```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(), task_E(), task_F();

extern void Yield();

int task_A_released(void); // returns true when Task A is released
int task_B_released(void); // returns true when Task B is released
int task_C_released(void);
int task_D_released(void);

//void yeild(void);

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

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

TaskControlBlock TCB[NUM_TASKS];

const volatile TaskControlBlock *CurrentTask = TCB;

//
// Called in an interrupt context to select next task to run
//

//
// 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 *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;
```

```
 58   TCB[task].suspend = 0;
 59   TCB[task].priority = 0;
 60 }
 61
 62 uint32_t stack1[100];
 63
 64 void Task_A_Loop(void)
 65 {
 66   while(1)
 67   {
 68       while( ! task_A_released()){Yield();}
 69       task_A();
 70   }
 71 }
 72
 73 uint32_t stack2[100];
 74
 75 void Task_B_Loop(void)
 76 {
 77   while(1)
 78   {
 79       while( ! task_B_released()){Yield();}
 80       task_B();
 81   }
 82 }
 83
 84 uint32_t stack3[100];
 85
 86 void Task_C_Loop(void)
 87 {
 88   while(1)
 89   {
 90       while( ! task_C_released()){Yield();}
 91       task_C();
 92   }
 93 }
 94
 95 uint32_t stack4[100];
 96
 97 void Task_D_Loop(void)
 98 {
 99   while(1)
100   {
101       while( ! task_D_released()){Yield();}
102       task_D();
103   }
104 }
105
106 TaskControlBlock* scheduler()
107 {
108   static int n = 0;
109   n++;
110   switch(n%4)
111     {
112     case 0:return TCB+1;
113     case 1:return TCB+2;
114     case 2:return TCB+3;
```

```
115     default:return TCB+4;
116     }
117 }
118
119 int idle_count = 0;
120
121 int main(void)
122 {
123   // Vendor function to work around bugs in some versions of the hardware
124   CHIP_Init();
125
126   SystemCoreClock = 14000000; // 14 MHz for this device
127
128   // configure 1ms timer tick
129   if (SysTick_Config(0.5*SystemCoreClock / 1000)) while (1) ;
130
131   // create the real-time tasks
132   CreateTask(1,Task_A_Loop,stack1,100);
133   CreateTask(2,Task_B_Loop,stack2,100);
134   CreateTask(3,Task_C_Loop,stack3,100);
135   CreateTask(4,Task_D_Loop,stack4,100);
136
137   /* Infinite loop for aperiodic and sporadic tasks */
138   while (1)
139   {
140   idle_count++;
141   }
142 }
143
```