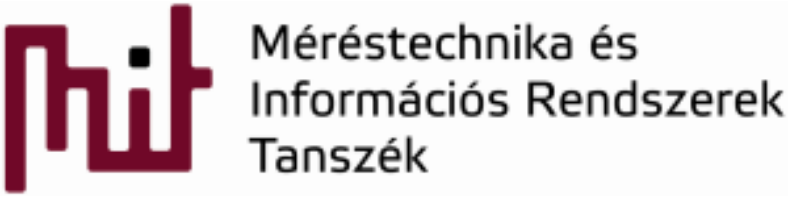
Nagyteljesítményű mikrovezérlők
10b. RTOS gyakorlatok

Scherer Balázs

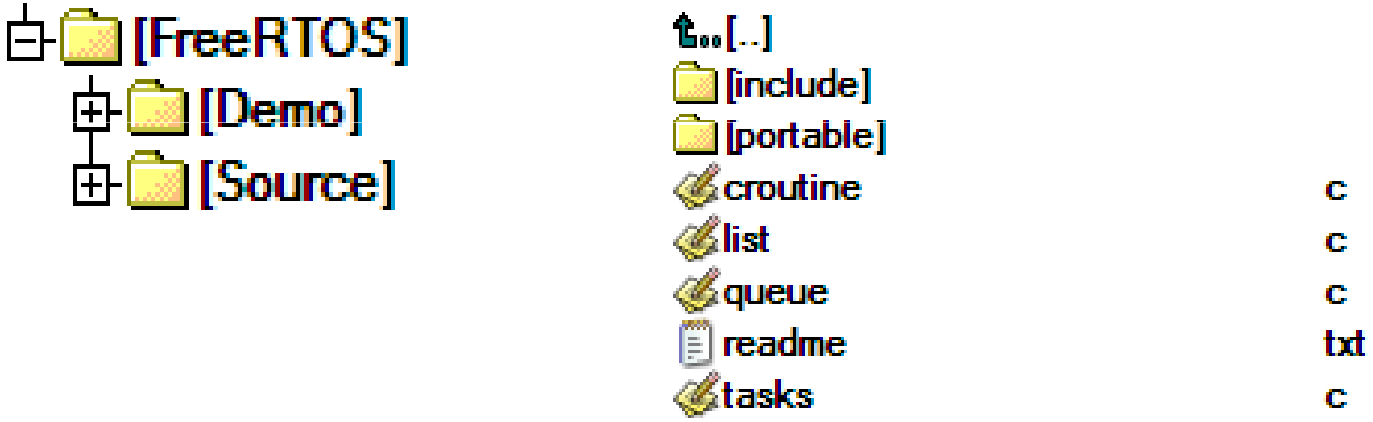


Budapest University of Technology and Economics
Department of Measurement and Information Systems

FreeRTOS

FreeRTOS forráskód elrendezés

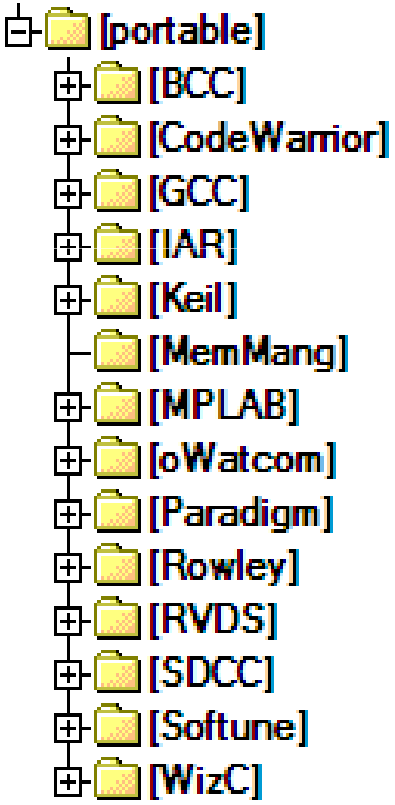
- Nagyon egyszerű alap kernel
  - tasks.c, queue.c, list.c. File-ok az Source könyvtárban



  - Ezek a file-ok tartalmazzák az alap taszk létrehozást és szinkronizációt.

FreeRTOS portolás

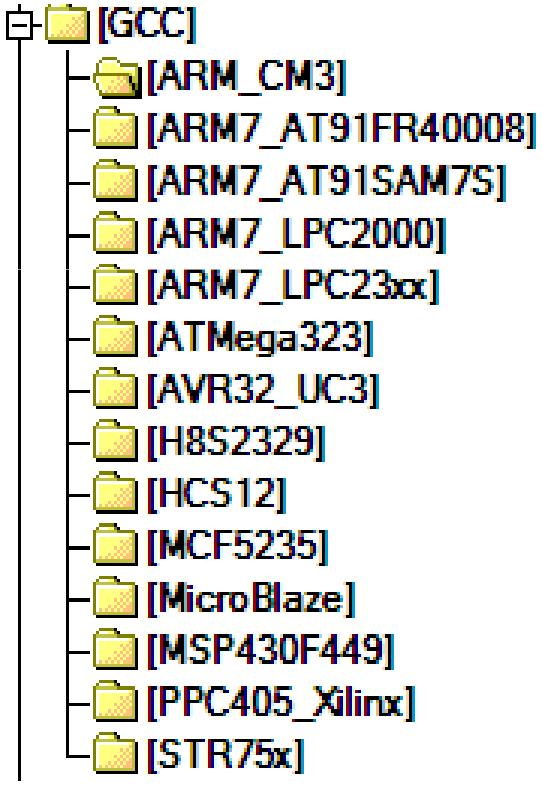
- A port specifikus kód részek a portable directoryban



- Task váltás, Sys tick timer, Critical szekcióba lépés és elhagyás
- Toolchain szerint rendezve

