Addressing the Challenges of Linux OS Development

WIND RIVER

Development Paradigm Traditional

WIND RIVER

HW Bring-up	Boot Loader Development	Kernel Bring-up	Application Development

Development Life Cycle

HW Bring-up

- Out-of-reset testing
- On-chip Debugging Interface test
- Minimal code execution test
- HW Peripherals test

Boot Loader Development

- Bootstrap code including Chip selects, Memory, MMU, interrupt controllers, timers, PCI devices and Cache
- Communication channels (RS-232, and Ethernet)

Kernel Bring-up

- Initializations, Device Drivers
- Mounting a File System
- Other OS component initializations

Application Development

- Debugging application with multiple processes and multiple threads

HW Bring-up

WIND RIVER

Challenge

Out-of-reset testing:

- Controlling the processor’s reset, and monitoring the out-of-reset process

HW Diagnostics:

- Connecting to the CPU with On-Chip Debug tools and putting the processor into background mode

- Testing and troubleshooting memory, data bus, address bus, and other peripherals problems without software to run on the CPU

HW Bring-up

WIND RIVER

Solution

Out-of-reset testing & HW Diagnostics:

- A full featured On-Chip Debugging emulator will enable you to issue a reset to the processor, monitor the reset sequence as it happens, and report problems
- Use the emulator's built-in code execution diagnostics to check if the processor is able to run basic code
- Use built-in HW diagnostics and other HW configuration utilities to test and configure your HW

Boot-loader

WIND RIVER

Challenge

Boot-strap code:

- The CPU and peripherals need to be initialized with the correct values and in the correct sequence

Execution and Control:

- Bootstrap/Bootloader code must be developed before the distribution software debugging tools can be used
- Programming the bootloader onto your board when there is no code running
- Software debugging tools require code and a communications channel (serial or ethernet) that are not available until the code and drivers are developed

Boot-loader

Solution

Use also an On-Chip Debugger to:

- Gain access to monitor core and peripheral registers
- Download and Debug the bootloader running from RAM
- Use breakpoints inside of the ISRs to verify execution of code through the vector regions
- Program flash with the bootloader that you have created
- Use HW breakpoints to debug your code execution in ROM

Kernel Bring-up

WIND RIVER

Challenge

Kernel and Device Drivers:

- The Linux kernel requires some kind of initializations provided by a bootloader before it can run
- Incorrect boot configuration or kernel initialization causes the kernel to crash early in execution before software tools can connect and provide visibility
- Kernel development frequently results in kernel crashes. When the kernel crashes, the kgdb debug channel also crashes.
- Tools are typically tailored to a specific distro, and in some cases a specific kernel version. Support of multiple distros or kernel versions requires multiple development toolsets

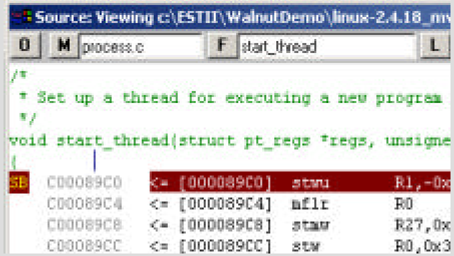
Kernel Bring-up

WIND RIVER

Solution

Use an On-Chip Debugging Emulator that supports:

- Debugging with MMU enabled, in both physical and virtual memory
- Setting HW and SW breakpoints in both physical and virtual memory



- Linux distribution and kernel version independent
- Source level debugging in the kernel without the intrusion of instrumentation or a software

Kernel Bring-up

Challenge

Mounting a File System:

- Linux requires a root file system (“Everything is a file”)
- NFS mounted root file system requires a network device driver to be in place and functioning reliably

Kernel Bring-up

Solution

Mounting a File System:

- Start with a RAM disk to debug kernel initialization and the connection channels required for your NFS mounted disk
- Use your On-Chip Debug connection to download the RAM disk contents to the target board
- Once the kernel boots and your connection channel is functioning reliably, substitute your NFS mounted file system and debug

Application Development

WIND RIVER

Challenge

Multiple Process/Thread Debugging:

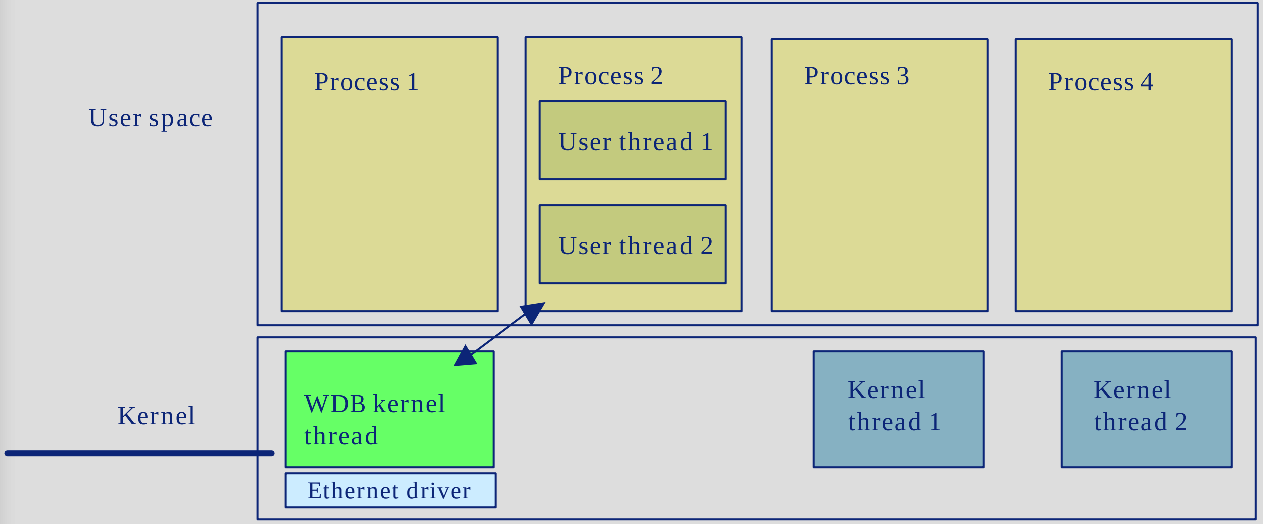
- Debugging a multi-threaded application using GDB is not reliable in a cross-development environment
- Debugging a single thread from within a process halts the entire process and all related threads
- Multiple debugging sessions for multiple processes sometimes confuses because multiple processes can only be debugged by launching separate gdb GUI and server sessions for each process
- Debugging a user application that accesses resources in the kernel space requires a separate (and different) debug agent for both the application and kernel

Application Development

Solution

Multiple Processes/Thread Debugging:

- Use a debug agent that:
  - Runs as a ‘polling’ RPC server kernel thread to enable both kernel and user application debugging
  - Leverages signals to debug processes
  - Works over serial and Ethernet



Questions

?