VEŽBA – FreeRTOS – 2

U ovoj vežbi radimo sa binarnim semaforima i softverskim tajmerima.

Nove sistemske funkcije FreeRTOS koje ćemo koristiti su navedene:

`SemaphoreHandle_t xSemaphoreCreateBinary( void );`	Kreira binarni semafor i vraća referencu (*handle*) na njega. Može imati samo dva stanja – dat (*given*) i neraspoloživ. Ako vrati nulti pokazivač (`NULL`), kreiranje nije uspelo. https://www.freertos.org/xSemaphoreCreateBinary.html
`BaseType_t xSemaphoreGive( SemaphoreHandle_t xSemaphore );`	Daje semafor čija je referenca (*handle*) data kao parametar. Vraća `pdPASS` ako je semafor uspešno dat ili `pdFAIL` ako je već bio dat pre poziva. https://www.freertos.org/a00123.html
`BaseType_t xSemaphoreTake( SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait );`	Pokušava da uzme (*take*) semafor. Vraća `pdPASS` ako je semafor uspešno uzet ili `pdFAIL` ako nije, a navedeni broj tick-ova je prošao. Pre isteka vremena, ako semafor nije raspoloživ, task koji je funkciju pozvao se uvodi u blokirano stanje. https://www.freertos.org/a00122.html
`TimerHandle_t xTimerCreate( const char *pcTimerName, const TickType_t xTimerPeriod, const UBaseType_t uxAutoReload, void * const pvTimerID, TimerCallbackFunction_t pxCallbackFunction );`	Kreira softverski tajmer čija je perioda određena brojem sistemskih tick-ova. Vraća referencu (*handle*) na tajmer objekat, nulti pokazivač (`NULL`) ukazuje na grešku. https://www.freertos.org/FreeRTOS-timers-xTimerCreate.html
`BaseType_t xTimerStart( TimerHandle_t xTimer, TickType_t xTicksToWait );`	Tek kreirani tajmer nalazi se u zaustavljenom stanju. Pozivom ove funkcije potrebno je pokrenuti ga. Pokretanje može biti neuspešno ako se zahtev za to ne prosledi u roku od zadatog broja tick-ova što rezultuje povratnom vrednošću `pdFAIL`. Inače se vraća vrednost `pdPASS`. https://www.freertos.org/FreeRTOS-timers-xTimerStart.html

Zadatak

Proširiti zadatak 1 iz prethodne vežbe kojeg sada ponavljamo:

Napisati program pod FreeRTOS-om koji:

a) kreira jedan task,

b) task menja stanje na displeju svakih 500 ms, tako da se naizmenični pojavljuje broj 1 i 0 u prvom segmentu displeja. Blokiranje taska na 500 ms izvesti pozivom funkcije `vTaskDelay`.

Izmeniti ovaj program tako da:

1.

a) kreira se tajmer sa periodom od 500 ms i odgovarajućom *callback* funkcijom,

b) kreira se binarni semafor preko kojeg će callback funkcija da obaveštava task o isteklom vremenskom intervalu od 500 ms,

c) task koji upravlja displejom se izmeni tako da čeka u blokiranom stanju sve dok ne dobije informaciju o isteklom intervalu preko semafora iz prethodne tačke.

2.

a) kreira se tajmer sa periodom od 10000 ms i istom *callback* funkcijom kao i tajmer iz 1.a)

b) C*allback* funkciju izmenuti tako da promena brojeva na displeju prestane kada prvi put istekne perioda novog tajemra, tj posle 10 s

Rešenje zadatka 1

a) kreiramo softverski tajmer, koji ima sličnu funkcionalnost kao hardverski tajmeri na koje ste navikli u mikrokontrolerima (https://www.freertos.org/RTOS-software-timer.html).

Softverski tajmer (ili samo „tajmer“) omogućava izvršavanje funkcije u određeno vremenskom periodu. Funkcija koju izvršava tajmer naziva se funkcijom povratnog poziva (callback). Vreme između pokretanja tajmera i njegove funkcije povratnog poziva, naziva se vremenskim periodom. Funkcija povratnog poziva tajmera se izvršava kada istekne period tajmera.

Funkcije povratnog poziva tajmera **nikada** ne smeju imati blokirajuće elemente. Na primjer, funkcija povratnog poziva tajmera ne smije pozivati vTaskDelay (), vTaskDelayUntil (), niti odrediti ne-nulto vrijeme blokade prilikom pristupa redu queue ili semaforu (što ćemo da nakandno objasnimo).