GCC specifikus részek

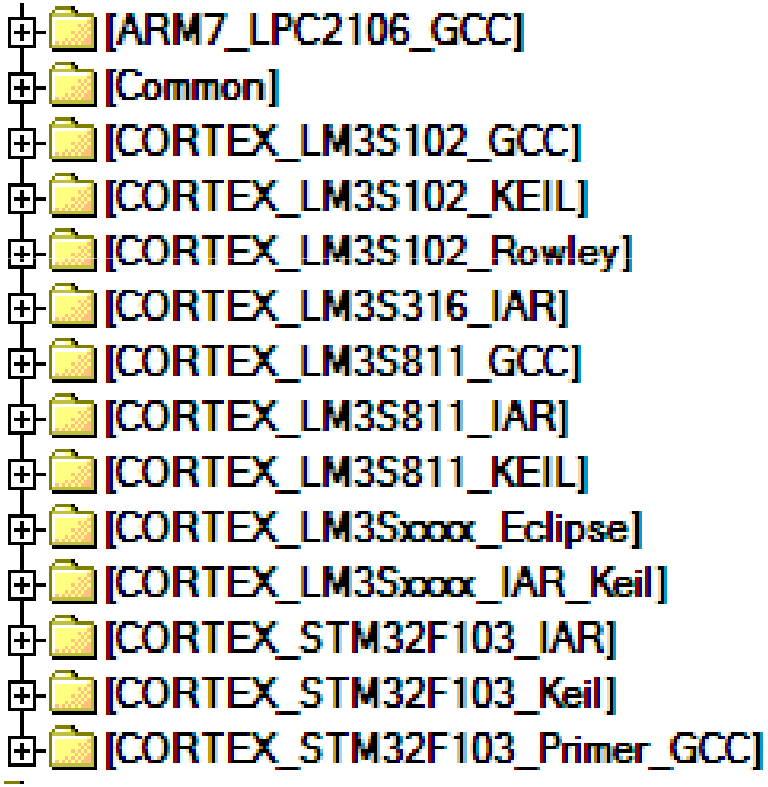
- GCC specifikus részek



- Egy port file

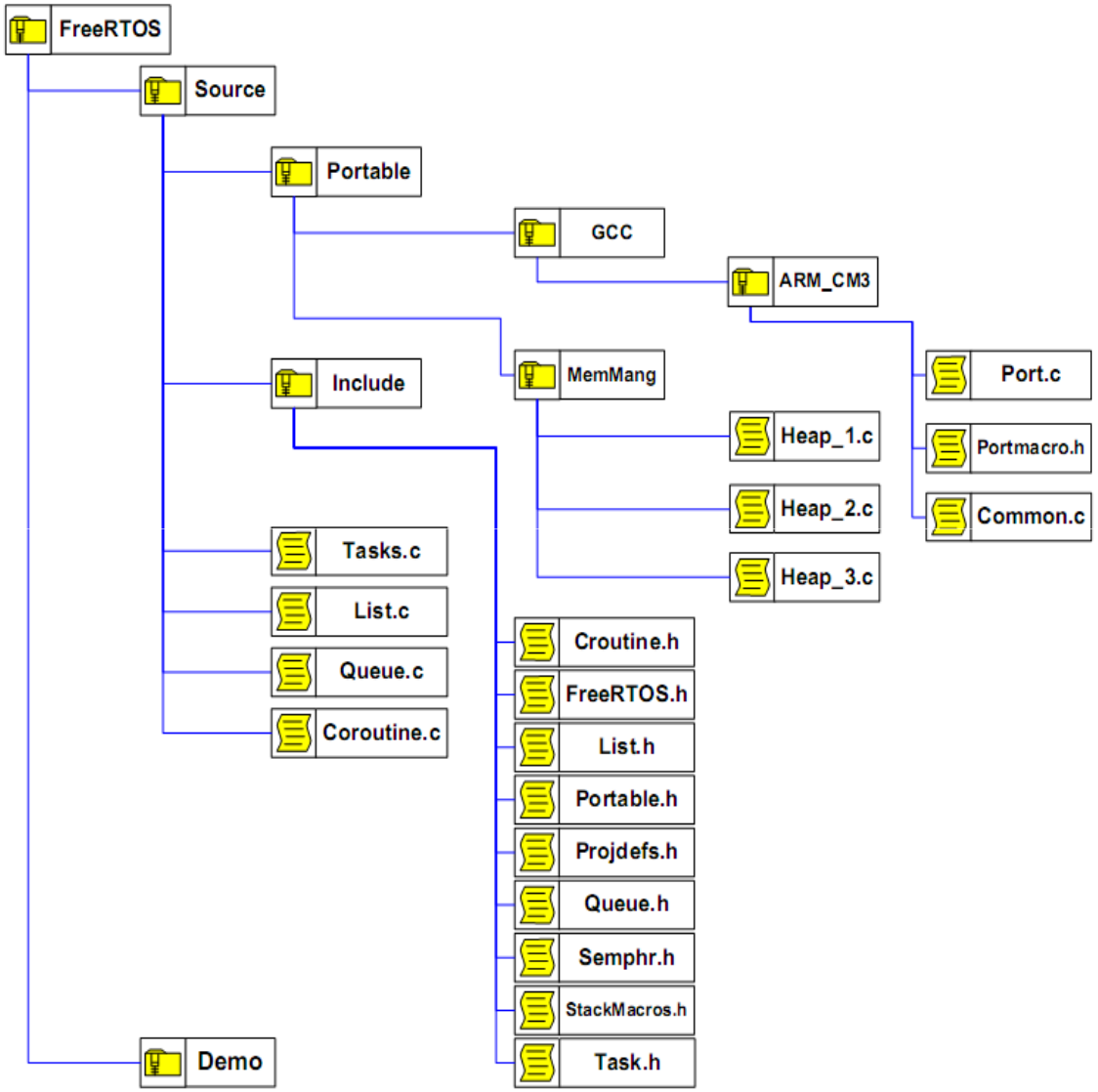
GCC demó projectek

- Kártya és fordító specifikus részek



- Startup kód
- Kártya specifikus kód

A FreeRTOS könyvtárszerkezete *áttekintés*



FreeRTOS konfiguráció

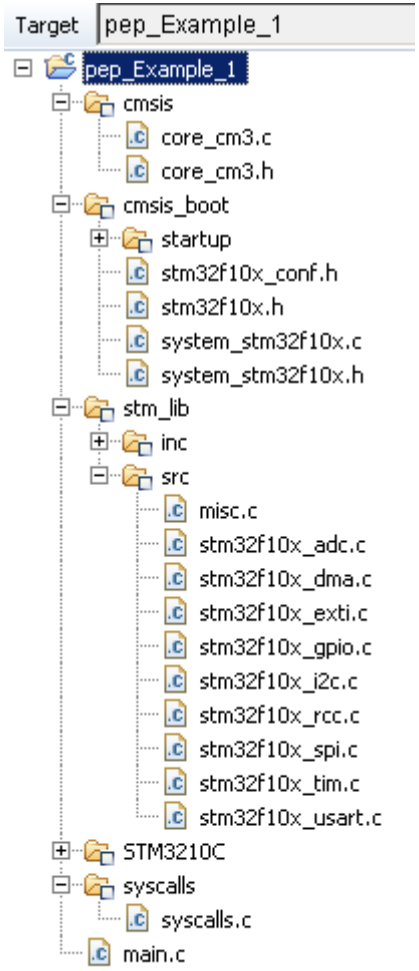
- FreeRTOS_Config.h

```
/*-----------------------------------------------------------
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *-----------------------------------------------------------*/

#define configUSE_PREEMPTION		1
#define configUSE_IDLE_HOOK		0
#define configUSE_TICK_HOOK		0
#define configCPU_CLOCK_HZ		( ( unsigned portLONG ) 20000000 )
#define configTICK_RATE_HZ		( ( portTickType ) 1000 )
#define configMINIMAL_STACK_SIZE	( ( unsigned portSHORT ) 70 )
#define configTOTAL_HEAP_SIZE		( ( size_t ) ( 7000 ) )
#define configMAX_TASK_NAME_LEN		( 10 )
#define configUSE_TRACE_FACILITY	0
#define configUSE_16_BIT_TICKS		0
#define configIDLE_SHOULD_YIELD		0
#define configUSE_CO_ROUTINES		0

#define configMAX_PRIORITIES		( ( unsigned portBASE_TYPE ) 5 )
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )
```

