DEBROS: A UNIX-like OS for 8-bit microcontrollers

MICHIGAN STATE UNIVERSITY

Mark A. Davis
Control Systems Software Group
National Superconducting Cyclotron Laboratory

NSCL

Introduction

Can low-end 8-bit microcontrollers based on decades-old technology help run an advanced experimental science facility? Yes! With the right tools, these low-end processors can support the basic functionality programmers expect from any UNIX-like operating system, making them a practical choice for many uses.

Motivation

The lack of a standard software platform for such 8 bit microcontrollers is an impediment to their use in projects they are otherwise well suited for. Proprietary tools and programming interfaces require longer learning curves and result in non-portable software and increased costs. At the National Superconducting Cyclotron Laboratory, these and other limitations brought development to a standstill and forced a search for alternatives. With no ready-made solution available, I decided to develop one myself.

Cooperative multitasking doesn't scale

Originally, our controllers used the cooperative multitasking mode, which works well when tasks are few and short. But inevitably, the number and capabilities of the tasks grew. To maintain acceptable response times, I was forced to break each task into smaller and smaller pieces, each responsible for managing its own CPU usage and saving/restoring the task state between calls. Tasks became larger, more complex, less efficient, and required more work to maintain.

```
int cooperativeTask1(unsigned long maxMS)
{
  static int state = 0;
  unsigned long startTime;

  switch (state)  {

    case 0:
      printf("Do initialization");
      state = 1;  break;

    case 1:
      printf("Do part 1");
      state = 2;  break;

    case 2:
      startTime = getMS();
      while (msSince(startTime) < maxMS))  {
        if (part2Done)  { state = 1;  break; }
        printf("Do a little more of part2");
      }
      break;

    default:
      printf("State machine error");
      return -1;
  }

  return 0;
}
```

Figure 1. Even simple tasks become complex when using cooperative multitasking.

Preemptive multitasking simplifies

Use of a preemptive multitasking kernel eliminates the need for each task to manage its CPU usage and save and restore its own state between task switches.

```
int preemptiveTask1(int argc, char **argv)
{
  printf("Do initialization");
  for (;;)  {
    printf("Do part 1");
    while (! part2Done)
      printf("Do a little more of part 2");
  }
  return 0;
}
```

Figure 2. Things are so much simpler when using preemptive multitasking.

High-end features on low-end hardware

First installed in 2006, DEBROS now runs on over 150 controllers at the NSCL, and its use continues to expand.

- Rabbit 3000 running at 44 MHz
- 768K RAM (200-265K unused)
- 512K Flash (used for basic filesystem)
- 100BASE-T Ethernet

Figure 3. Typical hardware specs

- Preemptive, priority-based, soft real-time multitasking kernel
- Hardware memory protection
- UNIX-compatible API (fcntl, stdio, sockets, pipes, semaphores, signals, etc)
- Multiple TCP and UDP network connections (including login shells)
- UNIX-like shell commands
- Simple Remote File System
- Concurrently writable system log, continuously backed up to remote file server
- EPICS code for auto discovery, configuration, and custom Records
- Extensive diagnostics includes remote monitoring and updating of boot images
- View plot of system variables in real-time
- Base functionality available to all applications includes:
  - Read/Write from/to digital and analog I/O signals
  - Bench and field-level calibrations for analog inputs and outputs
  - Hardware-specific initialization during startup
  - "Persistent" variables (saved to flash/BBRAM, restored on startup)

Figure 4. DEBROS software features

Is DEBROS right for your project?

If your project requires sub-millisecond response times and use of every last CPU cycle, then DEBROS may not be a good fit. But if you want a ready-to-use programming and runtime environment based on UNIX standards, then DEBROS could be just what you need.

Acknowledgments

Thanks to Andrew S. Tanenbaum and Linus Torvolds for concrete examples of what can be done if you put your mind to it. Thanks to Vasu Vuppula (NSCL) for urging me to publish my work. And thanks to my wife (Deborah) for her patience while I spent hours at home learning and working on DEBROS.

References

DEBROS Developer and User Manual, Mark A. Davis, Controls Software Group, NSCL; http://groups.nscl.msu.edu/controls/