Abstract

Debugging applications running on an embedded system is very difficult. Often the chips inconsideration has very little or no support for interactive debugging. This is due to the fact the most embedded chips are not a general-purpose processor and lack most of convenience we find on conventional PC.

This paper will present each and every one of the difficulties associated with a non-general purpose limited power chip then show how to work around it. The topics analyzed include the need and lack of debugger, communication facilities, operating system support, and intuitive perhaps visual interfaces. The paper revolves around a project that simulate these problems using a virtual x86 processor with bare environmental support as a target machine. Using examples, this paper will demonstrate each problem along with its workaround.

The goal of this paper is to present approaches taken to deal with specific yet promiscuous problems relating to a certain chip's deficiency. Reader can benefit from this paper as it paves way as fundamental approach and background information when developing software for embedded chips.

Introduction

This paper focuses on how to attach intrusively to a running process on a semi-crippled processing environment, then pausing, watching and manipulating its execution. On the other hand, debugging embedded applications sometimes denote debugging real time problems where a program does not behave well or expected under the real time constraint. That is more so referred to as debugging temporal behaviors in real time systems. This paper does not focus on any temporal behaviors. We'll be focusing on how to build an environment suitable for further use, such as real time debugging.

Usually a semi-crippled processor cannot provide the environments needed to create a flow blown debugging environment. The workaround for this problem is to do remote debugging, where the debugging intelligences runs on a remote and capable machine while the bare minimal debugging agent runs on the target machine. The two communicates to achieve its debugging purpose.

Communication issues arise with remote debugging. On some target machines, an operating system can provide some support for networking and through it the debugging agent can send messages back and forth with the more powerful machine. Other smaller chips may not have the capabilities to run an operating system nor the computing power to do network related tasks. In these cases, the debugger agent will relay on the other communication medium such as a serial port to communicate.

Debugging work that requires a debugging agent to utilize any serial port is tedious. It is best to create a simulator for the chip and debug using the simulator. E.g. machine that runs java programs can relay on other java virtual machine implementations to debug their software. There are numerous publicly available simulators for various chips.

The second problem with intrusive debugging is the need for an operating system. On a chip with operating system support (e.g. some chip may run Linux), there's minimal support for processes along with other things. The agent can simply run as a separate process on the target chip and manipulate the debugged process with the help of the operating system. On the other hand, an environment without operating support requires the debugging agent to be "in-process" with the code of which we want to debug. This is again very tedious.

Then the focus is on how to build an effective remote debugging agent.

Related work

Debugging code on a remote chip is something that all embedded engineers do. There are papers out there that will ease any debugging task by providing tools and information, which can readily apply to the debugging process.

Adequate communication is the bases for remote debugging; as it is the case for most embedded work. Xu [] presents a logic base language for networked agents. The paper discusses the java model as developing communication facilities for mobile agents. The idea can allow us to create a systematic way of communicating by uses of serial ports for debugging purposes. This of course assumes there is not already communication support.

Further, Hendrey [] discusses the uses of Ethernet as embedded network. His paper describes the delays and tolerance one can expect from a multi access device when used in embedded systems. Now chips that do not have network support can consider interfacing with simple Ethernet cards for testing purposes. Again, this proves useful for chips that do not already have some network connectivity.

On machine that already has communication facilities, more sophisticated tasks that involve programs running in some distributed real time constraint may need additional support. This is to say, the engineer need to take note of the temporal aspect of the code in addition to providing communication facilities. There needs to be a way to monitor and control the traffics that are part of the debugging process. Sung [] describes a “backplane” protocol that will monitor and control traffic so that the engineer can reproduce intermittent problems. The use of a backplane as a centralized master traffic control provides reproducible sequence of execution that a debugger may need.

The other necessity for remote debugging remotely is the operating system (or just simply processes) support. Although remote debugging can still be achieved on the target chip without the notion of process, it’s just much simpler to have system support. Also, the industry prefers to use a simulator for simple chips that does not run heavy software like the operating system. The fact that these chips are simple enough suggests that a simulator can be easily written.

Montague [] presented an operating system for the embedded java network computers. The idea is to create a semi-operational operating system (or some threading support) so that people can easily write a debugging agent to run on the target.

Röblitz [] presented the idea of using pthread to simulate embedded kernels. The idea can further develop into creating pthread package for embedded chip. With the minimal threading support engineers can separate the code that is being debugged and the code that does the debugging work.