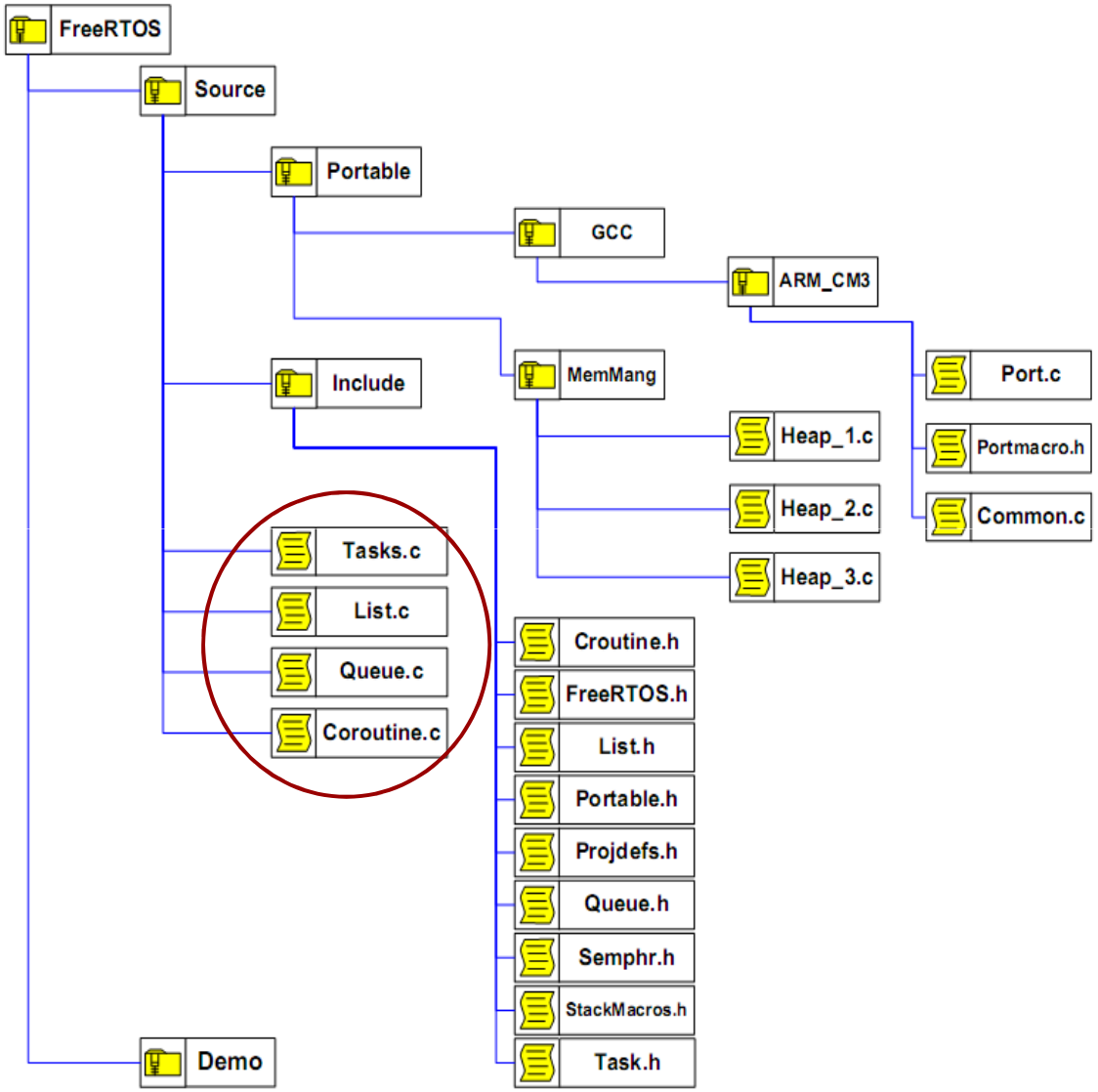
2. program
FreeRTOS beillesztése

FreeRTOS konfiguráció

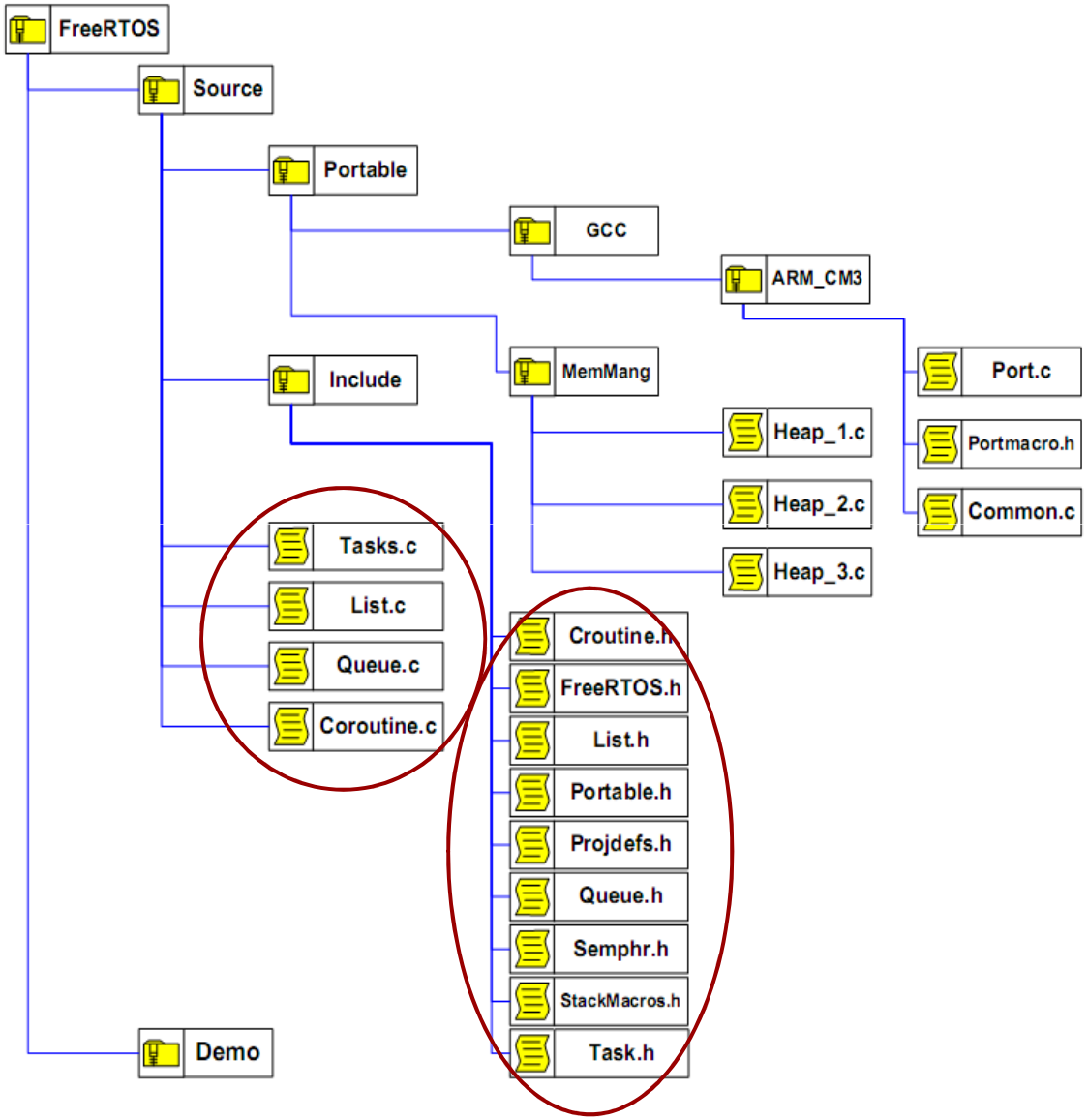
- Kiindulási alap az 1. program



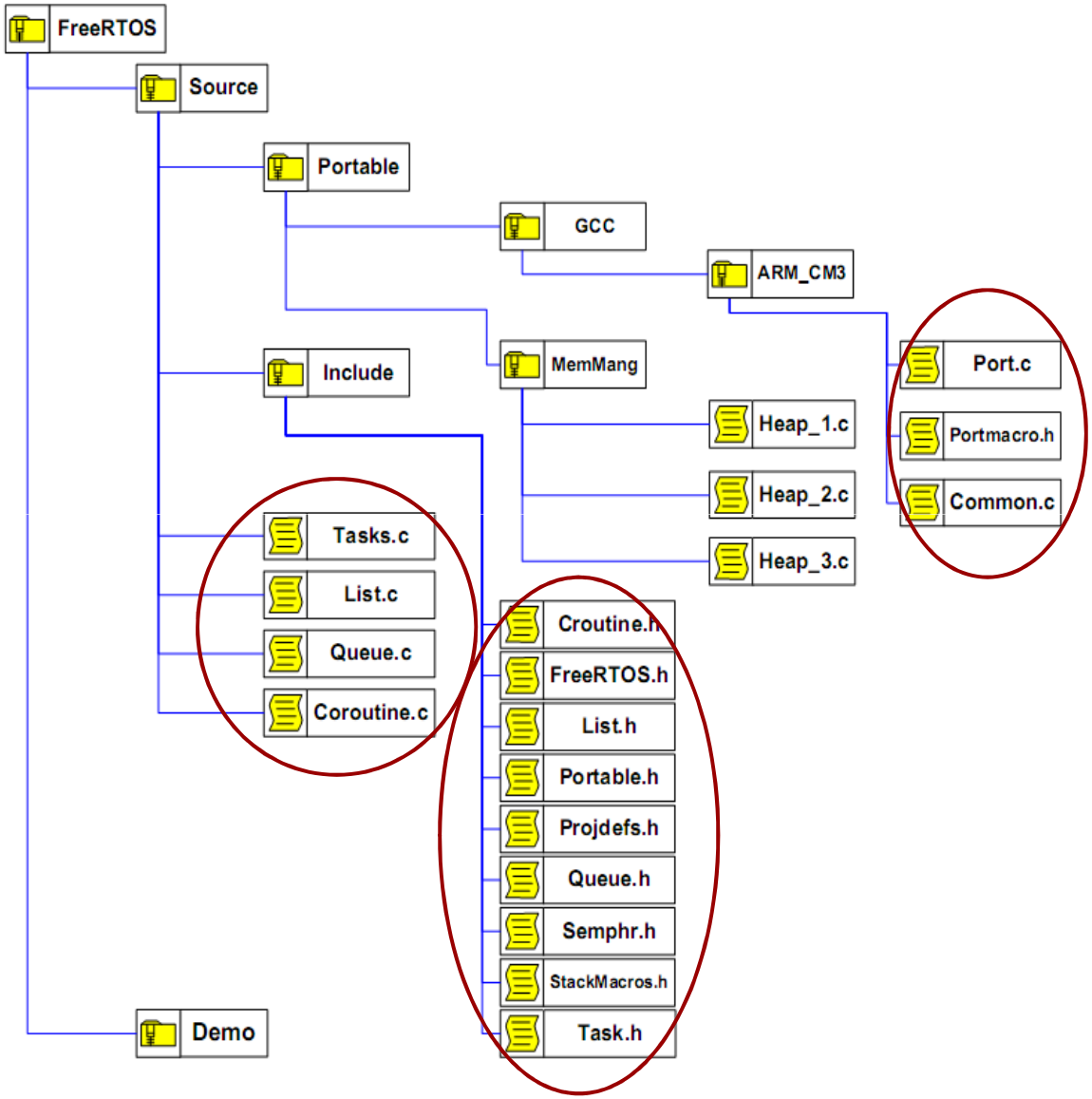
Hozzáadandó



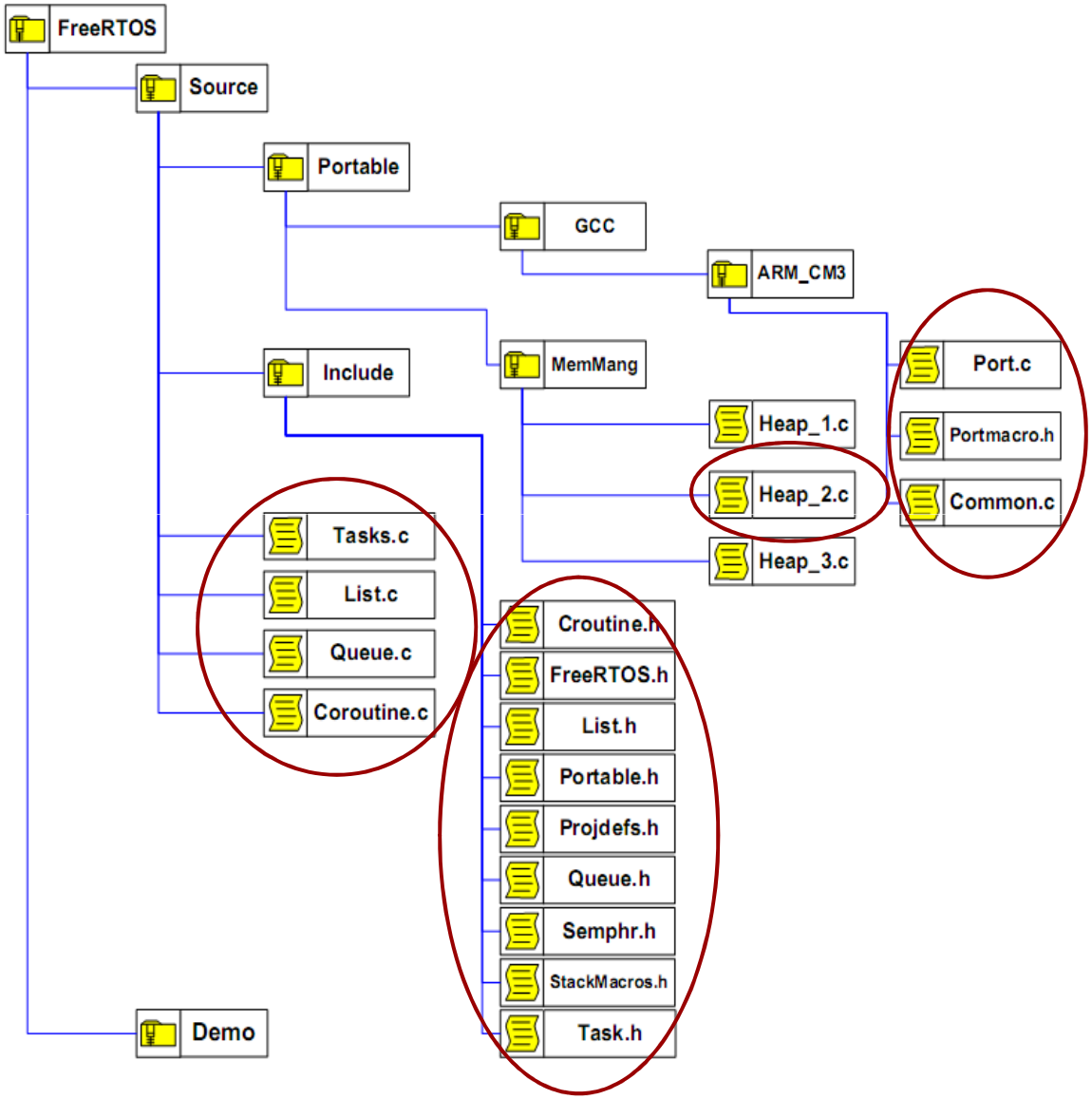
Hozzáadandó



Hozzáadandó



Hozzáadandó



Interupt vektorok

- 3 interupt vektor
  - Systick : oprendszer heartbeat timer
  - SVC: elindulásnál az első szál indítása
  - PendSVC: taszk váltás

- A portban meg vannak az implementációk
  - xPortSysTickHandler()
  - xPortPendSVHandler()
  - vPortSVCHandler()

