SZÉCHENYI EGYETEM
UNIVERSITY OF GYŐR

CYBERSECLAB

WebElektronika

Mikrovezérlők

FreeRTOS alapjai

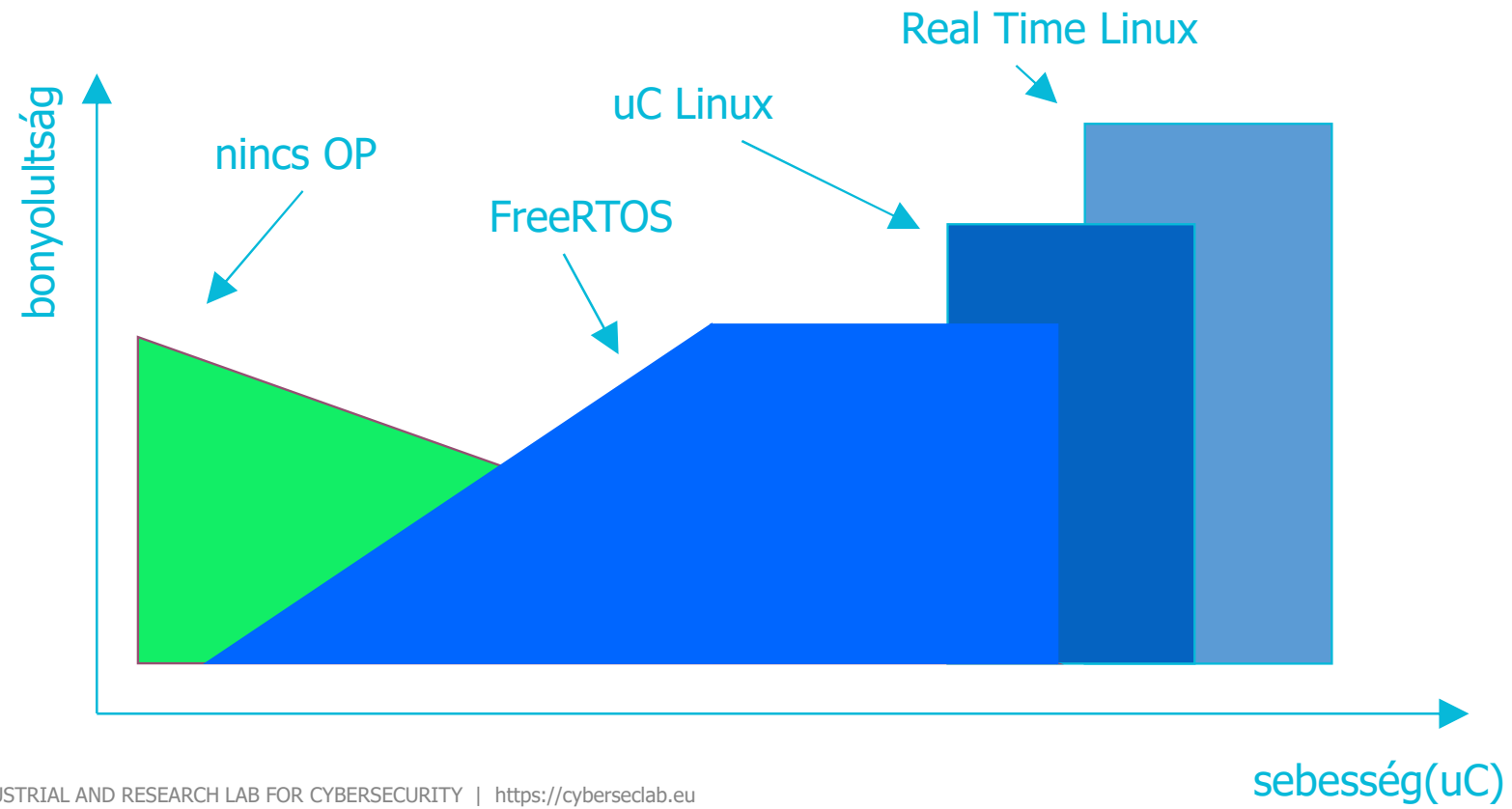
Dr. Hidvégi Timót
egyetemi docens

Tartalomjegyzék

Arduino IDE vs ESP-IDF

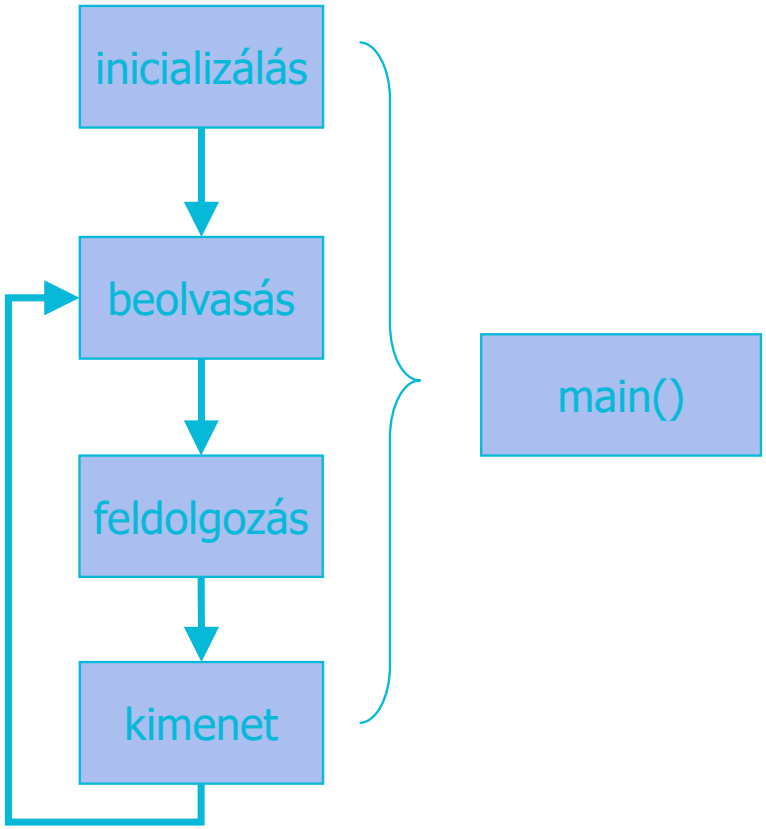
ESP-IDF	Arduino ESP32 core
✅ Native FreeRTOS support	— Limited RTOS support
✅ Task-based applications	— `setup()` and `loop()` functions
✅ Multi-core by default	— Single core by default
✅ Support for new ESP32 releases	— Limited support for new ESP32 releases
— Less beginner-friendly	✅ Beginner-friendly
— Smaller community	✅ Large community

