



Mikroračunarski sistemi

20ER6004



Napredno programiranje mikrokontrolera




MPLAB XC8 C



MPLAB XC8 C kompajler

- MPLAB XC8 C kompajler je samostalni, optimizovani ISO C99 kompajler
- Podržava sve 8-bitne PIC mikrokontrolere i dostupan je za mnoge popularne operativne sisteme (Microsoft® Windows® 7 (32/64 bit), Windows® 8 (64 bit), Windows® 10(64 bit), Ubuntu 16.04 (32/64 bit), Fedora 23 (64 bit) i Mac OS X 10.12 (64 bit))
- Postoji jedno zaglavlje, koje se tipično uključuje u sve izvorne C datoteke - <xc.h>
- To je generičko zaglavlje, koje uključuje ostala specifična zaglavlja, omogućuje pristup registrima specijalne namene na osnovu posebnih promenljivih, makroe za pristup memoriji i uključuje odgovarajuće instrukcije



Konfiguracija korišćenjem „#pragma config“

- **#pragma config <setting>=<named value>**
// Oscillator Selection bits : RC oscillator
// Watchdog Timer : WDT enabled
// Power-up Timer Enable bit : Power-up Timer is disabled
// Code Protection bit : Code protection disabled
#pragma config FOSC = EXTRC, WDT = ON, PWRTE = OFF, CP = OFF
- **#pragma config <setting>=<literal constant>**
#pragma config FOSC = 0x3, WDT = 0x1, PWRTE = 0x1, CP = 0x3FF
- **#pragma config <register>=<literal constant>**
#pragma config CONFIG = 0xFFFF
// IDLOC0 @ 0x2000, IDLOC1 @ 0x2001, IDLOC2 @ 0x2002, IDLOC3 @ 0x2003
#pragma config IDLOC0 = 0x127, IDLOC1 = 0x124



Registri specijalne namene

- Pristup kompletnim registrima:
 - INDF (0x00), TMRO (0x01), PCL (0x02), STATUS (0x03), FSR (0x04), PORTA (0x05), PORTB (0x06), EEDATA ((0x08), EEADR (0x09), PCLATH (0x0A), INTCON (0x0B), OPTION_REG (0x81), TRISA (0x85), TRISB (0x86), EECON1 (0x88), EECON2 (0x89)
- Primer pristupa
 - **PORTA** = 0x00;
- Pristup pojedinačnim bitovima
 - STATUSbits, PORTAbits, PORTBbits, PCLATHbits, INTCONbits, OPTION_REGbits, TRISAbits, TRISBbits, EECON1bits
- Pojednim bitovima
 - **PORTAbits.RA2** = 1;



Real-Time Operating Systems - RTOS



RTOS

- Sledeća stepenica u kreiranju složenih sistema
- Omogućuje izvršenje više zadataka i upravljanje vremenski kritičnim procedurama
- Koristi se u kombinaciji sa aplikacijama pisanim u C-u

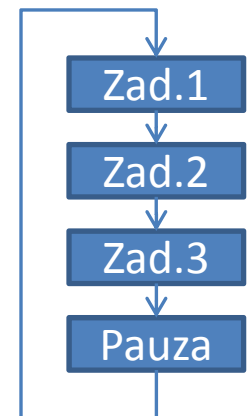


Kada treba koristiti

- RTOS koristi sistemske resurse (FLASH, RAM i procesorsko vreme); zato se koristi na PIC24, dsPIC i PIC32 familijama (ne preporučuje se na PIC18 ili slabijim)
- Potpomaže modularni dizajn i kod *reuse*
- Olakšava pisanje aplikacija koje koriste više kompleksnih biblioteka

Multi-tasking

- Kako sistem postaje složeniji, sve teže je balansirati mnogobrojne aktivnosti (svaka se bori za CPU vreme i time može izazvati značajno kašnjenje nekog drugog dela sistema)
- Zadatak (task) je deo programa koji rešava konkretan problem i ima konkretan rezultat
- Multi-tasking podrazumeva simultano izvršavanje više zadataka
- Obično smo do sada program organizovali u jednu „super-petlju“ gde se zadaci izvršavaju sekvencijalno
- Pauza na kraju petlje definiše frekvenciju ponavljanja ciklusa





Rad u realnom vremenu

- Petlja je najprostiji oblik multi-taskinga, jer primenjuje striktnu rotaciju zadataka
- Ako svi zadaci nisu istog prioriteta, petlja nije adekvatno rešenje (zadaci višeg prioriteta moraju se izvršavati pre onih nižeg)
- Zadaci mogu imati i ograničenje u vremenu za koje moraju biti izvršeni (*deadline*)
- Sistem koji radi u realnom vremenu, mora da obezbedi korektno izvršavanje zadataka u definisanim vremenskim rokovima



