Study on Operating System Designing in Computer-integrated Education

Liang Shi, Duoqian Miao
Tongji University, Shanghai, China

Abstract - In order to enhance students' understanding of computer composition principles and encourage them to compile operating system and compiler with the 32-bit MIPS based instruction subset processor designed by themselves, this paper will introduce main process and issues of Linux transplanting and discuss on possible solutions for transplanting integrity instruction set, besides, analyze their merits and disadvantages. In addition, it will fully introduce modification and reservation methods for exceptional instructions through the research of Linux source code. It will also capsule how to modify GCC compiler so as to provide solid theoretical ground for the follow-up practical experiment, by which similar theories based on Linux system transplanting can be further perfected. This work can be utilized as an instrumental reference for studies.

Keywords: MIPS32, Operating System, Experimental Reform

1 Introduction

With the rapid development of computer science, communication technology and network technology, embedded system has become an indispensable part in our daily life. Not like Windows, Linux has become the first choice for the developers in the embedded system filed for its good performances, clear structure and its open-source code. Linux has support for X86, ARM, MIPS, MIPS64, Sun SPARC, POWER PC, Motorola 68K, IBM S/390, Alpha, IA64, cris, parisc, sh and some other architectures.

Computer is a subject combined with science and engineering. Theoretical knowledge of computer science and practical use of computer engineering supplement each other, which take a giant stride forward in the subject and even the human society. In order to change the current phenomenon that students paid too much attention on books and they lacked of practice by their own hands, this paper expect to make an embedded Linux in a processor based on 32-bit MIPS instruction set, which provide a new idea in the reform of experimental practice in subjects like the operating system and the compiler theory. [1]

The development platform in this paper is an achieved processors based on 32-bit MIPS instruction set in the former part of integrated computer experiment. For some reason in designing, the instruction set is not the standard MIPS instruction set but its subset. The processor should be the same in logic after transplanting. [2]

2 Cross-Compilation Environment Build for Linux and Kernel Configuration and Compilation

Linux build is mainly including the cross-compiling environment build, editing codes on the host computer, compiling of programs, files and libs link and remote debug from host computer by a debug agent. We put our program into the ROM of development board to accomplish the experiment.

It is not feasible and practical to build a development environment on the development board because of its limitation of hardware. So we should build two platforms before transplanting the system. One is the host PC as development platform. The other is running platform on the Development Board. The relationship of them is shown in figure 1.

![Figure 1 Software development platform](image-url)

The Development process of software can be divided into three steps:

1. **Development Platform**
   - **Development environment**: Design, edit, compile, link, debug, text.

2. **Host OS**
   - Linux, windows, etc.

3. **PC, workstation, etc.**
4. **Operating platform**
   - **Debug agent**: Embedded Software, Bootloader, Embedded OS, Driver, Application
   - **Embedded hardware**
1 Source code writing.
2 Object modules building by cross compiling.
3 Link to modules and relative libs.

In order to generate binary codes which are suitable for running on the target platform, the codes should be compiled by the cross compiler between the platforms, which is different from developing software on a personal computer. In order to link the program codes with the correct data and libs from the memory in the target computer, we should use cross linker. Therefore, establishing a correct cross compiler is indispensable in the process of writing embedded software.

The main tools to establish a cross-compiler environment on MIPS instruction set including GCC, glibc, binutils and gdb as debugger.

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU tool chain.

Glibc is the link library and runtime of the program. For the reason that this link library and runtime is not based on development platform but the target platform, it should be compiled by the cross-compiler environment before being used.

Linux supports several popular computer architectures, which makes us choose the right configuration before cross compiling including processor architectures, file system types, board-level supports and device driver.

3 Ways to Solve the Problem With Integrity MIPS Instruction Set

Due to the students’ ability, the design of processor should not be too complicated in the former procedure of writing the processor in the integrated computer experiment. So we removed some less-frequently-used instructions and some complex instructions in coding. The whole processor is based on a subset of the standard MIPS instruction set. The main problem in the transplantation is how to keep the logical equivalence and how to make sure the program will be executed correctly.

After the kernel is compiled, if the corresponding instruction is not found in the target instruction set, the processor will throw out a reserved instruction exception. There are two ways to solve this question:

1) Change the kernel codes
2) Change the compiler.

