EXPERIENCE OF USING UCLINUX-BASED CAMAC CONTROLLERS IN VEPP-5 CONTROL SYSTEM

D. Bolkhovityanov*, R. Gromov, BINP, Novosibirsk, Russia

Abstract

In late 90s a need to replace aging "Odrenok" crate controllers in VEPP-5 control system has become apparent. In 2001 BINP electronics department designed a Motorola mc5200-based controller which has onboard 100M Ethernet and runs uClinux (Linux version for microcontrollers w/o MMU). In the course of a year VEPP-5 team has tested this controller and integrated it into control system infrastructure. This work had shown both pros of new controller (mainly due to unification of control system environment) and its cons (CPU architecture differs from that of host PCs, uClinux lacks some important features of Linux).

1 WHY INTELLECTUAL CONTROLLERS?

Historically CAMAC is the most used hardware standard in BINP, and this will remain for many years. Host computers are typically PCs running Linux. Two types of CAMAC controllers are being used: *dumb* and *intellectual* ones. Dumb controllers are simple devices which only accept commands to execute a single NAF. Intellectual controllers have their own CPU and are able to perform very complicated tasks, not disturbing host computer at all.

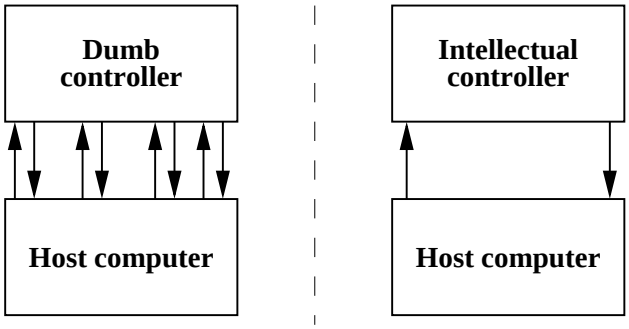

Figure 1: Data interchange diagrams between host computer and dumb and intellectual controllers for a typical "channel read" operation.

Dumb controllers are very simple in use, but they are suitable only for very small facilities and such systems scale badly. Additionally, dumb controllers aren't suitable for realtime systems, since host computer has to run a GUI, which almost always introduce latency. (In fact, there is one more problem: regular Linux/x86 has a task scheduling quant of 10ms, which is often insufficient.)

Intellectual controllers are used in more complicated tasks, and are able to take all expenses of control except the user interface.

BINP have been using home-made intellectual controllers "Odrenok" for many years (it has 24-bit ICL-1900 instruction set), and the latest version has become Ethernet-enabled[1]. Ethernet turned out to be a nice fieldbus. However, "Odrenok" hardware is too ancient and required replacement.

So, in 1995 several BINP teams began to use another home-made intellectual controller, which was based on Inmos T414/T805 RISC processors[2]. At that time it was a relatively powerful unit — a 25MHz 32-bit CPU with 2M RAM and integrated FPU, which fulfilled most of our requirements. However, that controller had several problems in itself:

- Inmos hardware required a dedicated network controller in the host PC, and the network had a hierarchical tree topology, which is very inconvenient.
- There was no OS and no dynamic program loading – microcontrollers booted with a monolithic image.
- Inmos processors were rather exotic, and used arcane programming language.

2 THE CM5307 MICROCONTROLLER

The CM5307 microcontroller is built around Motorola MCF5307 microprocessor. MCF5307 belongs to the m5200 "Coldfire" family[3]. The Coldfire family has RISC architecture, but its instruction set is a subset of the 68K family instruction set.

The CM5307 microcontroller contains:

- 66MHz MCF5307 CPU.
- 16M SDRAM.
- 128K boot ROM.
- 4M flash-disk, formatted with ext2 filesystem.
- Real time clock, one RS-232 port and one 100Mbit/s Ethernet controller.
- CAMAC bus controller.

The microcontroller can boot either from a flash disk or via BOOTP. The flash disk is large enough to contain Linux kernel, a telnet daemon and a basic set of utilities – cp/ls/ping etc.