Nedostaci „super-petlje“

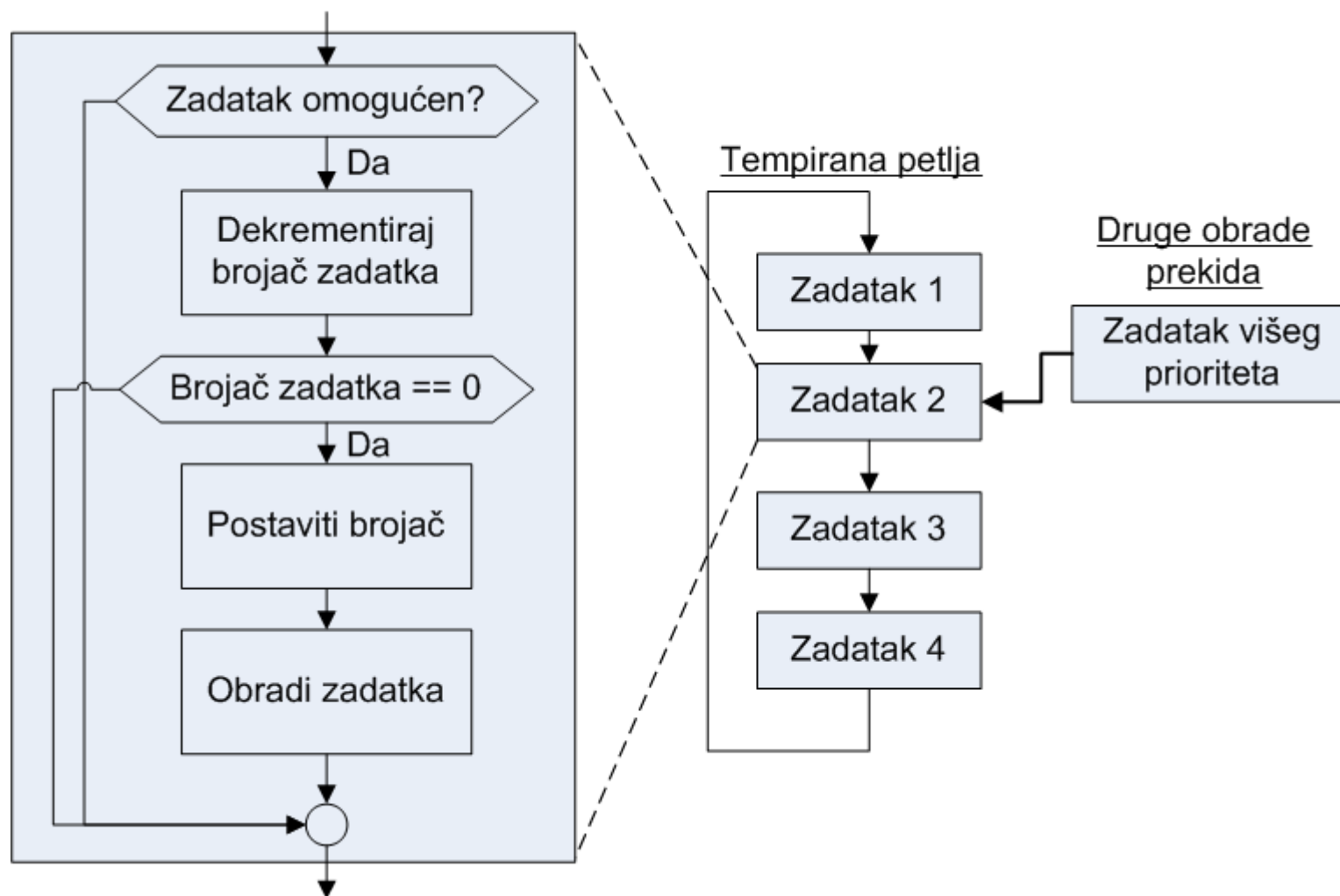
- Vreme izvršenja petlje nije konstantno (zavisno od uslova, neki zadaci se mogu izvršavati duže ili kraće)
- Zadaci ometaju jedni druge (kada jedan zadatak dobije CPU, neće ga pustiti dok se ne završi, iako je možda javio događaj koji zahteva trenutni odgovor)
- Zadaci višeg prioriteta ne dobijaju potrebnu pažnju (svi imaju isti prioritet u petlji)



Unapređenje „super-petlje“

- Zadaci mogu biti aktivirani
 - vremenski (*time-triggered*) ili
 - događajem (*event-triggered*)
- Prioriteti preko prekida (prebacivanje zadataka višeg prioriteta u prekide)
- Okidanje „petlje“ tempiranim (vremenskim) prekidom (preciznije definisanje vremena izvršenja „pozadinskih“ procesa)
- Omogućavanje zadataka i definisanje učestalosti izvršenja (uvođenje *enable* flega i brojača zadatka; što je manji broj upisan u brojač, to je veća učestalost prozivanja).

Upravljanje zadacima pri sekvencijalnom programiranju





Operativni sistem

- Preuzima upravljanje nad zadacima:
 - kada će i koliko dugo da se izvršavaju,
 - omogućuje komunikaciju i sinhronizaciju između njih,
 - upravlja pristupom sistemskim resursima
- RTOS je zapravo program opšte namene
- Može se nezavisno razviti ili kupiti gotovo (komercijalno) rešenje

