EE290O
Homework 1
Code Review

Shannon Zelinski
and
Chris Cortopassi

Portable and Complicated Embedded Machine Code - not working perfectly

```
//#include<stdio.h>
//#include<unistd.h>

#include <conio.h>
#include <dsound.h>
#include <dsensor.h>


//_____________________________________________________________
//DATA STRUCTURES

struct trigger
{
	float fCompareValue;
	int (*pPredicate)(void); //returns whether trigger should fire
	void (*pActivate)(void); //initializes fCompareValue
};

struct port
{
	float (*pGetValue)(void);
	void  (*pSetValue)(float);
};

struct task
{
	float fInputValue, fOutputValue;
	struct port  inputPort, outputPort;
	void (*pRun)(void);    //computes on fInputValue an puts output in
fOutputValue
};

struct driver   //link between two ports
{
	struct port* pInputPort;
	struct port* pOutputPort;
};
```

```
//_____________________________________________________________
struct trigger_node    //a trigger address pair.  The trigger q is made up of these.
{
        struct trigger *pTrigger;
        int iECodeAddress;
};

#define MAX_TRIGGERS 10
struct trigger_node aTriggerList[MAX_TRIGGERS];      //the trigger q.
//a trigger is activated if pTrigger isn't NULL


#define CALL 0
#define SCHEDULE 1
#define FUTURE 2
#define RETURN 3

#define OPCODE 0
#define ARG1 1
#define ARG2 2

void EmbeddedMachine(int aiProgram[][3])
{
        int iPC = 0, i = 0, iTrigger = 0;
        int bFoundReturn = 0;

        for(i = 0; i < MAX_TRIGGERS; i++) //search for an enabled trigger
        {
                if(aTriggerList[i].pTrigger != 0) //trigger was activated
                {
                        if(aTriggerList[i].pTrigger->pPredicate()) //and trigger is
enabled
                        {
                                for(iPC = aTriggerList[i].iECodeAddress;
!bFoundReturn; iPC++) //start at the specified E Code address and keep going
until we hit a RETURN instruction
                                {
                                        switch(aiProgram[iPC][OPCODE])
                                        {
                                        case CALL: //driver
                                                ((struct driver*) aiProgram[iPC][ARG1])-
>pOutputPort->pSetValue(((struct driver*) aiProgram[iPC][ARG1])->pInputPort-
>pGetValue());
                                                break;

                                        case SCHEDULE: //task execi
```

```
                                        execi( ((struct task*)
aiProgram[iPC][ARG1])->pRun, 0, 0, PRIO_LOWEST,
DEFAULT_STACK_SIZE); //LEGOS
                                        break;

                                case FUTURE: //add to trigger list
                                        for(iTrigger = 0; iTrigger <
MAX_TRIGGERS; iTrigger++)
                                        {
                                                if(aTriggerList[iTrigger].pTrigger
== 0) //found one
                                                        break;  //get out of for loop
                                        }

                                        aTriggerList[iTrigger].pTrigger = (struct
trigger*) aiProgram[iPC][ARG1]; //apTriggers[aiProgram[iPC][ARG1]];
//remember the trigger
                                        aTriggerList[iTrigger].iECodeAddress =
aiProgram[iPC][ARG2]; //remember the E code address
                                        aTriggerList[iTrigger].pTrigger-
>pActivate(); //activate the trigger (updates the compare value)
                                        break;

                                case RETURN:
                                        bFoundReturn = 1; //get out of for loop
                                        break;

                                default:
//                                      printf("ERROR:  invalid opcode\n");
                                }
                        }

                        aTriggerList[i].pTrigger = 0; //remove trigger from list
                        break; //get out of for loop
                }
        }
    }
}
```

```
//_____________________________________________________________
//TRIGGERS

struct trigger clockTrigger;

int ClockTriggerPredicate(void)
{
	int iTime = sys_time;

	return(iTime >= clockTrigger.fCompareValue);
}

void ClockTriggerActivate(void)
{