The RS-232 port is very useful during initial configuration of the controller and significantly eases troubleshooting.

CAMAC bus is mapped to CPU address space, so CAMAC can be accessed by simply reading or writing to memory in a special region. But a more reliable way is to use kernel driver, available via `/dev/camac`, which also provides access to LAM requests.

* bolkhov@inp.nsk.su

Up to now two series of CM5307 controllers were produced, with cost about 0.5K$/unit. The second series had several improvements.

3 CHOICE OF OS

We had a choice of two OSes: uClinux[4] and RTEMS[5].

uClinux is a Linux clone for processors without a memory management unit (MMU). Besides that, it is a regular Linux.

RTEMS, on the other hand, is a "half-Unix": "*It supports about 70% of the POSIX 1003.1b-1996 standard.*". RTEMS is good in multithreading, which is vital for CAMAC controller software. But RTEMS has one tremendous disadvantage: *it is not a true OS*, it doesn't support dynamic program loading. Our experience with Inmos-based controllers has shown that "monolithic image" approach becomes a nightmare in a constantly changing control system.

So, we chose uClinux.

4 PROS

4.1 Linux

Running the same OS in a host PC and in the microcontroller significantly eases the life.

Software for CM5307 is written in the very same C language. We can use the same libraries and header files (it is very handy: we had experience of maintenance of two separate "protocol definitions" when using Inmos controllers, and keeping them in sync wasn't a trivial task).

We even use the same set of Makefiles for both controller's and host's parts of the control system (the calls to "gcc" are simply replaced by calls to cross-compiler).

Controller is able to use host's filesystem via NFS, so that updating its software is trivial and can be done on the fly.

4.2 Ethernet

Ethernet is very appropriate in a role of fieldbus.

All communication between hosts and controllers is done via IP – there is no need to use non-standard protocols, often hard-to-understand and poorly designed.

Since there are no special communication hardware, a need to care about that hardware's drivers on every kernel upgrade also disappears.

5 CONS

The first minus of CM5307 is that Motorola processors are *big endian*, while Intel's are *little endian*. This means that data should be converted between these nodes. While this technology is well-known and is essential for system programmers, converting each and every piece of data is a mundane job.

The second and the greatest problem is absence of MMU. So, there is no virtual memory (and no shared memory), which means that:

1. No shared libraries, so library code is duplicated many times. And there is no standard mechanism of using shared memory for interprocess communication.
2. No shared libraries means that there is no dynamic module loading (`dlopen()` call is missing), so we can't run our control system server[6] directly in the controller.
3. `malloc()` implementation is very limited.
4. All processes (and the kernel too!) live in the same address space, so a programming mistake in any of the programs can ruin any other program and even the whole system. And such errors are very hard to find. Due to lack of memory protection, there is no users separation in uClinux/Coldfire — all processes run with uid=0.
5. No copy-on-write ability — so, the `fork()` syscall, one of the Unix cornerstones, would become very expensive. So, uClinux simply doesn't support `fork()`.

The first four points are tolerable: there is enough RAM in the controller, and programming errors are "findable". In fact, the latest version of the microcontroller even has satisfactory memory protection: there are memory bound registers, which limit accessible space.

But the real problem is the absence of `fork()`. Of course, there is a replacement – `vfork()`, but it is non-standard and is extremely limited. Lack of `fork()` has two serious consequences:

- Majority of software for Linux simply wouldn't even compile under uClinux.
- Multitasking and multithreading becomes very expensive and tricky.

We tried to find a POSIX Threads implementation for uClinux/Coldfire, but all attempts to compile it had failed, so we finally gave up.

In fact, Linux without `fork()` and virtual memory looks more like a DOS, just with a Unix-like I/O library.

6 "ABSTRACT DRIVERS"

The VEPP-5 control system exploits several types of CAMAC controllers, and CAMAC blocks used are mainly the same. But several sets of drivers (with identical functionality, but for different CAMAC interfaces) have to be maintained. For example, drivers for a dumb crate controller are written for Linux in C, as well as drivers for CM5307. And all of the drivers operate with "channels" – units of data read from CAMAC block (e.g. voltage measured by ADC) or written into it.

