```c
/*
 * FILENAME:    main.c
 *
 * DESCRIPTION: Simple OS for AVR Based Systems
 *
 * Created: 9/16/2012 8:56:02 PM
 *  Author: Casey Stark <starkca@msoe.edu>
 *
 * Demonstrates usage of ATMEGA32 simple OS. Operates 4 processes (threads)
 * which currently outputs the process number to PORTB.
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include "OS.h"

// Functions for each process
void p0(void);
void p1(void);
void p2(void);
void p3(void);


/*************************************************************************
* Main Method. It Will:
*   1. Start OS
*   2. Spawn each new process
*   3. Execute each new process until system is shutdown
*************************************************************************/
int main(void)
{
    // Watchdog Reset
    uint8_t watchdog = MCUCSR;
    watchdog = watchdog & (1 << WDRF);
    // Watchdog was triggered
    if(!(watchdog == 0))
    {
        DDRC = 0xFF;
        PORTC = 0xFE;
        MCUCSR |= 1 << WDRF;
        while(1);
    }

    // ADC Left Adjusts, Take sample from ADC5, AVCC is reference
    ADMUX |= (1 << ADLAR) | (1 << MUX2) | (1 << MUX0) | (1 << REFS0);

    // ADC clock = 125 kHz (64 Prescaler), Enable ADC
    ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1);

    // Set PORTB to all outputs
    DDRB = 0xFF;
    PORTB = 0x00;
```

```
    // Initialize the OS
    startOS();

    // Add each process to the OS
    addProcess(p0);
    addProcess(p1);
    addProcess(p2);
    addProcess(p3);

    sei();

    while(1)
    {
        // Sit here untill context switcher kicks in
    }
}

void powerLoss(void)
{
    TCCR1B &= ~(1 << CS12);

    WDTCR |= 1 << WDTOE;
    WDTCR &= ~(1 << WDE);

    DDRB |= 0xFF;
    PORTB = 0xFF;

    while(1);
}

/***************************************************************************/
/* Process 0                                                               */
/***************************************************************************/
void p0(void)
{
    while(1)
    {
        PORTB = 0x00;

        DDRA = 0x00;

        ADCSRA |= 1 << ADSC;
        while(!(ADCSRA & (1 << ADIF)));
        ADCSRA |= 1 << ADIF;

        uint8_t adcValLow = ADCL;
        uint8_t adcValHigh = ADCH;

        if(adcValLow < 0x8F)
            powerLoss();
    }
}
/***************************************************************************
```

```c
* Process 1
***************************************************************************/
void p1(void)
{
     // PORTA all inputs
     DDRA = 0x00;
     // AVCC as reference
     ADMUX |= (1 << REFS0);
     // Enable ADC, set prescaler to clk/128
     ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
    while(1)
    {
        // Start the conversion
        ADCSRA |= 1 << ADSC;
        // Wait for the conversion to complete
        while(!(ADCSRA & (1 << ADIF)));
        // Clear the complete conversion flag
        ADCSRA |= 1 << ADIF;

        // Grab the data
        uint8_t lowData = ADCL;
        uint8_t highData = ADCH;

        if(ADCL > 3)
            PORTB |= 1 << PB7;
        else
            PORTB &= ~(1 << PB7);
    }
}


/***************************************************************************
* Process 2
***************************************************************************/
void p2(void)
{
    while(1)
    {
        PORTB = 0x02;
    }
}


/***************************************************************************
* Process 3
***************************************************************************/
void p3(void)
{
    while(1)
    {
        PORTB = 0x03;
    }
}
```

```c
/*
 * FILENAME:    OS.h
 *
 * DESCRIPTION: Simple OS for AVR Based Systems
 *
 * Created: 9/16/2012 8:56:02 PM
 *  Author: Casey Stark <starkca@msoe.edu>
 *
 * Contains the interface for a very simple operating
 * system for embedded system.
 * Provides a very simple round robin based scheduler
 * based upon the AVR Timer Counter 1
 */

#ifndef OS_H_
#define OS_H_

// Stack size for each process
#define STACKSIZE 0x80

// Starting address of stack
#define STACKSTART 0x017F

// Maximum number of running processes
#define MAXPROCESSES 4

/*************************************************************************
* Starts the operation of the OS. Process control blocks will be
* initialized, and calling process given process block 0
*************************************************************************/
void startOS(void);

/*************************************************************************
* Add new process to OS. Process in passed in as a void pointer
*
* PARAMETERS:
*   void* function: Function which is to be the new process
*
* RETURNS:
*   int8_t:         Assigned Process ID (pid) or -1 if process can not
*                       be added
*************************************************************************/
int8_t addProcess(void* function);

#endif /* OS_H_ */
```

```c
/*
* FILENAME: OS.c
*
* DESCRIPTION:  Simple OS for AVR Based Systems
*
* Created: 9/16/2012 8:56:02 PM
*  Author: Casey Stark <starkca@msoe.edu>
*
* Contains the implementation for a very simple operating
* system for embedded system.
* Provides a very simple round robin based scheduler
* based upon the AVR Timer Counter 0
* Outputs the Process information to PORTB.
*/

#include <avr/io.h>
#include <avr/interrupt.h>

#include "OS.h"

// Contains structure for a process control block.
// Includes:
//  1. Process ID
//  2. Priority setting
//  3. Number of times process ran
//  4. Process Stack Pointer
struct processStruct
{
    uint8_t PID;
    uint8_t priority;
    uint8_t switchCount;
    void* stackPtr;
};

// Holds the process control blocks (pcb) for the operating system.