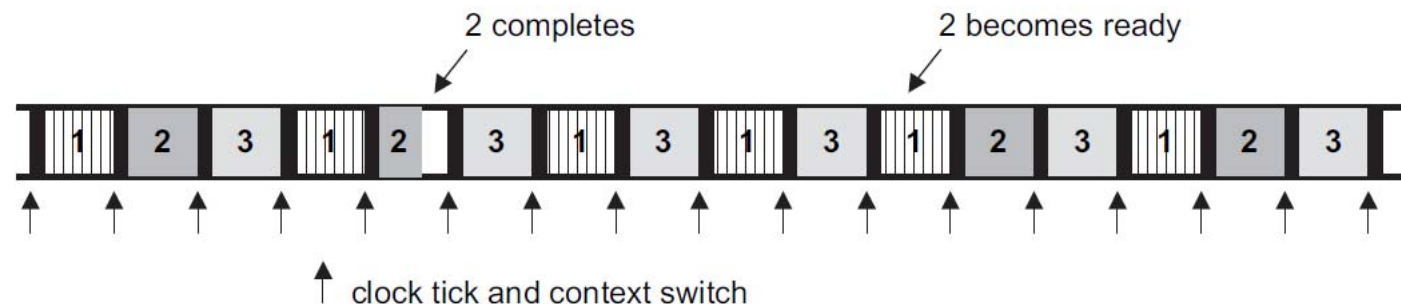
Planiranje

- Centralni deo RTOS-a je **planer** (*scheduler*)
- **Ciklično planiranje**
 - najjednostavnije planiranje
 - svaki zadatak se izvršava dok se ne završi
 - zadatak ne može biti prekinut dok radi
 - ima sve nedostatke „super-petlje“ i sekvencijalnog izvršenja



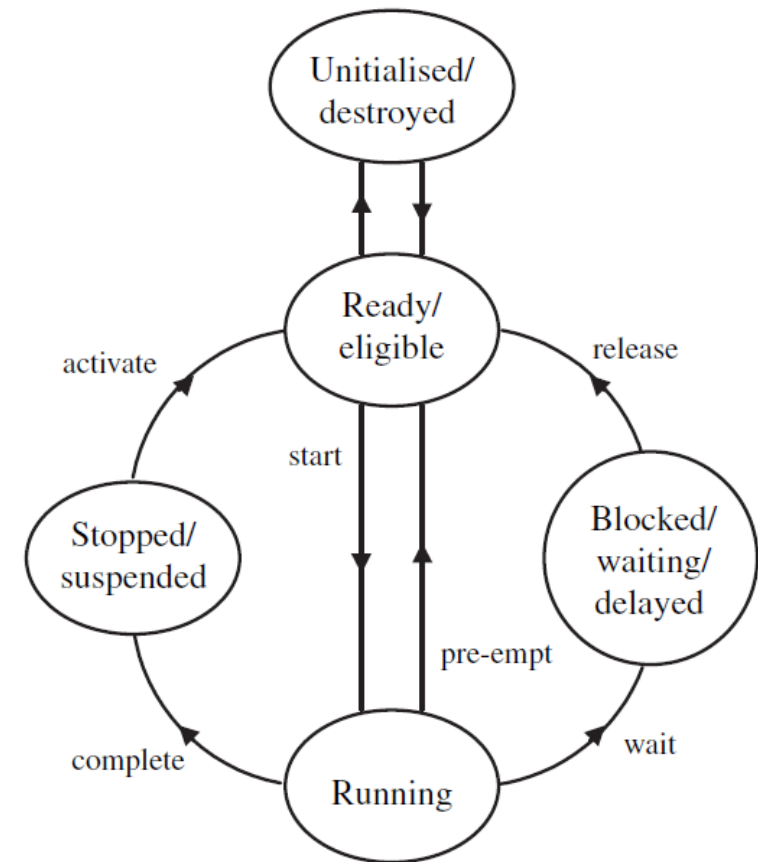
Planiranje

- **Round-robin**
 - Sistemom upravljaju regularni prekidi (otkucaji)
 - Zadaci se selektuju za izvršenje u fiksnoj sekvenci
 - Sa svakim otkucajem, tekući zadatak se prekida i upravljanje prenosi na sledeći
 - Prekidanje zadataka i promena konteksta ima svoju cenu (treba „snimiti“ flegove, registre, memorijske lokacije ...)
 - Smatra se da svi zadaci imaju isti prioritet



Stanja zadatka

- **Uninitialised** – zadatak ne postoji sa stanovišta RTOS-a (uništavanjem nepotrebnih zadataka skraćuje se lista i olakšava rad planera)
- **Ready** – spreman za izvršenje i počinje čim dobije CPU
- **Running** – zadatku je dodeljen CPU i on se izvršava
- **Blocked** – zadatak je spreman za izvršenje, ali zbog nečega mu nije dozvoljeno (čeka podatke ili resurs)
- **Stopped** – zadatak ne zahteva više CPU, završen je (ostaje u tom stanju dok se ponovo ne aktivira)

