/\*

\* kernel\_preemptive.c

\*

\* Created: 4/17/2020 3:11:09 PM

\* Author: Nathan Potvin

\*/

#include <avr/interrupt.h>

#include "kernel.h"

#ifdef PREEMPTIVE

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Local function declarations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void \_\_attribute\_\_ ((naked)) save\_context();

void \_\_attribute\_\_ ((naked)) restore\_context();

void \_\_attribute\_\_ ((naked)) schedule();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ISR definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Timer 2 compare match A ISR

\*

\* Set to run once per millisecond. Increments system clock and decrements

\* delay counters for any delayed threads.

\*/

ISR(TIMER2\_COMPA\_vect)

{

// increment the compare match to the next millisecond

OCR2A += (*F\_CPU* / USEC\_PER\_MILLIS) / TIMER2\_PRESCALLER - 1;

// decrement delay counters and clear delay\_status bit for any threads

// where counter reaches zero

*uint8\_t* msk = 0x01;

for (*uint8\_t* i = 0; i < MAX\_THREADS; ++i)

{

if ((kernel\_data.schedule\_ctrl.delay\_status & msk)

&& !(--kernel\_data.schedule\_ctrl.delay\_ctrs[i]))

{

kernel\_data.schedule\_ctrl.delay\_status &= ~msk;

}

msk <<= 1;

}

// increment system clock

++kernel\_data.system\_time;

}

/\*

\* Timer 2 compare match B ISR.

\*

\* Set to trigger at the end of the current threads time slice.

\*/

\_\_attribute\_\_ ((naked)) ISR(TIMER2\_COMPB\_vect)

{

// invloke the scheduler at the save context entry point

asm volatile ("rjmp save\_context");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Kernel function definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Initializes the system timer and the timer to invoke the

\* scheduler preemptively.

\*/

void init\_system\_timer()

{

// initialize all needed registers

*uint8\_t* com2a = 0b00; // output pin is disconnected

*uint8\_t* com2b = 0b00; // output pin is disconnected

*uint8\_t* wgm2 = 0b000; // Normal mode

*uint8\_t* foc2a = 0b0;

*uint8\_t* foc2b = 0b0;

*uint8\_t* cs2 = PRESCALLER\_SELECT; // use prescaller selected dynamically

// at compile time

*uint8\_t* ocie2a = 0b1; // enable compare match A interrupt

*uint8\_t* ocie2b = 0b1; // enable compare match b interrupt

*uint8\_t* toie2 = 0b0; // disable overflow interrupt

TCCR2A = (com2a << COM2A0) | (com2b << COM2B0) | ((wgm2 & 0b11) << WGM20);

TCCR2B = (foc2a << FOC2A) | (foc2b << FOC2B)

| (((wgm2 & 0b100) >> 2) << WGM22) | (cs2 << CS20);

OCR2A = (*uint8\_t*) ((*F\_CPU* / USEC\_PER\_MILLIS) / TIMER2\_PRESCALLER - 1);

OCR2B = (*uint8\_t*) (TIME\_SLICE / TIMER2\_PRESCALLER - 1);

TIMSK2 = (ocie2a << OCIE2A) | (ocie2b << OCIE2B) | (toie2 << TOIE2);

// set timer to zero

kernel\_data.system\_time = 0;

}

/\*

\* Delays the current thread and invokes the scheduler.

\*/

void delay(*uint16\_t* delay\_millis)

{

// atomically set the delay counter and set the delay status bit.

*ATOMIC\_BLOCK*(*ATOMIC\_RESTORESTATE*)

{

kernel\_data.schedule\_ctrl.delay\_ctrs[kernel\_data.schedule\_ctrl.cur\_thread\_id] =

delay\_millis;

kernel\_data.schedule\_ctrl.delay\_status |=

kernel\_data.schedule\_ctrl.cur\_thread\_msk;

}

asm volatile ("jmp save\_context");

}

/\*

\* Locks the current thread preventing the scheduler from being invoked.

\*/

void lock()

{

// clear the timer compare match B interrupt mask

TIMSK2 &= ~(0b1 << OCIE2B);

}

/\*

\* Unlock the thread to allow the scheduler to be invoked preemptively.

\*/

void unlock()

{

// set the timer compare match B interrupt mask

TIMSK2 |= (0b1 << OCIE2B);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Local function definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Saves the current thread context.

\* Acts as an entry point for the scheduler.

