```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;
```

```c
  TCB[task].priority = priority;
}

void vWaitUntil(int i)
{
  CurrentTask->ready_time = i;
  Yield();
}

void Task_A_Loop(void)
{
  int release_time = 10; // release all tasks at t = 10
  while (1)
  {
    vWaitUntil(release_time);
    task_A();
    release_time += 3;
  }
}

void Task_B_Loop(void)
{
  int release_time = 10; // release all tasks at t = 10
  while (1)
  {
    vWaitUntil(release_time);
    task_B();
    release_time += 5;
  }
}

void Task_C_Loop(void)
{
  int release_time = 10; // release all tasks at t = 10
  while (1)
  {
    vWaitUntil(release_time);
    task_C();
    release_time += 6;
  }
}

void Task_D_Loop(void)
{
  int release_time = 10; // release all tasks at t = 10
  while (1)
  {
    vWaitUntil(release_time);
    task_D();
    release_time += 10;
  }
}

TaskControlBlock* scheduler()
{
  int highest_priority = 0;
  int highest_priority_task = 0;
```

```c

  for(int i=1; i<NUM_TASKS; i++)
  {
    if(TCB[i].priority < 1)
    {
      continue;
    }
    if(SystemTick >= TCB[i].ready_time)
    {
      if(TCB[i].priority > highest_priority)
      {
        highest_priority = TCB[i].priority;
        highest_priority_task = i;
      }
    }
  }
  return TCB+highest_priority_task;
}

int idle_count = 0;

int main(void)
{
  // Vendor function to work around bugs in some versions of the hardware
  CHIP_Init();

  SystemCoreClock = 14000000; // 14 MHz for this device

  // configure 1ms timer tick
  if (SysTick_Config(1*SystemCoreClock / 1000)) while (1) ;

  // create the real-time tasks
  CreateTask(1,Task_A_Loop,stack1,100,4,0);
  CreateTask(2,Task_B_Loop,stack2,100,3,0);
  CreateTask(3,Task_C_Loop,stack3,100,2,0);
  CreateTask(4,Task_D_Loop,stack4,100,1,0);

  /* Infinite loop for aperiodic and sporadic tasks */
  while (1)
  {
  idle_count++;
  }
}

```