Operációs rendszerek alkalmazása



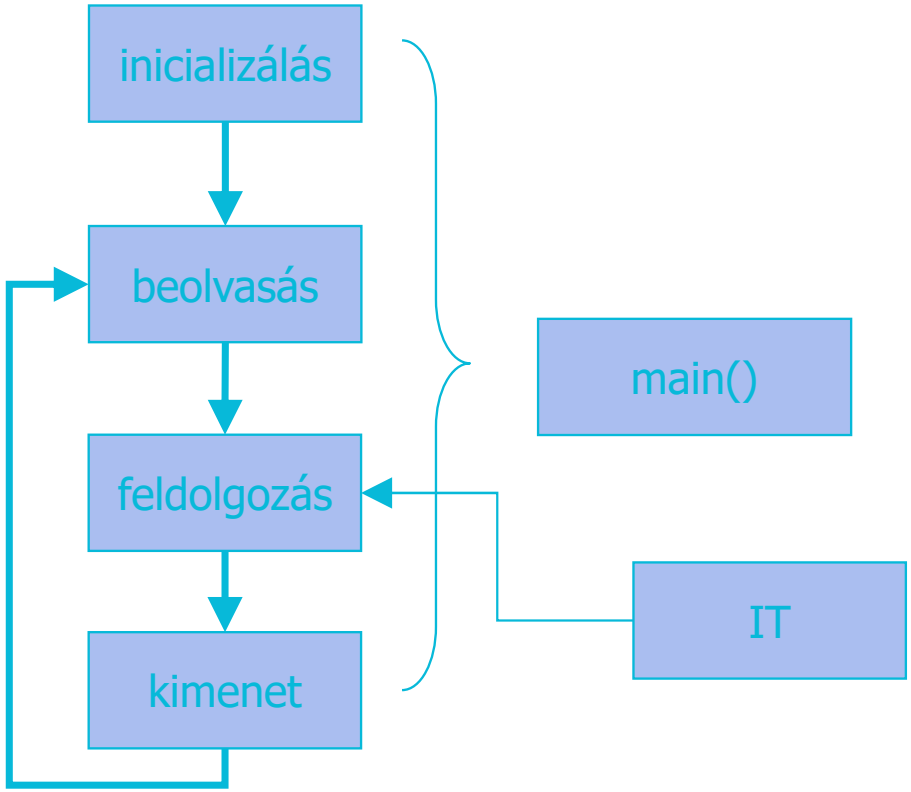
„Lineáris” programozás

- Egy főprogram
- Kisebb alkalmazások
- Tulajdonságok
  - Több fejlesztő nehezen dolgozik együtt
  - Nem kell „komolyabb” tudás
  - Nem energiatakarékos, állandóan fut



