The Embedded Machine
Predictable, Portable Real-Time Code

PLDI 2002

Thomas A. Henzinger, Christoph M. Kirsch

UC Berkeley

www.eecs.berkeley.edu/~cm
Mars, July 4, 1997
Lost contact due to embedded software failure
Funding Agencies Like It

Zurich, Switzerland, July, 2001
Autonomous Model Helicopter
Students, too

Berkeley, California, January, 2001
Swarms of Legos
Embedded Software

Environment

Environment Processes

Software Processes

Software
Environment vs. Platform Time

Environment Time

Platform Time

Software
I/O: Drivers

Environment

Software
Computation: Tasks

Environment

Software
Flow of Data

Environment

Software
Flow of Control

Environment

Embedded Machine

Software
Instruction: \texttt{call(a)}
Instruction: \texttt{call}(s)
Instruction: \texttt{schedule}(t)
Instruction: future($g, b$)
Synchronous vs. Scheduled Computation

Environment

b:
- call(a)
- call(s)
- schedule(t)
- future(g, b)

Software

Environment
Synchronous vs. Scheduled Computation

- Synchronous computation
- Kernel context
- Trigger related interrupts disabled

- Scheduled computation
- User context
The Zürich Helicopter
Helicopter Control Software

\[ g : c' = c + 5 \]
Block of synchronous code (nonpreemptable)

Scheduled tasks (preemptable)
b1: call(a_
ctuating)
call(s_ensing)
call(i
put)
schedule(Control [10])
schedule(Navigation [5])
future(g, b2)
E Code

\[ b2: \text{call}(s\_ensing) \]
\[ \text{schedule}(Navigation[5]) \]
\[ \text{future}(g,b1) \]
Platform Timeline: EDF
Platform Time is Platform Memory

- Programming as if there is enough platform time
- Implementation checks whether there is enough of it
Runtime Exceptions I

Environment

Software

call($a$)
Runtime Exceptions II

Environment

Software

\texttt{call(s)}
Runtime Exceptions III

Environment

schedule(t)

Software
An Exception Handler $e$

---

Environment

---

Software

---

b: 
- \text{call}(a)
- \text{call}(s)
- \text{schedule}(t)
- \text{future}(g,b)

\text{schedule}(t,e)

---

e: 
- \text{call}(a')

---

Software
Features

Environment

E Code:
• is *portable* real-time code (if environment-triggered)
• is *predictable* real-time code (if time-safe, or else exceptions)
• can be *linked/patched* (dynamically)

• *changes* perspectives: Schedulability = Program Analysis?
Implementations

- Linux/Windows: POSIX Threads/Semaphores
- OSEKWorks: VxWorks Tasks
- HelyOS: in Kernel, re-entrant interrupts
- LegOS: in Kernel
Dynamic Linking

E Machine

E Code

b:
- call(a)
- call(s)
- schedule(t)
- future(g, b)

Functionality Code

\( g \)

\( t \)
Instructions

Synchronous Driver: call(d)

Scheduled Task: schedule(t)

Triggering: future(g, b)
Time-Safety Checking for Embedded Programs

Thomas A. Henzinger, Christoph M. Kirsch, Rupak Majumdar, Slobodan Majumdar

UC Berkeley

www.eecs.berkeley.edu/~cm
The Time-Safety Game

Environment

Embedded Machine

Scheduler
The schedulability problem of E code is, given an E program and worst-case execution times for all tasks, to check that there is a scheduler so that all resulting traces of the program are time safe.

That is, to check that the scheduler has a winning strategy against the environment and the E machine.
The schedulability problem of E code is EXPTIME-complete (even for time-triggered programs).

Theorem: [Henzinger, Kirsch, Majumdar, Matic 2002]
Example

Scheduler

schedule(t2) if(c,b2) future(g,b3) b1:
schedule(t1) future(g,b4) b2:

b3!
Example

Scheduler

Environment

\begin{align*}
\text{future}(g, b3) \\
\text{if}(c, b2) \\
schedule(t2) \\
schedule(t1)
\end{align*}

b1: schedule(t1)

b2: future(g, b4)

b4!
That’s It