/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INTERRUPTS.C \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

/\*

This module implements the device interrupt exception

handler. This module will process all the device interrupts, including In-

terval Timer interrupts, converting device interrupts into V operations on

the appropriate semaphores.

A device interrupt occurs when either a previously initiated I/O request completes

or when either a processor Local Timer or the Interval Timer makes a 0x0000.0000

⇒ 0xFFFF.FFFF transition.

The only 2 timers that can cause an interrupt are the interval timer and the processor local timer

Interval Timer - Generates an interrupt to signal the end of processes' time slices

Local Timer - Generate an interrupt at the end of each 100 milliseconds

We must determine which Interrupt line and device was turned on first

Line 0 - Multi-core (ignored)

Line 1 and 2 - Clocks (ignored)

Line 3 - Disk device

Line 4 - Tape device

Line 5 - Network device (ignored)

Line 6 - Printer device

Line 7 - Terminal device (transmit or recieve? Look at the status for transmit and depending on what it is you can tell transmit or recieve)

Highest priority interrupt goes to the lowest line number by examining the CAUSE register

For device number we go to LOW ORDER memory to find it

Once we have the line number and device number...

-Determine the address of that device's device register

-Index of the semaphore for that device

Treat the interrupt as a "v" operation on the device's semaphore

-increment the semaphore

-test if the semaphore value <= 0 this mean we removedBlocked

-p->p\_s.v0 = status of the Interrupt

softBlockCount--

insertProcQ(P,readyQ)

ACK the interrupt

-set the command field to ACK

Return control to the process that was executing at the time of the Interrupt

\*/

/\* ==== Written By Solomon G and Edwin Cervantes ==== \*/

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* #INCLUDE MODULES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../h/const.h"

#include "../h/types.h"

#include "../e/initial.e"

#include "../e/scheduler.e"

#include "../e/pcb.e"

#include "../e/asl.e"

#include "/usr/local/include/umps2/umps/libumps.e"

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GLOBAL VARIABLES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

extern int semdTable[SEMALLOC];

extern int softBlockCount;

extern pcb\_PTR currentProcess;

extern pcb\_PTR readyQueue;

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LOCAL FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

HIDDEN int getDevNumber(unsigned int\* bitmap);

HIDDEN void copyState(state\_PTR src, state\_PTR dest);

HIDDEN void exitHandler(cpu\_t startTime);

/\*

Parameters: N/A

Returns: N/A

\*/

void interruptHandler(){

cpu\_t startTime, endTime; /\*Used to find out how much time we spend in our IHandler\*/

unsigned int causeReg, deviceNumber, lineNumber; /\*Cause register, line, and device\*/

device\_PTR deviceReg; /\*device register\*/

int\* sema4; /\*Only used in line 3\*/

int i; /\*We need an index for the semaphore\*/

int statusOfReg;

state\_PTR oldIntArea = (state\_PTR) INTERRUPTOLDAREA; /\*Where we find information over the interrupt\*/

pcb\_PTR p; /\*PCB\*/

STCK(startTime); /\*start clock\*/

causeReg = oldIntArea -> s\_cause; /\*Find out what was in the cause register that made the interrupt\*/

causeReg = (causeReg & IM) >> 8;

lineNumber = 0;

/\*Checking to see where the interrupt occured\*/

/\*This should never happen in Kaya OS. Invoke PANIC()\*/

if((causeReg & FIRST) != 0){/\*Multi-Core\*/

PANIC();

}

else if ((causeReg & SECOND) != 0){/\* Clock. Time over, invoke scheduler \*/

exitHandler(startTime);

}

else if ((causeReg & THIRD) != 0){/\*Clock\*/

LDIT(THOUSANDMS); /\*Loading 1000ms\*/

sema4 = (int\*) &(semdTable[PSEUDOCLOCK]); /\*Lets get to unblocking the sema4\*/

while(headBlocked(sema4) != NULL){

p = removeBlocked(sema4);

STCK(endTime);

if(p != NULL){

insertProcQ(&readyQueue, p); /\* Inserting \*/

(p->p\_time) = (p->p\_time) + (endTime - startTime); /\* Time stuff \*/

softBlockCount = softBlockCount - 1;

}

}

\*(sema4) = 0;

exitHandler(startTime);

}

else if ((causeReg & FOURTH) != 0){/\*Disk\*/

lineNumber = DISKNUM;

}

else if ((causeReg & FIFTH) != 0){/\*Tape\*/

lineNumber = TAPENUM;