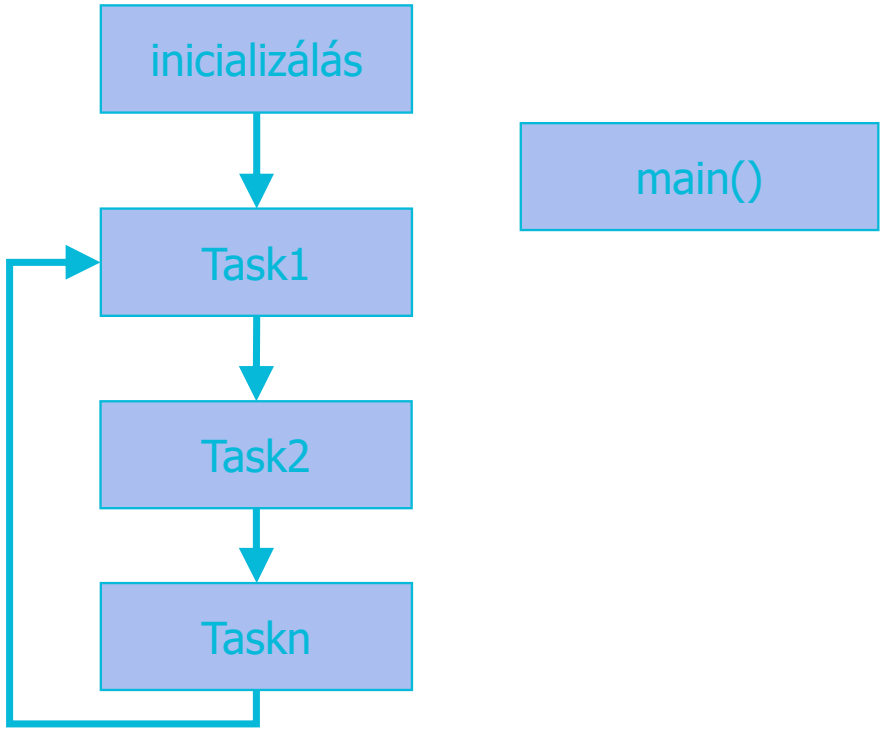
„Lineáris” programozás IT-vel

- Megszakítás használata
- Tulajdonságok
  - Nincs (?) pollingolás
  - Gyorsabb, hatékonyabb
  - Nehézkes a program átlátása nagy méretnél



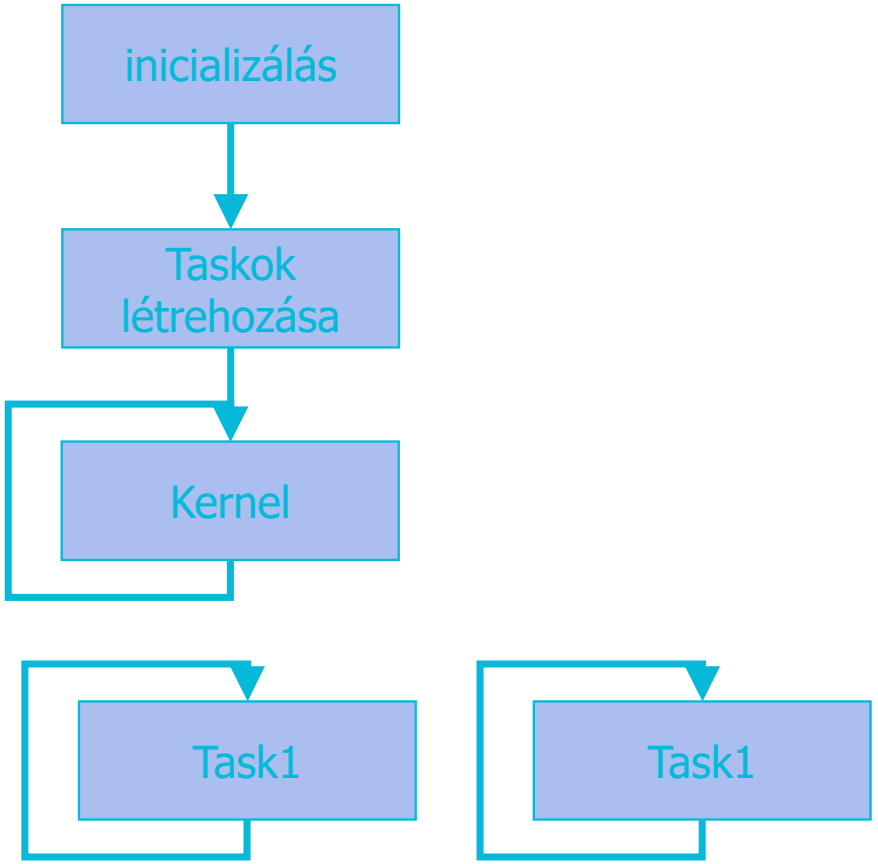
Kooperatív multitask

- Különböző függvények kerülnek alkalmazásra, ezekben nincs végtelen ciklus
- Tulajdonságok :
  - Átlátható a program
  - Több fejlesztő
  - Nem kell OP
  - Nehéz kezelni az energiatakarékosságot



Időosztásos multitask

- A kernel intézi a task-ok meghívását
- Tulajdonságok :
  - Könnyen módosítható platformváltáskor
  - A részfeladatok egymást nem zavarják
  - Nagy programoknál is kényelmes a használata
  - OP rendszer kell ehhez



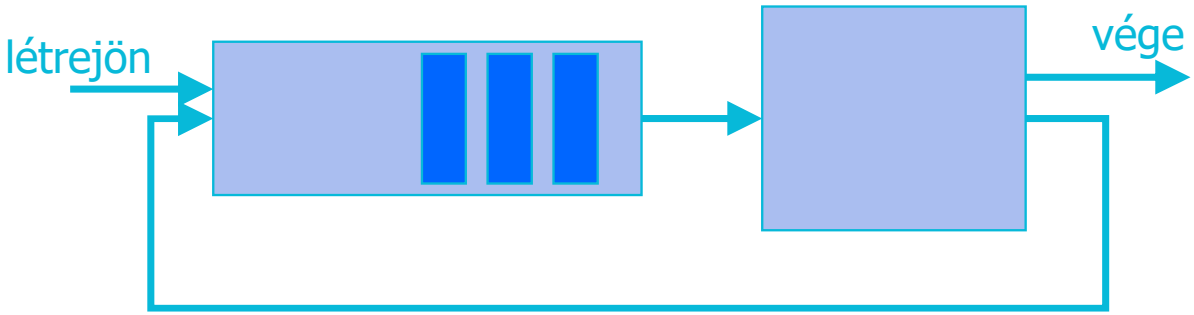
Operációs rendszerek I.

