje uporablja procese, monitorji in sinhronizacijske primitive

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **Real-Time Euclid – nad.**
  - Posebnosti:

1. Eksplicitno podajanje časovne enote

   REALTIMEUNIT(time-in-second)

2. Prepoved dinamičnih podatkovnih struktur
3. Časovno omejene zanke

   - maksimalno število ponovitev vsake zanke je znano že v času prevajanja

   - možen je predčasni izhod iz zanke

   EXIT [WHEN bool-exp]

4. Prepoved rekurzije

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **Real-Time Euclid – nad.**
  5. Procesi
     - definirani so statično
     - lahko so periodični ali aperiodični
     - prožimo jih s prekinitvami (dogodki), z drugimi procesi ali s časovnikom
     - vsi pogoji proženja so znani vnaprej
     - vsak proces ima eksplicitno podan časovni okvir, v katerem se mora končati ne glede na dogajanja v sistemu
  6. Pogojne spremenljivke in sinhronizacija med procesi
     - za sinhronizacijo med opravili uporablja razširjeno obliko semaforjev in monitorje

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **ADA**
  - Ada je nastal na zahtevo Oddelek za obrambo ZDA
  - Zamenjava za različne programske jezike, ki se uporabljajo za graditev vgrajenih sistemov
  - vsebuje značilnosti, ki omogočajo varnost, zanesljivost in napovedljivost obnašanja programa v pogojih kritičnih časovnih omejitev
  - dobro strukturiran, modularen in čitljiv programski jezik s strogim preverjanje tipov
  - omogoča neposredni dostop do vhodno/izhodnih naprav in prevajanje po modulih
  - vsebuje definicijo procesov, ki se lahko medsebojno sihronizirajo po principu asimetričnih randevujev

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **ADA – nad.**
  - vključuje obravnavo napak in omogoča sistemsko ali uporabniško obravnavo izjem
  - je velik in kompleksen in ne omogoča dobrega sistemskega načrtovanja
  - semantika Ade pogojuje dodeljevanje virov na osnovi prioritet in strategije FIFO
  - edina ukaza za delo z opravili sta iniciacija in terminacija
  - ne podpira časovnih odvisnosti (razen zakasnitev)
  - slaba podpora za prekinitve
  - ne vključuje zaščite pred smrtnimi objemi

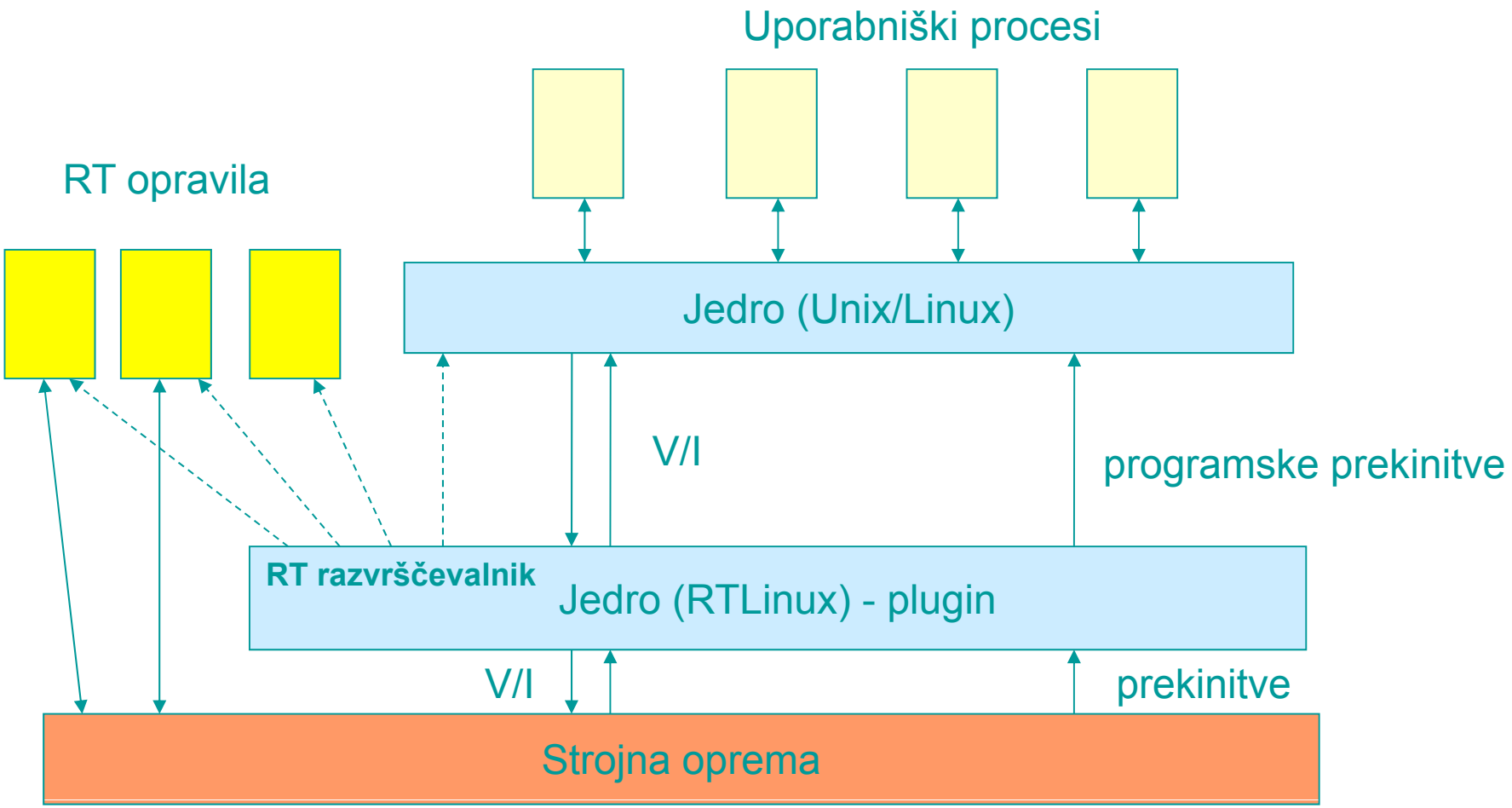
Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **Java**
  - nima podpore za direktni dostop do perifernih vmesnikov
  - nedeterminističnost zaradi dinamičnega dodeljevanja pomnilnika in »garbage colectorja«
  - ne podpira časovnih operacij
- Večino omejitev odpravlja RT-java

