

```

1/*****
2*Lab 09: Semaphore Implementation
3*****
4#include "sl_component_catalog.h"
5#include "sl_system_init.h"
6#include "app.h"
7#if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
8#include "sl_power_manager.h"
9#endif
10#if defined(SL_CATALOG_KERNEL_PRESENT)
11#include "sl_system_kernel.h"
12#else // SL_CATALOG_KERNEL_PRESENT
13#include "sl_system_process_action.h"
14#endif // SL_CATALOG_KERNEL_PRESENT
15#include "em_device.h"
16#include "em_chip.h"
17
18#include "segmentlcd.h"
19
20/*****
21 * Extern Includes for lab04
22 *****/
23extern void task_A(), task_B(), task_C(), task_D();
24
25extern void Yield();
26extern void SysTick_Handler(void);
27
28
29#define NUM_TASKS 5 // number of real-time tasks plus one
30
31//create a struct for the semaphores
32typedef struct
33{
34    int32_t id;
35    int32_t count;
36} semaphore;
37
38typedef struct
39{
40    uint32_t *stack_pointer;
41    uint32_t ready_time; // not used yet but will be later
42    int32_t priority; // not used yet but may be later
43    semaphore *smphr; //address of the semaphore
44} TaskControlBlock;
45TaskControlBlock TCB[NUM_TASKS];
46
47//create the semaphores and initialize them to NULL like in lab08
48semaphore semA;
49semaphore semB;
50semaphore semC;
51semaphore semD;
52
53semaphore semLCD; // semaphore for LCD display
54semaphore semCD; // semaphore for CD
55
56//use the same tick_count and prog variables from lab08
57int tick_count = 0;

```

```

58int prog = 0;
59
60//use the same s and n variables from lab08
61char s[8]; // string for LCD display
62int n; // number for LCD display
63
64volatile TaskControlBlock *CurrentTask = TCB;
65const volatile uint32_t SystemTick = 0; //SystemTick is found in context.s
66
67// stack space for each task
68uint32_t stack1[100];
69uint32_t stack2[100];
70uint32_t stack3[100];
71uint32_t stack4[100];
72
73void SemaphoreCreate(semaphore *sem, int32_t id, int32_t count)
74{
75    sem->id = id;
76    sem->count = count;
77}
78
79void give_semaphore(semaphore *sem)
80{
81    __disable_irq();
82    if(sem->count > 0)
83    {
84        sem->count ++;
85    }
86    else
87    {
88        sem->count = 1;
89        int highest = 0;
90        int index = 0;
91        for(int i = 1; i < NUM_TASKS; i++)
92        {
93            if(TCB[i].smphr == sem && TCB[i].priority > highest)
94            {
95                highest = TCB[i].priority;
96                index = i;
97            }
98        }
99        if(index == 0) // There are three (3) different cases that could happen
100        {
101            __enable_irq();
102            return;
103        }
104        else if(TCB[index].priority > CurrentTask->priority)
105        {
106            TCB[index].smphr = 0;
107            __enable_irq(); //make sure this is before Yield();
108            Yield();
109            return;
110        }
111        else
112        {
113            TCB[index].smphr = 0;
114            __enable_irq();

```

```

115    return;
116 }
117 }
118 }
119
120void take_semaphore(semaphore *sem)
121{
122    __disable_irq();
123    if(sem->count > 0){sem->count--;}
124    else
125    {
126        CurrentTask->smphr = sem;
127        __enable_irq();
128        Yield();
129        __disable_irq();
130        sem->count--;
131    }
132    __enable_irq();
133}
134
135// create a new task, set up the stack frame and mark it ready-to-go
136void CreateTask(int task, void (*func)(), void *stack, uint32_t stack_words, uint32_t
priority, uint32_t ready_time)
137{
138    uint32_t *ptr = (uint32_t *)stack + (stack_words-1); // last byte of stack
139    *ptr-- = 0x01000000; // xPSR, Thumb state only
140    *ptr-- = (uint32_t)func;
141    for (int i=0; i<6; ++i) *ptr-- = 0; // lr, r12, r3, r2, r1, r0
142    *ptr = -7; // exception link register
143    for (int i=0; i<8; ++i) *--ptr = 0; // r11, r10, r9, r8, r7, r6, r5, r4
144    TCB[task].stack_pointer = ptr;
145    TCB[task].ready_time = ready_time;
146    TCB[task].priority = priority;
147}
148
149void vWaitUntil(int i)
150{
151    CurrentTask->ready_time = i;
152    Yield();
153}
154
155void TaskA(void *params)
156{
157    int release_time = 10; // release all tasks at t = 10
158    (void) params; // suppress warning
159    for(;;)
160    {
161        vWaitUntil(release_time);
162        task_A();
163        release_time += 3; //period of task A
164    }
165}
166
167void TaskB(void *params)
168{
169    int release_time = 10; // release all tasks at t = 10
170    (void) params; // suppress warning