- Feladatok :
  - Task-ok futtatása
    - *Task :*
      - Részfeladatokat lát el
      - A futtató processzort minden task a sajátjaként kezeli
      - Ezek „párhuzamosan” futnak (látszólag)
  - Biztosítja a task-ok közötti kommunikációt
  - Memóriamenedzsment
  - Hardverfüggetlenség
  - (közös) erőforrások kezelése

- Task létrehozása
  - Létrehozásra kerül minden task-hoz egy „Task Control Block” (TCB)
    - *Lefoglalt memóriaterület (dinamikus)*
      - Ebben folyamatleírók vannak (és ezeknek az eltárolására szolgáló memóriaterület)
    - *Taskváltáskor a változókat, regisztereket menteni kell*
    - *Állapotok : Fut, futásra kész, várakozik*
- Várakozás
  - Egyszerre egy task futhat (övé a processzor), ezért várakozási sor lesz
  - Lista (adat és mutató, a mutató a következőre mutat)

Operációs rendszer II.

- Ütemező
  - A futásra kész task-ok között adja oda a processzort
  - Prioritás alapján dönt



- Valósidejű rendszer
  - Prioritások vannak
  - Meghatározott idő van a feldolgozásra
  - Azonos prioritású eseményeket bekövetkezés sorrendjében dolgozzák fel
  - Fajtái :
    - *Kemény : esemény feldolgozási ideje meg van határozva, késő válasz nem jó*
    - *Lágy : jó lehet a késő válasz is, átlagidő van meghatározva*
    - *Erős : keveréke az előzőeknek*
- Nem valósidejű rendszerek
  - Nem feltétlenül egymás után kerül feldolgozásra a bejövő esemény
  - A feldolgozási idő nincs mértéke nincs meghatározva

Operációs rendszerek

- Prioritás :
  - Minden task a prioritása alapján jut processzorhoz
  - Ez a prioritás változhat dinamikusan, illetve lekérhető
  - Hiearchikus felépítés
  - Ha két azonos prioritású task van, akkor az ütemező „körbe körbe jár” (egyforma időt kapnak egymás után)
- Ütemezési algoritmusok
  - First Come First Served (FCFS) A taskok érkezési sorrendben kapják meg a processzort.
    - *egyszerű*
    - *lassú, „csorda hatás”, sok múlik (idő) az érkezési sorrendtől*
  - Shortest Job First (SJF) Ha egy befejezi a feladatát, a futási jogot a legrövidebb kapja
    - *Legrövidebb a várakozási idő*
    - *Kiéheztetés (hosszú folyamatnál), előre kell tudni a folyamat hosszát, becslés (?), lehet a becslés rossz is….*
  - Round Robin (RR)

Operációs rendszerek

- Round Robin (körben forog)
  - A folyamatok, task-ok zárt körben vannak, előre definiált időre kapják meg a processzort. Azután sorvégére kerül.
  - Alkalmazhatók prioritások is
  - Interaktív rendszerekben használják inkább
  - Egyszerű, nincs kiéheztetés
  - Ha lejár az időszelet, el kell menteni a task állapotát (időveszteség)

