```c
/*****************************************************************************//**
*Lab 09: Semaphore Implementation
*****************************************************************************/
#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"

#include "segmentlcd.h"
#include "em_gpio.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

//create a struct for the semaphores
typedef struct
{
  int32_t id;
  int32_t count;
} semaphore;

typedef struct
{
  uint32_t *stack_pointer;
  uint32_t timer3_on;
  uint32_t task_mask;
  uint32_t ready_time;  // not used yet but will be later
  int32_t priority; // not used yet but may be later
  semaphore *smphr; //address of the semaphore
} TaskControlBlock;
TaskControlBlock TCB[NUM_TASKS];

//create the semaphores and initialize them to NULL like in lab08
semaphore semA;
semaphore semB;
semaphore semC;
semaphore semD;

semaphore semLCD; // semaphore for LCD display
semaphore semCD; // semaphore for CD
```

```
 58
 59 //use the same tick_count and prog variables from lab08
 60 int tick_count = 0;
 61 int prog = 0;
 62 //used to see what the timer3 is run at
 63 int time_measured = 0;
 64 int max_time = 0;
 65
 66 //use the same s and n variables from lab08
 67 char s[8]; // string for LCD display
 68 int n; // number for LCD display
 69
 70 volatile TaskControlBlock *CurrentTask = TCB;
 71 const volatile uint32_t SystemTick = 0; //SystemTick is found in context.s
 72
 73 // stack space for each task
 74 uint32_t stack1[100];
 75 uint32_t stack2[100];
 76 uint32_t stack3[100];
 77 uint32_t stack4[100];
 78
 79 void timer3_start()
 80 {
 81   TIMER3->CNT = 0; // reset timer to 0
 82   CurrentTask -> timer3_on = 1; // written to CMD when scheduled
 83   TIMER3->CMD = 1; // start the timer
 84 }
 85
 86 int timer3_stop()
 87 {
 88   TIMER3->CMD = 2; // stop timer
 89   CurrentTask -> timer3_on = 0;
 90   time_measured = TIMER3->CNT;
 91   return time_measured;
 92 }
 93
 94 void update_max_time(int time_measured)
 95 {
 96   if(time_measured > max_time)
 97   {
 98     max_time = time_measured;
 99   }
100 }
101
102 void SemaphoreCreate(semaphore *sem, int32_t id, int32_t count)
103 {
104   sem->id = id;
105   sem->count = count;
106 }
107
108 void give_semaphore(semaphore *sem)
109 {
110   __disable_irq();
111   if(sem->count > 0)
112   {
113     sem->count ++;
114   }
```

```
115   else
116   {
117     sem->count = 1;
118     int highest = 0;
119     int index = 0;
120     for(int i = 1; i < NUM_TASKS; i++)
121     {
122       if(TCB[i].smphr == sem && TCB[i].priority > highest)
123         {
124           highest = TCB[i].priority;
125           index = i;
126         }
127     }
128     if(index == 0)// There are three (3) different cases that could happen
129     {
130         __enable_irq();
131         return;
132     }
133     else if(TCB[index].priority > CurrentTask->priority)
134     {
135         TCB[index].smphr = 0;
136         __enable_irq(); //make sure this is before Yield();
137         Yield();
138         return;
139     }
140     else
141     {
142         TCB[index].smphr = 0;
143         __enable_irq();
144         return;
145     }
146   }
147 }
148
149 void take_semaphore(semaphore *sem)
150 {
151   __disable_irq();
152   if(sem->count > 0){sem->count--;}
153   else
154   {
155     CurrentTask->smphr = sem;
156     __enable_irq();
157     Yield();
158     //__disable_irq();
159     sem->count--;
160   }
161   __enable_irq();
162 }
163
164 // create a new task, set up the stack frame and mark it ready-to-go
165 void CreateTask(int task, void (*funct)(), void *stack, uint32_t stack_words, uint32_t
    priority, uint32_t ready_time)
166 {
167   uint32_t *ptr = (uint32_t *)stack + (stack_words-1); // last byte of stack
168   *ptr-- = 0x01000000; // xPSR, Thumb state only
169   *ptr-- = (uint32_t)funct;
170   for (int i=0; i<6; ++i) *ptr-- = 0; // lr, r12, r3, r2, r1, r0
```