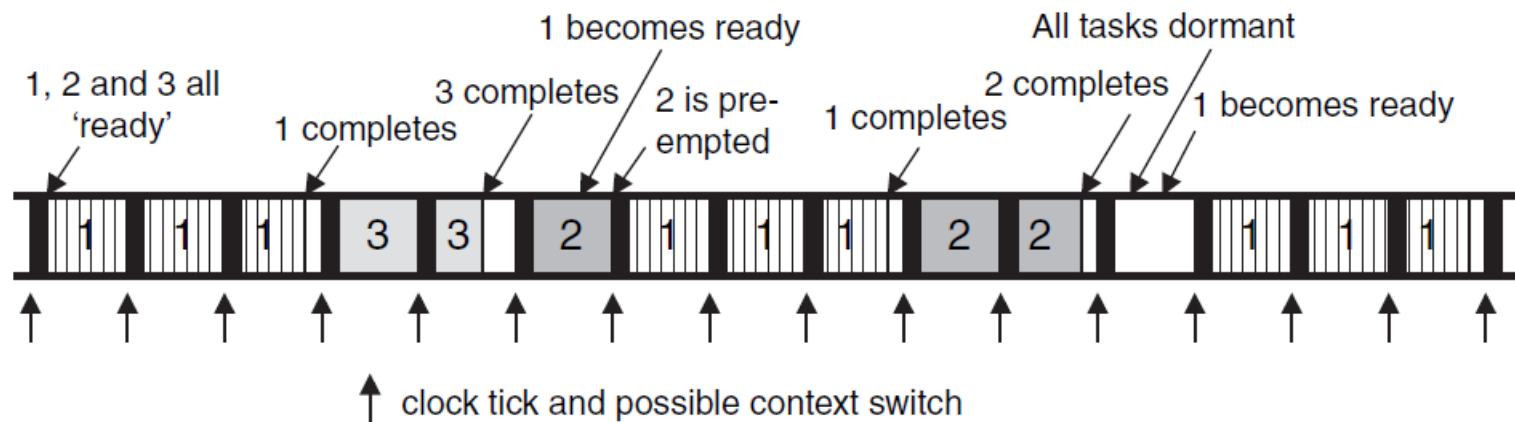




Planiranje - nastavak

- ***Pre-emptive* planiranje sa prioritetom**
 - Zadaci imaju dodeljene prioritete
 - Sistemom i dalje upravljaju regularni prekidi (otkucaji)
 - Na svaki otkucaj, planer proverava koji spreman zadatak ima najviši prioritet i njemu dodeljuje CPU
 - Ako zadatak najvišeg prioriteta i dalje zahteva CPU, on ga zadržava za sebe
 - Zadatak nižeg prioriteta se prekida ako zadatak višeg prioriteta postane spreman (ready)

Primer *pre-emptive* planiranja sa prioriteto



Task	Priority	Duration (in time slices)
1	1 (highest)	2.7
2	3	2.8
3	2	1.5



Planiranje - nastavak

- **Kooperativno planiranje**
 - Svaki zadatak oslobađa CPU u skladu sa poslom koji obavlja i kada sam odabere
 - Time zadatak sam upravlja snimanjem konteksta i smanjuje opterećenje sistema (ne snima se sve, jer bira najpogodniji trenutak)
 - Sistem ima duži odziv nego u prethodnom slučaju, ali zahteva manje memorije i promena konteksta se brže obavlja
 - Koristi se kod malih sistema
- Prekide ne bi trebalo koristiti za realizaciju zadataka, već samo za prosleđivanje urgentnih informacija zadacima ili planeru



Definisanje zadataka

- Broj ne bi trebalo da bude veliki, jer komplikuje planiranje i troši resurse (svaka promena konteksta troši i vreme i memoriju)
- Aktivnosti koje imaju blisko vreme završetka (*deadline*) ili su blisko povezane funkcionalnošću i razmenjuju veliku količinu podataka, treba da budu grupisane u isti zadatak



Pisanje zadatka i postavljanje prioriteta

- Zadatak treba napisati kao da se izvršava kontinualno, kao samostalni i poluautonomni program, bez obzira što može biti prekinut od strane planera
- Ne treba da poziva sekciju koda drugog zadatka, ali može koristiti biblioteke i pozivati zajedničke funkcije
- Zadatak može zavisiti od usluge drugog zadatka i tada treba obezbediti sinhronizaciju
- RTOS dozvoljava postavljanje prioriteta:
 - **statičkih** (fiksni) i
 - **dinamičkih** (mogu se menjati tokom izvršenja)
- Obično se alociraju 3 nivoa prioriteta:
 - **najviši** – zadaci bitni za „opstanak“ sistema,
 - **srednji** – zadaci bitni za korektan rad sistema i
 - **niski** – zadaci potrebni za adekvatan rad sistema, ali povremeno mogu biti „potisnuti“ i prihvatljivo je dodatno kašnjenje za njihovo izvršenje



Zaštita podataka i resursa

- Nekoliko zadataka mogu zahtevati pristup istom hardverskom (memorija, periferija) ili softverskom resursu (zajednički modul)
- Semafor se dodeljuje svakom deljenom resursu i njime se signalizira da li je trenutno u upotrebi
- „Binarni semafor“ omogućuje uzajamno isključivanje zadataka (samo jedan može da pristupi resursu, ostali prelaze u stanje blokiran)
- „Brojački semafor“ se koristi za skup identičnih resursa i inicijalno se postavlja na br. jedinica na raspolaganju; kada zadatak koristi jedan resurs iz skupa, dekrementira semafor, a kada ga oslobodi inkrementira (semafor definiše broj slobodnih jedinica)
- Semafori se mogu koristiti i za sinhronizaciju između zadataka



