Porting Dyninst to VxWorks

Jeff Hollingsworth <hollings@cs.umd.edu>
Ray Chen <rchen@cs.umd.edu>
Larry Michele <lmichele@ieee.org>
VxWorks OS Overview

- **General properties**
  - Designed for embedded systems
  - Hard real-time scheduler
  - Fully preemptive kernel

- **Wide range of deployed environments**
  - Simple embedded systems
    - Consumer hardware
      - Linksys routers
  - Scientific research equipment
    - Aerospace robotics
      - NASA JPL’s Mars rovers
VxWorks OS Overview

- **Architecture support**
  - 68K/CP132, ARM, ColdFire, i960, MIPS, PowerPC, SH, SPARC, x86/Pentium/IA-32, XScale
  - We target PowerPC for the port

- **Highly configurable kernel**
  - Optional components
    - Interactive shell
    - File system
    - Virtual memory
VxWorks Memory Model

● Like UNIX
  - Individual address space for each process
    • Executable data, stacks, heap, etc.
  - Multiple processes can exist in memory simultaneously

● Unlike UNIX
  - Processes do not overlap in virtual memory
  - Advantageous for:
    • Context switching
    • Debugging
    • Cross platform development
VxWorks Applications

● **Real-time process**
  - Full POSIX support except
    - No mmap()
    - No process creation via fork() and exec()
    - No file ownership

● **Kernel Based**
  - Simply another task launched under the kernel
  - No protection from misbehaving application
  - The norm, according to initial research

● **Basic execution unit: Task**
VxWorks Development

- Separate development from runtime
  - Cross compiler on “host” machine
  - Upload binary and execute on “target” machine

- Debugging must involve both systems
  - Functionality provided by Target Agent (target)
  - Physical link managed by Target Server (host)
    - Ethernet, serial, USB, etc.
    - Modular to provide for future
VxWorks Debugging

- One more level of indirection introduced
  - Allows for flexible connection to target
  - Reduction of communication overhead
    • Kernel binary stored in memory
    • Can return cached results to multiple tools
Target Agent

- **Compile kernel with target agent**
  - Akin to compiling with debug information

- **Basic debugging features**
  - Reading/writing task registers
  - Reading/writing process memory
  - Event callback system
    - Task creation/deletion
    - Breakpoints
    - Watchpoints
  - Cache flush/invalidate
Target Agent

- **Advanced features**
  - Loading/launching RTP/kernel tasks
  - Memory disassembler
  - Target function call
  - Symbol query system
    - Includes adding and removing symbols
    - In core memory only
  - Loading modules from host
    - Kernel or real-time process
WTX Protocol

- Protocol for debugging tools
  - Used to send requests to Target Server
- User friendly libraries for 3rd party use
  - C interface libraries provided
  - Integrate easily with modular design of Dyninst
- Allows any Dyninst platform to be a host
  - WTX libraries must exist on platform
Porting Challenges

- **Limited resources**
  - VxWorks designed for embedded systems
    - Minimum 1 MB RAM
    - 4MB is encouraged
  - libsymtabAPI.so alone is 1.5MB
  - Where should our runtime library go?

- **Solution**
  - Leverage Target Agent for all functionality
    - Already required for debugging
  - All other processing can be done on host
Porting Challenges

- **No file system**
  - Nothing extra on the target system
  - No wealth of libraries for dynamic functionality
    - Applications fully-linked when compiled

- **Solution**
  - Load everything needed via WTX
    - Including dynamic loader if needed
  - Limited resources may be problematic
Porting Challenges

- **WTX Protocol functionality gaps**
  - Conspicuous lack of “ps” functionality
  - Cannot perform attach

- **Solution**
  - Write custom kernel module
    - Module with ps functionality: 3Kb
  - Force target to load module and return data

- **Problems**
  - Unsure of size restrictions on kernel module
    - 3Kb may still be too big
Porting Challenges

- Incorrect or incomplete symbol info

<table>
<thead>
<tr>
<th>findSym grid*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: gridPrint</td>
</tr>
<tr>
<td>Name: gridIsBallAt</td>
</tr>
<tr>
<td>Name: gridInit</td>
</tr>
<tr>
<td>Name: gridDeleteBall</td>
</tr>
<tr>
<td>Name: gridAddBall</td>
</tr>
<tr>
<td>Name: grid</td>
</tr>
</tbody>
</table>

- Solution
  - Supplement WTX routines with SymtabAPI
    - Run on host to save target resources

Univertsity of Maryland
Current Status

- WTX functionality fully investigated
- Parsing VxWorks Binary
  - Kernel and executables use ELF
- Loading task for debugging
  - processCreate() functionality working
  - processAttach() functionality working
- Task instrumentation successful
  - Kernel and RTP tasks tested and working
Future Work

- **Move remaining functionality into Dyninst**
  - Mostly involves wrapping WTX library calls
- **Investigate latency of WTX**
  - Mutator effectively 2 indirections away
    - Worse if using a serial line to target
  - Compare to latency of TLB emulation
    - Can it handle heavyweight instrumentation?
    - Eg. Memory instrumentation
    - Can it handle on-demand instrumentation?
Future Work

- Possibility of using WDB directly
  - Reduce to 1 external indirection
- General solution to resource limitation
  - What if raw instrumentation exceeds RAM?