```
171   *ptr = -7; // exception link register
172   for (int i=0; i<8; ++i) *--ptr = 0; // r11, r10, r9, r8, r7, r6, r5, r4
173   TCB[task].stack_pointer = ptr;
174   TCB[task].ready_time = ready_time;
175   TCB[task].priority = priority;
176   TCB[task].task_mask = 1<<(task-1); //shift
177 }
178
179 void vWaitUntil(int i)
180 {
181   CurrentTask->ready_time = i;
182   Yield();
183 }
184
185 void TaskA(void *params)
186 {
187   int release_time = 10; // release all tasks at t = 10
188   (void) params; // suppress warning
189   for(;;)
190   {
191     vWaitUntil(release_time);
192     task_A();
193     release_time += 1; //period of task A
194   }
195 }
196
197 void TaskB(void *params)
198 {
199   int release_time = 10; // release all tasks at t = 10
200   (void) params; // suppress warning
201   for(;;)
202   {
203     vWaitUntil(release_time);
204     task_B();
205     release_time += 25; //period of task B
206     take_semaphore(&semLCD); // wait for LCD semaphore
207     SegmentLCD_ARing(prog,0); // turn off previous segment
208     prog = (prog +1) & 7;
209     SegmentLCD_ARing(prog,1); // turn on next segment
210     give_semaphore(&semLCD); // give LCD semaphore
211   }
212 }
213
214 void TaskC(void *params)
215 {
216   int release_time = 10; // release all tasks at t = 10
217   (void) params; // suppress warning
218   for(;;)
219   {
220     vWaitUntil(release_time);
221     take_semaphore(&semCD);
222     task_C(s);
223     release_time += 7; //period of task C
224     give_semaphore(&semCD);
225     take_semaphore(&semLCD);
226     SegmentLCD_Write(s);
227     give_semaphore(&semLCD);
```

```
228   }
229 }
230
231 void TaskD(void *params)
232 {
233   int release_time = 10; // release all tasks at t = 10
234   (void) params; // suppress warning
235   for(;;)
236   {
237     vWaitUntil(release_time);
238     timer3_start();
239     take_semaphore(&semCD);
240     task_D(&n);
241     release_time += 10; //period of task D
242     give_semaphore(&semCD);
243     take_semaphore(&semLCD);
244     SegmentLCD_Number(n);
245     give_semaphore(&semLCD);
246   }
247 }
248
249 TaskControlBlock* scheduler()
250 {
251   int highest_priority = 0;
252   int highest_priority_task = 0;
253   for(int i=1; i<NUM_TASKS; i++)
254   {
255     if(TCB[i].priority < 1){continue;}
256     if(SystemTick >= TCB[i].ready_time && TCB[i].smphr == 0)
257     {
258       if(TCB[i].priority > highest_priority)
259       {
260         highest_priority = TCB[i].priority;
261         highest_priority_task = i;
262       }
263     }
264   }
265   return TCB+highest_priority_task;
266 }
267
268 int idle_count = 0; // used to count idle time
269
270 int main(void)
271 {
272   SystemCoreClock = 14000000; // 14 MHz for this device
273   // Vendor function to work around bugs in some versions of the hardware
274   CHIP_Init();
275   // enable clock to GPIO & TIMER3
276   CMU->HFPERCLKEN0 |= 0x2100;
277   // Divide by 8, 1.75 MHz count rate
278   TIMER3->CTRL = 0x03000000;
279
280   // set pins 0-5 to mode 4 (output)
281   GPIO->P[gpioPortD].MODEL = (GPIO->P[gpioPortD].MODEL & ~0x0FFFFFF) | 0x00444444;
282
283   // Initialize the LCD
284   SegmentLCD_Init(false);
```

```
285   // Write to the display
286   SegmentLCD_Number(1234);
287   SegmentLCD_Write("HELLO");
288
289   // create the semaphores
290
291   SemaphoreCreate(&semLCD, 1, 1); //semaphore &semLCD, id 1, count 1
292   SemaphoreCreate(&semCD, 2, 1); //semaphore &semCD, id 2, count 1
293
294   // configure 5ms timer tick
295   if (SysTick_Config(5*SystemCoreClock / 1000)) while (1) ;
296
297   // create the real-time tasks
298   CreateTask(1,TaskA,stack1,100,4,0);
299   CreateTask(2,TaskB,stack2,100,3,0);
300   CreateTask(3,TaskC,stack3,100,2,0);
301   CreateTask(4,TaskD,stack4,100,1,0);
302
303   /* Infinite loop for aperiodic and sporadic tasks */
304   while (1)
305   {
306   idle_count++;
307   }
308 }
309
310
```