/\*

\* kernel\_cooperative.c

\*

\* Created: 4/17/2020 3:11:28 PM

\* Author: Nathan Potvin

\*/

#include "kernel.h"

#ifndef PREEMPTIVE

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

\* Local function declarations

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

void init\_system\_timer();

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)

{

// decrement each delay counter and clear delay\_status bit if 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;

}

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

\* Kernel function definitions

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

/\*

\* Delays the current thread by at least the given number of milliseconds.

\* The interrupt enable bit will be restored on the return of this function.

\*

\* delay\_millis: the minimum number of milliseconds for the thread to be

\* delayed

\*/

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

{

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

{

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

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

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

}

yield();

}

/\*

\* Disabled the specified thread blocking it from being scheduled by the

\* scheduler.

\* If the specified thread is the current running thread, the thread will

\* yield.

\*

\* tid: thread id of the thread to be disabled

\*/

void disable(*uint8\_t* tid)

{

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

{

kernel\_data.schedule\_ctrl.disable\_status |= 1<<tid;

if (kernel\_data.schedule\_ctrl.cur\_thread\_id == tid)

{

yield();

}

}

}

/\*

\* Saves the current thread's context then invokes the scheduler.

\*

\* Acts as an entry point to the scheduler.

\*/

void \_\_attribute\_\_ ((naked)) yield()

{

// push all callee save registers to stack

// any caller save registers should already be saved

asm volatile ("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 r28\n\

push r29");

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

\*STACK\_POINTER;

asm volatile ("rjmp schedule");

}

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

\* Local function definitions

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

/\*

\* Initializes the system timer using timer2.

\* Set up causes an interrupt to be triggered every millisecond

\*/

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 = 0b010; // CTC mode with TOP at OCRA

*uint8\_t* foc2a = 0b0;

*uint8\_t* foc2b = 0b0;

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

// at compile time

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

*uint8\_t* ocie2b = 0b0; // disable 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 = (*F\_CPU* / USEC\_PER\_MILLIS) / TIMER2\_PRESCALLER - 1;

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

// set timer to zero

kernel\_data.system\_time = 0;

}

/\*

\* Schedules the next thread using round robin scheduling and sleeps if

\* no threads are ready.

\* Any context must be saved when this function is called.

\* Will invoke the sleep function if no thread is ready and continue

\* to invoke it until one becomes ready.

\*/

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

{

// disable interrupts to schedule atomicly

*cli*();

// verify current thread stack 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\

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));

// restore the scheduled thread

\*STACK\_POINTER =

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

asm volatile ("pop r29\n\

pop r28\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");

// enable interrupts

*sei*();

asm volatile ("ret");

}

#endif /\* PREEMPTIVE \*/