

```

1  /*
2  * FILENAME:    main.c
3  *
4  * DESCRIPTION: Simple OS for AVR Based Systems
5  *
6  * Created: 9/16/2012 8:56:02 PM
7  * Author: Casey Stark <starkca@msoe.edu>
8  *
9  * Demonstrates usage of ATMEGA32 simple OS. Operates 4 processes (threads)
10 * which currently outputs the process number to PORTB.
11 */
12
13 #include <avr/io.h>
14 #include <avr/interrupt.h>
15 #include "OS.h"
16
17 // Functions for each process
18 void p0(void);
19 void p1(void);
20 void p2(void);
21 void p3(void);
22
23
24 /*****
25 * Main Method. It Will:
26 *   1. Start OS
27 *   2. Spawn each new process
28 *   3. Execute each new process until system is shutdown
29 *****/
30 int main(void)
31 {
32     DDRB = 0xFF;
33     PORTB = 0x00;
34
35     startOS();
36
37     addProcess(p0);
38     addProcess(p1);
39     addProcess(p2);
40     addProcess(p3);
41
42     sei();
43
44     while(1)
45     {
46
47     }
48 }
49
50 /*****
51 * Process 0
52 *****/
53 void p0(void)
54 {
55     while(1)
56     {
57         PORTB = 0x00;
58     }
59 }
60 /*****
61 * Process 1
62 *****/
63 void p1(void)
64 {
65     while(1)
66     {

```

```
67     PORTB = 0x01;
68 }
69 }
70
71
72 /*****
73 * Process 2
74 *****/
75 void p2(void)
76 {
77     while(1)
78     {
79         PORTB = 0x02;
80     }
81 }
82
83
84 /*****
85 * Process 3
86 *****/
87 void p3(void)
88 {
89     while(1)
90     {
91         PORTB = 0x03;
92     }
93 }
```

```
1 /*
2  * FILENAME:    OS.h
3  *
4  * DESCRIPTION: Simple OS for AVR Based Systems
5  *
6  * Created: 9/16/2012 8:56:02 PM
7  * Author: Casey Stark <starkca@msoe.edu>
8  *
9  * Contains the interface for a very simple operating
10 * system for embedded system.
11 * Provides a very simple round robin based scheduler
12 * based upon the AVR Timer Counter 1
13 */
14
15 #ifndef OS_H_
16 #define OS_H_
17
18 // Stack size for each process
19 #define STACKSIZE 0x80
20
21 // Starting address of stack
22 #define STACKSTART 0x017F
23
24 // Maximum number of running processes
25 #define MAXPROCESSES 4
26
27 /*****
28  * Starts the operation of the OS. Process control blocks will be
29  * initialized, and calling process given process block 0
30  *****/
31 void startOS(void);
32
33 /*****
34  * Add new process to OS. Process in passed in as a void pointer
35  *
36  * PARAMETERS:
37  *   void* function: Function which is to be the new process
38  *
39  * RETURNS:
40  *   int8_t:      Assigned Process ID (pid) or -1 if process can not
41  *               be added
42  *****/
43 int8_t addProcess(void* function);
44
45 #endif /* OS_H_ */
```

```
1 /*
2 * FILENAME: OS.c
3 *
4 * DESCRIPTION: Simple OS for AVR Based Systems
5 *
6 * Created: 9/16/2012 8:56:02 PM
7 * Author: Casey Stark <starkca@msoe.edu>
8 *
9 * Contains the implementation for a very simple operating
10 * system for embedded system.
11 * Provides a very simple round robin based scheduler
12 * based upon the AVR Timer Counter 0
13 * Outputs the Process information to PORTB.
14 */
15
16 #include <avr/io.h>
17 #include <avr/interrupt.h>
18
19 #include "OS.h"
20
21 // Contains structure for a process control block.
22 // Includes:
23 // 1. Process ID
24 // 2. Priority setting
25 // 3. Number of times process ran
26 // 4. Process Stack Pointer
27 struct processStruct
28 {
29     uint8_t PID;
30     uint8_t priority;
31     uint8_t switchCount;
32     void* stackPtr;
33 };
34
35 // Holds the process control blocks (pcb) for the operating system.
36 // Each process gets their own pcb assigned to it
37 volatile struct processStruct pcbs[MAXPROCESSES];
38
39 // Holds the currently running process
40 volatile uint8_t currentProcess;
41
42 volatile static uint8_t firstRun;
43
44 // Tracks how many process are currently running
45 volatile uint8_t processCount;
46
47 // Private Function Prototypes
48 void enableTimer(void);
49
50
51 /*****
52 * Start up the OS. Initializes Process Control Blocks,
53 * calling process is assigned block 0.
