PVS - P - 1

18.2.2013

Ocene → projektno delo! ☺

TEST
TEST

Laboratorijske vaje cca. 14 dni kasneje

skripta!

Vgrajeni sistemi

več povezanih (vgnezdenih) enot

1. Multitasking
2. Real-time

PVS - P - 2

20.2.2013

Embedded sistemi v avtu:

- modularno
- manjša poraba bakra za napajanje
- večja zanesljivost (manjša kompleksnost - posamezni sklopi)
- posamezni sklopi so bolj zanesljivi!
- lažje izvajanje Multitasking

PVS - P - 3

25.2.2013

Večopravilnost

Realni čas

Bonsai...

↳ Urnik

Časovne nezine

Pon
Tor
Sre
Čet
Pet
Sob
Ned /

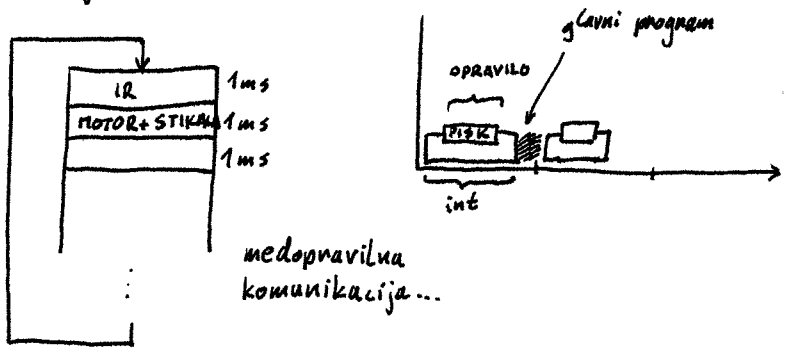
PVS - P - 4

Časovno rezinjenje

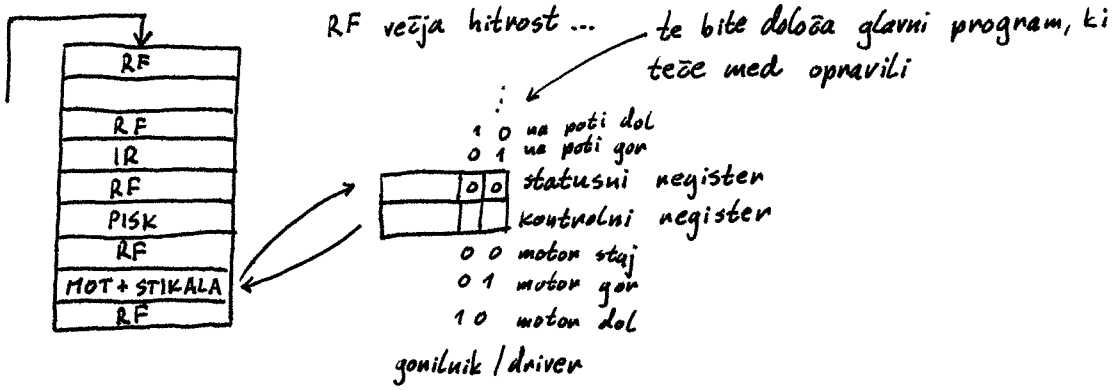
1. ČASOVNO REZINJANJE (Time slicing)

2. RAZVRŠČANJE (Scheduling)

- vse časovne rezine enako velike
- opravila se končajo v svoji časovni rezini
- urnik je ciklični in konstanten
- ni zunanjih prekinitev

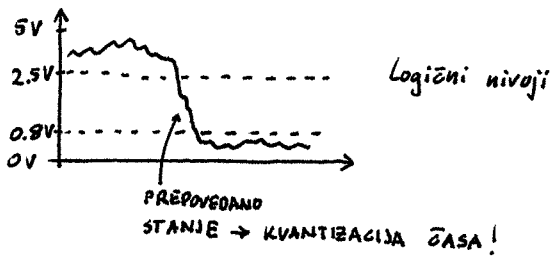


za našo rampo:

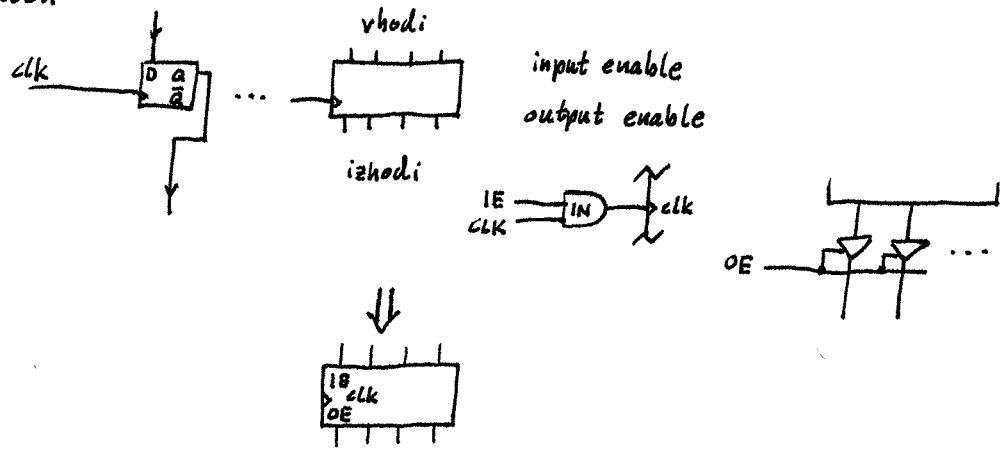


NVS - P - 5

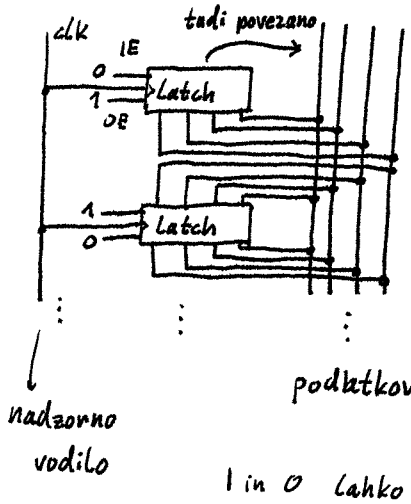
27.2.2013



Latch

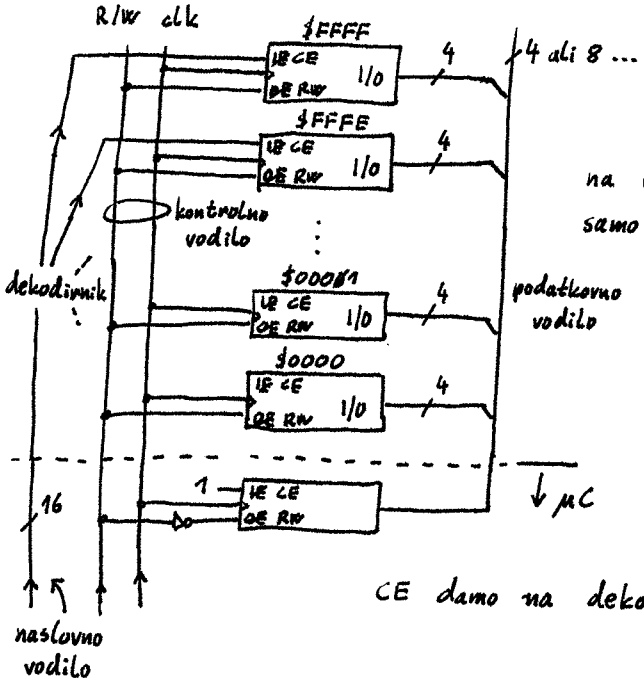


Vodila



vsi OE so na 0 razen tistega, ki oddaja
vsi IE so na 0 razen tistega, ki sprejema

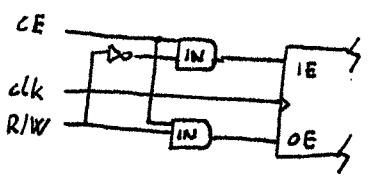
1 in 0 lahko povežemo skupaj → 1/0



na vodilu je aktiven samo en vhod IE ali OE!

chip enable

IE	OE	CE	R/W	
0	0	0	X	
0	1	1	1	pomeni read
1	0	1	0	
1	1			

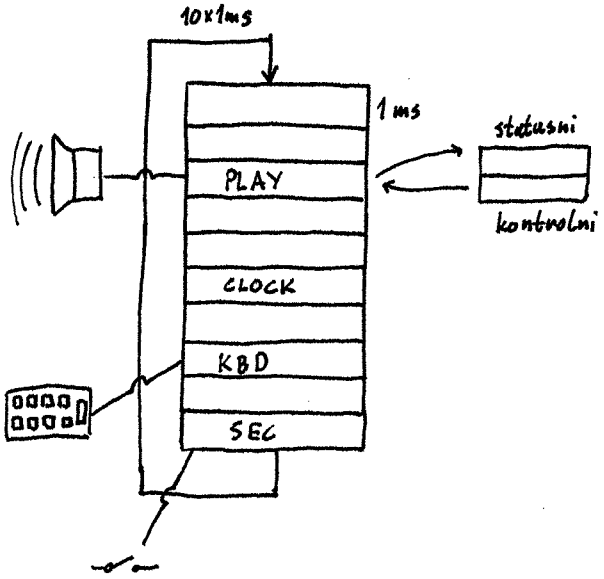


vse R/W damo skupaj

CE damo na dekodinnik $16 \rightarrow 2^{16}$

PVS - P - 7

4.3.2013



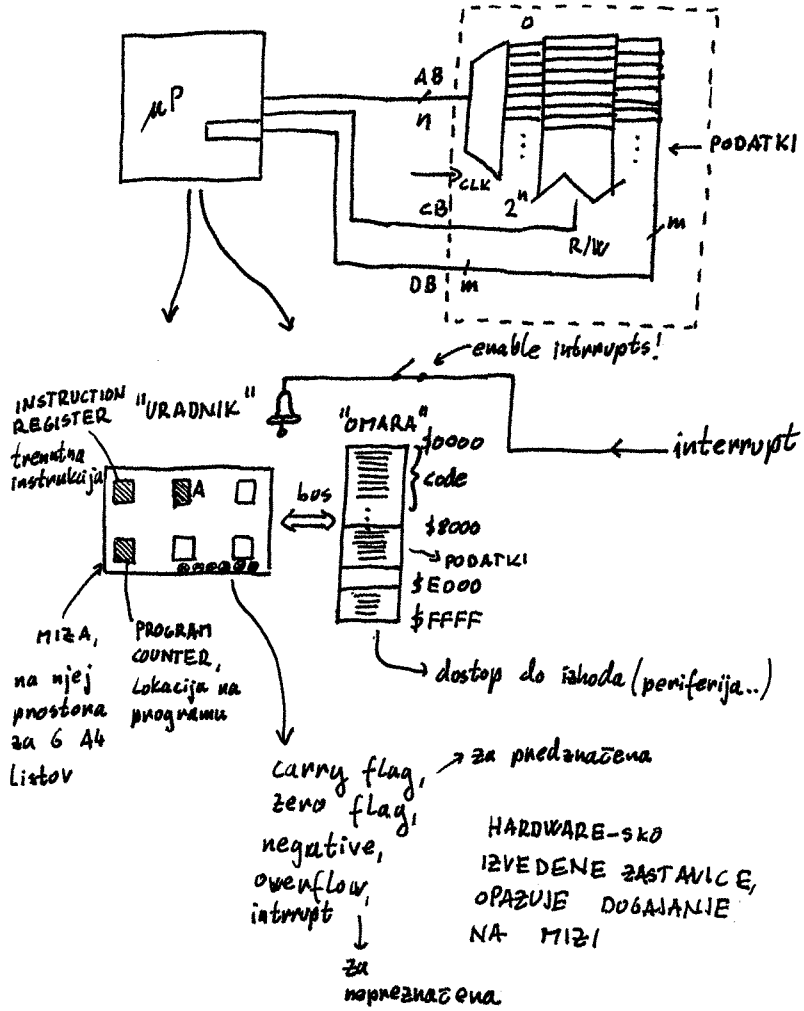
atom → časovno nedeljiva sekvenca
urinih ciklov

sch_syn (14)

zanka preverja če je še dovolj časa do prikinitve ...
po tasku je dovolj časa in se taknat izvede atom

NVS - P - 8

6.3.2013



ob intrrruptu se postavimo na nek naslov (intrrupt rutina), kar je na mizi damo v stack (data segment), $8000 in naprej... po končani interrupt rutini se na mizo naloži program iz stacka (stack pointer)

PVS - P - 8

11. 3. 2013

*.c
↓ compiler
*.s
↓ download
flash (omara)

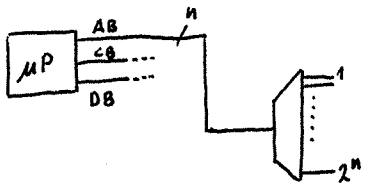
UVOD

- Roki (deadlines)
  1) Hard deadlines (obvezni)
  2) Firm deadlines (neobvezni) če bi se odzvali prepozno, se ne odzovemo (recimo telefonija)
  3) Soft deadlines (mehki) lahko se odzovemo tudi kasneje (prenašanje podatkov, ne govora)

- Splošni pogoji za RT
  a) Kompleksnost
  b) Zanesljivost (HW+SW)
  c) Varnost (človeška škoda)
  d) Primeren za verifikacijo (debug)
  e) Primeren za validacijo (dokaži, da sistem izpolnjuje predpisane specifikacije)
  f) Modularnost -kaj naročnik želi od produkta!
  g) Omejena sredstva (baterija..., vibracije, vlaga, temp. ...)

