Porting to H8/300 Architecture

Yoshinori Sato
linux H8/300 porting project
1-6-19 Yokomachi gosen city niigata pref. japan
ysato@users.sourceforge.jp

Abstract

I performed porting to H8/300 architecture of Renesas as very small GNU/Linux system environment. uClinux-2.4.20 runs with advanced mode of H8/300H and H8S CPU. And H8/300 architecture support is done merge of with linux-2.5.68, and development progresses currently. This thesis describes a part peculiar to in H8/300 support. And examination of method to perform real-time processing with CPU which is low-speed and problem to occur in a small system.

1 Introduction

I ported linux kernel to H8/300 architecture in small embedded environment. H8/300CPU is 16/32bit original architecture of renesas technology.

This result was done merge of with 2.5.68 of a series of 2.5. And a series of 2.0/2.4 is distributed as one part of uClinux.

I did not need to use full-scale OS till now. But I get possible to let the advanced application that I utilized file system or network connection because linux.

This thesis describes general description of H8/300 architecture and examines a problem of case letting linux work.

Evasion methods of constraint explain implementation and a porting method of kernel in detail. And I easily explain porting of libc and a userland.

I easily describe future expantion such as correspondence to linux-2.6 or Realtime extension last.

2 H8/300 prosessor

Internal 32bit registers. and instruction set which was limited processor. Memory space of 16MByte, and linear can access entirely.

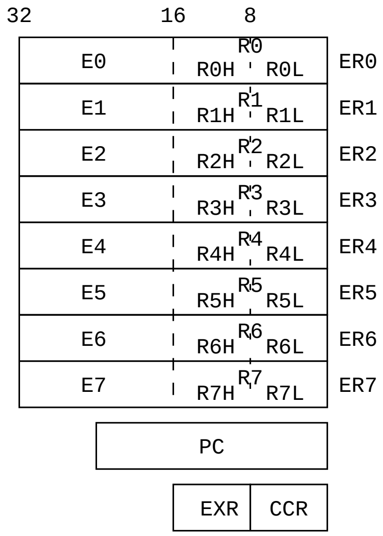

FIGURE 1: *Register set*

It is used as register of 16bit length usually, but can use two register as 32bit length in a mass. gcc has features to form such a code. I utilized this features and get possible to port linux kernel.
There are the following things in a restriction to obstacle when port it otherwise.

- A concept of kernel/user mode does not exist.

  Linux kernel requires with kernl/user mode. But it is not supported in H8/300 architecture.

- Interrupt vector table is fixed address.

Location of interrupt vector table is fixed. And that is assigned to ROM normally. Implementation of interrupt handler becomes difficult as it is because kernel cannot change interrupt vector freely.

It needed to solve such a problem in order to let linux kernel work with H8/300 architecture.

3 porting linux kernel

3.1 implementation of kernel/user mode

Current implementation of linux kernel requires with what execution mode more than 2 level exists, and stack area separates in each mode. Because there is not support with hardware about this features, it comes true with software entirely. Able to tell the current mode by unused bit of status register. check flag in exception handling starting and change mode when it is necessary. Changeover of mode performs renewal of sp and makes exception stack frame to kernel stack.2 Make exception stack frame alike and renew sp when switch over from kernel mode to user mode.

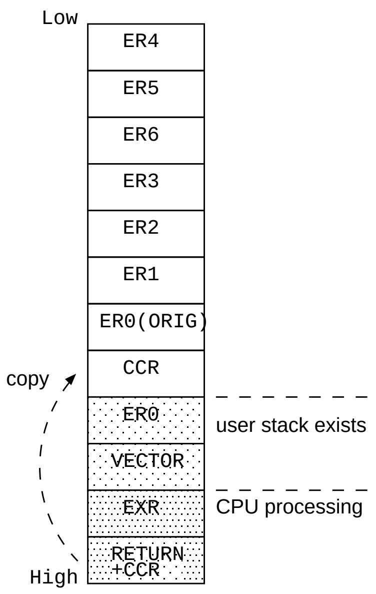

FIGURE 2: *exception stack frame*

3.2 implementation of vector table

I built a virtual table in RAM in order to make a change of vector table possible. Carry this out by seeing one table in RAM at the real table which there is in ROM.