54 *****/
55 void startOS(void)
56 {
57     firstRun = 1;
58
59     // Set current process index
60     currentProcess = 0;
61
62     // Initialize process counter
63     processCount = 0;
64
65     // Enable Timer Interrupt
66     enableTimer();
```

```

67 }
68
69
70 /*****
71 * Add new process to OS. Process in passed in as a void pointer
72 *
73 * PARAMETERS:
74 *   void* function: Function which is to be the new process
75 *
76 * RETURNS:
77 *   int8_t:          Assigned Process ID (pid) or -1 if process can not
78 *                   be added
79 *****/
80 int8_t addProcess(void* function)
81 {
82     uint8_t retVal = -1;
83
84     if(processCount < MAXPROCESSES)
85     {
86         // Setup PCB for new process
87         pcbs[processCount].stackPtr = (void*)(STACKSTART + (STACKSIZE * processCount));
88
89         // Place return to function on stack
90         *(uint8_t*)pcbs[processCount].stackPtr = (0x00FF & (uint16_t)function);
91         pcbs[processCount].stackPtr--;
92         *(uint8_t*)pcbs[processCount].stackPtr = (0xFF00 & (uint16_t)function) >> 8;
93         pcbs[processCount].stackPtr--;
94
95         // Pad the stack
96         pcbs[processCount].stackPtr = pcbs[processCount].stackPtr - 33;
97
98         // Initialize each process block items
99         pcbs[processCount].priority = processCount;
100        pcbs[processCount].switchCount = 0;
101        pcbs[processCount].PID = processCount + 1;
102
103        processCount = processCount + 1;
104        retVal = processCount;
105    }
106
107    return retVal;
108 }
109
110
111 /*****
112 * Enables Timer 1 as periodic timer. When Fired, scheduler will
113 * run to determine next process to execute
114 *
115 * PARAMETERS:
116 *   VOID
117 *
118 * RETURNS:
119 *   VOID
120 *****/
121 void enableTimer(void)
122 {
123     // Compare register to 0x04E2
124     OCR1A = 0x04E2;
125
126     // Enable Compare Interrupt
127     TIMSK |= 1 << OCIE1A;
128
129     // Start counting at 0
130     TCNT1 = 0x00;
131
132     // Timmer 1 for CTC with Prescaler of 256

```

```
133     TCCR1B |= (1 << WGM12) | (1 << CS12);
134 }
135
136
137 /*****
138 * Cause scheduler to run. Determine next process to execute
139 *****/
140 ISR(TIMER1_COMPA_vect, ISR_NAKED)
141 {
142     asm("cli");
143
144     // Store all registers to stack
145     asm("push r0");
146     asm("in r0, 0x3f");
147     asm("push r0");
148     asm("push r1");
149     asm("push r2");
150     asm("push r3");
151     asm("push r4");
152     asm("push r5");
153     asm("push r6");
154     asm("push r7");
155     asm("push r8");
156     asm("push r9");
157     asm("push r10");
158     asm("push r11");
159     asm("push r12");
160     asm("push r13");
161     asm("push r14");
162     asm("push r15");
163     asm("push r16");
164     asm("push r17");
165     asm("push r18");
166     asm("push r19");
167     asm("push r20");
168     asm("push r21");
169     asm("push r22");
170     asm("push r23");
171     asm("push r24");
172     asm("push r25");
173     asm("push r26");
174     asm("push r27");
175     asm("push r28");
176     asm("push r29");
177     asm("push r30");
178     asm("push r31");
179
180     if(!firstRun)
181     {
182         PORTB = 0xFF;
183         PORTB = 0x00;
184         PORTB = 0xFF;
185
186         pcbs[currentProcess].stackPtr = (void*)SP;
187
188         pcbs[currentProcess].switchCount = pcbs[currentProcess].switchCount + 1;
189
190         if(currentProcess == (MAXPROCESSES - 1))
191         {
192             // Wrap back to initial process
193             currentProcess = 0xFF;
194         }
195
196         currentProcess = currentProcess + 1;
197
198         SP = (uint16_t)pcbs[currentProcess].stackPtr;
```

```
199
200     // Output process Information as requested
201
202     PORTB = pcbs[currentProcess].PID;
203
204     PORTB = pcbs[currentProcess].priority;
205
206     PORTB = pcbs[currentProcess].switchCount;
207
208     PORTB = (uint8_t)(0x00FF & (uint16_t)pcbs[currentProcess].stackPtr);
209
210     PORTB = (0xFF00 & ((uint16_t)pcbs[currentProcess].stackPtr)) >> 8;
211
212     // Restore all of the registers to their previous state.
213     asm("pop r31");
214     asm("pop r30");
215     asm("pop r29");
216     asm("pop r28");
217     asm("pop r27");
218     asm("pop r26");
219     asm("pop r25");
220     asm("pop r24");
221     asm("pop r23");
222     asm("pop r22");
223     asm("pop r21");
224     asm("pop r20");
225     asm("pop r19");
226     asm("pop r18");
227     asm("pop r17");
228     asm("pop r16");
229     asm("pop r15");
230     asm("pop r14");
231     asm("pop r13");
232     asm("pop r12");
233     asm("pop r11");
234     asm("pop r10");
235     asm("pop r9");
236     asm("pop r8");
237     asm("pop r7");
238     asm("pop r6");
239     asm("pop r5");
240     asm("pop r4");
241     asm("pop r3");
242     asm("pop r2");
243     asm("pop r1");
244     asm("pop r0");
245     asm("out 0x3f, r0");
246     asm("pop r0");
247
248     asm("reti");
249 }
250
251 else
252 {
253     firstRun = 0;
254
255     SP = ((uint16_t)pcbs[currentProcess].stackPtr) + 33;
256
257     asm("reti");
258 }
259
260 }
```