}

else if ((causeReg & SIXTH) != 0){/\*Network\*/

lineNumber = NETWORKNUM;

}

else if ((causeReg & SEVENTH) != 0){/\*Printer\*/

lineNumber = PRINTERNUM;

}

else if ((causeReg & EIGHTH) != 0){/\*Terminal\*/

lineNumber = TERMINT;

} else {

PANIC();

}

/\*Fetch device number. Subtract 3 since first 3 devices do not count\*/

deviceNumber = getDevNumber((unsigned int\*)(INTBIT + ((lineNumber - THREE) \* LENGTHWORD)));

/\*Do we even need an interrupt\*/

if (deviceNumber == INTNOTNEEDED){

PANIC();

}

/\*Lets go get the register\*/

deviceReg = (device\_PTR) (INTDEV +((lineNumber - THREE) \* 0x80) + (deviceNumber \* 0x10));

/\*If not a terminal\*/

if(lineNumber != 7){

statusOfReg = deviceReg -> d\_status; /\*Store away device status\*/

deviceReg -> d\_command = ACK; /\*Send ACK message\*/

i = EIGHT \* (lineNumber - THREE) + deviceNumber; /\*Fidning the semaphore we need to perform a V opertaion on\*/

}else{ /\*i.e.: We have a terminal interrupt. Write on 3, 4, and 5, read otherwise\*/

int transm = (deviceReg -> t\_transm\_status & PRIVATEINSTUC);

if(transm == 3 || 4 || 5){

i = (EIGHT \*(lineNumber - THREE)) + deviceNumber; /\*Get index\*/

statusOfReg = deviceReg -> t\_transm\_status; /\*store away status\*/

deviceReg -> t\_transm\_command = ACK; /\*ACK it\*/

}else{

i = (EIGHT \*(lineNumber - THREE + 1) + deviceNumber); /\*Get index\*/

statusOfReg = deviceReg -> t\_recv\_status; /\*store away status. When recieving\*/

deviceReg -> t\_recv\_command = ACK; /\*ACK it\*/

}

}

/\*Finally time to V the semaphore we want\*/

sema4 =&(semdTable[i]);

(\*sema4) = (\*sema4) + 1;

/\*wake up\*/

if((\*sema4) <= 0){

p = removeBlocked(sema4);

if(p != NULL){

p -> p\_state.s\_v0 = statusOfReg;

insertProcQ(&readyQueue, p);

softBlockCount = softBlockCount - 1;

}

} else {

}

exitHandler(startTime);

}

/\*

Parameters: uint(bitmap)

Returns: int

\*/

HIDDEN int getDevNumber(unsigned int\* bitMap){

    unsigned int alarm = \*bitMap;

    if((alarm & FIRST) != 0) {

        return 0;

    }

    else if((alarm & SECOND) != 0){

        return 1;

    }

    else if((alarm & THIRD) != 0){

        return 2;

    }

    else if((alarm & FOURTH) != 0){

        return 3;

    }

    else if((alarm & FIFTH) != 0){

        return 4;

    }

    else if((alarm & SIXTH) != 0){

        return 5;

    }

    else if((alarm & SEVENTH) != 0){

        return 6;

    }

    else if((alarm & EIGHTH) != 0){

        return 7;

    }

return -1;

}

/\*Finally time to exit the handler

We are not charge the current process for time used in the handler

Return to scheduler

Goodbye

\*/

/\*

Parameters: cpu\_t

Returns: N/A

\*/

HIDDEN void exitHandler(cpu\_t startTime){

cpu\_t endTime;

state\_PTR inter = (state\_PTR) INTERRUPTOLDAREA;

if(currentProcess != NULL){ /\*Check to see if we're waiting. Boring\*/

STCK(endTime);

TODStart = TODStart + (endTime - startTime); /\*Charge time used in handler and add to TOD\*/

copyState(inter, &(currentProcess -> p\_state));

insertProcQ(&readyQueue, currentProcess);

}

scheduler();

}

HIDDEN void copyState(state\_PTR src, state\_PTR dest) {

int i;

/\* id \*/

dest->s\_asid = src->s\_asid;

/\* register cause \*/

dest->s\_cause = src->s\_cause;

/\* pc \*/

dest->s\_pc = src->s\_pc;

/\* status register \*/

dest->s\_status = src->s\_status;

/\* each register \*/

for (i=0; i < STATEREGNUM; i++){

dest->s\_reg[i] = src->s\_reg[i];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END OF ITERRUPTS.C \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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