Poznati RTOS

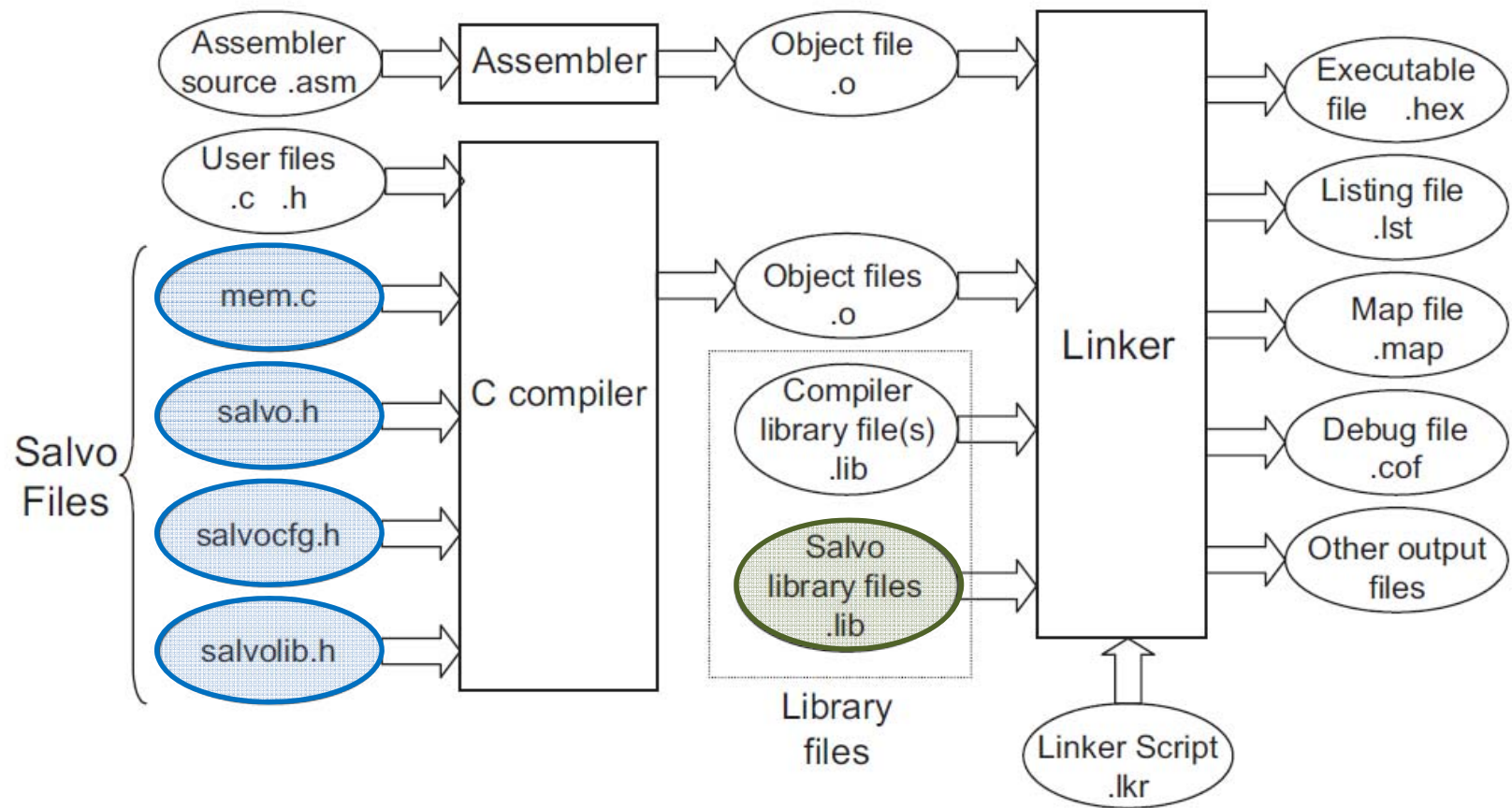
- Salvo
- CCS compiler
- CMX-Tiny+
- PICos18
- MicroC/OS-II
- FreeRTOS
- OSA-RTOS