Diagnostika napak

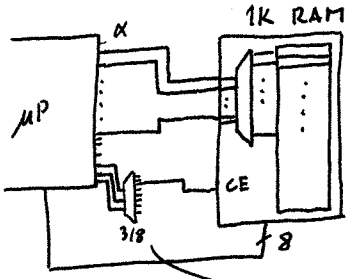
Dekodiranje
(naslovno vodilo)



osnova dekodirnik 3/8

Pomnilniška mapa:

naslov:	aktiven:	
0x0000	R_0	1K RAM (1024 celic po 1 byte)
0x0001	R_1	
⋮		
0x03FF	R_{1023}	
0x0400	R_0	
⋮		



→ razdeli RAM na 8 kosov

če imamo x-e nad naslovno vrstico se celice izmenjuje $R_0, R_0, R_1, R_1, \ldots$

če so x pod naslovno vrstico se celice ponavljajo $R_0, R_1, \ldots, R_{1023}, R_0, R_1 \ldots$

Naslavljanje:

1) statično dinamično (povezava do naslovov se spreminja - dinamično)
2) eksplicitno implicitno
3) popolno nepopolno (1 naslov, ena celica)
4) simetrično asimetrično (dekodirnik na zgornjih bitih simetrično razdeli RAM)

Verifikacija in validacija

```
function fac(n:int) return int;
begin
   if n=0 then return 0;   (1 za n!)
   else return n*fac(n-1);
end
```

fja naj bi računala faktoriele (n!)

$n \geq 0$
$n \leq 7$ } pogoji

Determinističen odziv

(predvidljiv)

vedno se sistem odzove enako!
tudi glede na čas izvajanja

WCET

"registrske spremenljivke"
kot na miši "vradnika"

za merjenje WCET je pametno izklopiti optimizacije

izvori nedeterminizma:

- semantika jezika

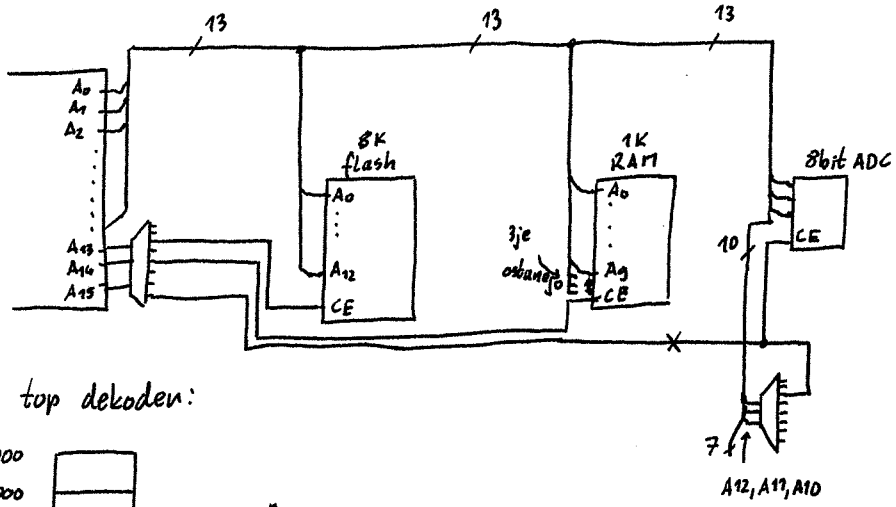
```
x=0
if 4<x and 3<func(10) then
```

(ali se ta izvede?)

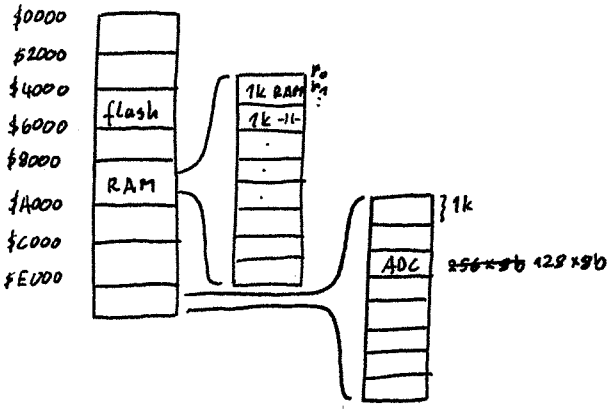
- prevajalnik
  ne vemo kako bo prevajalnik optimiziral program
- Data cache
  vmesni hitri / majhen pomnilnik
  za embedded sisteme NE uporabimo cache-a
- DMA
  procesor mora včasih počakati, ker nekdo drug dostop do vodila → pomnilnika
  za embedded NE uporabimo
- OS - paraleliżem - NE

NVS - P - 11

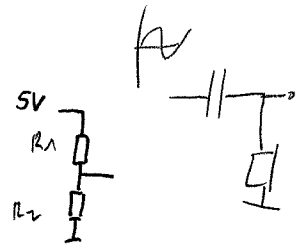
20.3.2013



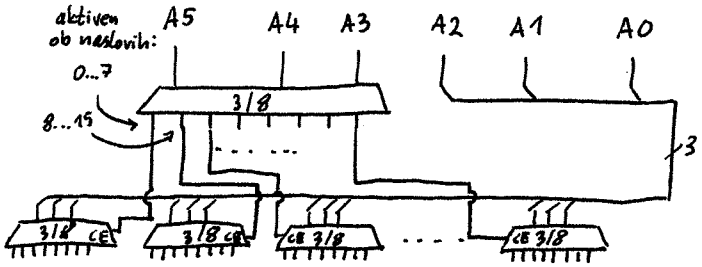
top dekoder:



Kaskadni dekodirniki

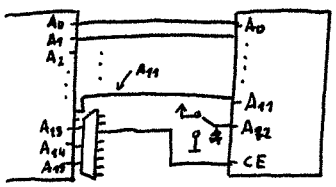


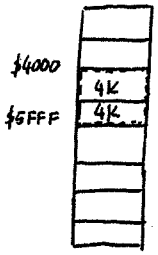
želimo 6/64



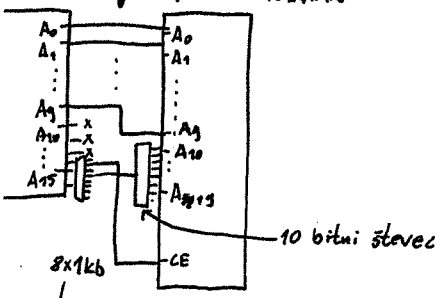
64 izhodov

2x4kb pomnilnika





dva zrcaljena, ze dodamo stikalo nasljavljamo en in zgdnug del



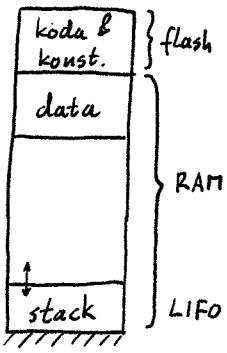
dummy write - ko dostopamo števec ta preskopi na naslednjo stran

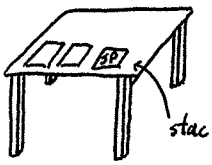
procesor

Upravljanje s pomnilnikom:

(memory management)

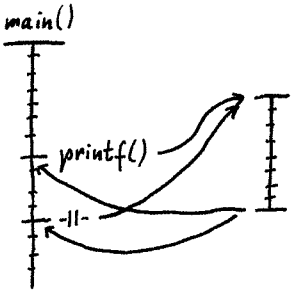
- stack
- heap





stack pointer (nakazuje prvo prosto lokacijo stacka)

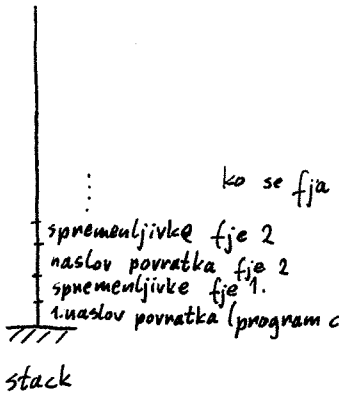
stack ni fiksna struktura



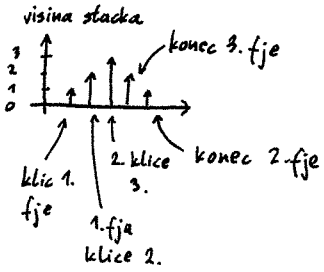
pred klicem funkcije si shranimo lokacijo, od kje je bila fja klicana

tudi spremenljivke (lokalne) se dajejo na sklad (te spremenljivke samo za čas izvajanja fje).

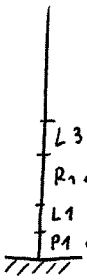
Lahko bi shranili programski števec ob klicu, vendar nastane problem, če fja kliče še eno fjo → uporabimo stack



ko se fja 2 konča, pobrišemo te spremenljivke in se vrnemo navzdol po stacku



ob prekinitvi se morajo delovni registri nekam shranit (vsi razen stack pointerja)

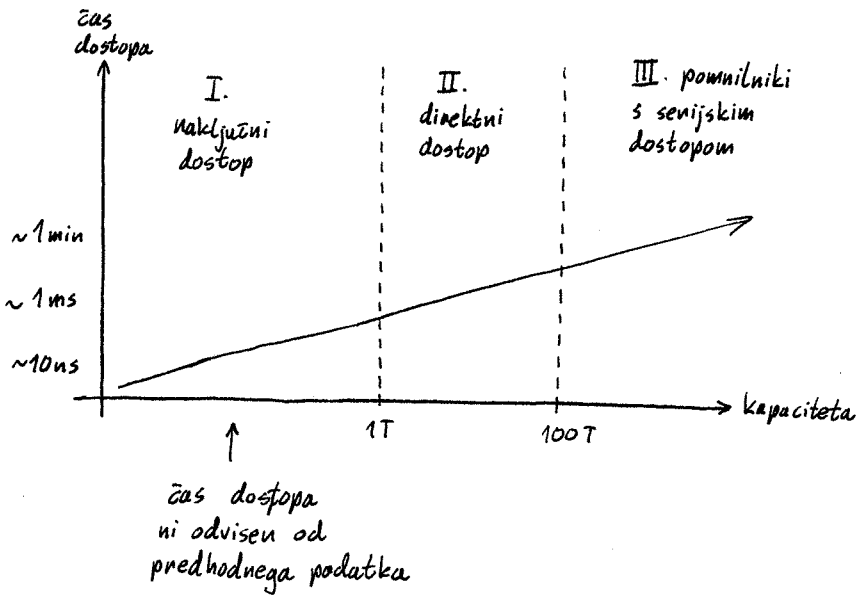


daljše instrukcije se izvedejo preden se začne izvajati prekinitev

NVS - P - 13

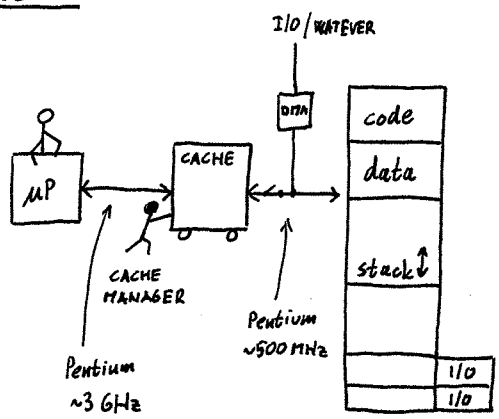
27.3.2013

Pomnilniki



Cache

3.4.2013



za real-time nočemo cache-a in DMA!

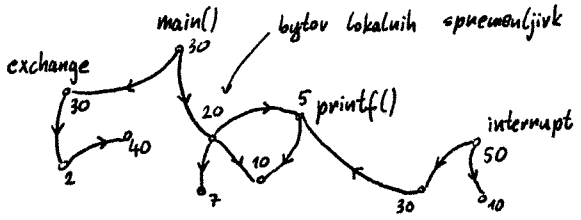
PVS - P - 14

8.4.2013

Stack

koliko nabimo stacka?

metoda: usmerjen graf klicev

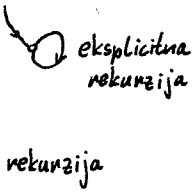


najdaljša pot 102 → če vzamemo 1k je ziher!

interrupt najdaljše 85

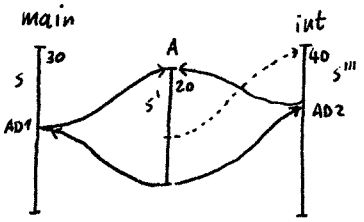
$\Sigma = 187$

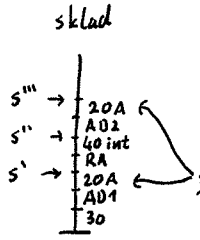
kaj pa takole?



Re-entrancy

funkcija je prekinjena med izvajanjem in se po koncu prekinitve izvaja naprej - prekinitev kliče to funkcijo!





če so lokalne spremenljivke so vredu

se ne motijo podatki

če damo globalne spremenljivke, je problem!

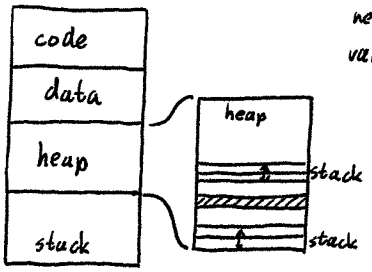
Dinamično dodeljevanje pomnilnika

(Heap management)

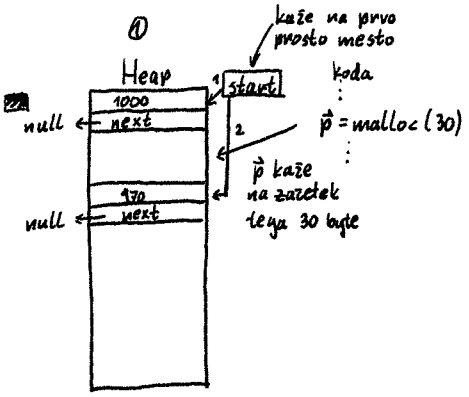
$\vec{p}$ = malloc (400)

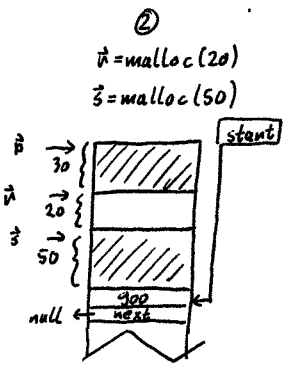
free ( )

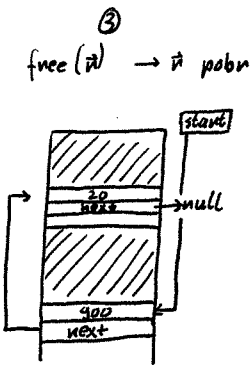
⋮

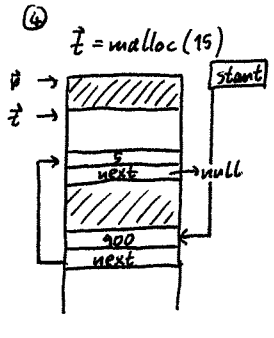


necimo sistem za varovanje vstopa

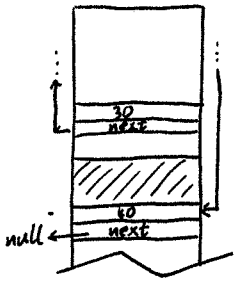








PVS - P - 15



če sedaj pobrišemo ta sektor, dobimo tni fragmentirane,

program mora prevenit sosednje sisteme, kar je časovno nedeterministično

NVS - P - 15

10.4.2013

Implicitno dekodiranje

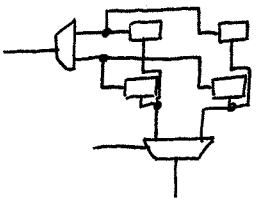
v čipu dejansko nimamo dekodirnika, ker bi nabili preveč linij (miljon za 1 mega)

a) linearno (kot že poznamo)

b) kvadratasto (mreža)

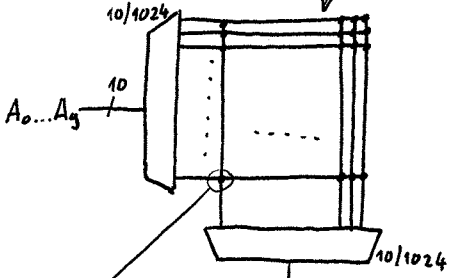
ena opcija:

mrežno:



za večino 1 Mb

bitno polje (1024·1024 presečišč)



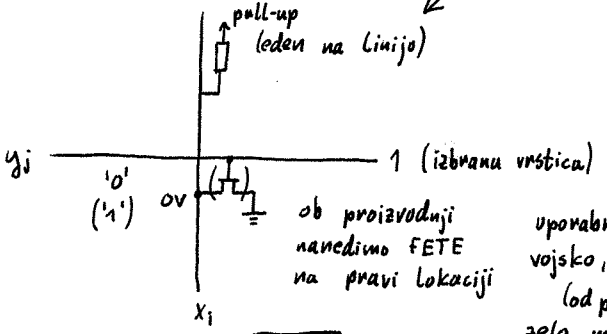
Linearni so redki (so pa hitrejši)

10
$A_{10} \dots A_{19}$

vsako presečišče bo shranilo en BIT

teh bitnih navnin polj (bitnih polj) je lahko več, oba kodinnika lahko nadzinata več navnin

ena vrstica ROM

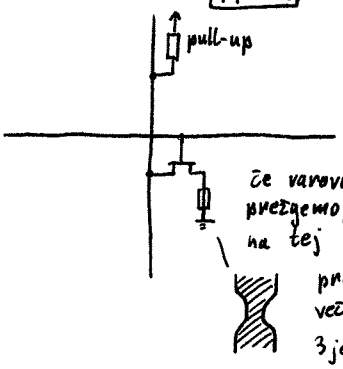


ob proizvodnji nanesemo FETE na pravi lokaciji

uporabno za vojsko, vesolje (odporno na gamma žarke)

zelo majhna poraba!