U FreeRTOSConfig.h header fajlu (gde se inače nalaze sva konfiguraciona podešavanja FreeRTOS-a) se nalaze 4 konfiguracione opcije

```
/* Software timer related configuration options. */
#define configUSE_TIMERS                                1
#define configTIMER_TASK_PRIORITY                       ( configMAX_PRIORITIES - 1 )
#define configTIMER_QUEUE_LENGTH                        20
#define configTIMER_TASK_STACK_DEPTH                    ( configMINIMAL_STACK_SIZE * 2 )
```

- `configUSE_TIMERS` podešeno na 0 ako ne želimo da koristimo tajmere (da uštedimo RAM Sistema), a na 1 kada želimo koristiti tajmere
- `configTIMER_TASK_PRIORITY` prioritet tajmerskog taska, po defaultu maksimalan

- `configTIMER_QUEUE_LENGTH` maksimalan broj tajmerskih komandi
- `configTIMER_TASK_STACK_DEPTH` veličina steka, tj RAM memorijskog prostora, koji tajmerski task može da koristi

Detaljniji opis konfiguracije tajmera možete naći na https://www.freertos.org/Configuring-a-real-time-RTOS-application-to-use-software-timers.html

```
/* Kreiramo softverski tajmer,ali ga jos ne pokrecemo */
	myTimer1= xTimerCreate(
	NULL,/* tekstualni naziv tajmera, nije nepohodan, koristi se samo za debug  */
	xTimerPeriod,		/* perioda softverskog tajmera u "ticks". */
	pdTRUE,  /* xAutoReload je podesen na pdTRUE, tako da je ovo ciklicni tajmer */
	NULL,				/* indetifikacija ovog tajmera, ne koristimo ga u ovom primeru */
	TimerCallback);/* ova funkcija se izvrsava kada se zavrsi jedna perioda tajmera */

if (myTimer1 == NULL) { while (1); } // proveravamo da li se tajmer pravilno kreirao, ako nije zakucaj program
```

Kreiranje softverrskog tajmera se vrši pomoću sistemske funkcije FreeRTOS-a `xTimerCreate`, koja sadrži više parametara, od kojih je za ovaj zadatak značajni drugi,koji određuje periodu tajmera, kao i 5 tj zadnji,koji određuje funkciju (`TimerCallback`) u kojoj se implemetira, tj izvršava ovaj task. Ako je tajmer uspešno kreiran, funkcija `xTimerCreate` vraća referencu (*handle*) na tajmer objekat, inače nulti pokazivač (`NULL`) ukazuje na grešku, što proveravamo preko `if (myTimer1 == NULL)`. Ako task nije uspešno kreiran blokiramo program preko while(1) .

```
xTimerStart(myTimer1, portMAX_DELAY); //start tajmera
```

Pomoću funkcije `xTimerCreate` kreira se jedan softverski tajmer, ali se taj tajmer startuje tek kad se pomoću funkcije `xTimerStart` (i kada se pokrene `vTaskStartScheduler`) posalje komanda za startovanje tajmera. Ova funkcija ima dva parametra , prva je referenca na tajmer objekat koji želimo pokrenuti, a drugi vreme koje čekamo da se ova komanda izvrši (ako ima puno tajmerskih komandi na čekanju). `portMAX_DELAY` predstavlja maksimalno vreme u sistemu (može se gledati kao beskonačno dugo vreme). Ovu funkciju pozivamo iz funkcije `first_task`, tj tajmer se krene izvršavati prvi put kada se zvrši prvi task.

```
binSem1 = xSemaphoreCreateBinary();
	if (binSem1 == NULL) while (1); // proveravamo da li se semafor pravilno kreirao
```

sistemska funkcija *xSemaphoreCreateBinary* kreira binarni semafor i vraća referencu (*handle*) na njega. Može imati samo dva stanja – dat (*given*) i neraspoloživ. Ako vrati nulti pokazivač (`NULL`), kreiranje nije uspelo.
Binarni semafori služe za sihronizaciju, između taskova i između taskova i interapta. Često se koriste da bi izvršila sihronizacija između interapta i taksa, tj da se task odblokira kada se desi interapt. Za detalje oko binarnih semafora pogledajte ovde https://www.freertos.org/Embedded-RTOS-Binary-Semaphores.html

```
static void first_task(void* pvParams)
{
	xTimerStart(myTimer1, portMAX_DELAY); //start tajmera
	while (1) {
			//first_taskvTaskDelay(pdMS_TO_TICKS(500)); iz prethodne vezbe
```

```
                    xSemaphoreTake(binSem1, portMAX_DELAY);
                    Display_Toggle_0();
                }
}
```