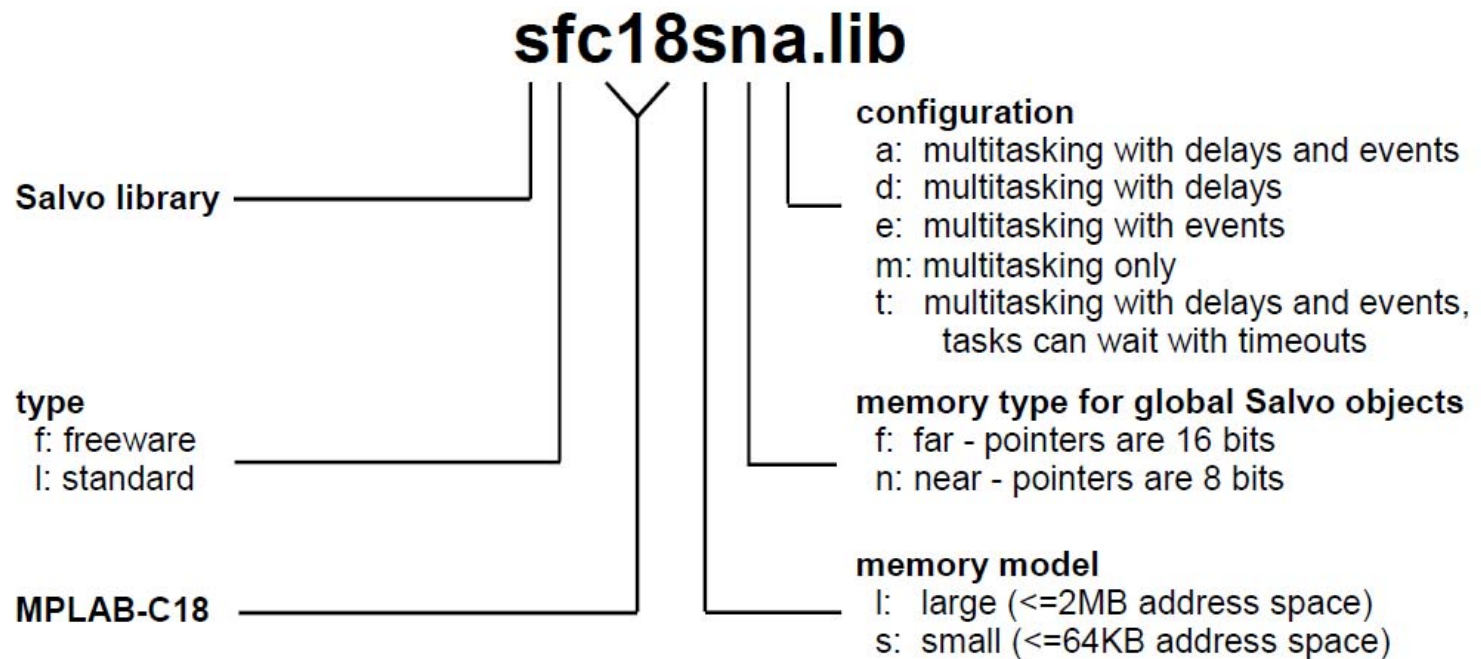
Salvo RTOS

- Komercijalni RTOS firme Pumpkin Inc. (<http://www.pumpkininc.com>) koji može raditi i na slabijim mikrokontrolerima
- Postoji i besplatna verzija (Salvo Lite)
- Inicijalno je razvijen u assembleru, ali je kasnije „prepisan“ u C-u i sada zahteva C kompajler
- Koristi kooperativni planer i podržava više zadataka (u punoj verziji ograničen samo količinom RAM-a) sa 16 nivoa prioriteta, uz malo zauzeće memorije; zadaci istog prioriteta izvršavaju se po *round-robin* algoritmu
- Podržava binarne i brojačke semafore, poruke, redove poruka (*message queues*) i flegove događaja
- Za PIC12, PIC16, PIC17 i PIC18 MCU koristi se MPLAB C18 kompajler (ili neki od sledećih: HI-TECH PICC, HI-TECH PICC-18 ili IAR PIC18 C)

Salvo proces



Nomenklatura Salvo biblioteke



Library config.	a	d	e	m	t
Multitasking	+	+	+	+	+
Delays	+	+			+
Events	+		+		+
Idling	+	+	+		+
Task priorities	+	+	+		+
Time outs					+



Minimalna Salvo aplikacija

```
#include "main.h"  
#include <salvo.h>  
int main( void )  
{  
    Init(); // Inicijalizacija uredjaja  
    OSInit();  
    while (1) {  
        OSSched();  
    }  
}
```

OSInit() inicijalizuje sve Salvo strukture podataka, pokazivače i brojače; mora se pozvati pre bilo koje druge Salvo funkcije.

OSSched() poziva Salvo planer i omogućuje aktiviranje jednog od spremnih zadataka; da bi se omogućio multitasking, OSSched() se mora stalno pozivati.



Kreiranje, startovanje i promena zadatka

```
#include "main.h"
#include <salvo.h>
void TaskA( void )
{
    while (1) {
        OS_Yield();
    }
}
void TaskB( void )
{
    while (1) {
        OS_Yield();
    }
}
```

```
int main( void )
{
    Init();
    OSInit();
    OSCreateTask(TaskA, OSTCBP(1), 10);
    OSCreateTask(TaskB, OSTCBP(2), 10);
    while (1) {
        OSSched();
    }
}
```



Kreiranje i pokretanje zadataka

- Salvo zadaci su funkcije bez parametara
- Sadrže inicijalizaciju i beskonačnu petlju
- Prilikom kreiranja (korišćenjem `OSCreateTask()` funkcije), eksplicitno mu se dodeljuje *task control block* (**tcb**) i prevodi se u stanje **stopped** (ili **ready**)
- Da bi se startovao, zadatak mora da bude u **ready** stanju, što se postiže pozivom `OSStartTask()`, međutim, Salvo direktno prevodi zadatak u stanje **ready** nakon kreiranja, pa je poziv često nepotreban
- Kada je jednom preveden u stanje **ready**, zadatak se aktivira (**running**) čim bude zadatak sa najvišim prioritetom od svih spremnih zadataka



Bitne funkcije

- **OS_Yield()** – безусловna promena konteksta. Svaki Salvo zadatak mora da sadrži najmanje jedan ovakav poziv. Obično je na kraju beskonačne petlje, ali može se javiti bilo gde u kodu. Sve metode koje sadrže uslovnu ili безусловnu promenu konteksta počinju sa OS_.
- **OSCreateTask()** – kreira zadatak sa zadatom početnom adresom, **tcb** pointerom i prioritetom.
 - **Početna adresa** je obično početak zadatka, definisan njegovim imenom.
 - Svaki zadatak zahteva jedinstven **tcb**, koji sadrži sve informacije potrebne Salvu da bi upravljao njime (početna i povratna (*resume*) adresa, stanje, prioritet, itd). Ukupno ima OSTASKS **tcb**-a, označenih brojevima od 1 do OSTASKS. OSTCBP() je makro koji olakšava pribavljanje pointera na odgovarajući **tcb**. Npr. OSTCBP(2) je pokazivač na drugi tcb.
 - **Prioritet** se kreće u granicama od 0 (najviši) do 15 (najniži)

Dodavanje funkcionalnosti

```
#include "main.h"
#include <salvo.h>
unsigned int counter;
void TaskCount( void )
{
    while (1) {
        counter++;
        OS_Yield();
    }
}
void TaskShow( void )
{
    InitPORT();
    while (1) {
        PORT = (PORT & ~0xFE) | ((counter >> 8) & 0xFE);
        OS_Yield();
    }
}

int main( void )
{
    Init();
    OSInit();
    OSCreateTask(TaskCount, OSTCBP(1), 10);
    OSCreateTask(TaskShow, OSTCBP(2), 10);
    counter = 0;
    while (1) {
        OSSched();
    }
}
```