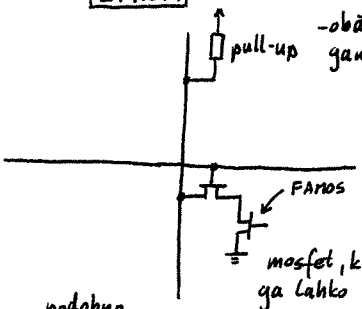
PROM



če varovalko prežgemo, dobimo enico na tej lokaciji

prežgemo z večjo napetostjo 3 je ~ms pulzi

EPROM



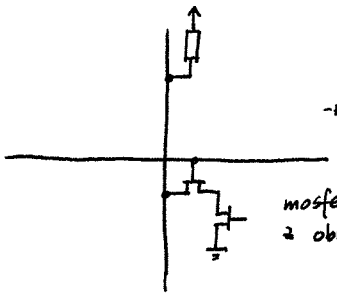
-občutljiv na gamma!

mosfet, ki ga lahko "zapečemo" s prebojem, v gate se nekako vgradi kapacitivnost, brišemo z UV! izbije elektrone iz gatea

podobno programiranje

NVS - P - 16

EEPROM ≐ flash

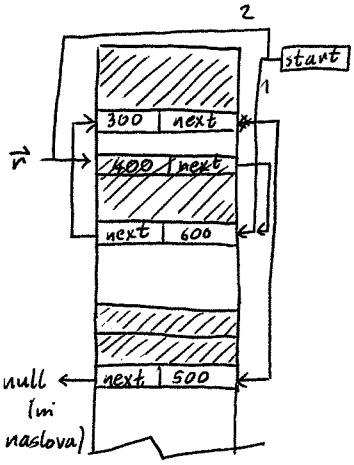


PVS - P - 16

15. 4. 2013

Heap (parkirišče) se hitro fragmentira

za računalnik je zelo učinkovito, da se da podatke "kam nekam", shranjujemo pa pozicijo

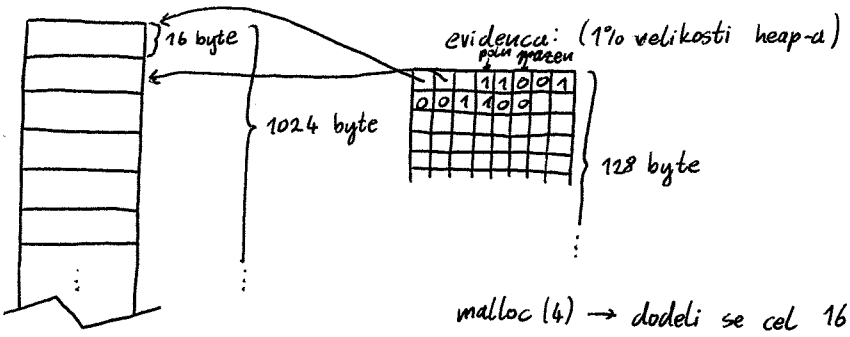


malloc (400)

free (r) ∇ defragmentacija!! časovno nedeterministično

Bitno polje:

heap razdeljen na bloke znane velikosti, krmilnik ima tabelo lokacij



malloc (4) → dodeli se cel 16 byten blok

Za vgrajene sisteme!

Memory leak:

preveč malloc, premalo free

spomina zmanjkuje → "memory is leaking"

Memory fault:

uporaba že sproščenega dela pomnilnika

Metode za dodeljevanje naslova:

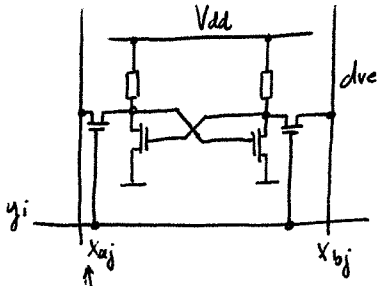
- Best-fit
- First-fit
- Next-fit (~~ne prvega zaporednega naslednjega~~) išče od tam, kjer je ostal prej
- Buddy-fit (predvideva, da bodo naslednji podatki podobno veliki)
- Worst-fit (minimizira fragmentacijo)

Knuttovo pravilo - samo 66% izkoristek pomnilnika

NVS - P - 17

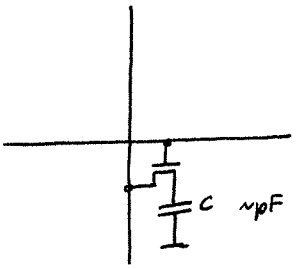
18.4.2013

SRAM Pomnilna celica:



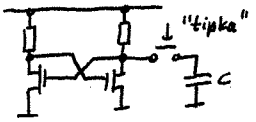
zelo hitro pisanje!

DRAM:



porabi zelo malo prostora

osveževanje z S-RAM celico



PVS - P - 17

22.4.2013

Razvrščanje

Opravila (tasks)

- sinhrona (periodic, polling)
- asinhrona (event-driven, naključno) "heuristic"

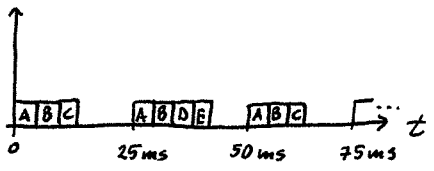
trije tipi razvrščanja:

1) CIKLIČNO (ročno) — časovne reziine daljše od opravil — not-preemptive
2) TIME-SLICED scheduling — preemptive
3) PRIORITY BASED scheduling — preemptive

preemptive → predčasen (odzovemo pred časom)

CIKLIČNO primer:

Task	perioda	WCET
A	25 ms	10 ms
B	25 ms	8 ms
C	50 ms	5 ms
D	50 ms	4 ms
E	100 ms	2 ms



- recimo, da želimo nekam vriniti prekinitev, problem nastane, če se sproži tik pred novo režino, interrupt pa želimo obdelati takoj
- interrupt mora vpisati čas, koliko ga je pokuril, zato da kasneje RTOS ta čas uporabi, da se naslednja režina ne začne prepozno
- za interrupt poznamo WCET in periodo (kako pogosto pričakujemo interrupt)
- problem je tudi v skupnem dostopu opravila in interrupta

NVS - P - 18

24.4.2013

Komunikacijska vodila

za I/O periferijo zunanje vodilo!

za zunanje enote ne potrebujemo naslovnega vodila zaradi manjšega števila enot (?)

