Linux Debugging Overview

This article is intended to give the reader a general overview of the debugging tools or methods available at the each of the major software layers: User space, Kernel space and Boot-loader space. A brief description of the characteristics of these three spaces is offered under each corresponding section.

Contents

User Space
Debugging Tools:
Kernel Space
Debugging tools
Boot-loader
Debugging Tools:

User Space

The user space layer is where most of the customization or product differentiation software resides. In Linux, as in most high-level operating systems, multiple applications often written by different developers can co-exists and run at the same time; therefore, in order to avoid contention for hardware resources, applications access hardware via pre-defined software driver APIs as opposed directly (improves portability of application to another hardware platform with slightly different hardware). Similarly, user space applications cannot access physical memory directly, but rather access memory via virtual memory pointer; this virtual to physical memory conversion and vice-versa is taken care of by the Virtual Memory Manager (VMM) software module which resides in the kernel space.

Debugging Tools:

- MEMWATCH (http://www.linkdata.se/sourcecode.html): MEMWATCH is an simple to use open source tool for detecting a variety of memory errors, including memory leaks, erroneous frees, overflow and underflow ... All you need to do is simply add a header file (memwatch.h) to your source code and use the -DMEMWATCH option in your gcc statement. MEMWATCH tells you if you free an already freed pointer or if you have unfreed memory and the line number which has the problem; it also displays statistics including how much memory was leaked, how much was used, and the total amount allocated.
- **strace** : strace is a powerful utility included in virtually all Linux distributions. It captures and displays information for every kernel system call executed by a Linux application and can even be used with programs for which no source code is available (no debug symbols necessary). The linux man pages (man strace) will give you a quick reference on how to use strace. In addition to using strace to trace thru hierarchy of system calls, using strace with the '-c' option (strace -c) produces high level profiling of your application.
- **GNU debugger**: GDB is perhaps the most important tool for developers. GDB can be used to debug post-mortem core dumps or to conduct a live debugging session. In addition, many graphical front ends have been written on top of command line GDB, ranging from the simple open source applications such as Data Display Debugger (DDD) to high-end commercial IDEs such as DevRocket. GDB manual is rather large and GDB can take some time to master; however, this article gives a brief introduction on how to use command line GDB on our DaVinci platforms
    - **DDD** : Data Display Debugger (DDD) is a popular open source graphical front end to GDB. To find out more information on using DDD, you can read the DDD Manual (http://www.gnu.org/manual/ddd/). In addition, this App Note (http://focus.ti.com/general/docs/litabsmultiplefilelist.tsp?literatureNumber=spraap9) gives a brief introduction on using DDD with one of our platforms (DM6446 EVM); however, the steps described in the App Note can easily be extended to other DaVinci and OMAP platforms.
    - **DevRocket IDE**: DevRocket is a commercial IDE available from MontaVista. Once a user purchases the correct License from MontaVista, the user typically gets access to MV Zone software update site where they can download the DevRocket software as well as the necessary documentation. In addition, MontaVista's online MELD community (http://meld.mvista.com/default.aspx?&ref=ls) is a great source of support for this tool.

Kernel Space

This layer normally contains all the software drivers which abstract the hardware resources (memory, peripherals...) from the user space applications. Most of the software drivers adhere to an industry standardized APIs (e.g. V4L2 for capture, driver, ALSA for audio, I2C, USB....), hence the driver developer writers abide by these APIs; in the rare cases where there is no standardized industry API for a particular hardware peripheral, then the kernel developer can define their own API and provide corresponding documentation. Unlike user space software, kernel software is not restricted from accessing physical memory directly. Also, in addition to drivers which control the hardware, the kernel also includes software modules which implement various operating system managers (e.g. VMM discussed in previous section) and file-system types (EXT3, JFFS, YAFFS2...).

Debugging tools

- **Kernel source level debugger (kgdb)**: The kgdb program allows for remote bebugging of host Linux kernel through gdb. KGDB was originally implemented as a patch to Linux kernel, but it has been included in the official kernel in 2.6.26. This patch allows the remote host running gdb to connect to the target machine (running the kernel to be debugged) during the boot process. This allows you to begin debugging as early as possible. You can then break into the kernel, set break points, examine data, and so on (similar to how you would use gdb on an application program). Two machines are required to use kgdb: one of these is a host machine, and the other is a target machine. A serial line (null-modem cable) connects the machines through their serial ports. For a quick toturial on using kgdb on one of our DaVinci platforms, see Debugging on DaVinci using kgdb
- **Oops**: Kernel oops contain much useful information to help you trouble shoot the cause; however, much on the information displayed requires good knowledge of the underlying processor architecture. It is mentioned here for completeness.
- **Code Composer Studio**: CCS is a JTAG based debug environment. CCSv4 and later support Linux Aware Debug enabling you to debug kernel modules, view active processes and threads and debug individual threads.

Boot-loader

The boot-loader software is likely much smaller than the kernel or user space software; its goal is to initialize the basic (or minimal) hardware necessary to upgrade software, load the kernel and eventually user space software. This can be challenging to debug since only basic hardware is initialized at this point and hence many of the useful utilities available at the kernel and user space are not present. Some vendors include some basic diagnosis utilities, but these are very vendor specific and not covered here. Fortunately, the need to customize boot-loader is minimal and changes are often straight-forward; therefore, most users should not need to deal with debugging boot-loader at all.

Debugging Tools:

- **JTAG** : JTAG probes use a low-level method of communication originally employed for boundary scan testing of integrated circuits defined by Joint Test Action Group (JTAG); most modern CPUs today include a JTAG interface designed to provide software debugging capabilities. When it comes to debugging problems in the early stages of the boot process, JTAG probes have become the tool of choice. In the world of JTAGs, you have your ARM-based JTAGs (normally 20--pins) and you TI JTAGs (normally 14-pins, see JTAG Connectors). If you are using a device with ARM9 or ARM 11, you will need to make sure you have an emulator which is capabile of Adaptive Clocking. The ARM 20 pin and TI 20 pin connectors do provide for system reset, which is very useful for debugging driver code, especially right after boot.
  - **BDI2000** : This is a very popular choice for debugging ARM processors; you can find more information here
  - **XDS510/560 series**: These family of JTAGs are available from various vendors such as Blackhawk and Spectrum Digital (http://www.spectrumdigital.com) and can be used to debug both ARM and DSP on TI processors. More details at: XDS510 and XDS560. There is also a low cost product line called XDS100.

{{

1. switchcategory:MultiCore=
   - For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
   - For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

   Please post only comments related to the article **Linux Debugging Overview** here.

Keystone=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **Linux Debugging Overview** here.

C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **Linux Debugging Overview** here.*

DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **Linux Debugging Overview** here.*

MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **Linux Debugging Overview** here.*

OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Linux Debugging Overview** here.*

OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Linux Debugging Overview** here.*

MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **Linux Debugging Overview** here.*

*For technical s please post yo questions at http://e2e.ti.co Please post or comments abo article **Linux Debugging O** here.*

}}

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity