# VEŽBA – RTOS – 1

## Pregled

Operativni sistem omogućava apstrakciju sistema na kom se izvršava – drugim rečima, omogućuje aplikaciji koja se na sistemu izvršava da komunicira sa uprošćenim i dobro definisanim softverskim okruženjem nezavisnim od konkertnog hardvera, tzv. *virtuelnom mašinom*.

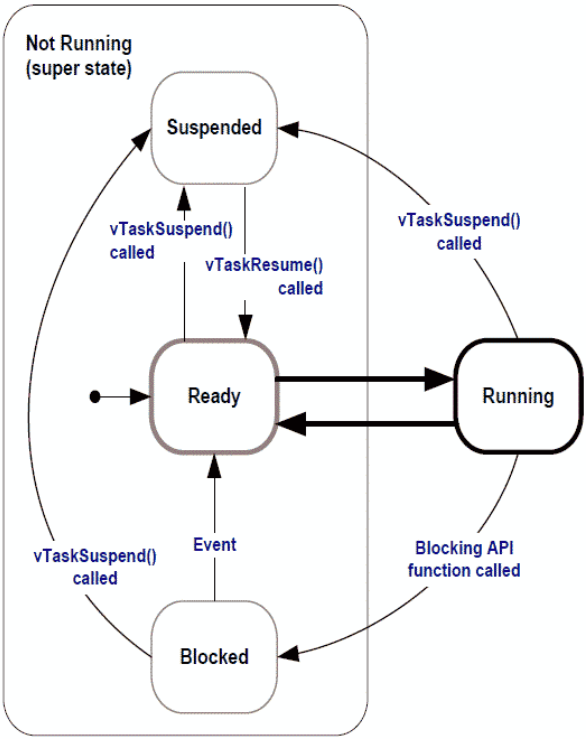
FreeRTOS (Free Real Time Operating System) je operativni sistem namenjen savremenim embeded sistemima skromnijih kapaciteta koji omogućuje ustanovljavanje više paralelnih niti ili zadataka (engl. thread, task) programa, kao i osnovne elemente bezbedne komunikacije među njima – semafori i redovi (engl. semaphore, queue).

Kompletna komunikacija sa operativnim sistemom se obavlja preko skupa dokumentovanih funkcija, tzv. API-ja (Application Programming Interface).

## FreeRTOS

U nastavku je dato nekoliko osnovnih API funkcija neophodnih za pokretanje sistema, kreiranje tajmera i uspostavljanje osnovne komunikacije između paralelnih zadataka (taskova).

Operativni sistem FreeRTOS preuzima upravljanje sistemom nakon poziva funkcije `vTaskStartScheduler`. Od tog trenutka nadalje svi taskovi se izvršavaju u skladu sa svojim stanjem, a promene stanja prikazane su na šemi na slici 1.




*Slika 1 - Tranzicija stanja taska*

U narednoj tabeli dat je kratak pregled API funkcija koje su potrebne za rešavanje zadataka ove laboratorijske vežbe:

| | |
|---|---|
| `BaseType_t xTaskCreate( TaskFunction_t pvTaskCode, const char * const pcName, unsigned short usStackDepth, void *pvParameters, UBaseType_t uxPriority, TaskHandle_t *pxCreatedTask );` | Funkcija koja kreira `Task` objekat. Vraća vrednost `pdPASS` ako je objekat uspešno kreiran. Svaki kreirani task počinje sa radom nakon pokretanja *raspoređivača*. Ako je raspoređivač već pokrenut, task kreće u rad odmah nakon kreiranja. https://www.freertos.org/a00125.html |
| `void vTaskStartScheduler( void );` | Pokreće *raspoređivač* – nakon ovog poziva operativni sistem preuzima upravljanje sistemom. https://www.freertos.org/a00132.html |

| | |
|---|---|
| `void vTaskDelay( TickType_t xTicksToDelay );` | Prebacuje task u blokirano stanje na vreme koje odgovara zadatom broju tick-ova. https://www.freertos.org/a00127.html |

# Zadaci

1. 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`.
2. Proširiti prethodni zadatak tako da:
   a) kreira drugi task,
   b) task menja stanje na displeju svakih 1000 ms, tako da se naizmenični pojavljuje broj 1 i 0 u DRUGOM segmentu displeja. Blokiranje taska na 1000 ms izvesti pozivom funkcije `vTaskDelay`.

# Dodatne informacije

Kao što smo već napomeni, koristićemo FreeRTOS verziju za windows, u projektu koji vam šaljem postoji veliki broj fajlova, ali skoro svi su sistemski fajlvoi FreeRTOS-a, i ne postoji potreba da ih modifikujemo ili proučavamo sta tačno rade (osim ako vas ne zanima kako je tačno realizovan sam FreeRTOS, što nam nije cilj na ovim vežbama, već kako ga možemo koristiti). Urađen je displej na kom se mogu prikazati vrednosti promenjivih, njegovo pozivanje je realizovano preko drajveru u folderu HWS_driver.

