Vaja 6

Preprost operacijski sistem v realnem času (RTOS, real time operating system)

Naloga

Napišite program za preprosto digitalno uro, ki šteje sekunde.

Na prikazovalniku LCD naj bo ves čas prikazano trenutno stanje ure: hh:mm:ss (ure, minute, sekunde)

Tipke:

- T1 → poveča ure
- T2 → poveča minute
- T3 → ustavi uro (dokler je pritisnjena)
- T4 → postavi čas na 00:00:00

Uporabite preprost operacijski sistem v realnem času

RTOS (real time operating system)

Osebni računalnik

- Časovni potek programov običajno ni pomemben oz. določen
- Želimo čim hitrejše izvajanje

Krmilni sistemi

- Časovna programska sled je natančno določena
- Časovna opredeljenost in predvidljivost je bolj pomembna kot hitrost → programi tečejo v realnem času
- Pri več opravilih je potreben nadzorni mehanizem → RTOS
  - Skrbi, da vsakemu opravilu pripada določen delež procesorskega časa
  - Skrbi, da se opravila izvedejo v okviru predpisanih časovnih zahtev

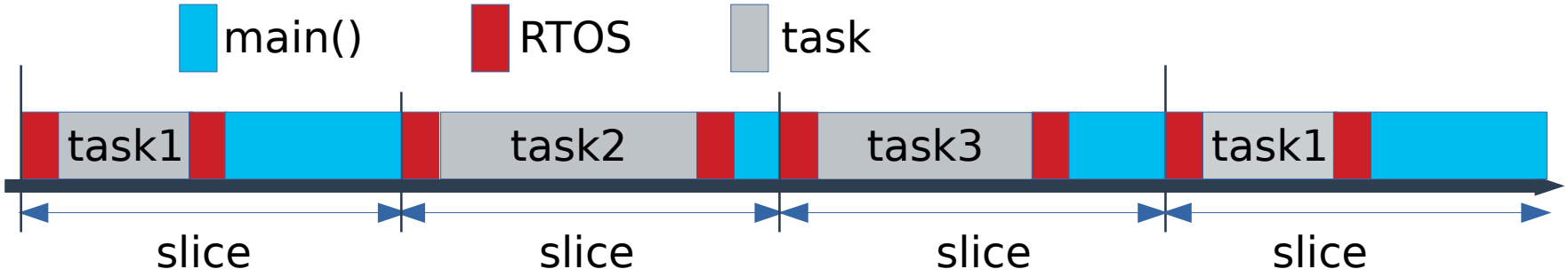
Preprost RTOS – lastnosti in omejitve

Nespremenljive časovne rezine

Vsa opravila so krajša od časovne rezine

Vnaprej določeno konstantno število opravil

Krožno razvrščanje opravil (round robin)



Opravila se izvajajo ciklično v enakem zaporedju, kot so podana v tabeli opravil

Vsa opravila imajo enako periodo izvajanja: dt = Ntasks * time_slice

Preprost RTOS – lastnosti in omejitve

Kooperativni (non-preemptive)

- Opravila sama prepustijo procesor drugim
- Opravila se končajo pred zaključkom časovne rezine
- Opravila ne morejo biti prekinjena s strani drugega opravila
- → manj težav s hkratnim dostopom do skupnih enot (ne potrebujemo mutex-ov, semaforjev)

Brez dinamičnega nadzora nad pomnilnikom

- Opravila, medpomnilniki, itd. so vnaprej definirani

Brez MPU (memory protection unit)

Omejena fleksibilnost, učinkovit, zelo preprost razvrščevalnik (scheduler)

Preprost RTOS – SysTick (system tick)

Del procesorskega jedra

glej "thirdparty/CMSIS/include/core_cm3.h"

24 bitni števec

šteje navzdol, ko doseže vrednost 0 → lahko sproži izjemo (exception) in reset na določeno vrednost (egister SysTick→LOAD)

Funkcija SysTick_Config(...) je že del ASF

Nastavi števec in prekinitev ter takoj zažene sistem → **ne bomo je uporabili** (lahko je za vzorec)

Sami želimo nadzor nad zagonom in ustavitvijo sistema

→ Napišimo svojo rtos_init(...) funkcijo

Preprost RTOS

uint32_t rtos_init(uint32_t slice_us);

Inicializacija sistema (konfiguracija SysTick za pravilno dolžino časovne rezine)

uint32_t slice_us: dolžina časovne rezine v mikrosekundah

void rtos_disable(void);

Izklop RTOS (dve možnosti):

1. Ustavitev SysTick števca→ zakasnitev vseh opravil
2. Maskiranje SysTick prekinitve → števec teče dalje, prekinitve ne bo
   → ob ponovnem vklopu prekinitve je časovna rezina lahko prekratka za opravilo

void rtos_enable(void);

Vklop RTOS

RTOS inicializacija

void rtos_init(uint32_t slice_us){

...

}

Konfiguracija SysTick

- Nastavi uro SysTick števca (MCK ali MCK/8)
- Nastavi SysTick "reload" register
  - Število ciklov števca, ki ustreza dolžini časovne rezine
- Omogoči SysTick prekinitev
- Nastavi prioriteto SysTick prekinitve
- Ne zažene števca → rtos_enable() / rtos_disable()

Razvrščevalnik opravil

void SysTick_Handler(void){

...

}

Beleži naj število časovnih rezin (tick)

Ob vsakem klicu naj izvede eno od opravil iz tabele in krožno pomakne indeks opravil na naslednje

Za opravilo lahko tudi zabeleži trenutek zadnjega zagona

Preveri naj, če je slučajno opravilo predolgo (pred koncem trenutne SysTick prekinitve je že sprožena nova)

Postavljen bit 26 (SCB_ICSR_PENDSTSET_Msk) v registru SCB→ICSR (system control block, interrupt control and state reg.) naj ujame program v neskončni zanki

Opravila

typedef void (* ptr_function)(void);

Nov podatkovni tip z imenom **ptr_function**: kazalec na funkcijo

```
struct _task{
  ptr_function function;  /* kazalec na funkcijo */
  uint32_t last_tick;     /* trenutek zadnjega zagona opravila */
};
typedef struct  _task   rtos_task_t;
```

Nov podatkovni tip z imenom **rtos_task_t**

Strukturo lahko enostavno razširimo z dodatnimi polji

Npr. klic opravila z argumenti, prioritete, interval, sklad, ...

Primer opravila – LCD gonilnik

```
rtos_task_t  lcd_task = {
   .function = lcd_driver,
   .last_tick = 0
};
```

rtos_task_t *rtos_task_list[] = {&lcd_task};

Globalna zbirka **kazalcev** na strukture opravil

Ker so kazalci → RTOS lahko spreminja vsebino struktur (npr. last_tick)

Ker je globalna → preprostejša uporaba, občutljivost na napaka

Statična zbirka → se ne bo spreminjala → preprostejši razvrščevalnik, ne potrebujemo dinamečnega nadzora nad pomnilnikom (memory management)

Primer klica funkcije opravila (opravi razvrščevalnik):

```
rtos_task_list[current_task_idx]→function ();
```

Naloga

Opravila:

1. Gonilnik ure (clock_task)
2. Priprava teksta za izpis (lahko tudi v main())
3. Gonilnik za LCD
4. Gonilnik za tipke

Zaenkrat zelo preprost in neposredno nadzira drugo opravilo (gonilnik ure)

Kasneje bomo uvedli FIFO za tipke

→ gonilnik ne bo imel direktnega vpliva na druga opravila

→ to bo delo glavnega programa