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **C/C++**
  - Nima neposredne podpore za SRČ
  - Za delo z opravili, sinhronizacijo, … uporablja funkcije operacijskega sistema
  - Obravnava prekinitev je odvisna od prevajalnika
  - Za delo s časom Ansi C definira tip **time_t** in nekatere osnovne funkcije za delo z njim
  - Omogoča neposredni dostop do registrov perifernih vmesnikov

RTLinux


Uporabniški procesi
RT opravila
Jedro (Unix/Linux)
V/I
programske prekinitve
RT razvrščevalnik
Jedro (RTLinux) - plugin
V/I
prekinitve
Strojna oprema

RTLinux aplikacije

- RT opravila so implementirana kot Linux moduli, ki se neposredno naložijo v pomnilnik računalnika
- Imajo neposredni dostop do strojne opreme in ne uporabljajo virtualnega pomnilnika
- Ne uporabljajo standardnih sistemskih klicev jedra Linux-a
- Uporablja POSIX definicije niti, sinhronizatorjev, …
- www.opentech.at
  http://www.fsmlabs.com/developers/man_pages/function_list.htm

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **PEARL (Process and Experiment Automation Realtime Language)**
  - primernejši za SRČ kot Ada
  - obstajata tri verzije: Basic PEARL, Full PEARL, Distributed PEARL (standardizirane v Nemčiji z DIN in mednarodno z ISO)
  - vsebuje osnovne podatkovne tipe ter CLOCK in DURATION
  - podpira modularno izdelavo kompleksnih programov, omogoča ločeno prevajanje modulov
  - modul lahko vsebuje sistemski del in več problemskih delov

Pregled najpogosteje uporabljenih jezikov za sisteme v realnem času - nadaljevanje

- **PEARL – nad.**
  - omogoča nizkonivojske ukaze za izmenjavo informacij s procesnimi vmesniki (preko virtualnih podatkovnih postaj)
  - dobra podpora razvrščanja in kontrole izvajanja opravil ter za izražanje časovno pogojenega obnašanja
  - nima strukturiranih sinhronizacijskih konstruktov s časovnim nadzorom
  - nestrukturirana obravnava izjem
  - ne omogoča ugotavljanja stanja opravil in sredstev
  - možno je napisati program, ki se izvaja nedefinirano dolgo časa
  - ne omogoča neposredne analize razvrstljivosti

ANALIZA ČASOV IZVAJANJA OPRAVIL IN ANALIZA RAZVRSTLJIVOSTI

- Zakaj je potrebna analiza časa izvajanja opravil
- Analiza razvrstljivosti
- Možnosti izvedbe časovne analize
- Pogoji za doseganje napovedljivosti v višjem programskem jeziku

Zakaj je potrebna analiza časa izvajanja opravil

- Osnovna lastnost SRČ je, da se bodo vsa opravila zaključila do predvidenega skrajnega roka.
- To moramo preveriti še preden damo tak sistem v uporabo oz. ga sploh implementiramo - analiza razvrstljivosti (schedulability analysis).
- Največji problem pri tem predstavlja natančna ocena časa izvajanja:
  - Preveč optimistične ocene $\Rightarrow$ med izvajanjem aplikacije prišlo do prekoračitve rokov.
  - Preveč pesimistične ocene $\Rightarrow$ nepotrebni stroški za doseganje napovedljivosti
- Skrajne čase izvajanja opravil potrebujemo tudi zaradi njihovega razvrščanja (razvrščanje po skrajnih rokih).