// Each process gets their own pcb assigned to it
volatile struct processStruct pcbs[MAXPROCESSES];

// Holds the currently running process
volatile uint8_t currentProcess;

volatile static uint8_t firstRun;

// Tracks how many process are currently running
volatile uint8_t processCount;

// How many times has the process switched
volatile uint8_t switchCounter;

// Private Function Prototypes
void enableTimer(void);
```

```c
/*************************************************************************
* Start up the OS. Initializes Process Control Blocks,
* calling process is assigned block 0.
*************************************************************************/
void startOS(void)
{
    firstRun = 1;

    // Set current process index
    currentProcess = 0;

    // Initialize process counter
    processCount = 0;

    // Enable Timer Interrupt
    enableTimer();
}


/*************************************************************************
* Add new process to OS. Process in passed in as a void pointer
*
* PARAMETERS:
*   void* function: Function which is to be the new process
*
* RETURNS:
*   int8_t:         Assigned Process ID (pid) or -1 if process can not
*                       be added
*************************************************************************/
int8_t addProcess(void* function)
{
    uint8_t retVal = -1;

    if(processCount < MAXPROCESSES)
    {
        // Setup PCB for new process
        pcbs[processCount].stackPtr = (void*)(STACKSTART + (STACKSIZE * processCount));

        // Place return to function on stack
        *(uint8_t*)pcbs[processCount].stackPtr = (0x00FF & (uint16_t)function);
        pcbs[processCount].stackPtr--;
        *(uint8_t*)pcbs[processCount].stackPtr = (0xFF00 & (uint16_t)function) >> 8;
        pcbs[processCount].stackPtr--;

        // Pad the stack
        pcbs[processCount].stackPtr = pcbs[processCount].stackPtr - 33;

        // Initialize each process block items
        pcbs[processCount].priority = processCount;
        pcbs[processCount].switchCount = 0;
        pcbs[processCount].PID = processCount + 1;
```

```
        processCount = processCount + 1;
        retVal = processCount;
    }

    return retVal;
}


/************************************************************************
* Enables Timer 1 as periodic timer. When Fired, scheduler will
* run to determine next process to execute
*
* PARAMETERS:
*   VOID
*
* RETURNS:
*   VOID
************************************************************************/
void enableTimer(void)
{
    // Compare register to 0x04E2
    OCR1A = 0x04E2;

    // Enable Compare Interrupt
    TIMSK |= 1 << OCIE1A;

    // Start counting at 0
    TCNT1 = 0x00;

    // Timmer 1 for CTC with Prescaler of 256
    TCCR1B |= (1 << WGM12) | (1 << CS12);
}


/************************************************************************
* Cause scheduler to run. Determine next process to execute
************************************************************************/
ISR(TIMER1_COMPA_vect, ISR_NAKED)
{
    asm("cli");

    // Store all registers to stack
    asm("push r0");
    asm("in r0, 0x3f");
    asm("push r0");
    asm("push r1");
    asm("push r2");
    asm("push r3");
    asm("push r4");
    asm("push r5");
    asm("push r6");
    asm("push r7");
    asm("push r8");
```

```
    asm("push r9");
    asm("push r10");
    asm("push r11");
    asm("push r12");
    asm("push r13");
    asm("push r14");
    asm("push r15");
    asm("push r16");
    asm("push r17");
    asm("push r18");
    asm("push r19");
    asm("push r20");
    asm("push r21");
    asm("push r22");
    asm("push r23");
    asm("push r24");
    asm("push r25");
    asm("push r26");
    asm("push r27");
    asm("push r28");
    asm("push r29");
    asm("push r30");
    asm("push r31");

    if(!firstRun)
    {
        PORTB = 0xFF;
        PORTB = 0x00;
        PORTB = 0xFF;

        switchCounter++;

        if(switchCounter < 800)
            asm("wdr");

        pcbs[currentProcess].stackPtr = (void*)SP;

        pcbs[currentProcess].switchCount = pcbs[currentProcess].switchCount + 1;

        if(currentProcess == (MAXPROCESSES - 1))
        {
            // Wrap back to initial process
            currentProcess = 0xFF;
        }

        currentProcess = currentProcess + 1;

        SP = (uint16_t)pcbs[currentProcess].stackPtr;

        // Output process Information as requested

        PORTB = pcbs[currentProcess].PID;
```

```
        PORTB = pcbs[currentProcess].priority;

        PORTB = pcbs[currentProcess].switchCount;

        PORTB = (uint8_t)(0x00FF & (uint16_t)pcbs[currentProcess].stackPtr);

        PORTB = (0xFF00 & ((uint16_t)pcbs[currentProcess].stackPtr)) >> 8;

        // Restore all of the registers to their previous state.
        asm("pop r31");
        asm("pop r30");
        asm("pop r29");
        asm("pop r28");
        asm("pop r27");
        asm("pop r26");
        asm("pop r25");
        asm("pop r24");
        asm("pop r23");
        asm("pop r22");
        asm("pop r21");
        asm("pop r20");
        asm("pop r19");
        asm("pop r18");
        asm("pop r17");
        asm("pop r16");
        asm("pop r15");
        asm("pop r14");
        asm("pop r13");
        asm("pop r12");
        asm("pop r11");
        asm("pop r10");
        asm("pop r9");
        asm("pop r8");
        asm("pop r7");
        asm("pop r6");
        asm("pop r5");
        asm("pop r4");
        asm("pop r3");
        asm("pop r2");
        asm("pop r1");
        asm("pop r0");
        asm("out 0x3f, r0");
        asm("pop r0");

        asm("reti");
    }

    else
    {
        firstRun = 0;

        SP = ((uint16_t)pcbs[currentProcess].stackPtr) + 33;
```

```
        switchCounter = 1;

        asm("reti");
    }

}
```