```c
#include "sl_component_catalog.h"
#include "sl_system_init.h"
#include "app.h"
#if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
#include "sl_power_manager.h"
#endif
#if defined(SL_CATALOG_KERNEL_PRESENT)
#include "sl_system_kernel.h"
#else // SL_CATALOG_KERNEL_PRESENT
#include "sl_system_process_action.h"
#endif // SL_CATALOG_KERNEL_PRESENT
#include "em_device.h"
#include "em_chip.h"

/*******************************************************************************//**
 * Extern Includes for Lab04
 ******************************************************************************/
extern void task_A(), task_B(), task_C(), task_D();

extern void Yield();
extern void SysTick_Handler(void);


#define NUM_TASKS 5 // number of real-time tasks plus one

typedef struct
{
  uint32_t *stack_pointer;
  uint32_t ready_time;  // not used yet but will be later
  int32_t priority; // not used yet but may be later
} TaskControlBlock;

TaskControlBlock TCB[NUM_TASKS];

volatile TaskControlBlock *CurrentTask = TCB;
const volatile uint32_t SystemTick = 0; //SystemTick is found in context.s

// stack space for each task
uint32_t stack1[100];
uint32_t stack2[100];
uint32_t stack3[100];
uint32_t stack4[100];

//
// create a new task, set up the stack frame and mark it ready-to-go
//
void CreateTask(int task, void (*funct)(), void *stack, uint32_t stack_words, uint32_t
  priority, uint32_t ready_time)
{
  uint32_t *ptr = (uint32_t *)stack + (stack_words-1); // last byte of stack
  *ptr-- = 0x01000000; // xPSR, Thumb state only
  *ptr-- = (uint32_t)funct;
  for (int i=0; i<6; ++i) *ptr-- = 0; // lr, r12, r3, r2, r1, r0
  *ptr = -7; // exception link register
  for (int i=0; i<8; ++i) *--ptr = 0; // r11, r10, r9, r8, r7, r6, r5, r4
  TCB[task].stack_pointer = ptr;
  TCB[task].ready_time = ready_time;
```

```
 57   TCB[task].priority = priority;
 58 }
 59
 60 void vWaitUntil(int i)
 61 {
 62   CurrentTask->ready_time = i;
 63   Yield();
 64 }
 65
 66 void Task_A_Loop(void)
 67 {
 68   int release_time = 10; // release all tasks at t = 10
 69   while (1)
 70   {
 71     vWaitUntil(release_time);
 72     task_A();
 73     release_time += 3;
 74   }
 75 }
 76
 77 void Task_B_Loop(void)
 78 {
 79   int release_time = 10; // release all tasks at t = 10
 80   while (1)
 81   {
 82     vWaitUntil(release_time);
 83     task_B();
 84     release_time += 5;
 85   }
 86 }
 87
 88 void Task_C_Loop(void)
 89 {
 90   int release_time = 10; // release all tasks at t = 10
 91   while (1)
 92   {
 93     vWaitUntil(release_time);
 94     task_C();
 95     release_time += 6;
 96   }
 97 }
 98
 99 void Task_D_Loop(void)
100 {
101   int release_time = 10; // release all tasks at t = 10
102   while (1)
103   {
104     vWaitUntil(release_time);
105     task_D();
106     release_time += 10;
107   }
108 }
109
110 TaskControlBlock* scheduler()
111 {
112   int highest_priority = 0;
113   int highest_priority_task = 0;
```

```
114
115   for(int i=1; i<NUM_TASKS; i++)
116   {
117     if(TCB[i].priority < 1)
118     {
119       continue;
120     }
121     if(SystemTick >= TCB[i].ready_time)
122     {
123       if(TCB[i].priority > highest_priority)
124       {
125         highest_priority = TCB[i].priority;
126         highest_priority_task = i;
127       }
128     }
129   }
130   return TCB+highest_priority_task;
131 }
132
133 int idle_count = 0;
134
135 int main(void)
136 {
137   // Vendor function to work around bugs in some versions of the hardware
138   CHIP_Init();
139
140   SystemCoreClock = 14000000; // 14 MHz for this device
141
142   // configure 1ms timer tick
143   if (SysTick_Config(1*SystemCoreClock / 1000)) while (1) ;
144
145   // create the real-time tasks
146   CreateTask(1,Task_A_Loop,stack1,100,4,0);
147   CreateTask(2,Task_B_Loop,stack2,100,3,0);
148   CreateTask(3,Task_C_Loop,stack3,100,2,0);
149   CreateTask(4,Task_D_Loop,stack4,100,1,0);
150
151   /* Infinite loop for aperiodic and sporadic tasks */
152   while (1)
153   {
154   idle_count++;
155   }
156 }
157
```