Zasad sve što treba da proučimo su fajlovi main.c i i_vezba.c, gde su realizovani traženi zadaci. Koristan je i FreeRTOSConfig.h, gde se mogu podesiti parametri vezani za funkcionisanje FreeRTOS-a (kao neki *settings* operativnog sistema) ali o tome ćemo malo više u sledećim vežbama.

U main.c osim main funkcije postoje i sledeće funkcije:

```
void vApplicationMallocFailedHook(void);
void vApplicationStackOverflowHook(TaskHandle_t pxTask, char* pcTaskName);

void vAssertCalled( unsigned long ulLine, const char * const pcFileName );
```

ove funkcije su sistemske, tj moraju se nalaziti u mainu, ali ne postoji potreba da ih modifikujemo niti je to dozvoljeno, pošto služe za proveru da li je kreiran neki objekat u operativnom sistemu i da obrišu memoriju u idle tasku. U *main.c* se vrši poziv za funkciju `I_vezba_1` (I pod komentareom `I_vezba_2`) gde se nalaze realizacije zadataka.

**Rešenje zadatka 1**.- Pogledajte fajl: I_Vezba_1_zadatak.c u projektu

Može se videti da main funckcija poziva funkciju `I_vezba_1()`. Osim ove funkcije, sve promenjive i funkcije u fajlu I_Vezba_1_zadatak.c kao i u I_Vezba_2_zadatak.c su tipa *static*, da bi imale doseg samo unutar ovog fajla. U ovoj funkciji se prvo vrši inicijalizacija displeja pomoću funkcija `mxDisp7seg_Init` i `mxDisp7seg_Open` čija se definicija nalazi u fajlu `mxDisp7seg.c`.

Posle tog prelazi se na kreiranje taska pomoću sistemske funkcije FreeRTOS-a `xTaskCreate`, koja

sadrži više parametara, od kojih je za ovaj zadatak najvažniji prvi parameter koji određuje funkciju ( *first_task* ) u kojoj se implemetira, tj izvršava ovaj task. Ako je task uspešno kreiran ( tj, sistem je uspeo da obezbedi resurse da se task kreira), funkcija `xTaskCreate` vraća vrednost `pdPASS,` što proveravamo preko `if (myTask != pdPASS).` Ako task nije uspešno kreiran blokiramo program preko while(1) .

Posle toga poziva se `vTaskStartScheduler(),` sistemska funkcija koja pokreće *raspoređivač* – nakon ovog poziva operativni sistem preuzima upravljanje sistemom. Posle ove funkcije mora ići beskoančna petlja.
Pošto je kreiran jedan task, on se posle pokretanja *raspoređivača* stalno izvršava, tj izvršava se funkcija `first_task.` Funkcija preko koje je implementiran task se **nikad** ne sme završiti, tj. u njoj se se mora nalaziti beskonačna petlja.

U while(1) petlji funkcije `first_task` se koristi sistemska funkcija FreeRTOS-a `vTaskDelay,` koja prebacuje task u blokirano stanje na vreme koje odgovara zadatom broju tick-ova, u ovom slučaju na 500 ms. Ovo se u radu programa manifestuje kao delay, kako joj i ime kaže. Pošto se svi vremenski parametri prosleđuju u vidu broja sistemskih tick-ova, koristimo makro funkcija koja konvertuje vreme u milisekundama u broj sistemskih tick-ova: *pdMS_TO_TICKS( vreme_u_milisekundama ),* čime se obezbeđuje da vremenske periode možemo podešavati u milisekundama.
U while(1) petlji funkcije *first_task* nalazi se i funkcija `Display_Toggle_0(),` koja se izvršava kada prodje delay, tj. svakih 500 ms. Pomoću funkcije `mxDisp7seg_SelectDigit` aktivira se određena cifra displeja, prvi parametar ove funkcije je refenca na objekat displeja, u ovom slučaju `myDisp,` a drugi parameter je redni broj cifre displeja koju želimo aktivirati, prva cifra (gledano sa leva na desno) ima vrednost 0. Preko funkcije `mxDisp7seg_SetDigit` podešavamo šta se ispisuje na aktivnoj cifri, može se kontrolisati svaki segment cifre. Upisivanjem odgovrajućeg člana niza `character`[] u cifru mogu se ispisati svi jednocifreni decimalni brojevi, od 0 do 9.

**Rešenje zadatka 2**.- Pogledajte fajl: I_Vezba_2_zadatak.c u projektu

Prosirujemo prethodni zadatak dodavanjem još jednog taska ponovnom pozivanjem `xTaskCreate,` koji je impleentiran pomoću funkcije `second_task` . U ovoj funkciji se u beskonačnoj petlji svakih 1000 ms izvršava funkcija `Display_Toggle_1.` Jedina razlika u odnosu na funkciju *Display_Toggle_1* je što je ovde aktivna druga cifra sa leva, pod rednim brojem 1.

Može se videti da se ova dva taska izvršavaju nezavisno jedan od drugog, tj. svaki od taskova izvršava kod koji izvršava u “paraleli“ sa drugim taskom. U praksi se naravno oba taska izvršavaju jedan za drugim, iako zbog multitaskinga izgleda da se izvršavaju istovremeno. https://www.freertos.org/implementation/a00004.html