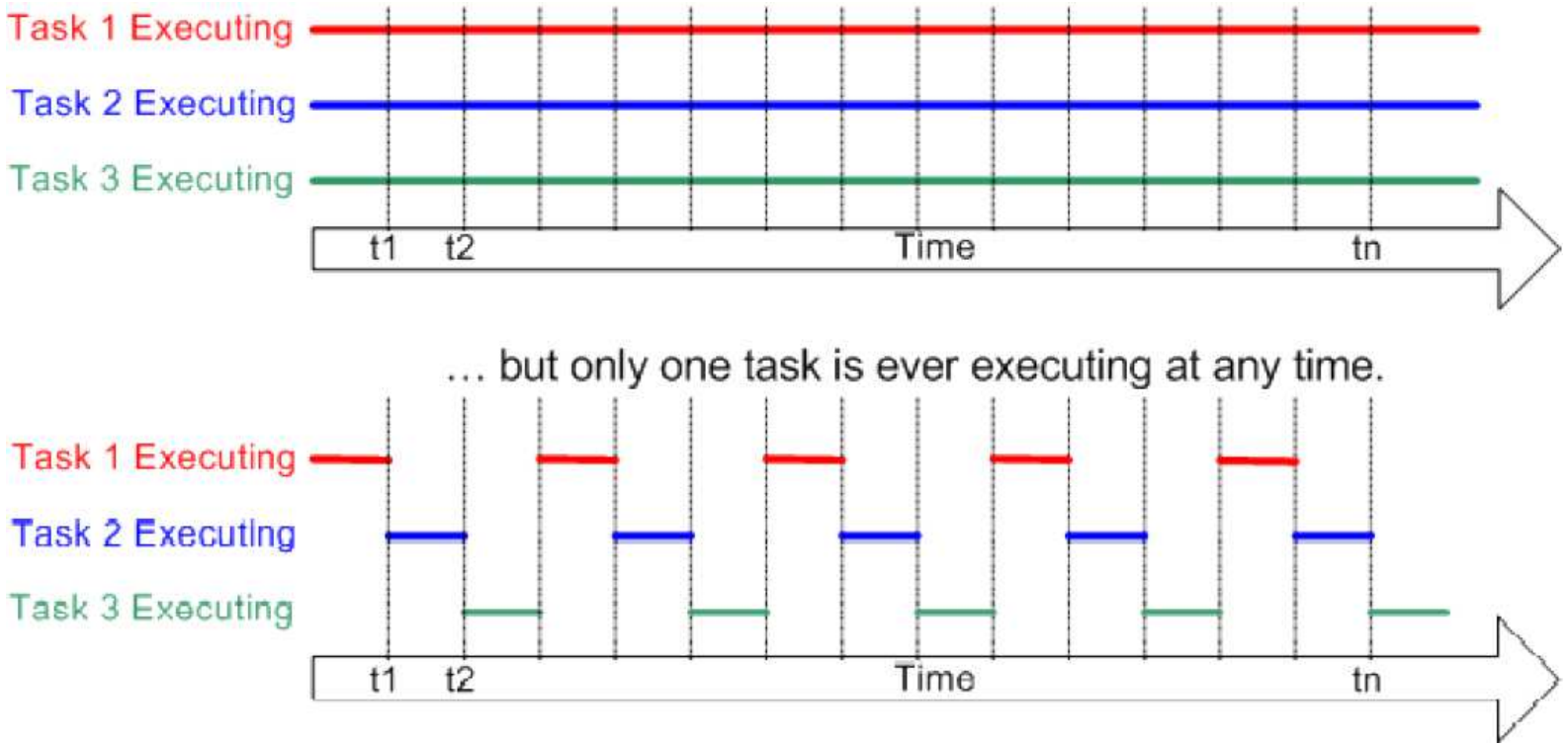
RTOS

- RTOS vs egyszerű C program?
  - A valós idejű operációs rendszerek kulcsfontosságú tényezői a minimális megszakítási késleltetés és a minimális szálváltási késleltetés.
  - A valós idejű operációs rendszert inkább azért értékelik, hogy milyen gyorsan és mennyire kiszámíthatóan reagál a feladatok adott időn belüli elvégzésére.
- RTOS főbb típusai
  - Kemény RTOS
    - *a feladat adott határidőn belüli elvégzésére kötelezett.*
  - Firm RTOS
    - *határidőhöz kötött, de ha lekésik a határidőt, az elfogadható, de a Hard RTOS esetében nem.*
  - Lágy RTOS
    - *nincs határidőhöz kötve*
- Pár példa
  - FreeRTOS, AAOS, LynxOS, RTLinux, VxWorks, OSE, QNX, Windows CE

FreeRTOS

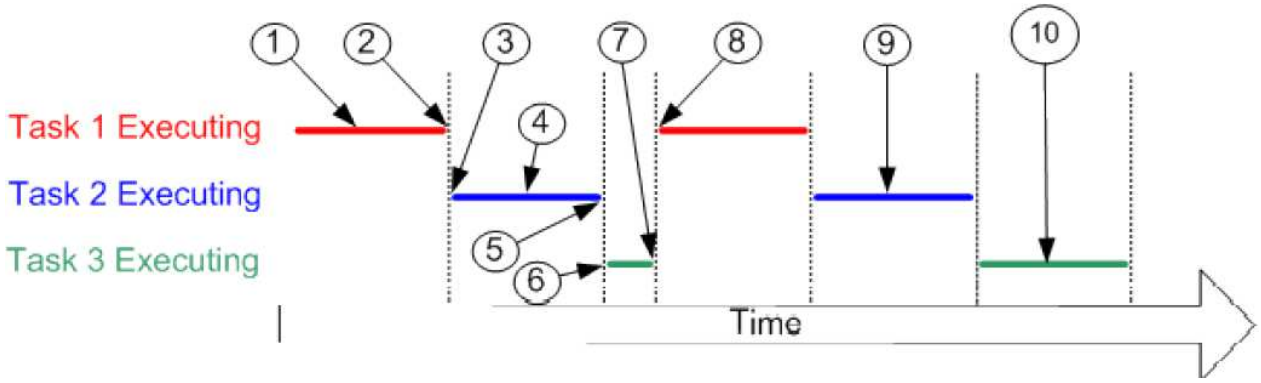
- Nyílt forráskódú „C” nyelven írt
- 16, 32 bites uC-nél használható, PIC18F-nél megkötésekkel
- Op rendszer együtt fordul le a fejlesztő kódjával
- Van fizetős verzió is :
  - OpenRTOS
  - SecureRTOS
- https://www.freertos.org
- https://github.com/ExploreEmbedded/Arduino_FreeRTOS/archive/master.zip