Korišćenje događaja (1)

```
#include "main.h"
#include <salvo.h>
#define TASK_COUNT_P OSTCBP(1) /* task #1 */
#define TASK_SHOW_P OSTCBP(2) /* task #2 */
#define PRIO_COUNT 10 /* task priorities*/
#define PRIO_SHOW 10 /* "" */
#define BINSEM_UPDATE_PORT_P OSECBP(1) /* binsem #1 */
unsigned int counter;
void TaskCount( void )
{
    while (1) {
        counter++;
        if (!(counter & 0x01FF)) {
            OSSignalBinSem(BINSEM_UPDATE_PORT_P);
        }
        OS_Yield();
    }
}
```



Korišćenje događaja (2)

```
void TaskShow( void )
{
    InitPORT();
    while (1) {
        OS_WaitBinSem(BINSEM_UPDATE_PORT_P,
                     OSNO_TIMEOUT);
        PORT = (PORT & ~0xFE) | ((counter >> 8) & 0xFE);
    }
}
```



Korišćenje događaja (3)

```
int main( void )
{
    Init();
    OSInit();
    OSCreateTask(TaskCount, TASK_COUNT_P, PRIO_COUNT);
    OSCreateTask(TaskShow, TASK_SHOW_P, PRIO_SHOW);

    OSCreateBinSem(BINSEM_UPDATE_PORT_P, 0);
    counter = 0;

    while (1) {
        OSSched();
    }
}
```



Bitne funkcije

- **OSCreateBinSem()** – kreira binarni semafor sa definisanim **ecb** (*event control block*) pokazivačem i početnom vrednošću. Semafor se mora kreirati pre nego što neki zadatak pokuša da ga prevede u signalizirano stanje ili čeka na njega.
- **OSSignalBinSem()** – prevodi semafor u signalizirano stanje. Ako nema zadatka koji čeka na njega, on se samo inkrementira. Ako jedan ili više zadataka čeka na njega, tada se zadatak najvišeg prioriteta aktivira.
- **OS_WaitBinSem()** – zadatak čeka na binarni semafor, dok semafor nije u signaliziranom stanju. Ako je vrednost semafora 0, zadatak se prevodi u stanje čekanja (**blocked**). Zadatak se ponovo aktivira (**ready**) kada je semafor u signaliziranom stanju, ali se pokreće (**running**) tek kada on postane najprioritetniji od svih aktivnih. Ako je semafor 1, kada zadatak pozove ovu funkciju, vrednost semafora se menja na 0, a zadatak nastavlja sa izvršenjem.



Dodatne napomene

- Salvo koristi **event control block** (ecb) pokazivače kao hendlere za događaje (*events*)
- Prefiks **OS_** u pozivu OS_WaitBinSem() ukazuje da funkcija dovodi od **promene konteksta**.
- Uvek mora da se definiše *timeout* interval čekanja na binarni semafor. Ako želimo čekati beskonačno dugo, koristi se predefinisana konstanta **OSNO_TIMEOUT**.
- U prethodnom primeru, **OS_WaitBinSem()** se korisiti umesto poziva **OS_Yield()**, jer on u sebi implicitno sadrži promenu konteksta, tj. OS_Yield() poziv.



Definisanje vremena na koje se izvršava zadatak

- U prethodnom primeru, zadaci se izvršavaju u „otvorenoj petlji“. Dodavanjem novog zadatka istog ili višeg prioriteta smanjiće se frekvencija inkrementiranja brojača (tj. izvršavanja zadatka).
- Odlaganjem izvršenja zadatka (*delay*), zadatak prestaje da se izvršava dok dato vreme ne istekne, a planer je dužan da po isteku vremena vrati zadatak u stanje **ready**.



Primer za odlaganje izvršenja (1)

```
#include "main.h"
#include <salvo.h>
#define TASK_COUNT_P OSTCBP(1) /* task #1 */
#define TASK_SHOW_P OSTCBP(2) /* "" #2 */
#define TASK_BLINK_P OSTCBP(3) /* "" #3 */
#define PRIO_COUNT 10 /* task priorities*/
#define PRIO_SHOW 10 /* "" */
#define PRIO_BLINK 2 /* "" */
#define BINSEM_UPDATE_PORT_P OSECBP(1) /* binSem #1 */
unsigned int counter;

void TaskCount( void )
{
    while (1) {
        counter++;
        if (!(counter & 0x01FF)) {
            OSSignalBinSem(BINSEM_UPDATE_PORT_P);
        }
        OS_Yield();
    }
}
```



Primer za odlaganje izvršenja (2)

```
void TaskShow( void )
{
    while (1) {
        OS_WaitBinSem(BINSEM_UPDATE_PORT_P,
                     OSNO_TIMEOUT);
        PORT = (PORT & ~0xFE) | ((counter >> 8) & 0xFE);
    }
}
```

```
void TaskBlink( void )
{
    InitPORT();
    while (1) {
        PORT ^= 0x01;
        OS_Delay(50);
    }
}
```



Primer za odlaganje izvršenja (3)

```
void main( void )
{
    Init();
    OSInit();

    OSCreateTask(TaskCount, TASK_COUNT_P, PRIO_COUNT);
    OSCreateTask(TaskShow, TASK_SHOW_P, PRIO_SHOW);
    OSCreateTask(TaskBlink, TASK_BLINK_P, PRIO_BLINK);

    OSCreateBinSem(BINSEM_UPDATE_PORT_P, 0);

    counter = 0;

    enable_interrupts();

    while (1) {
        OSSched();
    }
}
```



