Wind River Sensor Point Technology

Felix Burton

Monitoring/Tracing Workshop
Jan 29-30, 2008
Sensor Points

- Dynamic instrumentation of functions running on “live” devices or systems
  - Instrument applications written in C or C++
  - Instrument kernel, ISRs, and device drivers
  - Instrument third-party code
  - No pre-instrumentation required

- Software instrumentation modules
  - Sensor points written in ANSI-C with custom directives
  - No application, kernel, or third-party source code needed
  - Same scope as any function in which it is inserted
    - Access to local and global variables

- Highly efficient, minimal overhead logging framework

- Minimally intrusive

- Small footprint
Major Use Cases

- Dynamically probe running Systems
  Ex. Retrieve and modify variables, data and execution flow

- Capture system state at point of failure
  Ex. Eliminate need to reproduce in lab

- Rapidly iterate to isolate root causes
  Ex. ‘What if’ analysis w/o recompile/restart

- Patch running equipment
  Ex. ‘hot patch’ to verify fixes before committing to code base

- Enhance QA process
  Ex. Inject faults, Measure performance, Code coverage, simulate I/O
Sensor Point Architecture

1. Dynamically instrument
2. Enable instrumentation
3. Log data

```c
void foo(int param) {
    int state;
    state = globalist[param].state;
    if (state < 10)
        return state;
    return 0;
}
```

```c
static count = 0;
sensorpoint thread {
    sensorpoint "foo"
    on_line(5):
        log($state, count++)
}
```
Sensor Point Execution Path

foo( )

Branch to patch

Sensor point
{
  ....
}

Push the stack

Save registers

Load PatchID

Call sensor point

Restore registers

Pop stack

Original instruction

Branch back
Sensor Point Language (1)

• ANSI C with extensions
  – Language extensions enable Sensor Points to describe instrumentation address, and access symbols in the target application
  – Sensor Points can include all standard C primitives, such as variable and function declarations, type definitions, etc

• Sensor Point Directives
  – sensorpoint
    • The sensorpoint directive describes the context in which subsequent directives execute
  – on_entry
    • Specifies Sensor Point address as the entry of a function, a thread or start of a program (depending on the sensorpoint directive above)
  – on_exit
    • Specifies the exit of a function, a thread or termination of a program
  – on_line and on_offset
    • These directives specify a line number or a hexadecimal offset as the Sensor Point address, within the context of a function
Sensor Point Language (2)

• Target expressions allow Sensor Points to reference objects in target application space

• Example target expressions
  – Access Registers
    • $$EAX, $$r3 : Access registers EAX or r3.
  – Access local and global variables by name
    • $myVar: Access variable myVar in the target application name space
  – Positional parameters
    • $1: Access first of the function call parameters
  – Return value
    • $0 or $return: Set the return value of target function (only in on_exit)

• Stub Function
  – sp_StubRoutine: Skip a function completely (only in on_entry)

• Stack Trace
  – sp_PrintTraceback, sp_LogTraceback: Print or log stack trace for the target function
Nesting of Sensor Points

• Sensor Points can be lexically nested.
  – The ability to nest Sensor Points can be a very powerful feature
  – Sensor Points are nested to control the activation of the Sensor Points and to control the up-scope visibility of data items declared in the Sensor Points

• Nesting allows creation of umbrella for nested Sensor Points.
  – The inner Sensor Point is executed only if the enclosing (umbrella) Sensor Point is active.
Logging facilities

• Logging is designed to be highly efficient and minimally intrusive
  – High performance locking mechanism allows multiple threads to access log buffers with minimum overhead
  – Binary logs to maximize efficiency during logging
  – Constant data is not logged during execution, instead it is inserted during log formatting
  – Simultaneous writing and reading while maintaining data integrity
  – Built-in logging of context information (thread id, time-stamp) for effective log analysis
    • High precision timer (ns) is used when available

• Easy to use log visualization tool
Challenges

- System integration
- Reliable stack walk
- Common log framework
- Variable length instruction patching
WIND RIVER