

ZRT – A Real-Time Operating System for the Zilog Z8 Microcontroller

Abstract

A small preemptive multitasking operating system has been written to run on the Zilog Z8 microcontroller. The system consists of light-weight threads, a round-robin scheduler, binary semaphore and an ability to voluntarily relinquish control back to the scheduler. The Z8! Encore evaluation board was used to develop this system.

Motivation

When you have several isolated tasks each needing periodic servicing you can execute each one in sequence as shown in Fig. 1. If each of these tasks is short, like sampling the state of a switch, this is a reasonable approach. But what happens when you have a long compute-bound task and also need to periodically perform other tasks. Or worse, maybe you don't even know how long your tasks take to execute or they change dynamically. It is these non-deterministic factors that led to the development of preemptive multitasking operating systems.

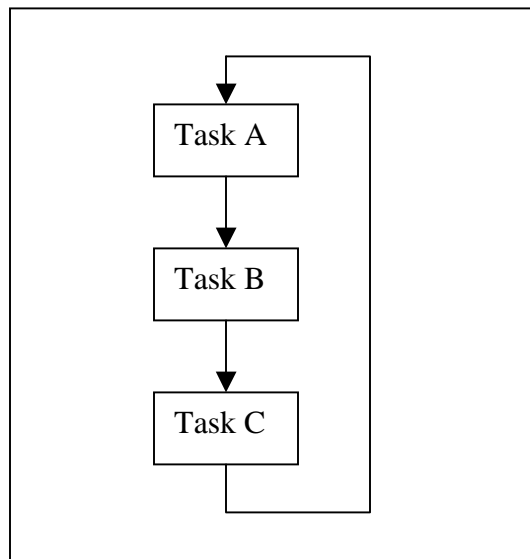


Figure 1 – Several short tasks can be handled without preemption.

These features can also lead to software that is easier to write and maintain. For example, the task of computing an output value and presenting that value can now be decoupled. In the ZRT sample code there are 3 competing tasks: a simple counter, an algorithm to compute pi and a routine to display those results. The display routine has attributes of the Observer Pattern described in Gamma et al. [4] where the change of a value by one sub-system is propagated to other sub-systems. Using patterns like this is a good way to produce more robust software.

The Stack Frame

When a function is called, 4 things are pushed on the stack. First come the function arguments followed by the return address. R15 and the variables local to the scope of the called function are pushed last as shown in Fig. 2. The C code and corresponding assembly language for a simple function that produces this kind of stack frame is shown in Listing 1.

```

char simple(char p)
{
    char l;
    l = p;
    return l;
}

*PUSH  R15
*LDX   R15, SPL
*SUBX  R15, #1

LD     R0,3(R15)
LD     -1(R15),R0
LD     R0,-1(R15)

*LDX   SPL, R15
*POP   R15
*RET

```

Listing 1 – C code and corresponding assembly listing for a simple function.

Context Switch

Each thread or function has a context or state associated with itself. This state consists of the current program counter, the stack, the content of the current register group (R0 – R15) and the register pointer itself. This context is defined in `struct thread` in `zrt.h` (see Listing 2). The scheduler maintains an array of these structures, one for each thread.

```

struct thread {
    unsigned int sp;    // Stack pointer
    unsigned char r15; // This is similar to the base pointer BP in Intel's 8086 architecture.
    unsigned char rp;  // The Register Pointer, points to a group of 16 registers. Used to access r15.
    unsigned char r14, r13, r12, r11, r10, r9, r8, r7, r6, r5, r4, r3, r2, r1, r0;
    unsigned char inUse;
};

```

Listing 2 – A thread's context. Note that the program counter isn't stored here because it's value, like the flags register, is pushed onto the stack when the interrupt routine is called.

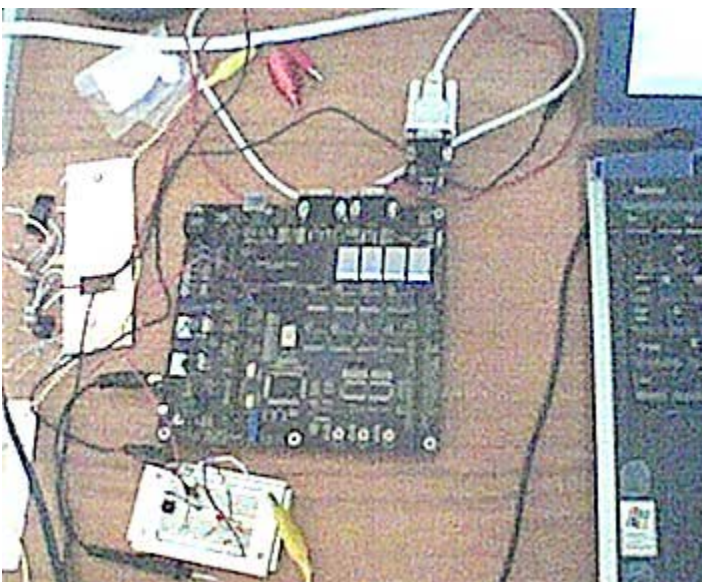


Figure 6. The Z8 Encore system used to develop ZRT.