Bitne funkcije

- **OSTimer()** – definiše otkucavanje sistemskog časovnika i potrebno je zvati sa stabilnom učestalošću (vidi naredni kod). Prekidi moraju biti omogućeni da bi OSTimer mogao biti pozvan (otuda i `enable_interrupts()` poziv pre startovanja planera)
- **OS_Delay()** – zamenjuje `OS_Yield()` i odlaže izvršenje zadatka za onoliko otkucaja sistemskog časovnika koliko je navedeno kao parametar ove funkcije (za prethodni slučaj, delay je $50 * 10\text{ms} = 500\text{ms}$)
- Salvo nadgleda odložene zadatke na svaki poziv OSTimer-a i overhead ne zavisi od broja odloženih zadataka, osim ako se ne dogodi da period odlaganja za dva ili više zadataka ističe istovremeno.



Definisanje sistemskog časovnika

```
#include <salvo.h>
#define TMRO_RELOAD 156 /* for 100Hz ints @ 4MHz*/

void interrupt IntVector( void )
{
    if (TOIE && TOIF) {
        TOIF = 0;
        TMRO -= TMRO_RELOAD;
        OSTimer();
    }
}
```




Signalizacija preko poruka (1)

```
#include "main.h"  
#include <salvo.h>  
#define TASK_COUNT_P OSTCBP(1) /* task #1 */  
#define TASK_SHOW_P OSTCBP(2) /* "" #2 */  
#define TASK_BLINK_P OSTCBP(3) /* "" #3 */  
#define PRIO_COUNT 12 /* task priorities*/  
#define PRIO_SHOW 10 /* "" */  
#define PRIO_BLINK 2 /* "" */  
#define MSG_UPDATE_PORT_P OSECBP(1) /* sem #1 */  
  
unsigned int counter;  
  
char CODE_B = 'B';  
char CODE_C = 'C';
```



Signalizacija preko poruka (2)

```
void TaskCount( void )
{
    counter = 0;
    while (1) {
        counter++;
        if (!(counter & 0x01FF)) {
            OSSignalMsg(MSG_UPDATE_PORT_P,
                        (OStypeMsgP) &CODE_C);
        }
        OS_Yield();
    }
}
```



Signalizacija preko poruka (3)

```
void TaskShow( void )
{
    OStypeMsgP msgP;
    InitPORT();
    while (1) {
        OS_WaitMsg(MSG_UPDATE_PORT_P, &msgP,
                  OSNO_TIMEOUT);
        if (*(char *)msgP == CODE_C) {
            PORT = (PORT & ~0xFE) | ((counter >> 8) & 0xFE);
        }
        else {
            PORT ^= 0x01;
        }
    }
}
```



Signalizacija preko poruka (4)

```
void TaskBlink( void )
{
    OStypeErr err;
    while (1) {
        OS_Delay(50);
        err = OSSignalMsg(MSG_UPDATE_PORT_P,
                          (OStypeMsgP) &CODE_B);
        if (err == OSERR_EVENT_FULL) {
            OS_SetPrio(PRIO_SHOW+1);
            OSSignalMsg(MSG_UPDATE_PORT_P,
                          (OStypeMsgP) &CODE_B);
            OS_SetPrio(PRIO_BLINK);
        }
    }
}
```



Signalizacija preko poruka (5)

```
void main( void )
{
    Init();
    OSInit();
    OSCreateTask(TaskCount, TASK_COUNT_P, PRIO_COUNT);
    OSCreateTask(TaskShow, TASK_SHOW_P, PRIO_SHOW);
    OSCreateTask(TaskBlink, TASK_BLINK, PRIO_BLINK);

    OSCreateMsg(MSG_UPDATE_PORT_P, (OStypeMsgP) 0);

    enable_interrupts();

    while (1) {
        OSSched();
    }
}
```



Bitne funkcije

- **OSCreateMsg()** – kreira poruku. Poruka mora biti kreirana pre nego što neki zadatak čeka na nju ili proverava stanje. Pokazivač na poruku mora biti tipa `OStypeMsgP`.
- **OSSignalMsg()** – signalizira poruku. Navode se *ecb* pokazivač i pokazivač na sadržaj poruke. Zadatak najvišeg prioriteta koji čeka na poruku postaje **ready**. Pokazivač na poruku se mora korektno „kastovati“ da bi se pravilno izvršilo dereferenciranje.



Bitne funkcije

- **OS_WaitMsg()** – aktivira čekanje na zadatu poruku. Poruka se prosleđuje zadatku kroz pokazivač na poruku. Da bi se izvršila ekstrakcija poruke, pokazivač mora da se „kastuje“ u odgovarajući tip.
- **OS_SetPrio()** – menja prioritet tekućeg zadatka i trenutno vrši promenu konteksta.
- **OSSetPrio()** – zadatak menja prioritet, ali novi prioritet stupa na scenu tek nakon što zadatak preda upravljanje planeru.

Literatura

- **Designing Embedded Systems with PIC Microcontrollers, Principles and Applications, Book • 2nd Edition • 2010**



- Opciono:
 - Ch.14 Introducing C, str.443-465.
 - Ch.15 C and the embedded environment, str.467-480.
 - Ch.16 Acquiring and using data with C, str.481-500.
 - Ch.17 More C and the wider C environment, str.501-524.
- Ch.18 Multi-tasking and the real-time operating system, str. 525-539.
- Ch.19 The Salvo real-time operating system, str.541-571.

- [illegible]