\* This function enters scheduler after completion.

\*/

void \_\_attribute\_\_ ((naked)) save\_context()

{

// save general purpose registers to stack

asm volatile ("push r0\n\

push r1\n\

push r2\n\

push r3\n\

push r4\n\

push r5\n\

push r6\n\

push r7\n\

push r8\n\

push r9\n\

push r10\n\

push r11\n\

push r12\n\

push r13\n\

push r14\n\

push r15\n\

push r16\n\

push r17\n\

push r18\n\

push r19\n\

push r20\n\

push r21\n\

push r22\n\

push r23\n\

push r24\n\

push r25\n\

push r26\n\

push r27\n\

push r28\n\

push r29\n\

push r30\n\

push r31");

// save status register

asm volatile ("in r0, 0x3f\n\

push r0");

// save stack pointer

kernel\_data.thread\_ctrl\_tbl[kernel\_data.schedule\_ctrl.cur\_thread\_id].stack\_ptr =

\*STACK\_POINTER;

// jump to the scheduler

asm volatile ("rjmp schedule");

}

/\*

\* Restores the context of the current thread.

\* Is invoked after the scheduler runs.

\*/

void \_\_attribute\_\_ ((naked)) restore\_context()

{

// increment OCR2B to match on at the end of the next time slice

OCR2B = TCNT2 + (TIME\_SLICE / TIMER2\_PRESCALLER - 1);

// enable the TIMER2\_COMPB interrupt to allow for rescheduling

TIMSK2 |= 0b1<<OCIE2B;

// restore stack pointer

\*STACK\_POINTER =

kernel\_data.thread\_ctrl\_tbl[kernel\_data.schedule\_ctrl.cur\_thread\_id].stack\_ptr;

// restore status register

asm volatile ("pop r0\n\

out 0x3f, r0");

// restore general purpose registers

asm volatile ("pop r31\n\

pop r30\n\

pop r29\n\

pop r28\n\

pop r27\n\

pop r26\n\

pop r25\n\

pop r24\n\

pop r23\n\

pop r22\n\

pop r21\n\

pop r20\n\

pop r19\n\

pop r18\n\

pop r17\n\

pop r16\n\

pop r15\n\

pop r14\n\

pop r13\n\

pop r12\n\

pop r11\n\

pop r10\n\

pop r9\n\

pop r8\n\

pop r7\n\

pop r6\n\

pop r5\n\

pop r4\n\

pop r3\n\

pop r2\n\

pop r1\n\

pop r0");

// the next address on the stack should be the address of the next instruction

// to be executed in the current thread

asm volatile ("reti");

}

/\*

\* Verifies the stack canary and schedulers the next thread.

\* If no thread is enabled, the scheduler enters a sleep mode.

\*/

void \_\_attribute\_\_ ((naked)) schedule()

{

// disable TIMER2\_COMPB interrupt to prevent the scheduler from being invoked

// during sleep

TIMSK2 &= ~(0b1<<OCIE2B);

// verify canary

if (\*(kernel\_data.thread\_ctrl\_tbl[kernel\_data.schedule\_ctrl.cur\_thread\_id].canary\_ptr) != CANARY)

{

stack\_overflow();

}

// compute ready status and sleep if no threads are ready

*uint8\_t* ready\_status;

do

{

ready\_status = kernel\_data.schedule\_ctrl.disable\_status;

ready\_status |= kernel\_data.schedule\_ctrl.delay\_status;

ready\_status = ~ready\_status;

if (!ready\_status)

{

// enter the extended standby sleep mode if no threads are ready

SMCR = SLEEP\_MODE\_EXT\_STANDBY | (0b1 << SE);

*sei*();

asm volatile ("sleep");

*cli*();

}

} while (!ready\_status);

// schedule the next thread

do

{

kernel\_data.schedule\_ctrl.cur\_thread\_id =

(kernel\_data.schedule\_ctrl.cur\_thread\_id + 1) & 0x07;

asm volatile ("lsl %1\n\

clr r1\n\

adc %1, r1"

: "=r" (kernel\_data.schedule\_ctrl.cur\_thread\_msk)

: "r" (kernel\_data.schedule\_ctrl.cur\_thread\_msk));

} while (!(kernel\_data.schedule\_ctrl.cur\_thread\_msk & ready\_status));

// jump to restore the new current thread's context

asm volatile ("rjmp restore\_context");

}

#endif /\* PREEMPTIVE \*/