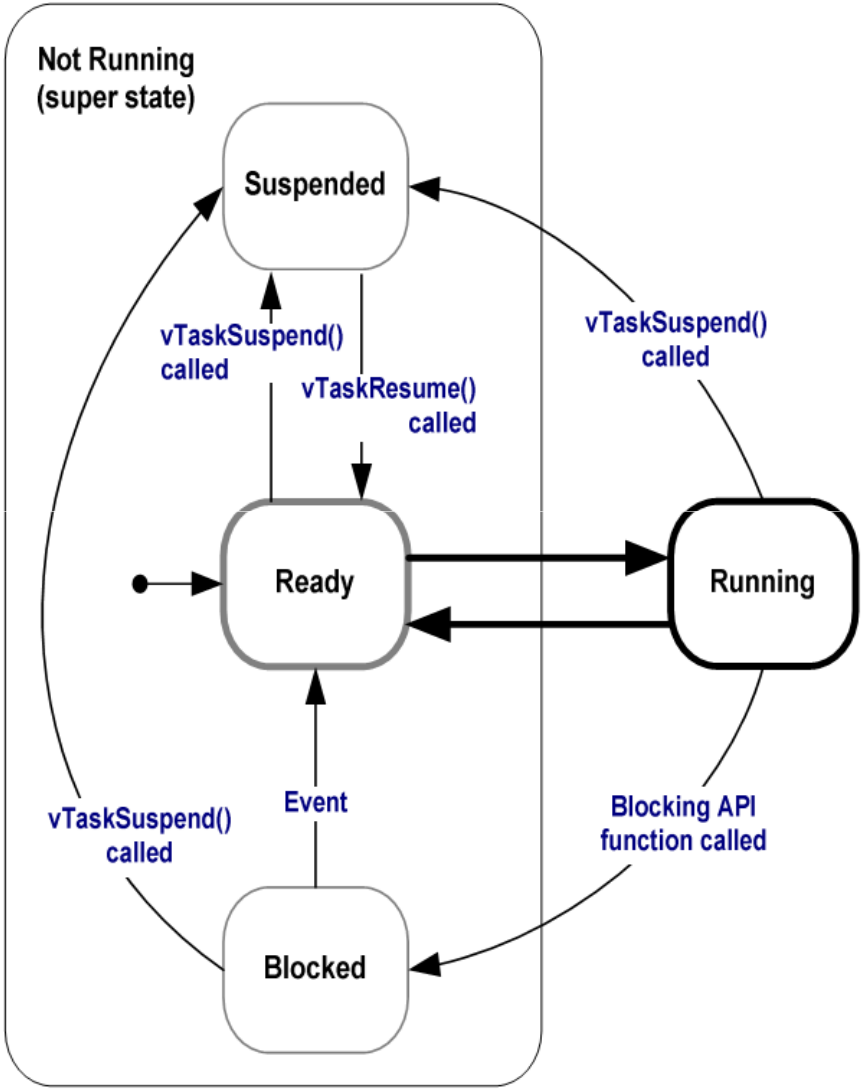
- CMSIS startup részében kell módosítani

Startup kód változtatás

- Hivatkozni a külső file-ban elérhető függvényekre
  - extern void xPortSysTickHandler(void);
  - extern void xPortPendSVHandler(void);
  - extern void vPortSVCHandler(void);
- A megfelelő vektorokat feülírni

```
130 __attribute__ ((section(".isr_vector")))
131 void (* const g_pfnVectors[])(void) =
132 {
133   /*----------Core Exceptions------------------------------------------------------*/
134   (void *)&pulStack[STACK_SIZE-1],     /*!< The initial stack pointer          */
135   Reset_Handler,                 /*!< Reset Handler                            */
136   NMI_Handler,                   /*!< NMI Handler                              */
137   HardFault_Handler,             /*!< Hard Fault Handler                       */
138   MemManage_Handler,             /*!< MPU Fault Handler                        */
139   BusFault_Handler,              /*!< Bus Fault Handler                        */
140   UsageFault_Handler,            /*!< Usage Fault Handler                      */
141   0,0,0,0,                       /*!< Reserved                                 */
142   vPortSVCHandler,                  /*!< SVCall Handler                           */
143   DebugMon_Handler,              /*!< Debug Monitor Handler                    */
144   0,                             /*!< Reserved                                 */
145   xPortPendSVHandler,               /*!< PendSV Handler                           */
146   xPortSysTickHandler,              /*!< SysTick Handler                          */
147
```

Szálak lehetséges állapotai



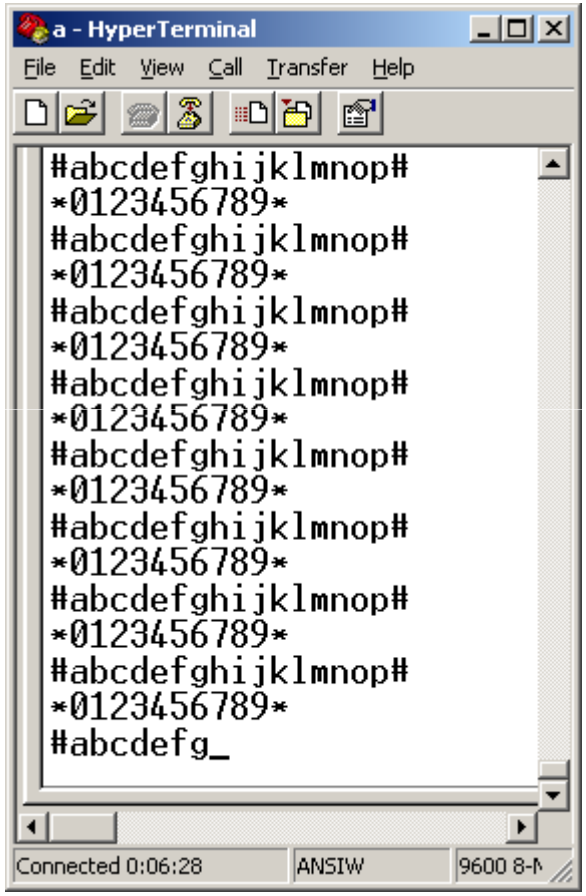
Szálak létrehozása, szálkezlés

- xTaskCreate()
  - Függvény pointer a task-ra
  - Task neve (a FreeRTOS nem használja)
  - Stack mérete szavakban
  - Task paraméterek
  - Prioritás
  - Task handle ami a létrejött taszkra mutat

```
xTaskCreate( vTaskA,
             ( signed portCHAR * ) "TASK_A",
             comSTACK_SIZE,
             NULL,
             1,
             ( xTaskHandle * ) NULL );
```

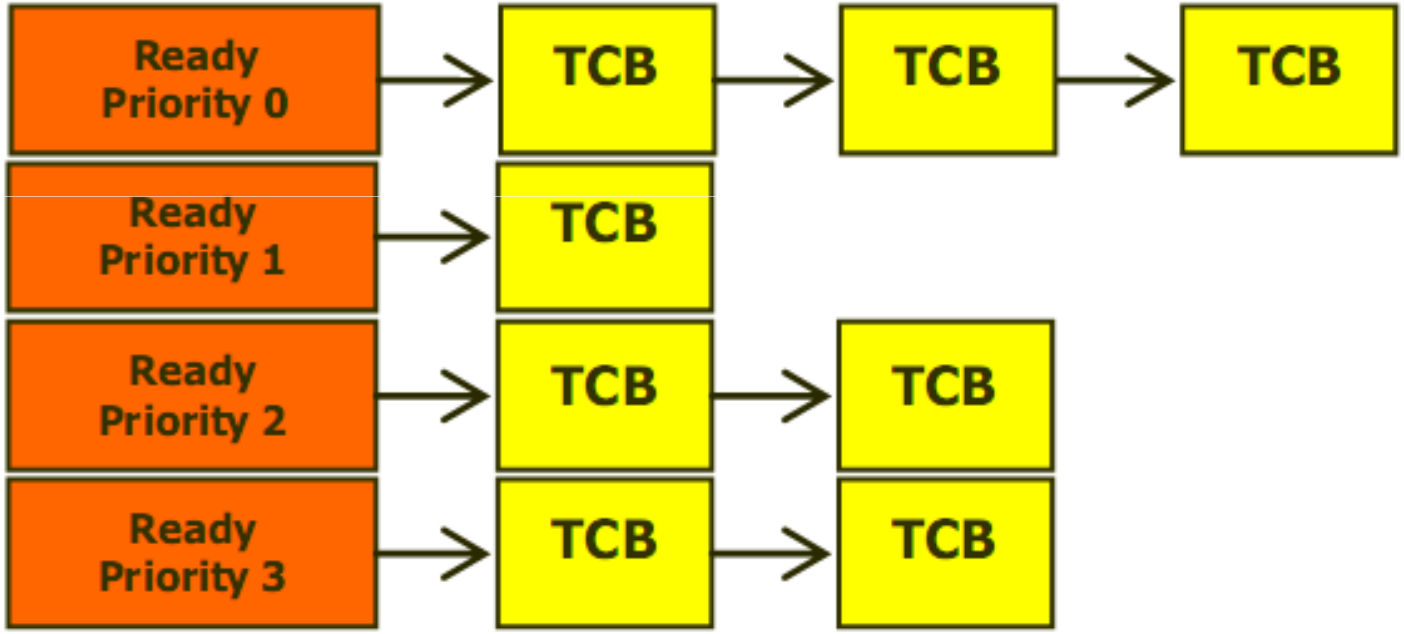
. példa két párhuzamos szál

- TaskA, TaskB
  - Fix periódus
- Toljuk el a TaskA kezdését picit
- Melyiknek nagyobb a priorítása?
  - Próbáljuk ki a configUSE_PREEMPTION konfiguráció opciót