```

```

171    for(;;)
172    {
173        vWaitUntil(release_time);
174        task_B();
175        release_time += 125; //period of task B
176        take_semaphore(&semLCD); // wait for LCD semaphore
177        SegmentLCD_ARing(prog,0); // turn off previous segment
178        prog = (prog +1) & 7;
179        SegmentLCD_ARing(prog,1); // turn on next segment
180        give_semaphore(&semLCD); // give LCD semaphore
181    }
182}
183
184void TaskC(void *params)
185{
186    int release_time = 10; // release all tasks at t = 10
187    (void) params; // suppress warning
188    for(;;)
189    {
190        vWaitUntil(release_time);
191        take_semaphore(&semCD);
192        task_C(s);
193        release_time += 29; //period of task C
194        give_semaphore(&semCD);
195        take_semaphore(&semLCD);
196        SegmentLCD_Write(s);
197        give_semaphore(&semLCD);
198    }
199}
200
201void TaskD(void *params)
202{
203    int release_time = 10; // release all tasks at t = 10
204    (void) params; // suppress warning
205    for(;;)
206    {
207        vWaitUntil(release_time);
208        take_semaphore(&semCD);
209        task_D(&n);
210        release_time += 49; //period of task D
211        give_semaphore(&semCD);
212        take_semaphore(&semLCD);
213        SegmentLCD_Number(n);
214        give_semaphore(&semLCD);
215    }
216}
217
218TaskControlBlock* scheduler()
219{
220    int highest_priority = 0;
221    int highest_priority_task = 0;
222    for(int i=1; i<NUM_TASKS; i++)
223    {
224        if(TCB[i].priority < 1){continue;}
225        if(SystemTick >= TCB[i].ready_time && TCB[i].smphr == 0)
226        {
227            if(TCB[i].priority > highest_priority)

```

```
228     {
229         highest_priority = TCB[i].priority;
230         highest_priority_task = i;
231     }
232 }
233 }
234 return TCB+highest_priority_task;
235 }
236
237 int idle_count = 0; // used to count idle time
238
239 int main(void)
240 {
241     SystemCoreClock = 14000000; // 14 MHz for this device
242     // Vendor function to work around bugs in some versions of the hardware
243     CHIP_Init();
244     // Initialize the LCD
245     SegmentLCD_Init(false);
246     // Write to the display
247     SegmentLCD_Number(1234);
248     SegmentLCD_Write("HELLO");
249
250     // create the semaphores
251
252     SemaphoreCreate(&semLCD, 1, 1); // semaphore &semLCD, id 1, count 1
253     SemaphoreCreate(&semCD, 2, 1); // semaphore &semCD, id 2, count 1
254
255     // configure 1ms timer tick
256     if (SysTick_Config(1*SystemCoreClock / 1000)) while (1) ;
257
258     // create the real-time tasks
259     CreateTask(1, TaskA, stack1, 100, 4, 0);
260     CreateTask(2, TaskB, stack2, 100, 3, 0);
261     CreateTask(3, TaskC, stack3, 100, 2, 0);
262     CreateTask(4, TaskD, stack4, 100, 1, 0);
263
264     /* Infinite loop for aperiodic and sporadic tasks */
265     while (1)
266     {
267         idle_count++;
268     }
269 }
270
271
```