Task-ok futtatása



Task-ok futtatása

- Task1 fut
- A kernel felfüggeszti a Task1-et
- A kernel elindítja a Task2-t



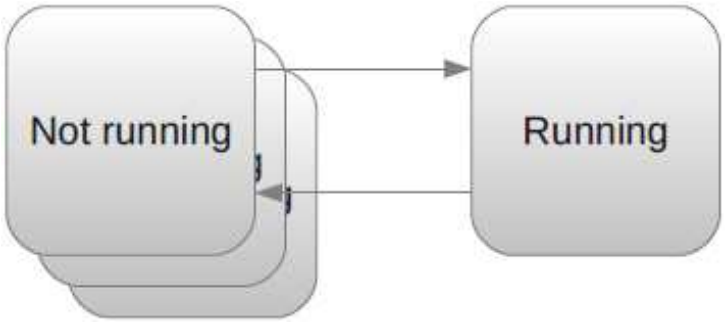
Főbb tulajdonságok

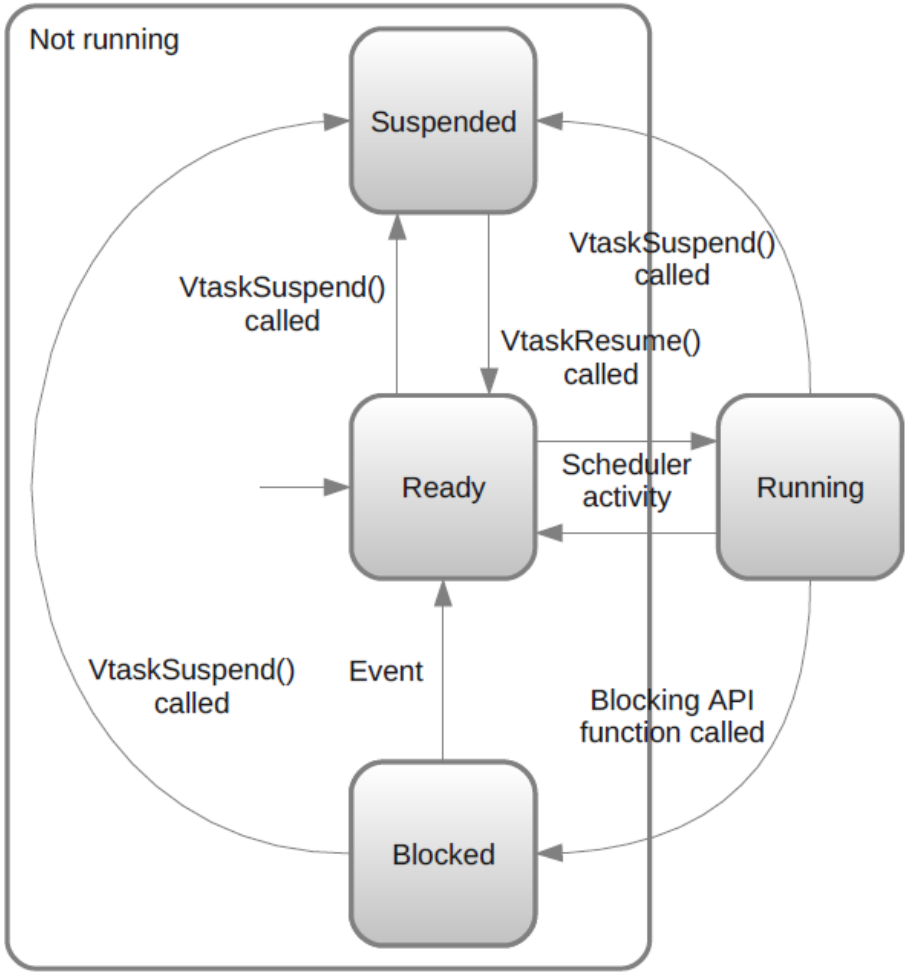
- Nyílt forráskódú „C” nyelven írt
- 16, 32 bites uC-nél használható, PIC18F-nél megkötésekkel
- Op rendszer együtt fordul le a fejlesztő kódjával
- Van fizetős verzió is
  - OpenRTOS
  - SecureRTOS

Nevek

- A függvénynév mindig a visszatérési értékkel kezdődik.
  - Például :
    - *vTaskDelete() : void-ot ad vissza, a task.c-ben található*
    - *vSemaphoreCreateBinary() : void-ot ad vissza, a queue.c-ben van*
  - Láthatóság :
    - *Ha privát a láthatóság, akkor „prv” előtagot kap a függvény, ekkor a saját file-ban érhetők csak el.*

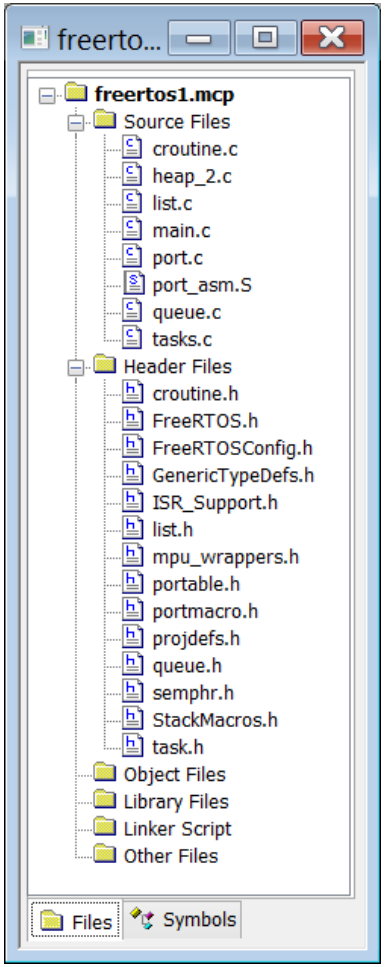
Szálak állapota





Állományok (régebben)

- Rendszermag :
  - tasks.c
  - croutine.c
  - list.c
  - queue.c
- Fejléc
  - tasks.h
  - croutine.h
  - list.h
  - queue.h
- Fejlécállományok :
  - FreeRTOS.h
- Hardverabsztrakciós réteg :
  - port.c
  - portmacro.h
- Memóriamenedzsment :
  - heap_1.c
  - heap_2.c
  - heap_3.c
- Konfigurációs állomány :
  - FreeRTOSConfig.h



Task-ok létrehozása

- A függvény, amelyből a task-ot létrehozzuk, végtelen ciklussal rendelkezik. Lehet törölni a szálat.
- Létrehozás :
  - xTaskCreate(. . .)
  - Ha a szál létrejön, akkor a fgv pdPASS értéket ad vissza
- Törlés :
  - vTaskDelete(xTaskHandle . . .)
- A szál általában végtelen kialakítású, nem érhet véget.

```
void vTaskCode( void *pvParameters )
{
        for( ;; )
        {
                kód
        }
}
```

```
xTaskCreate( vTaskCode, "Nev", 256, NULL, 1Y, NULL);
```