An obvious idea comes to mind: can all these sets of drivers be unified?

We chose a following approach:

- Each driver's text exists in a form of a .h-file (we call these .h-files "abstract drivers"). This .h-file contains several functions — Init, Read and Write. These functions are declared using `ABSTRACT_XXXX_FUNC` macros, so that exact types of parameters are hidden.
- For each CAMAC interface there is an interface library, called "driver interface", which does all the "dirty work". On one side, it knows how to communicate with driver's "user" (in case of CM5307 it implements the controller side of the host↔controller communication protocol). And it knows when to call driver's Init/Read/Write functions. On the other side, it provides the abstract driver with an "abstract CAMAC interface" in a form of DO_NAF() function.
- When building binary drivers, abstract drivers' texts are `#include`'d by driver interfaces.

The simplest driver (for a dataway display, which has 1 register) looks like this:

```
ABSENT_INIT_FUNC

ABSTRACT_READ_FUNC
{
    DO_NAF(ref, N1, 0,  0, value);
    return 0;
}

ABSTRACT_FEED_FUNC
{
    DO_NAF(ref, N1, 0, 16, value);
    return 0;
}
```

This approach has an additional advantage: CAMAC drivers become very small and easy to write even for programmers with little skills. And probability of programming mistakes in such small pieces of code significantly decreases.

Of course, there are other approaches – for example, by making each driver and each CAMAC interface C++ classes, and mixing them to get a driver for a certain block on a certain interface. But this one seems to be one of the simplest, little resource-consuming and portable.

7 CONCLUSION

CM5307 is an improvement over previous intellectual CAMAC controllers used in BINP.

In fact, ARM, PowerPC or an x86 clone would be a better choice of CPU for a microcontroller, but Coldfire was chosen because of abilities of BINP electronics department.

Use of uClinux was determined by CPU. Linux turned out to be a very good OS for a microcontroller, and true Linux would be even better.

So now BINP electronics department plans to develop a CAMAC controller based on an x86-compatible CPU. This CPU will be a "ready-to-use" board in SODIMM standard, which are widely available on the market (see, for example, [7, 8]). These SODIMM boards contain complete microcomputer, with CPU, RAM, flash-disk, Ethernet and several other I/O controllers (usually serial port and, often, VGA). These boards are PC-compatible and allow to run any of PC OSes, including Linux.

Using exactly the same architecture and OS in both host computers and microcontrollers would give us a completely unified environment.

While that new, better crate controller isn't ready, we'll continue to use CM5307, and use of "abstract drivers" will ease the transition to new controller.

8 REFERENCES

[1] A.Aleshaev et al, "VEPP-4 Control System Upgrade", Proc. ICALEPCS'97, http://www.aps.anl.gov/icalepcs97/paper97/p057.pdf

[2] D.Yu. Bolkhovityanov, Yu.I. Eidelman, "Control System for VEPP-5 electron-positron complex", Proc. ICALEPCS'97, http://www.aps.anl.gov/icalepcs97/paper97/p066.pdf

[3] "68K/ColdFire", http://www.motorola.com/COLDFIRE

[4] Lineo, Inc., "Embedded Linux/Microcontroller Project", http://www.uclinux.org/

[5] OAR Corp., RTEMS, http://www.rtems.com/RTEMS/rtems.html

[6] D. Bolkhovityanov, I. Pivovarov, O. Tokarev, "Evolution and Present Status of VEPP-5 Control System", PCaPAC'2002 poster MO-P15.

[7] FS FORTH-SYSTEME GmbH, "DIMM-520", http://www.fsforth.de/produkt.cfm?color=blue&prodid=49

[8] JUMPtec Industrielle Computertechnik AG, "DIMM-PC/386-IE", http://www.jumptec.de/product/data/dimmpc/dimm386iee.html