	clockTrigger.fCompareValue = clockTrigger.fCompareValue + 500;
}

//_____________________________________________________________
//PORTS

struct port touchSensorPort;

float TouchSensorPortGetValue(void)
{
	return(TOUCH_1); //LEGOS return current state of touch sensor
}



static const note_t sound[] = { { PITCH_D4,  1 }, { PITCH_END, 0 } };   //LEGOS

struct port beepPort;

void BeepPortSetValue(float fArg) //emit a beep if we're told to
{
	if(fArg)
		dsound_play(sound); //LEGOS
}


struct port lightSensorPort;

float LightSensorPortGetValue(void)
{
	return(LIGHT_2); //LEGOS return current state of light sensor
}
```

```
struct port lcdPort;

void LcdPortSetValue(float fArg) //put the given integer value on the LCD
{
	lcd_int((int) fArg); //LEGOS printf("%f\n", fArg);
}

//_____________________________________________________________
//TASKS

struct task beepTask;

void BeepTaskInputSetValue(float fValue)
{
	beepTask.fInputValue = fValue;
}

float BeepTaskOutputGetValue(void)
{
	return(beepTask.fOutputValue);
}

void BeepTaskRun(void)
{
	beepTask.fOutputValue = !beepTask.fInputValue; //beep if touch sensor
isn't pressed
}


struct task lightTask;

void LightTaskInputSetValue(float fValue)
{
	lightTask.fInputValue = fValue;
}

float LightTaskOutputGetValue(void)
{
	return(lightTask.fOutputValue);
}

void LightTaskRun(void)
{
	lightTask.fOutputValue += (lightTask.fInputValue - 80) / 10; //time integrate
the light
}
```

```
//_____________________________________________________________
//DRIVERS

struct driver touch2BeepTaskDriver;
struct driver beepTask2BeepDriver;

struct driver light2LightTaskDriver;
struct driver lightTask2LcdDriver;

//_____________________________________________________________

int aiMyProgram[][3] =
{
	{CALL,		(int) &touch2BeepTaskDriver,	-1}, //0
	{CALL,		(int) &beepTask2BeepDriver,		-1},
	{CALL,		(int) &light2LightTaskDriver,	-1},
	{CALL,		(int) &lightTask2LcdDriver,	-1},
	{SCHEDULE,	(int) &beepTask,			-1},
	{SCHEDULE,	(int) &lightTask,			-1},
	{FUTURE,	(int) &clockTrigger,		8},
	{RETURN,	-1,					-1},

	{CALL,		(int) &light2LightTaskDriver,	-1}, //8
	{CALL,		(int) &lightTask2LcdDriver,	-1},
	{SCHEDULE,	(int) &lightTask,			-1},
	{FUTURE,	(int) &clockTrigger,		0},
	{RETURN,	-1,					-1}
};

void EMachineThread(void)
{
	while(1)
	{
		EmbeddedMachine(aiMyProgram);
		msleep(10);
	}
}

int main(int argc, char** argv)
{
	int iTime = sys_time;

	//INITIALIZE TRIGGERS
	clockTrigger.pPredicate = ClockTriggerPredicate;
	clockTrigger.pActivate  = ClockTriggerActivate;
	clockTrigger.fCompareValue = iTime;
```

```
	//INITIALIZE PORTS
	touchSensorPort.pGetValue = TouchSensorPortGetValue;
	beepPort.pSetValue = BeepPortSetValue;

	lightSensorPort.pGetValue = LightSensorPortGetValue;
	lcdPort.pSetValue = LcdPortSetValue;

	//INITIALIZE TASKS
	beepTask.inputPort.pSetValue = BeepTaskInputSetValue;
	beepTask.outputPort.pGetValue = BeepTaskOutputGetValue;
	beepTask.pRun = BeepTaskRun;

	lightTask.inputPort.pSetValue = LightTaskInputSetValue;
	lightTask.outputPort.pGetValue = LightTaskOutputGetValue;
	lightTask.pRun = LightTaskRun;

	//INITIALIZE DRIVERS
	touch2BeepTaskDriver.pInputPort = &touchSensorPort;
	touch2BeepTaskDriver.pOutputPort = &(beepTask.inputPort);

	beepTask2BeepDriver.pInputPort = &(beepTask.outputPort);
	beepTask2BeepDriver.pOutputPort = &beepPort;

	light2LightTaskDriver.pInputPort = &lightSensorPort;
	light2LightTaskDriver.pOutputPort = &(lightTask.inputPort);

	lightTask2LcdDriver.pInputPort = &(lightTask.outputPort);
	lightTask2LcdDriver.pOutputPort = &lcdPort;

	ds_active(&SENSOR_2); //LEGOS

	aTriggerList[0].pTrigger = &clockTrigger; //put a trigger in the q to get
started
	aTriggerList[0].iECodeAddress = 0;

   if(-1 == execi(EMachineThread, 0, 0, PRIO_NORMAL,
DEFAULT_STACK_SIZE)); //LEGOS
		cputs("err");

/*	while(1)
	{
		EmbeddedMachine(aiMyProgram);
	}			*/

	return(0);
}
```

More Simple Non Portable Embedded Machine Code - it works

```
// embedded machine

#include <conio.h>
#include <unistd.h>
#include <dsensor.h>
#include <dsound.h>
#include <dmotor.h>

#define b1              1
#define b2              2
#define print  2
#define beep            1

volatile time_t trigger_compare;
volatile time_t clock_time;
static note_t sound[2]={{PITCH_D4 , 1}, {PITCH_END, 0}};
int sound_toggle;
int count;
int atomic = b1;
int enable_task[3];