PARALELNA / SERIJSKA

↓ več podatkov hkrati (ob istem urinem ciklu)

SINHRONA / ASINHRONA

moderna vodila imajo boljše odkrivanje napak

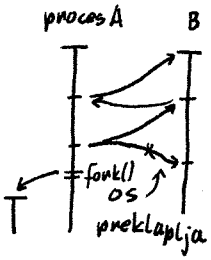
~~gpib~~	Ethernet	asinhroni podčrtani
GPIB	RS232	
LPT	USB	
ATA	CAN	
IDE	UART	
PCI	I2C	
SCSII	SPI	
SD	BT	
	IR	
	RF	
	morse	

PVS - P - 18

6.5.2013

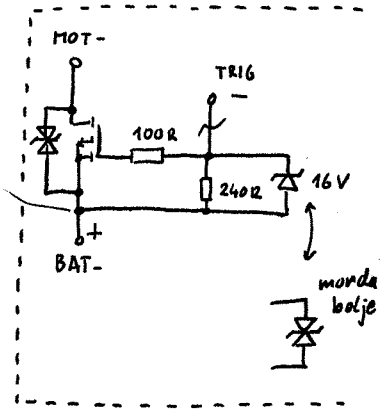
Linux		Win
process	=	application
thread	=	~~to~~ process
(task = RT thread)		

Sočasni programi



navadna fja:

fork:

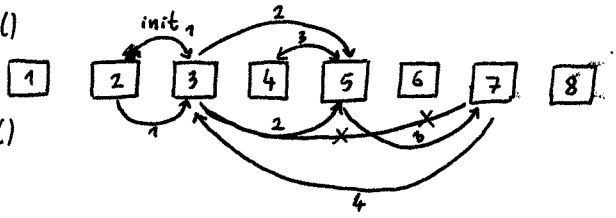


Stanja procesov

1) non-existing
2) created
3) initialized
4) runnable
5) running
6) blocked
7) completed
8) terminated

main()

task()

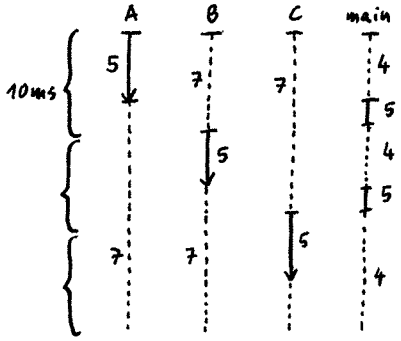


Načini razvrščevanja

1) ciklično — non-preemptive
2) časovno rezinjenje — pre-emptive
3) prioritetno razvrščanje — pre-emptive

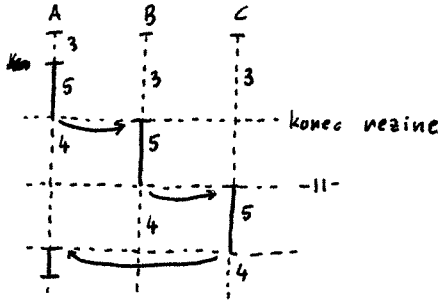
PVS - P - 19

CIKLIČNO:



RT ✓

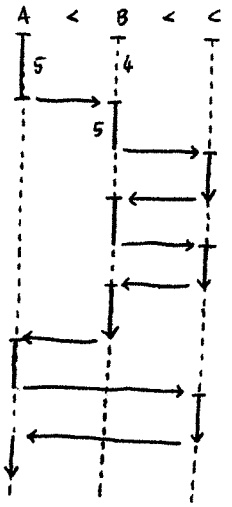
TIME-SLICING:



nabimo stack za vsak proces

~~RT~~

PRIORITETNO:



dovolj en stack

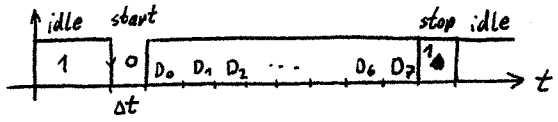
RT ✓✓

NVS - P - 19

8.5.2013

RS232

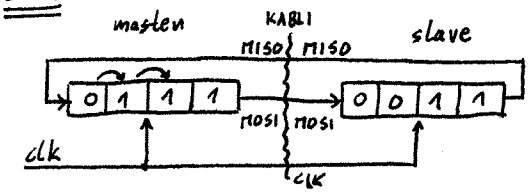
Logična 1 od 3 do 12 V
Logična 0 od -12 do 0 V



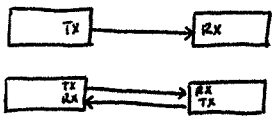
V praksi velikokrat beremo en bit trikrat (okoli sredinske lege) zaradi varnosti

stop bit zagotavlja fronto pri start bitu, če pošiljamo ničle

SPI



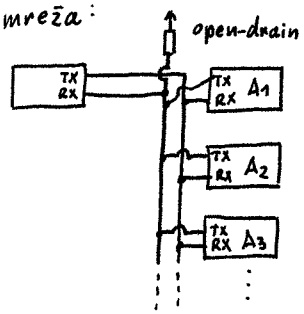
Simplex
Half-duplex
Full-duplex

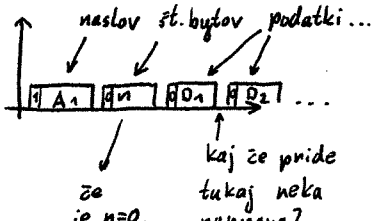


sim.

half. (ne hkrati)
full. (tudi hkrati)

mreža:

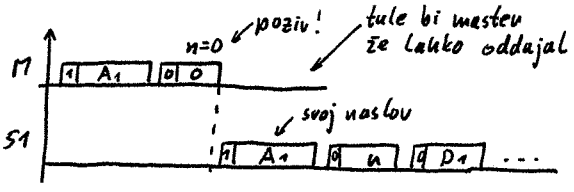




če je n=0, je to poziv za govor

kaj če pride tukaj neka naprava?

rečemo naslov 9 bitov (en bit 1!)
podatki 9 bitov (en bit 0!)



PVS - P - 20

13.5.2013

Monotono prioritetno razvrščanje

vedno se ve, kdo ima višjo prioriteto (glej prejšnjo stran)

proces	'T' perioda	'c' WCET	'P' prioriteta	
A	150 ms	12 ms	1	24 %
B	40 ms	10 ms	2	25 %
C	30 ms	10 ms	3 ← najvišja	33.3 %
				82.3 %

perioda 50 ms pomeni, da se mora A v teh 50 ms enkrat izvesti

ne zlavfa!
(problem pri 600 ms)

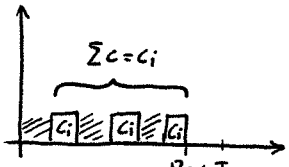
dve formuli za izračun:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} < N \cdot \left(2^{1/N} - 1\right)$$

zadosten pogoj
NE potreben!

