

Project Final Report

ASERT OS

A Simple Embedded Real-Time Operating System

04/25/08

Andrew Sweeney

ajs86@gwu.edu

Project Abstract

The goal of this project was to design a simple embedded real-time operating system for the ez8 CPU on the Zilog Z8 Development Board. Due to the complex and ever-evolving nature of this project I had no final completion in mind, rather milestones were set up by order of importance. The crux of the project was focused around multitasking, timers, and scheduling all of which are in a working state. The project was developed in C and assembly and hopefully will be continued in my spare time.

Status

Original Milestones:

1. Context Switching
2. Scheduler / Scheduling Schemes
3. Interrupt Handler
4. Process Communication
5. Timers
6. Threads and Thread control
7. Basic Device I/O
8. Memory Allocator (user/kernel)
9. Memory Protection
10. Nested Interrupts (Fooled with software?)
11. Shared Memory
12. Basic File systems

Out of my project milestones listed above I can say that I got 3 or 4 of them in a working state, however no where near where I would like them to be. Most of the problems I ran into occurred with the Zilog IDE compiler and poorly/undocumented information regarding what registers were being used as what during in the different stack frame schemes. Originally I wanted to use dynamic frames for my stack, however after several hours of finding weird results I decided to switch to the static frame, large memory model. I do think that I could use the static frame implementation however I feel like I would have to re-write a fair amount of the Zilog assembly code for how the stack frames are handled, as well as the interrupt request routines. I made no changes to my project itself as It is still a work in progress, there is just still more to be done.

Specification

All development was done on the:

ZDS II – Z8 Encore! Family 4.10.1 Build 07011201

Z8 Encore! Family IDE version 4.10 (06121401)
ZiLOG Z8 Encore! Macro Assembler Version 2.45 (06120402)
Z8 Encore! ANSI C Compiler Version 3.51 (06120402)
ZiLOG IEEE 695 Linker/Locator Version 6.22 (06120402)
ZiLOG IEEE 695 Object Librarian Version 4.11 (06120402)
GNU Make version 3.79 (ZiLOG enhancements v1.30)
Simulator Nexus Layer DLL version 1.00 (06120501)
eZ8 Disassembler version 1.04
eZ8 Encore! USB Smart Cable Nexus DLL version 1.10 (06120501)
eZ8 Encore! Serial Smart Cable Nexus DLL version 1.00 (06120501)
eZ8 Encore! Ethernet Smart Cable Nexus DLL version 1.00 (06120501)
Cycle-accurate eZ8 Instruction Simulator version 1.04

Relevant Build Flags:

CPU Family: Z8Encore_640_Family
CPU: Z8F6403

Code Generation:
 User Defined
 Limit Optimizations
 Frames: Static
 Memory Model: Large

Preprocessor Definitions:

_Z8F6403,_Z8ENCORE_640_FAMILY,_Z8ENCORE_F640X,_SIMULATE

C Startup Module – Standard

Use Default Libraries: C Runtime Library, Real

```
CC = @C:\PROGRA~1\ZiLOG\ZDSII_~1.1\bin\ez8cc
ASM = @C:\PROGRA~1\ZiLOG\ZDSII_~1.1\bin\ez8asm
LINK = @C:\PROGRA~1\ZiLOG\ZDSII_~1.1\bin\ez8link
LIB = @C:\PROGRA~1\ZiLOG\ZDSII_~1.1\bin\ez8lib
INCBASE = C:\PROGRA~1\ZiLOG\ZDSII_~1.1\include
```

```
CFLAGS = \
-const:RAM -define:_Z8F6403 -define:_Z8ENCORE_640_FAMILY \
-define:_SIMULATE -define:_Z8ENCORE_F640X -genprintf -keepasm \
-keeplst -Nolist -Nolistinc -model:L -optlink -promote -regvar:8 \
-reduceopt \
-stdinc:"$(INCBASE)\std;$(INCBASE)\zillog;$(INCBASE)\zillog\Z8Encore_F640X" \
-debug -revaa -cpu:Z8F6403 \
-asmw:" -cpu:Z8F6403 -define:_Z8F6403=1 -define:_Z8ENCORE_640_FAMILY=1
-define:_SIMULATE=1 -define:_Z8ENCORE_F640X=1 -include:$(INCBASE)\std;$(
(INCBASE)\zillog;$(INCBASE)\zillog\Z8Encore_F640X -revaa"
```

```
AFLAGS = \
-define:_Z8F6403=1 -define:_Z8ENCORE_640_FAMILY=1 \
-define:_SIMULATE=1 -define:_Z8ENCORE_F640X=1 \
-include:"$(INCBASE)\std;$(INCBASE)\zillog;$(INCBASE)\zillog\Z8Encore_F640X" \
-list -Nolistmac -name -pagelen:56 -pagewidth:80 -quiet -sdipt \
-warn -debug -NOigcase -revaa -cpu:Z8F6403
```

File List and brief description:

– arch_thread.c, arch_thread.h

These files hold the main thread information, functions. These are architecture dependent and are in control of creating new threads, and switching threads.

