Overview of Embedded Linux Analysis Tools

John “JT” Thomas
Area Sales Manager
Types of Analysis Tools

- **Static**
  - Analyzes source or object code for common mistakes and/or patterns
- **Debugging**
  - Controls program execution and provides detailed state information
- **Instrumentation**
  - Inserts source code into program to track activities
- **Statistical**
  - Utilizes facilities outside of program to perform analysis
Static

• Source Code Analysis
  – Analyzes application without actually executing it
  – Provides security vulnerability and defect detection
  – Architectural and build-over-build trend analysis for multiple languages

• Examples
  – lint
  – Klocwork
  – Coverity
Debugging

- Runtime (Dynamic) Analysis
  - Interactive
  - Halts execution
  - Broad view of a program state

- Loads program and utilizes debug information to “see” into the application

- Detailed state information
  - Registers
  - Memory
  - Dissassembly
  - Stack variables
GDB – GNU Debugger

- de-facto FOSS debugger for GCC toolchains
- Covers a wide range of architectures
- Support remote debugging with GDBserver
- Can debug over serial, TCP/IP, and/or USB
- Can drive JTAG debuggers
Remote Debugging

Command-line input

Text output

GDB

GDBserver
Analysis Tools

• Typically - work independently from a debugger

• Answer specialized questions
  – Where are the memory leaks?
  – Where are the performance bottlenecks?
  – How much memory am I using?
  – Which system events have transpired and when?
Memory Analysis

• Memory Usage
  – How much memory is my application using?
  – What about the kernel? What parts are responsible for the most usage?

• Memory Debugging
  – How do I find the source of a memory leak?
  – Are there any memory access violations?

• Examples:
  – dmalloc
  – electric fence
  – mpatrol
  – valgrind
/proc

• /proc is a virtual filesystem that resides in the kernels memory

• A direct reflection of the system kept in memory and represented in a hierarchal manner

• View and manage information about kernel and currently running processes
  – Statistical Information
  – Hardware Information
  – Runtime Parameters
  – Network and Host Parameters
  – Memory and Performance Information
mpatrol

- Linux userspace program and library that diagnoses run-time errors that are caused by the incorrect use of dynamically allocated memory, including writing to free memory and memory leaks

- Replaces calls to existing C and C++ memory allocation functions as seamlessly as possible

- Summarizes all of memory allocations listed by size and the function that allocated them and a list of memory leaks with the call stack of the allocating function.
Performance Analysis

• Determine program bottlenecks
  – Which functions are using the most CPU time?

• Asymptotic (big-O) analysis
  – Which data structure is the right choice?

• Examples:
  – OProfile
  – gprof
  – VTune
OProfile

• A system-wide profiler for Linux systems, capable of profiling all running code at low overhead.

• Consists of a kernel driver and a daemon for collecting sample data, and several post-profiling tools for turning data into information.

• Leverages the hardware performance counters of the CPU to enable profiling of a wide variety of interesting statistics, which can also be used for basic time-spent profiling.

• All code is profiled: hardware and software interrupt handlers, kernel modules, the kernel, shared libraries, and applications.
Trace Analysis

• Accurate timing of all system events
  – Which events happened when (micro-second accuracy)?
  – How long did an event take (interrupt latency)?

• Event correlations
  – Which events caused other events?
  – Which class of events transpired?

• Examples
  – LTT, LTTng
  – SystemTap
LTTng (Linux Trace Toolkit next generation)

- LTTng is the kernel tracer that generates traces of an instrumented Linux kernel
- Based on kernel instrumentation and a high-speed relay file system to copy the information from the kernel space to the user space
- Supports native C types and alignment
- Registers new instrumentation sites from dynamically loaded modules on-the-fly
Implications for embedded Linux development

• Analysis applications need to run on device
  • Ensure you have enough RAM on a development device
  • Some application require compiler options (like backtrace and debug info)

• Some analysis tools require kernel support
  • Requires kernel configuration and rebuild
  • New kernel must be bootable on the target

• Data collection is typically stored on the device
  • Ensure you have enough storage for file writing
  • Keep data collection periods as small as possible
  • Clean up log files after analysis
MontaVista Analysis Tools

- Platform Developer Kit
  - Kernel and Application Debugging
  - System Trace
  - System and Application Profiling
  - System Memory Usage with Kernel Details
  - Kernel Function Timing

- Application Developer Kit
  - Application Debugging
  - Application Profiling
  - Application Memory Leak Detection
  - Application Memory Usage
Questions?