3.1 Change the kernel codes

While the processor is running, if the next instruction does not exist, the system will throw out a reserved instruction exception and jump into the corresponding interrupt handler. We can achieve these instruction logically by adding actual operations for these losing instructions in the interrupt handler. That’s the principle and the procedure of how to change the kernel.

For the MIPS instruction set in the kernel, the reserved instruction exception initialization is located in the /arch/mips/kernel/traps.c. The function asmlinkage void do_ri(struct pt_regs *regs) is the function which handle the reserve instruction exception. Its main structure is as follows:

```c
asmlinkage void do_ri(struct pt_regs *regs)
{
    unsigned int __user *epc = (unsigned int __user *)exception_epc(regs);
    unsigned long old_epc = regs->cp0_epc;
    unsigned int opcode = 0;
    int status = -1;
    if (notify_die(DIE_RI, “RI Fault”, regs, 0, regs_to_trapnr(regs), SIGILL) == NOTIFY_STOP)
        return;
    die_if_kernel(“Reserved instruction in kernel code”, regs);
    if (unlikely(compute_return_epc(regs)<0))
        return;
    if (unlikely(get_user(opcode, epc)<0))
        status = SIGSEGV;
    if (!cpu_has_llsc && status < 0)
        status = simulate_llsc(regs, opcode);
    if(status < 0)
        status = simulate_rdhwr(regs, opcode);
    if(status < 0)
        status = simulate_sync(regs, opcode);
    if(status < 0)
        status = SIGILL;
    if(unlikely(status > 0))
    { 
        reg->cp0_epc = old_epc;
        force_sig(status, current);
    }
}
```

We can add our own functions into exception handler for unrealized instructions here, the code is structured as follows:
int simulate_otherOP(struct pt_regs *regp, unsigned int inst)
{
    register unsigned long rs;
    register unsigned long rt;
    unsigned long va;
    unsigned long mem;
    unsigned int byte;
    rs = _GPR_STKOFFSET(_RS_(inst));
    rt = _GPR_STKOFFSET(_RS_(inst));
    va = rs + (unsigned long)((short)_OFFSET_(inst));
    byte = va & 3;
    switch(_OP_(inst))
    {
        case 0x33:
            return 1;
        default:
            return 0;
    }
}

_OP_ is a macro operation. Its function is to make out the foundation for the switch statement by fetch the op code from 32-bit machine code. In the different case statement, we add corresponding appropriate handler by the fetched op code to achieve these instructions that will caused reserved instruction exception.

For example, in MIPS instruction set, the op code 0x22 corresponds to the LWL instruction. We add the appropriate command processing codes in the case statement to calculate the offset of non-aligned instruction then return from the function. The codes are shown as below:

```c
Case 0x22:
    mem = *(unsigned long*)(va-byte));
    mem = mem << (3 - byte) * 8;
    rt = (rt & ~(1UL >> byte * 8) | mem;
    _GRP_STKOFFSET(_RT_(inst)) = rt;
    break;
```

At last we add our custom handler into do_ri() function.

```c
If(simulate_otherOP(reg, opcode))
{
    Compute_return_epc(regs);
}
```

After the custom function finished and retuned 1, the Linux system will invoke compute_return_epc() to assign the value back into pc register from epc register. If not, the system will continue processing other exception handler.

We can change the Linux source code based on C language directly in this way. It is simple to starter and easy to master. The equivalence can be easily ensured after being modified so that this method has high practical value. However, this method is based on the interrupt handler of reserved instruction exception. If the code contains a huge amount of unachieved instructions, the system performance will be reduced. This factor must be considered before the method is invoked.

### 3.2 Change the compiler

Changing the Linux source code to achieve the custom operation in reserved instruction exception handler may make the system have low efficiency in the practice. Therefore, modifying the GCC compiler directly may also have its impact.

The main compilation process of GCC is shown in Figure 2:

![Figure 2 compilation process of GCC](image)

**Figure 2 compilation process of GCC**

## 4 Conclusions

During the instruction generation, we can change part of the GCC compiler source code files. While compiling, we can use achieved equivalent instructions sequences to instead of unachieved instructions which may be generated to solve this problem. The main GCC compile file includes inst-emit.c, inst-flags.h, inst-config.h, inst-code.h and so on. This method can solve the problem of system transplantation with the uncompleted instruction set in an efficient way. However, due to the high threshold and the efficiency of the GCC compiler, the experimental cycle of computer experiment will be extended greatly while the system efficiency is greatly improved. This section need to be further improved and followed up in the future work.
5 References