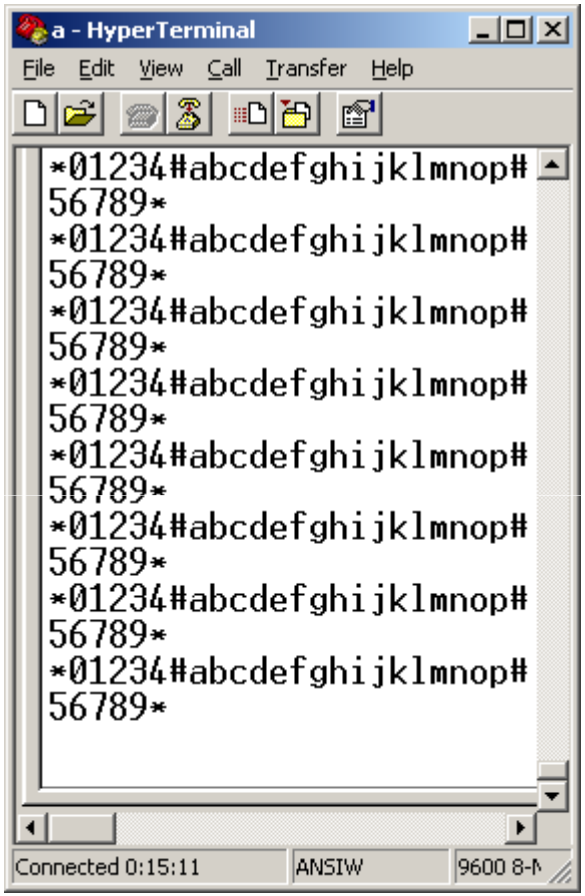
Priorítás kezelés ütemező

- Több soros preemtív ütemező



2. példa két delay közötti különbség

- Használjuk a vTaskDelayUntil -t



3. Közös előforrás probléma megoldása

- Gyorsítsuk fel a szálakat 100ms-re
- RTOS Kernel Control-ból a

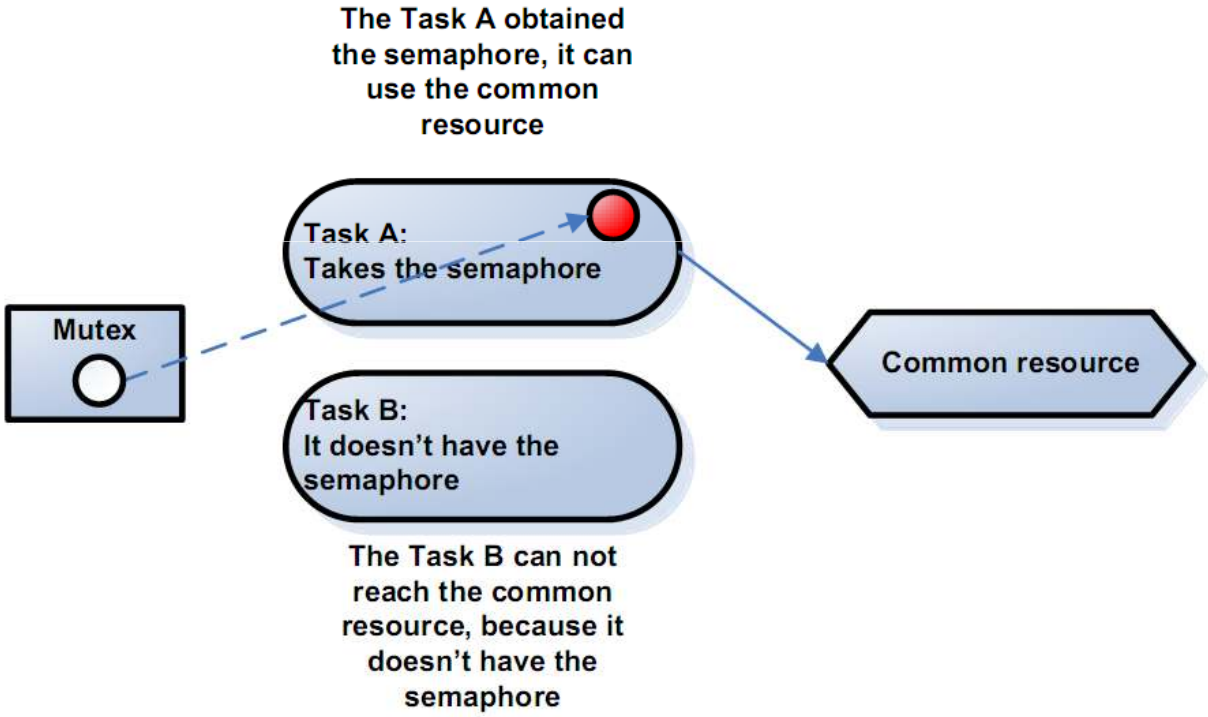
  taskENTER_CRITICAL

  taskEXIT_CRITICAL

  Használata
- Mi a gond?
  - Hozzunk létre egy gyors nagy priorítású szálat
    - 5 ms-est 2ms-el eltolva
  - Nyújtsuk meg a printf-et