Analiza razvrstljivosti

- Predstavlja kompleksen problem. Zahteva pregled vseh možnih stanj, ki lahko nastopijo v sistemu in njihov vpliv na zahtevo za izpolnitev skrajnih rokov.
- Na stanje sistema vplivajo različni notranji in zunanji dogodki: časovni dogodki, prekinitve, vzporedno izvajanje, kritična območja, ipd. – veliko število možnih kombinacij.
- Zato običajno postavimo nekatere omejitve (npr. minimalni čas med pojavo dveh dogodkov)
- Edini obstoječi sistem, ki jo podpira, je RTEuclid z nekatere omejitvami v delovanju sistema: izvajanje opravil v točno definiranih časovnih okvirjih, dodane so omejitve v programskem jeziku.

Možnosti izvedbe časovne analize

- *Analiza specifikacij* - najbolj groba, na osnovi skrajnih rokov in pogojev proženja posameznih opravil
- *Analiza izvorne programske kode* - poznati moramo približne čase izvajanja posameznih ukazov ali delov programa

Možnosti izvedbe časovne analize – nad.

- *Analiza prevedene programske kode* – na nivoju posameznih ukazov mikroprocesorja, poznati moramo strukturo programa, problem optimizacije kode in sodobnih računalniških struktur
- *Povezava prevajalnika in časovnega analizatorja* – upošteva tako strukturo programa kot optimizacijo kode, lahko doda kodo za povečanje performančnih rezerv sistema.
- *Neposredno merjenje časa izvajanja kode* – najnatančnejša

Ugotavljanju časa izvajanja opravil na nivoju ukazov višjega programskega jezika

1. Zaporedje stavkov

sz ::= s1 s2 .. Sn

$$t_m(s_z) = \sum_{i=1..n} \left(t_m(s_i) \right)$$

Možnosti napak:

```
for i:=1 to n do
...
end for
for j:=n+1 to m do
...
end for
```

Ugotavljanju časa izvajanja opravil na nivoju ukazov višjega programskega jezika

2. Odločitveni stavki

2.1 Stavek if

*sif ::= **if** pogoj **then** sthen **else** selse*

$$t_m(s_{if}) = t_m(pogoj) + \max\{t_m(adm_{then}) + t_m(s_{then}), t_m(adm_{else}) + t_m(s_{else})\}$$

Podrobnejši izračun:

$$t_m(s_{if}) = \max\{t_m(pogoj_{true}) + t_m(adm_{then}) + t_m(S_{then}), t_m(pogoj_{false}) + t_m(adm_{else}) + t_m(s_{else})\}$$

Ugotavljanju časa izvajanja opravil na nivoju ukazov višjega programskega jezika

2.2 Stavek case

scase ::= case izraz of

pogoj1 : s1

pogoj2 : s2

...

pogojn : sn

end case

$$t_m(s_{case}) = t_m(izraz) + \max_{i=1..n}\left\{ t_m(s_i) + t_m(adm_i) + \sum_{j=1..i} t_m(pogoj_j) \right\}$$

Ugotavljanju časa izvajanja opravil na nivoju ukazov višjega programskega jezika

3. Ponavljalni stavki (zanke)

3.1 Zanka for

sfor ::= for id = sp_meja to zg_meja do

stelo

end for

$$t_m(s_{for}) = t_m(adm_{for1}) + (t_m(s_{telo}) + t_m(adm_{for2})) * \max\{\text{zg\_meja} - \text{sp\_meja} + 1, 0\}$$

Ugotavljanju časa izvajanja opravil na nivoju ukazov višjega programskega jezika

3.2 Zanki while in repeat

swhile ::= while pogoj do maxloop zg_meja

stelo

end while

srepeat ::= repeat maxloop zg_meja

stelo

until pogoj

$$t_m(s_{while}) = (t_m(pogoj) + t_m(adm_{while}) + t_m(s_{telo})) * \text{zg\_ meja}$$

$$t_m(s_{repeat}) = (t_m(s_{telo}) + t_m(pogoj) + tm(adm_{until})) * \text{zg\_ meja}$$

Lastnosti programskih jezikov za doseganje napovedljivosti

- Ni stavkov GOTO
- Število ponovitev zank mora biti strogo omejeno
- Izključitev dinamičnih podatkovnih struktur, kazalcev in rekurzije
- Vsi sinhronizacijski ukazi morajo biti časovno omejeni

Lastnosti programskih jezikov za doseganje napovedljivosti

- Možnost eksplicitnega podajanja časa izvajanja oz. dodatni navodil prevajalniku za realnejšo oceno časa izvajanja posameznih delov kode
- Izključitev sekundarnih pomnilnih enot (diski)
- Izključitev podatkovnih baz, lupin ekspertnih sistemov itd.