Task-ok

- Ha magasabb prioritású task blokkolva lesz, alacsonyabb fog futni
- Ha mindegyik szál blokkolva van, akkor a pihenőszál fog elindulni

- vTaskDelay - „pontatlan” várakozás, adott ideig alszik
- vTaskDelayUntil - „pontos” várakozás, az utolsó felébredéstől adott ideig alszik a szál
- vTaskPrioritySet
- vTaskSuspend
- vTaskResume

Task-ok

- void vTaskSuspend(xTaskHandle pxTaskToSuspend);
  - Felfüggeszti a task-ot

```
xTaskCreate(vTask1, „Név", 256, NULL, 1, &xHandle );
vTaskSuspend( xHandle );
```

- void vTaskResume(xTaskHandle pxTaskToResume);
  - „visszaveszi” a felfüggesztett task-ot

```
xTaskCreate( vTask1, „Név", 256, 1, &xHandle );
vTaskSuspend( xHandle );
vTaskResume( xHandle );
```

Task-ok (bead, kivesz)

- Egy memóriaterületre („postaláda”) adat tehető be vagy vehető ki :
  - fogyasztó : adatot vesz ki
  - termelő : adatot tesz be
- Egy memóriaterületet ilyen célra többen is használhatják egyidőben

Sorok

- Taskok közötti kommunikációra szolgál, rögzített méretű terület
- Erőforrás, tehát blokkolhat task-okat!
  - Tele a sor
  - Üres a sor
- Ha több értéket adunk a task-nak -> struktúra használata
- Létrehozás
  - xQueueCreate( )
- Elhelyezés a sor elejére
  - xQueueSendToFront( )
- Elhelyezés a sor végére
  - xQueueSendToBack( )
- Adatot felhasználhatunk :
  - Kivétellel :
    - *xQueueReceive( )*
  - Kivétel nélkül :
    - *xQueuePeek( )*

Rendszer vezérlése

- vTaskStartScheduler
- vTaskEndScheduler
- vTaskSuspendAll
- vTaskResumeAll

- void vTaskDelay(portTickType xTicksToDelay);
  - Tick-ek száma alapján késleltet (adott ideig késleltet)

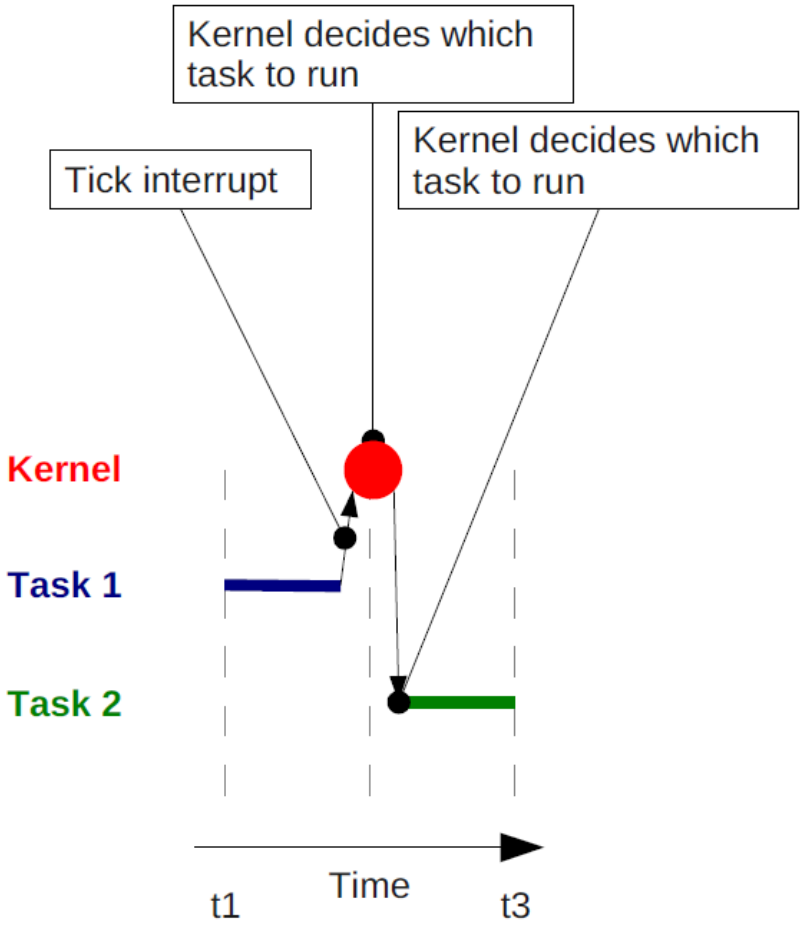
  portTickType xDelay

  . . . .

  vTaskDelay(xDelay);
- void vTaskDelayUntil(portTickType *pxPreviousWakeTime, portTickType xTimeIncrement)
  - Utolsó felébredéstől ideig alszik

Szálak állapota

- Két állapot lehet :
  - Fut
  - Nem fut
    - *Futásra kész (magasabb, vagy ugyanolyan prioritású szál fut, a szál várakozik)*
    - *Felfüggesztett (suspended) (ritkán használt) A task nem versenyez a futási jogért*
    - *Blokkolt (lehet erőforráshiány is, nem versenyez másokkal)*
- Mindig csak egy szál lehet „fut” állapotban
- Az állapotváltásnál adminisztrációs idő lép fel