```

```
//*********
//  TASKS
//*********

//*******************************
//              function: beep_task
//              toggles beep pitches
//*******************************
void beep_task(void)
{
        switch(sound_toggle)                            //sound_toggle is used to
toggle the sound pitch
        {
                case 1:
                        sound[0].pitch = PITCH_A4;
                        sound_toggle = 2;
                break;
                case 2:
                        sound[0].pitch = PITCH_D4;
                        sound_toggle = 1;
                break;
                default:
                break;
        }
        enable_task[beep] = 0;                          //once the task is
complete, it is disabled
}                                                                       //only the
scheduler can reenable a task

//*******************************
//              function print_task
//              increments count to be printed
//*******************************
void print_task(void)
{
        count = count + 1;                              //increment counter
        enable_task[print] = 0;                         //disable task
}
```

```
//**********************************
//                EMBEDDED MACHINE FUNCTIONS
//**********************************

//**********************************
//                function: call
//                int driver: print or beep                //call drivers
//**********************************
void call(int driver)
{
        switch(driver)
        {
                case beep:
                        dsound_play(sound);                                        //beep
driver
                break;

                case print:
                        lcd_int(count);                                            //print
driver
                break;

                default:
                break;
        }
}

//**********************************
//                function: schedule
//                int task: print or beep
//**********************************
void schedule(int task)
{
          enable_task[task] = 1;                                  //enable task
}

//**********************************
//                function: future
//                int emnumber: b1 or b2
//**********************************
void future(int emnumber)
{
        atomic = emnumber;
        //choose future atomic block
        trigger_compare = trigger_compare + 500;        //enable next trigger
}
```

```
//**********************************
//                EMBEDDED MACHINE
//**********************************

//**********************************
//                function: embedded_machine
//                int emnumber: b1 or b2
//**********************************
void embedded_machine(emnumber)
{
        switch(emnumber)
        {
                case b1:
                        call(print);
                        call(beep);
                        schedule(print);
                        schedule(beep);
                        future(b2);
                break;

                case b2:
                        call(print);
                        schedule(print);
                        future(b1);
                break;

                default:
                break;
        }
}
```

```
//**********************************
//**********************************
// REAL TIME OPERATING SYSTEM
//**********************************
//**********************************

int main(int argc, char **argv)
{
        count = 0;                                                      //init(task1)
        sound_toggle = 1;                                       //init(task2)
        trigger_compare = sys_time + 500;       //activate(trigger)

        while(1)
        {
                clock_time = sys_time;
        if (trigger_compare==clock_time)//check if trigger has been activated
        {
                embedded_machine(atomic);

                        if (enable_task[print]==1)   //execute enabled tasks
                        {
                                print_task();
                        }
                        if (enable_task[beep]==1)
                        {
                                beep_task();
                        }
        }
        }
        return 0;
}
```