N	$N \cdot (2^{\frac{1}{N}} - 1)$
1	1
2	0.828
3	0.780
4	0.755
⋮	
∞	ln 2 = 0.693...

proces	response-time
A	52 ms
B	20 ms
C	10 ms



zaokroževanje navzgor!

$$R_i = C_i + \sum_{j=i+1}^{N} C_j \cdot \left\lceil \frac{R_i}{T_j} \right\rceil$$

višja prioriteta

- $R_C = 10\,ms$
- $R_B = 10\,ms + 10\,ms \cdot \frac{R_B}{30\,ms} = 0, 10, 20, 20$
  ↳ $R_B = 20\,ms$
- $R_A = 12\,ms + 10\,ms \left(\frac{R_A}{40\,ms}\right) + 10\,ms \left(\frac{R_A}{30\,ms}\right) = 0, 12, 32, 42, 52, 52$
  ↳ $R_A = 52\,ms$

$R_i \leq T_i$

20.5.2013

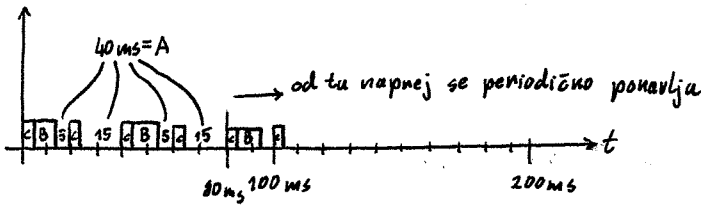
proces	T	C	P	μ	R
A	80 ms	40 ms	1	0.5	80 ms
B	40 ms	10 ms	2	0.25	15 ms
C	20 ms	5 ms	3	0.25	5 ms
				Σ = 1.0	

po metodi fiksne točke (kot prej...)

$$R_A = 40\,ms + 10\,ms \cdot \left\lceil \frac{R_A}{40\,ms} \right\rceil + 5\,ms \cdot \left\lceil \frac{R_A}{20\,ms} \right\rceil = 0, 40, 60, 75, 80$$

procesor dela ves čas!

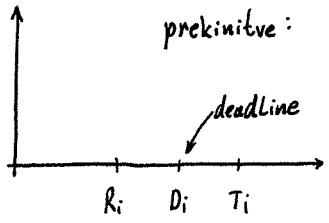
vedno je pametno narisat!



PVS - P - 21

proces	T	C	P	R
A	50ms	12ms	1	52 ms
B	40ms	10ms	2	20 ms
C	30ms	10ms	3	10 ms

tudi za prekinitve poznamo neko periodo (znotraj katere ne pričakujemo nove prekinitve)



$R_i \leq D_i \leq T_i$

A in B sta prekinitve

proc.	T	D	C	P	R
A	20ms	5ms	3 ms	4 ~~2~~	10ms
B	15 ms	7 ms	3 ms	3 ~~3~~	7 ~~ms~~
C	10 ms =	10 ms	4 ms	2 ~~4~~	4 ~~ms~~
D	20 ms ≠	20 ms	3 ms	1 ~~1~~	

← računamo, tako kot prej

pri prekinitvah za periodo vzamemo deadline

R
3 ms
6 ms
10 ms
20 ms

$R_B = 3ms + 3ms \left\lceil \frac{R_B}{20ms} \right\rceil = 0,3,6$

$R_C = 4ms + 3ms \left\lceil \frac{R_C}{15ms} \right\rceil + 3ms \left\lceil \frac{R_C}{20ms} \right\rceil$
$= 0,4,10$

Za "mehkejše" RTOS uporabljamo enake formule, vendar namesto WCET vzamemo AET (average execution time)

Arbitraža

čakanje procesov, ki upravljajo z eno napravo (recimo več ljudi pred dvigalom)

PVS - P - 21

27.5.2013

Programski atomi

- instrukcija se mora dokončati (daljše instrukcije več ciklov...)
- nalaganje naslova povratka, sklad

po defaultu se pri procesorjih izklopijo prekinitve med časom izvajanja neke prekinitve

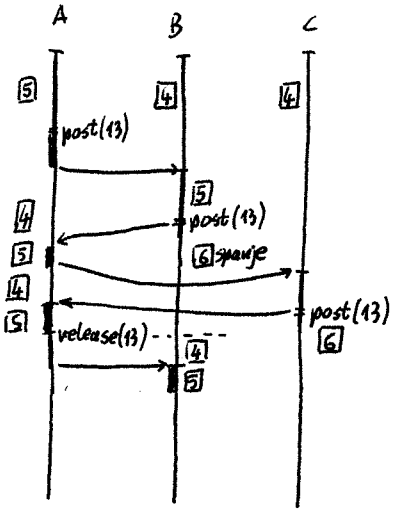
Semafonji

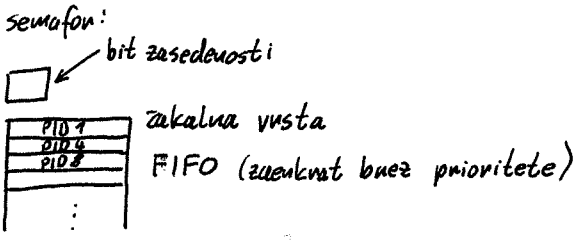
včasih task z višjo prioriteto ne sme prekiniti nekega z nižjo, ze ta sekvenčno bene nek pomemben podatek

Semaforji

- binarni semafor (mutex)
- števni semafor

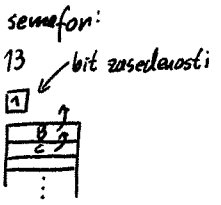
MUTEX:





post(...) "prijava v semafor"

release(...) "končal"

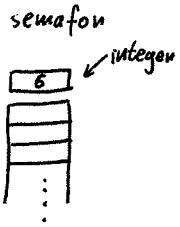



```
post(m)
{ int=0
  if not m.flag then
     m.flag = true,
  else
     enqueue m.queue ID
     ID → 6
} int=1
```

```
release(m)
{ int=0
  if not empty m.queue then
     dequeue m.queue
     ID → 4
  else m.flag = false
} int=1
```

atomski funkciji!

ŠTEVNI:



če int ≠ 0, zmanjšaj za 1

če je 0, pojdi v čakalno vrsto

NVS - P-23

29.5.2013

Dokumentiranje

pomanjkovanje dokumentacije pri programiranju
je pogosto zaradi preprostati spreminjanja programa

Nivo	Jezik
- Povpraševanje (opis izdelka, ki ga neka firma želi)	- slo
- Ponudba	- slo
- Projektna dok.	- slo-ang
- Programska dok.	- ang (~~slo~~)
- In-line dok. (komentarji v kodi)	- ang (~~(slo)~~)
- Imena spremenljivk in rutin	- ang (~~(slo)~~)