Može se videti da je razlika u realizaciji funkcije *first_task* iz prethodne vežbe i ovog zadatka samo u primenu funkcije xSemaphoreTake . Ovo je blokirajuća funkcija koja pokušava da uzme (*take*) semafor binSem koji je naveden kao prvi parametar u funkciji, u periodu određen drugim parametrom (ovde portMAX_DELAY) tj dok ne istekne *timeout* . Vraća pdPASS ako je semafor uspešno uzet ili pdFAIL ako nije, a navedeni *timeout* je prošao. Sve dok semafor nije raspoloživ, task koji je funkciju pozvao se uvodi u blokirano stanje, tj. izvršenje taska čeka da uzme (*take*) semafor. U slučaju da je za *timeout* podešen portMAX_DELAY, *timeout* nikad neće proći, tj task će biti blokiran beskonačno dugo, ako ne uzme semafor.

```
static void TimerCallback(TimerHandle_t tmH)
{
	xSemaphoreGive(binSem);
}
```

Tajmerska povratna funkcija *TimerCallback* , koja se poziva kada *myTimer1* perioda prođe, poziva samo jednu funkciju xSemaphoreGive, koja daje (*give*) semafor čija je referenca (*handle*) data kao parametar. Vraća pdPASS ako je semafor uspešno dat ili pdFAIL ako je već bio dat pre poziva.

Znači algoritam ovog programa je sledeći, svaki put kada prođe jedna perioda tajmera myTimer1 (ovde 500ms) poziva se njegova callback funkcija TimerCallback, koja daje ili "puni" semafor binSem1. Pre toga task koji se izvršava u funkciji *first_task* blokira se kod funkcije xSemaphoreTake i čeka da dobije taj semafor, i kada se to desi task se odblokira, izvrši funkciju *Display_Toggle_0* , a onda opet čeka da dobije semafor, što se dešava posle 500 ms, kada se ponovo pozove xSemaphoreGive iz TimerCallback.

Rešenje zadatka 2

Kreiramo novi softverski tajmer (myTimer2) koji poziva istu funkciju (TimerCallback) kao i myTimer1.

```
myTimer2 = xTimerCreate(
                NULL,/* tekstualni naziv tajmera, nije nepohodan, koristi se samo za debug  */
                pdMS_TO_TICKS(5000UL),          /* perioda softverskog tajmera u "ticks". */
                pdTRUE,         /* xAutoReload je podesen na pdTRUE, tako da je ovo ciklicni tajmer */
                (void*)1,       /* indetifikacija ovog tajmera, ne koristimo ga u ovom primeru */
                TimerCallback); /* poziva istu funkciju kao i myTimer1*/
        if (myTimer2 == NULL) { while (1); } // proveravamo da li se tajmer pravilno kreirao, ako nije
zakucaj program
```

Može se videti da ovde koristimo indetifikaciju tajmera (tj 4 parametar funkcije xTimerCreate) i dodeljujemo mu kao vrednost argumenta vrednost 1. U kodu možete videti da smo modifikovali i myTimer1 i dodelili mu indetifikator 0.
Pošto oba tajmera pozivaju istu funkciju kada im završi perioda, preko indetifikatora možemo da proverimo koji tajmer je pozvao funkciju.

TimerCallback funkcija je modifikovana:

```
static void TimerCallback(TimerHandle_t tmH)
{
	uint32_t timer_number;
	timer_number = (uint32_t)pvTimerGetTimerID(tmH);
	if (timer_number==1)  xTimerStop(myTimer1, 0);

	if (timer_number == 0) xSemaphoreGive(binSem1);
}
```

Preko funkcije `pvTimerGetTimerID` možemo proveriti koja je indetifikacija tajmera, ovoj funkciji prosleđuje se kao argument referenca tajmera koji je pozvao callback funkciju, što se očitava preko reference `tmH`. Znači svaka tajmerska callback funkcija mora imati kao parameter referencu tipa `TimerHandle_t`, u koju se kao argument referenca tajmera koju ju je pozvao.
Ako je indetifikacija tajmera jednak1 0 (što upisujemo u promenjivu `timer_number`), `callback` funkcija je pozvana od `myTimer1` i radi isto što i u zadatku 1, tj. daje semafor svakih 500 ms.
Ako je indetifikacija tajmera jednaka 1, `callback` funkcija je pozvana od `myTimer2` i preko funkcije `xTimerStop` isključuje `myTimer1`, što se desava posle 10 sekundi, kada `myTimer2` prvi put završi periodu i pozove callback funkciju.

DOPUNA:
Pored navedenih funkcija vezanih za softverski tajmer, u FreeRTOS-u se koristi veći broj tajmerskih funkcija, navedenih ovde https://www.freertos.org/FreeRTOS-Software-Timer-API-Functions.html .