– clock.c, clock.h

These are the main system clock files. This is the general clock that is used for the OS which currently is TIMER0. This timer is used to get the system time, as well as increment each timer, and task switching.

– main.c

This is the main program. All tasks are set up here.

- **process.c, process.h**

These hold the TCB (task control block/process control block). All tasks are started through the functions defined here. Currently this also contains the scheduler.

- **scheld.c, scheld.h**

Currently these are empty, however the scheduler should be moved there.

- **sem.c, sem.h**

Basic semaphore library. Since there is no event handling currently these are really simple and can be used in any program.

- **time.c, time.h**

This contains time structs, which should be later used to implement unix like timers, and time of day functionality.

- **timers.c, timers.h**

This contains a very simple timer library which just works off the system clock. They can be used in any application. Currently there is no timer queue however.

- **z8def.h**

This contains basic system wide definitions.

Implementation & Construction

The starting point was to construct a basic task switcher. This proved to be quite difficult due to problems with my original design. Although the design itself was not flawed (at least in terms of ANSI C compatibility), the Zilog C compiler did not like the implementation and I reluctantly changed to a simpler design. The basic idea is simple. Every process has its own process struct. Currently there is a fixed predetermined number of tasks there can be this is because I have yet to implement my own memory allocation scheme yet. Each process is added

to a spot on the list. A currently executing process can give up its CPU time by calling yield which in turn will cause the scheduler to run. The scheduler will pick the next task to run and execute it. There is an interrupt request routine which when triggered will update a tick count which is used as the system clock. After so many ticks the routine will also call schedule to schedule a new task to run.

The threads themselves are rather simple, but work a little differently than standard threads in how the data is stored. Each thread contains a reference to its own stack. I do mean its own stack.... not just a stack pointer. A threads stack is simply an 8 bit array of a predetermined size. When a context switch occurs the running thread pushes all of its data onto its stack, then the next stack to run switches the global stack pointer to point to its stack and pops off the information. This design allows for relatively simple light weight threads which I thought would go well with the simple 8 bit processor.

The scheduler is very crude and no where near a practical implementation. Tasks are not scheduled with any real priority except for the time they should be on the stack to run for. I am not sure of the fairness of it since I did not have enough time to really dig down and analyze/create a proper scheme.

The clock count is currently off. Some operations that need to happen atomically I believe are delaying the clock such as context switches. I did not have time to accurately get a measure of how long these take but it seems like it takes a bit. Due to this the software timers are affected and incorrect. I don't think it should be to bad of a problem to fix once I figure out whats truly causing the problem.

process.h

```
struct process{
    pid_t pid;
    u8_t state;
    int priority;
    clock_t ticks;
    struct arch_thread thread;
};
```