Now this paper will present a debugging environment created for one particular setup involving a minimal target x86 machine. It will mainly cover the communication and process support problem arising from this setup.

Semi-compiled References

Below is a list reference(not fully compiled) along with some that I might want to consult later on.

Dianxiang Xu, Guoliang Zheng, Xiaocong Fan
http://citeseer.nj.nec.com/xu98logic.html
A logic based language for networked agents (1998)
a Java based tool for developing mobile agent systems

Danny Patel
http://citeseer.nj.nec.com/469972.html
Object-Oriented Design of an Embedded Communication Protocol in UML
Describes communication protocols in UML.

Wonyong Sung and Soonhoi Ha
http://citeseer.nj.nec.com/sung98efficient.html
Efficient and Flexible Cosimulation Environment for DSP Applications (1998)
this paper defines and implements the backplane protocol for communication
and synchronization between client simulators

Uses a "master controller" to watch over all communication
going over the write and perhaps synchronize them to reproduce
problem.

Geoffrey R. Hendrey
http://citeseer.nj.nec.com/hendrey99standard.html
Standard Ethernet as an Embedded Communication Network (1999)
This paper shows what kind of delay performance to expect when Ethernet
is used for embedded networking, shows how to use...

Bruce R. Montague
http://citeseer.nj.nec.com/montague96jn.html
JN: An Operating System for an Embedded Java Network Computer (1996)

Thomas Röblitz, Oliver Bühn, Frank Mueller
http://citeseer.nj.nec.com/538188.html
LegoSim: Simulation of Embedded Kernels over Pthreads
using Pthreads as a means to resemble embedded task execution and
suggests an I/O-based representation of device information.

Lars Albertsson, Peter S Magnusson Computer and Network Architectures
http://citeseer.nj.nec.com/522464.html
Simulation-Based Temporal Debugging Of Linux (2000)
We present a temporal debugger, capable of examining tempora! behaviour
of operating systems

Below are ones I might consult:

http://citeseer.nj.nec.com/kiniry98idebug.html
IDebug: An Advanced Debugging Framework for Java (1998)
Infospheres Debug package (called IDebug [8]) to insert assertions
into the original program code to check the code against the specification

http://citeseer.nj.nec.com/kellomaki94psd.html
Psd a Portable Scheme Debugger (1994)
the source program is transformed into one that behaves as if
run under a conventional debugger.

http://citeseer.nj.nec.com/159506.html
Debugging in Standard ML of New Jersey
This paper describes how to setup and operate emacs as a
interface to the ML debugger.

http://citeseer.nj.nec.com/20306.html
Evaluating Testing Methods by Delivered Reliability (1998)
There are two main goals in testing software: (1) To achieve adequate
quality (debug testing); the objective is to probe the software for
defects so that these can be removed. (2) To assess existing quality

(operational testing); the objective is to gain confidence that the
software is reliable.

http://citeseer.nj.nec.com/liu96communication.html
Communication Issues in Heterogeneous Embedded Systems (1996)

http://citeseer.nj.nec.com/jaramillo99debugging.html
Debugging of Optimized Code through Comparison Checking (1999)
approach to the debugging of optimized code through comparison checking

http://citeseer.nj.nec.com/528882.html
A Heterogeneous and Distributed Co-Simulation Environment (2002)
This paper presents the implementation and evaluation of a hardware and
software co-simulation tool

http://citeseer.nj.nec.com/543545.html
Debugging Parallel Systems:
give an introduction to work presented in the area of debugging large
software systems with modern hardware architectures.

http://citeseer.nj.nec.com/thane00monitoring.html
Monitoring, Testing and Debugging of Distributed Real-Time Systems (2000)
In this thesis we try to remedy these problems by
presenting an integrated approach
to monitoring, testing, and debugging of distributed real-time
systems.

http://citeseer.nj.nec.com/503035.html
Debugging Using Time Machines Replay Your Embedded Systems History
the program is repeatedly reexecuted to track down errors when a failure
has been observed.

http://citeseer.nj.nec.com/528776.html
Simulation-Based Debugging of Soft Real-Time Applications (2001)
The debugger is based on a simulator modelling an entire workstation in
sufficient detail to run unmodified operating systems and applications