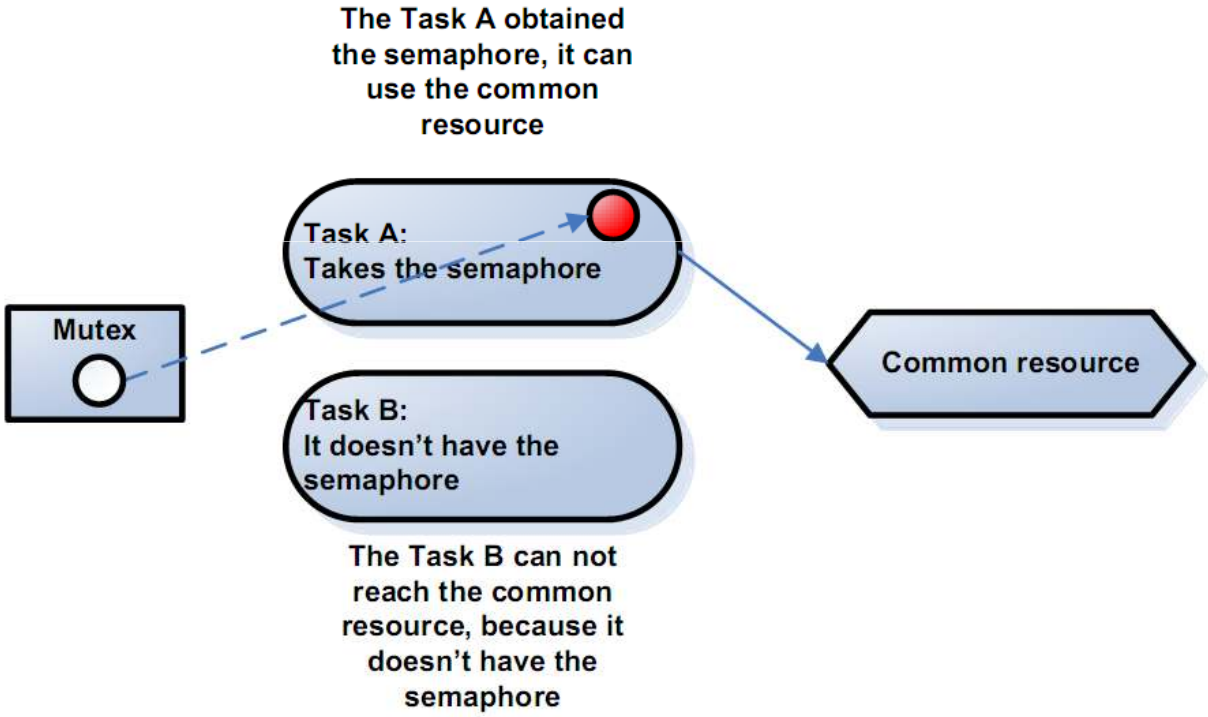
4. Mutex használata

- Hozzunk létre egy mutex-et és vizsgáljuk meg a hatását



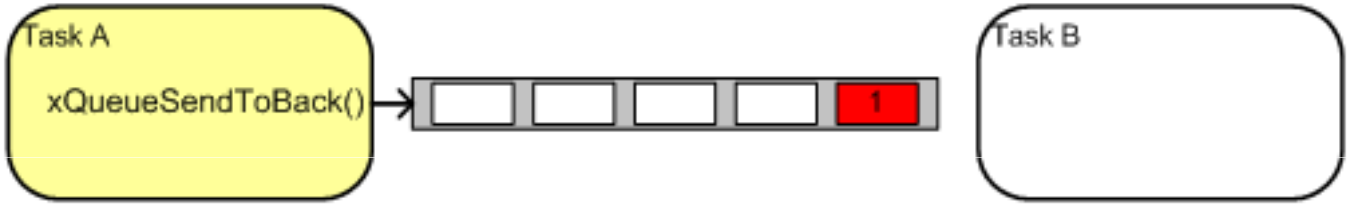
5. Mutex használata

- Legyen külön kiíratő függvény (handler)

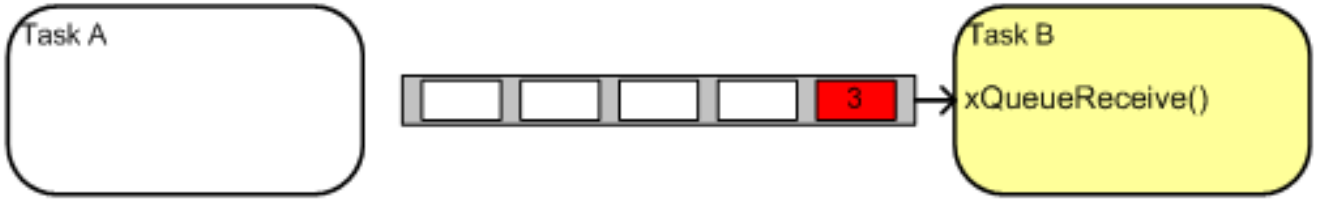


6. Queue használata

- Egy fogyasztó egy termelő megoldás
  - Termelés



  - Fogyasztás



7. Két termelő egy fogyasztó

- Két fogyasztó egy termelő megoldás
  - Tipikus megoldása a kijelző és más kommunikációs eszközök kezelésének

8. Szoftveres timer

- Oprendszer timer megoldások
- Callback függvény hívás
- Nem kötődnek közvetlenül a hardware-hez
  - Tetszőleges számú lehet
  - Valójában egy rendszer szál szolgálja ki a kéréseket
  - A heartbeat timerhez erősen kötődik
- Sok opció
  - Periódikus nem periódikus …
- OS konfigurálás
  - #define configUSE_TIMERS 1
  - #define configTIMER_TASK_PRIORITY configMAX_PRIORITIES - 1
  - #define configTIMER_QUEUE_LENGTH 5
  - #define configTIMER_TASK_STACK_DEPTH 256

FreeRTOS működés vizsgálat memória kezelés

Hogyan áll össze a memória

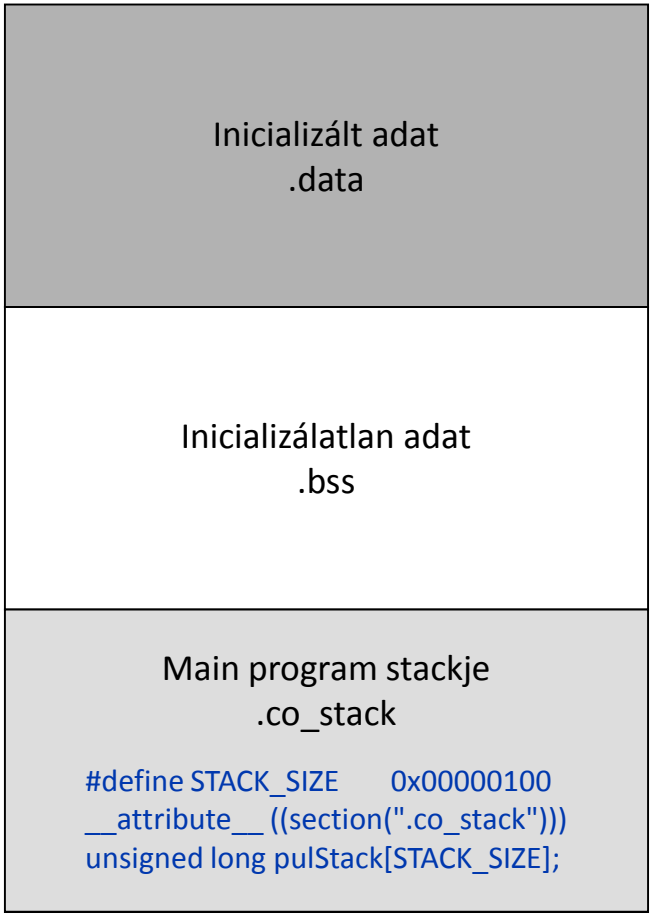
- Linker script ROM rész

```
OUTPUT_FORMAT ("elf32-littlearm", "elf32-bigarm", "elf32-littlearm")
/* Internal Memory Map*/
MEMORY
{
        rom (rx)  : ORIGIN = 0x08000000, LENGTH = 0x00040000
        ram (rwx) : ORIGIN = 0x20000000, LENGTH = 0x00010000
}

_eram = 0x20000000 + 0x00010000;
/* Section Definitions */
SECTIONS
{
   .text :
   {
      KEEP(*(.isr_vector .isr_vector.*))
      *(.text .text.* .gnu.linkonce.t.*)
      *(.glue_7t) *(.glue_7)
      *(.rodata .rodata* .gnu.linkonce.r.*)
   } > rom
```

Hogyan áll össze a memória

- Linker script RAM rész

```
.data : AT (_etext)
{
   _sdata = .;
   *(.data .data.*)
   . = ALIGN(4);
   _edata = . ;
} > ram

/* .bss section which is used for uninitialized data */
.bss (NOLOAD) :
{
   _sbss = . ;
   *(.bss .bss.*)
   *(COMMON)
   . = ALIGN(4);
   _ebss = . ;
} > ram

/* stack section */
.co_stack (NOLOAD):
{
   . = ALIGN(8);
   *(.co_stack .co_stack.*)
} > ram
```



Normál malloc használata

- Syscalls.c
  - A RAM terület végére

```
caddr_t _sbrk ( int incr )
{
  static unsigned char *heap = NULL;
  unsigned char *prev_heap;

  if (heap == NULL) {
    heap = (unsigned char *)&_end;
  }
  prev_heap = heap;

  heap += incr;

  return (caddr_t) prev_heap;
}
```