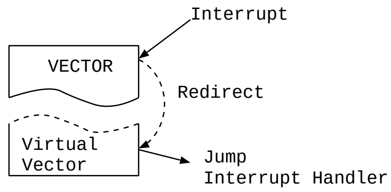

FIGURE 3: *exception handling flow*

Address of the table which there is in RAM analyzes contents of ROM in initialization and is decided. I mean by this that making of bootloader becomes easy.
And it became easy in this method to form vector address from an interrupt factor as a merit.
Implementation goes as follows. Jump to an individual address every each interrupt factor in H8/300. As for preparing an entry every an interrupt factor in order to pursue number, memory with many factors is inefficient. But can get return address to be different in interrupt factor because subroutine calls an interrupt entry from a table of RAM. More only than the case that prepared an individual interrupt entry became late in last method, but decided code size in last method than that because can small.
A comparison result

1. separation entries

```
vec_n:
    jmp @interrupt_n

intterrupt_n:
    mov.l er0,@-sp
    sub.l er0,er0
    mov.b #n,r0l
    jmp @common_interrupt_entry
```

size	12byte × factor
time	26cycle

2. common entries

```
vec_n:
    jsr @interrupt_entry

interrupt_entry:
    mov.l er0,@-sp
    mov.l @(4:16,sp),er0
    sub.l #vector_top,er0
    shlr  er0
    shlr  er0
    subs  #1,er0
```

size	20byte(common part) + 4byte × factor
time	42cycle

Because save process of register is added in real code, code size becomes big. Because a part except this occupies most as for the execution time, I mean that I can disregard a difference of last level.

3.3 Systemcall spec

Syscall trap	trapa #0 (software exception 0)
Syscall No	ER0
Syscall args	ER1 - ER6 (max 6)

Set system call number to register and generate a software exception. As for the argument, it is gone by way of register. Become procedure same as other architecture.
Because I examine PIC support of binary currently, I may change it in the future.

3.4 Device driver

There is you aiming at letting driver provided already work by some corrections. Driver of porting being done works by a correction of a part depending on a board.
This was able to carry out interrupt handler by imitating action of i386 linux. I examined similar technique about I/O access, but gave up it because address translation table burdened you.
Because that general-purpose I/O port shares it with more than one driver is not considered, kernel manages a situation inside not to compete. It is to be it in driver uses personal API and is to set a situation of I/O port thing. This is features to be specific H8/300.

4 Development

4.1 Support Target

Structure of a target supporting currently is a street of chart.

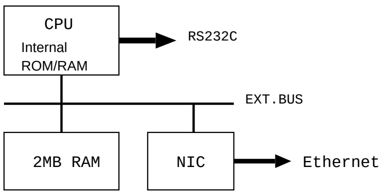

FIGURE 4: *aki3068net*

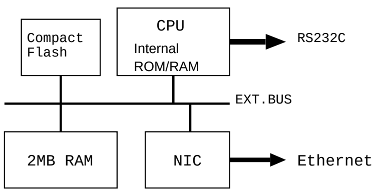

FIGURE 5: *H8MAX*

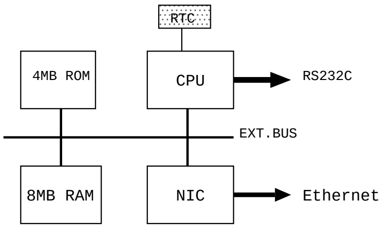

FIGURE 6: *EDOSK-2674*

Basic structure is almost same.
Interface with a host is to use RS232C (CPU built-in UART) and Ethernet.

4.2 Host Enviroment

I build cross development environment of h8300-elf in GNU/Linux environment.

	gcc	3.3 + patch
Useing version.	binutils	2.13.1
	gdb	5.2.1 + patch

A patch assuming long long with 64bit length comes to require in gcc. A patch to add requiring hardware simulation to minimally comes to require in gdb.

4.3 libc

Libc supporting currently is only uClibc. There is not support schedule more than this for the time being.

4.4 bootloader

If there is enough ROM capacity, it can boot directly. Can boot compressed kernel with a little ROM capacity. When reading is crowded from the outside (CompactFlash or tftp), I use RedBoot.

4.5 distribution

It became support with uClinux-dist recently. Can make environment to expect by this easily.

5 Future schedule

2.4 series performs a bugfix and review of code. 2.6 series is not stable yet. Driver support performs working for the present enough, too. And I proceed with improvement of toolchain.
Because computing performance of CPU is low as for the realtime extension, I mean for real in a simple method if there is not it. Because I theoretically cut interruption with hook from userland, I want to examine the implementation that made use of such a special feature.

6 Acknowledgement

Porting working of linux kernel took support of "MITO software project" of Information-technology Promotion Agency, and it was done. And I got contribution of a target and code from a lot of people. I thank these support.

References

[1] http://www.uclinux.org

[2] http://uclinux-h8.sourceforge.jp

[3] http://www.azpower.com/H8-uClinux/