time.h

```
/* simplified UNIX time struct */
struct tm {
    int tm_sec;           /* seconds after the minute [0, 59] */
    int tm_min;          /* minutes after the hour [0, 59] */
    int tm_hour;         /* hours since midnight [0, 23] */
    int tm_mday;         /* day of the month [1, 31] */
    int tm_mon;          /* months since January [0, 11] */
    int tm_year;         /* years since 1900 */
};

struct timespec {
    time_t    tv_sec;      /* time in seconds */
    long      tv_nsec;     /* time in nanoseconds */
};

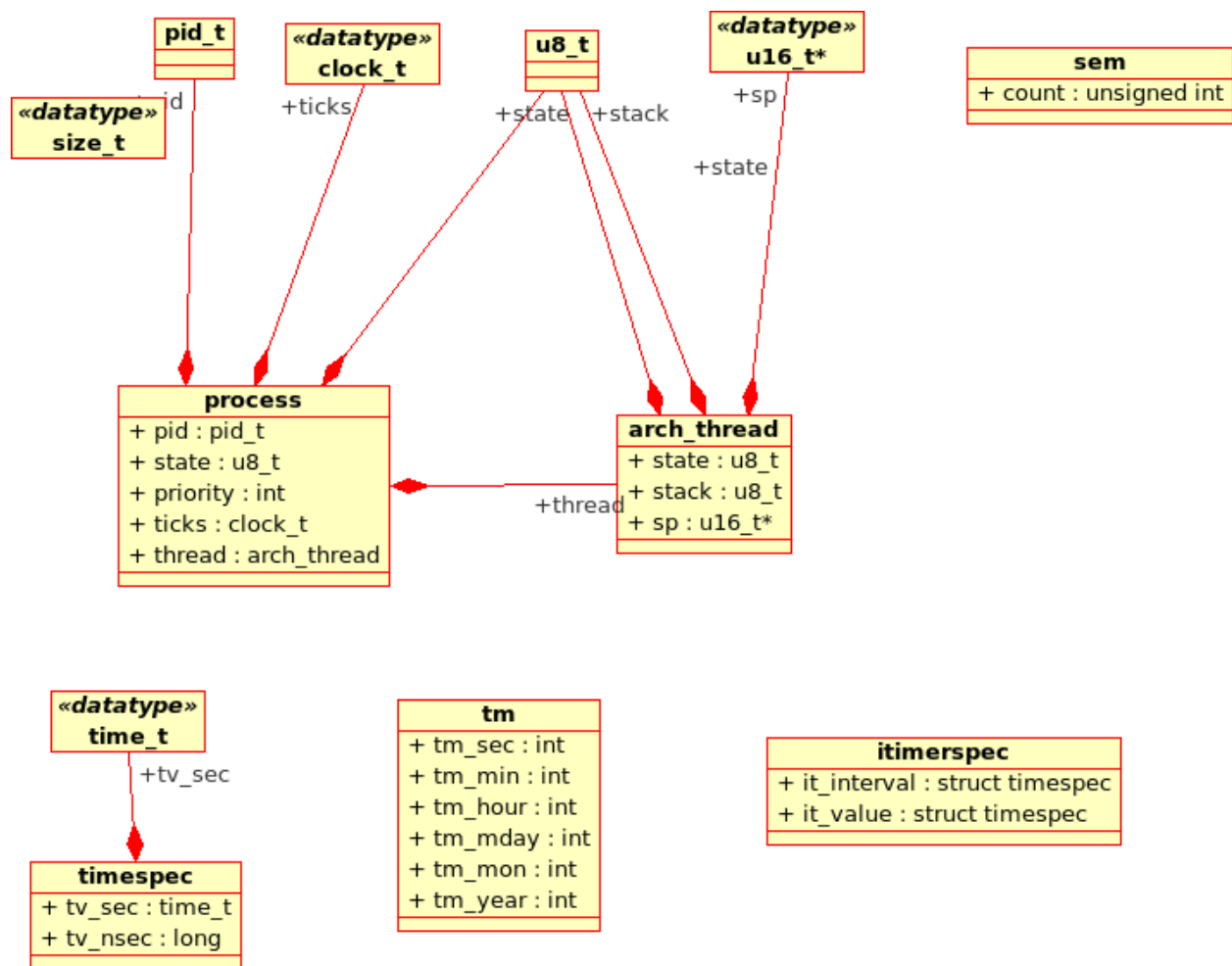
struct itimerspec {
    struct timespec it_interval; /* Timer period */
    struct timespec it_value;     /* Timer expiration */
};
```

sem.h

```
struct sem {
    unsigned int count;
};
```

arch_thread.h

```
struct arch_thread {
    u8_t stack[STACK_SIZE];
    u16_t *sp;
};
```



Retrospective

The design of an operating system need to be well developed from the start. Mine design, although well throughout was not as complete as it should have been, however I got away with this since it is relatively small. Before any more advancements are made on the project I need to rethink the task scheme as well as organization for the project. Originally I did have multiple folders however that got to be annoying and pointless since the IDE does not have a folder view. I also want to go back change completely some of the core libraries that Zilog uses since 1. I don't fully understand what they are doing sometimes since its not clearly documented and 2. Its just not appropriate for developing a real-time OS.

I found multiple annoyances when developing this project, however there was a lot of good documentation actually available through Zilog. I found the the ez8 CPU manual itself to very useful, the IDE and lack of documentation to what its doing and expecting was probably the worst part. I was originally trying to use dynamic frames however I could never wrap my head around how to fully get them working and whats written in the manual regarding how they work does not talk about what registers are used for what, i.e. Frame Pointer, Base Pointer, and all the other RR0 hidden 16 and 8 bit registers that I could not get full control of. I also want to re-write the IRQ handler so its more appropriate for this type of program.

I learned a great deal from this project and it only got me more excited about hardware and operating systems development. A good portion of my time was spent researching different techniques and implementations and because of this I also learned a great deal about some other open-source operating systems that I was not familiar with. I still have a great deal to learn with regards to programming at the operating system level as well as complex embedded systems, but I am confident that I have a strong enough basis to tackle these topics.

Attachments

Source Code.
Presentation Slides
Presentation Paper
Brochure

The program I used to generate the code UML Diagrams is called Umbrello. This document was typed in Open Office 2.4