FreeRTOS memória használata

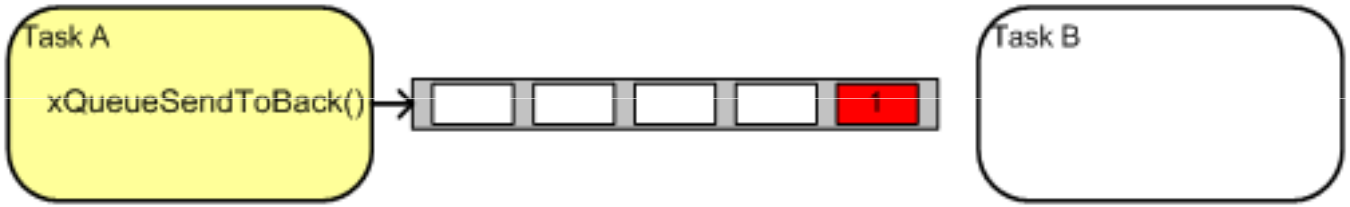
- Memmang stílus kiválasztása, illetve
  - heap_1: csak foglalás nincs felszabadítás
  - heap_2: foglalás és felszabadítás
  - heap_3: gyakorlatilag malloc, free thread safe-en

- FreeRTOS_config.h
  - #define configTOTAL_HEAP_SIZE((size_t) (30 * 1024))

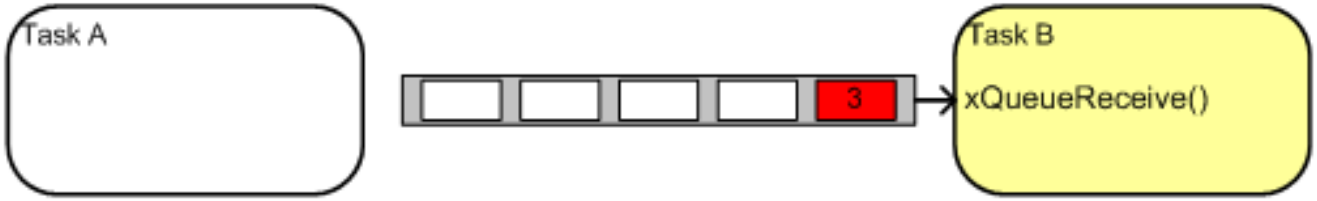
- Kezelő függvények
  - pvPortMalloc()
  - vPortFree()

1. példa Queue használata memória foglalással

- Egy fogyasztó egy termelő megoldás
  - Termelés



  - Fogyasztás



2. példa Stack méret monitorozás

- Mit éri meg monitorozni és mit?
- Stack water mark felkonfigurálása
- FreeRTOS_config beállítások
  - #define INCLUDE_uxTaskGetStackHighWaterMark0
  - uxTaskGetStackHighWaterMark(NULL);
- Miért nem számít, hogy hova rakjuk?

3. példa Stack eater megírása

- Mikor látszik a hatása?
- Mi a hatása

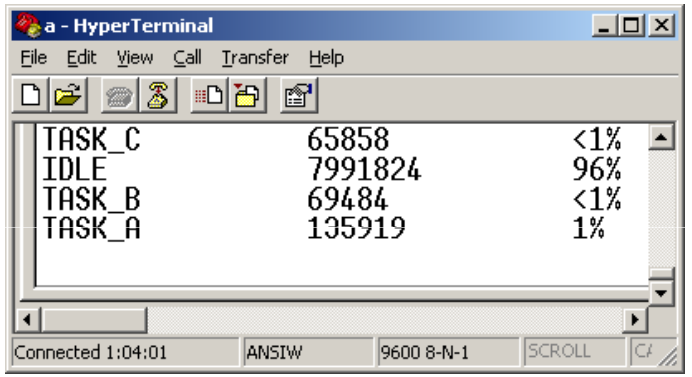
4. példa Stack overflow jelzés

- Hogy működik?
- FreeRTOS_config beállítások
  - #define configCHECK_FOR_STACK_OVERFLOW 2
- Implementálandó
  - vApplicationStackOverflowHook függvény létrehozása

FreeRTOS időzítési vizsgálatok

5. Run Time Stats

- Beépített műszerezés
  - Szükséges hozzá egy viszonylag nagy felbontású Timer
    - Hardware-es megvalósítás kell
- Kész megvalósítás
  - timertest.c
- FreeRTOS konfiguráció



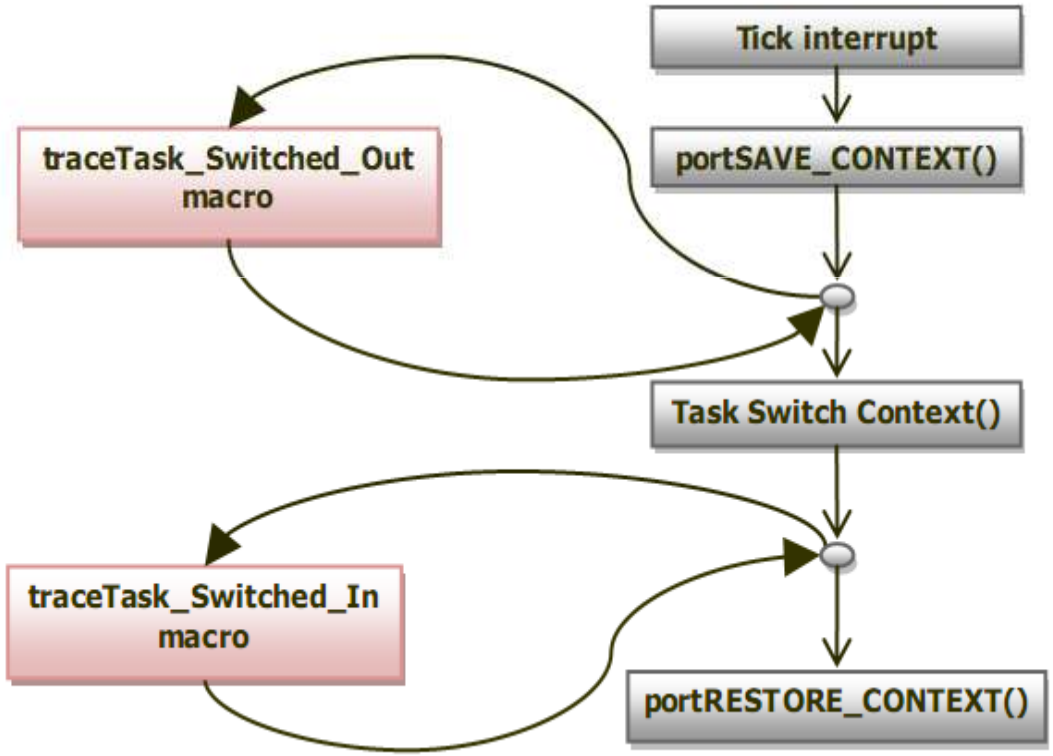

```
#define configGENERATE_RUN_TIME_STATS   1
```

- Megvalósítás

```
extern unsigned long ulRunTimeStatsClock;
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() ulRunTimeStatsClock = 0
#define portGET_RUN_TIME_COUNTER_VALUE() ulRunTimeStatsClock
```

Trace hookok

- Gyakorlatilag minden fontosabb belső lépéshez tartozik



6. Feladat

- Az aktuális task kirajzolása egy scope segítségével
- DAC használat: PA4-es láb
  - dac.c, dac.h
- FreeRTOS config
  - #define configUSE_APPLICATION_TASK_TAG 1

  - extern void my_dac_set_voltage(unsigned char dac);
  - #define traceTASK_SWITCHED_IN() my_dac_set_voltage(((int) pxCurrentTCB->pxTaskTag)*100)