Prioritás

- Prioritás rendelhető minden szálhoz, illetve ezek lekérdezhetők (vTaskPrioritySet())
- Prioritás 0 és configMAX_PRIORITIES-1 között lehet (FreeRTOSConfig.h)
- A prioritás futás közben megváltoztatható és lekérdezhető
- Ha magasabb prioritású task blokkolva lesz, alacsonyabb fog futni
- Ha mindegyik szál blokkolva van, akkor a pihenőszál fog elindulni

```
#define configUSE_PREEMPTION                    1
#define configUSE_IDLE_HOOK                     1   // 0- volt, de engedélyeztem
#define configUSE_TICK_HOOK                     0
#define configTICK_RATE_HZ                      ( ( portTickType ) 1000 )
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) 80000000UL )
#define configPERIPHERAL_CLOCK_HZ               ( ( unsigned long ) 40000000UL )
#define configMAX_PRIORITIES                    ( ( unsigned portBASE_TYPE ) 5 )
#define configMINIMAL_STACK_SIZE                ( 190 )
#define configISR_STACK_SIZE                    ( 400 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) 28000 )
```

Prioritás

- Lekérés
- unsigned portBASE_TYPE uxTaskPriorityGet(xTaskHandle pxTask);
  - Visszaadja a task prioritását

```
if( uxTaskPriorityGet( xHandle ) != tskIDLE_PRIORITY )
{
    kód
}
```

- Beállítás
- void vTaskPrioritySet(xTaskHandle pxTask, unsigned portBASE_TYPE uxNewPriority);
  - Beállítja egy task-nak a prioritását

```
vTaskPrioritySet( xHandle, tskIDLE_PRIORITY + 1 );
```

Pihenőszál

- Mi van akkor, ha minden szál blokkolva van? -> pihenőszálé lesz a vezérlés
- Neve állandó : void vApplicationIdleHook(void)
- Energiatakarékos módnál hasznos (Idle() makró)
- A pihenőszál nem tartalmazhat végtelen ciklust, blokkolást

```
#define configUSE_PREEMPTION                    1
#define configUSE_IDLE_HOOK                     1   // 0- volt, de engedélyeztem
#define configUSE_TICK_HOOK                     0
#define configTICK_RATE_HZ                      ( ( portTickType ) 1000 )
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) 80000000UL )
#define configPERIPHERAL_CLOCK_HZ               ( ( unsigned long ) 40000000UL )
#define configMAX_PRIORITIES                    ( ( unsigned portBASE_TYPE ) 5 )
#define configMINIMAL_STACK_SIZE                ( 190 )
#define configISR_STACK_SIZE                    ( 400 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) 28000 )
```

Memóriakezelés

- Három algoritmus, dinamikus memóriakezelés
  - heap_1.c
  - heap_2.c
  - heap_3.c

  } configTOTAL_HEAP_SIZE makró

```
#define configUSE_PREEMPTION                    1
#define configUSE_IDLE_HOOK                     1   // 0- volt, de engedélyeztem
#define configUSE_TICK_HOOK                     0
#define configTICK_RATE_HZ                      ( ( portTickType ) 1000 )
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) 80000000UL )
#define configPERIPHERAL_CLOCK_HZ               ( ( unsigned long ) 40000000UL )
#define configMAX_PRIORITIES                    ( ( unsigned portBASE_TYPE ) 5 )
#define configMINIMAL_STACK_SIZE                ( 190 )
#define configISR_STACK_SIZE                    ( 400 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) 28000 )
```

Memóriakezelés

- heap_1.c
  - Lefoglalni le tud, felszabadítani nem
  - Egyszerű
  - pvPortMalloc() fgv implementált csak
- heap_2.c
  - Lefoglal és felszabadít mem. területet
  - Best fit, legjobban illeszkedő memóriaterületet keresi
- heap_3.c
  - A memória méretét a fordítónál adjuk meg, a malloc() és a free() függvényeket használja memória kezelésére
  - malloc() és a free() működése alatt az ütemező nem működik

FreeRTOS az esp32-ben

- TaskCreatePinnedToCore() függvény a Task létrehozásához a core0 és a core1 számára
- Hét paraméter
  1. Task-ot megvalósító függvény neve
  2. Az a név, amelyet a feladatnak adunk, pl. Task1 vagy Task2.
  3. A következő a stack mérete, amelyet a feladatnak adunk pl. 10000.
  4. A Task bemeneti paramétere.
  5. A Task prioritása következik, ahol a 0 a legalacsonyabb prioritás.
  6. A Task kezelője, amelyet korábban definiáltunk, pl. &Task1 vagy &Task2.
  7. Végül megadjuk a mag azonosítóját, ahol az adott feladat futni fog, pl. 0 vagy 1, ahol a 0 a core0 és az 1 a core1.

```
xTaskCreatePinnedToCore(Task1Code,"Task1",10000,NULL,1,&Task1,0);
```

Industrial and Research Lab for Cybersecurity

- Web
  - https://cyberseclab.eu
- Facebook
  - https://www.facebook.com/IndustrialandResearchLab
- Github
  - https://github.com/cyberseclabor
- Linkedin
  - https://www.linkedin.com/company/industrial-and-research-lab-for-cybersecurity

SZÉCHENYI EGYETEM
UNIVERSITY OF GYŐR
GÉPÉSZMÉRNÖKI, INFORMATIKAI ÉS VILLAMOSMÉRNÖKI KAR

CYBERSECLAB

Industrial and Research Lab for Cybersecurity