A Thread-Aware Debugger with an Open Interface

Daniel Schulz
Qcentic GmbH
Max-Planck-Str. 39A
50858 Cologne (Germany)

e-mail:
mueller@informatik.hu-berlin.de
WWW:
http://www.informatik.hu-berlin.de/~mueller

Frank Mueller
Institut für Informatik
Humboldt-Universität zu Berlin
10099 Berlin (Germany)
Overview

- motivation
- design
- active debugging
- debugger components
- implementation
- evaluation
Motivation

- SMP programming $\rightarrow$ Threads
- POSIX Threads: many implementations
  - debugging support: no standardization
  - debugging difficult:
    - interleaving of control flows
    - sync. + async. suspension/resumption
    - partial ordering of executions (synchronization)

<table>
<thead>
<tr>
<th></th>
<th>traditionally</th>
<th>desirable</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakpoints</td>
<td>for all flows</td>
<td>thread-specific</td>
</tr>
<tr>
<td>synchronization</td>
<td>state invisible</td>
<td>inquiries</td>
</tr>
<tr>
<td>scheduling breakpoints</td>
<td>implicit</td>
<td>optional control</td>
</tr>
<tr>
<td>breakpoints</td>
<td>explicit</td>
<td>at context switches</td>
</tr>
</tbody>
</table>
- portability: open interface for debugging threads
- extensibility: query API application ↔ debugger
- flexibility: varying functionality
- activation: optional shared library

⇒ TDI: thread debug interface (generic)
⇒ TED: Thread extensions for debugging (implementation-dependent)
Design Options for Encapsulation

(a) Non-Generic Design

(b) Generic Design
Active Debugging

- enhance application code
- collect information / manipulate execution

Passive Debugging (traditional)
- enhance debugger
- probe application

<table>
<thead>
<tr>
<th>Issue</th>
<th>Active Debugging</th>
<th>Passive Debugging</th>
</tr>
</thead>
<tbody>
<tr>
<td>details of thread implementation known to debugger?</td>
<td>No (transparent)</td>
<td>Yes</td>
</tr>
<tr>
<td>change/add new thread impl.</td>
<td>no changes</td>
<td>enhance debugger</td>
</tr>
<tr>
<td>extract info from application</td>
<td>declarative</td>
<td>procedural</td>
</tr>
<tr>
<td>query overhead</td>
<td>lower, no redundancies</td>
<td>higher, redundant requests</td>
</tr>
<tr>
<td>post-mortem thread debugging</td>
<td>not possible</td>
<td>always</td>
</tr>
</tbody>
</table>
TED: Thread Extensions

- **uniform access** to internal thread ADTs
- **set manipulation** primitives
  - \( S_r : T_{DO} \to T_{DA} \) (read)
  - \( S_w : T_{DA} \times T_{DO} \to T_{DA} \) (write)
  - for types \( T \) of address domain \( DA \) / object domain \( DO \)
- **map objects** (thin layer) onto
  - an existing thread API or
  - a debug extension API
- **call-outs** thread API → TED
  - register objects
  - update object relations
- **interface** for
  - set iteration \( \to DA \)
  - attribute access \( \to \mathbb{N} \cup \{\text{NULL}\} \)
Booch Class Diagram of Object Classes

Thread
- Identifier
- Address of self
- Priority
- State
- User Function

WaitFor
- n

BlockedOn
- 1
- n

OwnedBy
- 1

Mutex
- Identifier
- Address of self

Condition Variable
- Identifier
- Address of self

SignaledBy
- 1

A Thread-Aware Debugger with an Open Interface

ISSTA’00
TDI: Thread Debug Interface

- abstracts from debugger and thread impl.
- keeps database of application’s state
- TED registers with TDI (thin interface):
  - object updates / queries
  - iteration / attribute access
  ⇒ all of these or subset
- supports persistent identifiers (e.g., thread IDs)
- communicates consistent state to debugger
  ⇒ uniform, extensible database query language
  - selections of relations with values
  - projections of relations with assignments
  ⇒ queries clustered → remove redundancies
  - responses reduced → no copies
Communication Structure

- active debugging ⇒ exchange more data
- OS support for debugging limits probe bandwidth
  ⇒ use IPC for query/response

- application stopped at breakpoint ⇒ cannot serve query
  ⇒ debugger calls handler function in application

- repeated calls for large responses ⇒ many context switches
  ⇒ fork child in application
- child fills IPC buffers ⇒ avoids consecutive calls
Communication between Debugger and Application

Target Process | IPC-Channel (buffered) | Debugger
---|---|---
```
call Target.call(<RequestHandler>)
call IPC.write(<TED-Request>)
call return
```  
```
call IPC.read()
call return
```  
```
call Target.call(<RequestHandler>)
call IPC.write(<Result #1>)
call return
```  
```
call IPC.read()
call return (Result #1)
call IPC.read()
call return (Result #n)
call IPC.read()
call return (Result #n)
call Target.call(<RequestHandler>)
call IPC.write(<Result #n>)
call return
```  
```
call fork
```  
```
call Request Handling
```  
```
call IPC.write(<TED-Request>)
call Request>
call return call
```  
```
call IPC.read()
call return (Result #1)
call IPC.read()
call return (Result #2)
call IRC.read()
call return (Result #2)
call IPC.read()
call return (Result #n)
call IPC.read()
call return (Result #n)
```  
```
) mutually exclusive execution
```  
```
(b) parallel execution
```
Debugger Extensions

- IPC interface to TDI client
- new user commands
- query / response handshake:
  - issue query
  - then call TDI server handler
  - TDI server parses query
  - updates state using TED
  - formats and sends response
**Implementation**

- GDB 4.18
- LinuxThreads, Solaris Threads, FSU Pthreads, MIT Pthreads
- application bound to support dynamic linking
- TDI server as DLL, only activated if debugged (flag in TED)
- Problem: assure consistent state
  - skew of TDI execution ⇒ deal with probe effect
  - event notification ⇒ postpone signals
  - blocking calls ⇒ replacement calls
⇒ must prevent preemption / suspension during TDI activity
Signal Handling during Active Debugging

POSIX Threads

Scheduling

pthread_debug_TDI_sig_ignore

off on

Record

Debugger

ptrace(POKE..., ...)

kill(pid, SIGALRM)

ptrace(PEEK..., ...)

pthread_debug_TDI_ignored_signals
Response Times: IPC vs. Ptrace

- Response time [sec]
- Instantiated threads

- ptrace() line
- TDI line
queries generated
- by TDI due to user commands
- explicitly by experienced user

thread:id,entry,state:state == 1 || mbo == 0 (1)
thread:id,prio=10,state: (prio+10<20) && cvwf ! =0x10 (2)

- parsed, transformed into tuples
- handled by query evaluator

responses translated into symbolic representation

8 804b238 2 #7 804b238 2 #10 804b238 1
Thread-Specific Breakpoints

- hit breakpoint
- check running thread ID (depends on thread impl.)
- upon mismatch, clean up and reset breakpoint
- resumption (2) may accept signals (and context switch)
  ⇒ disable signals during (2)

- breakpoint at context switch
  ⇒ trap at nextPC of all suspended threads

- forced suspension
- signal application ⇒ invoke scheduler
Resetting a Breakpoint

Breakpoint a[n] hit

1. Reset

2. Resume

3. Reset

Ready to Resume Execution
Thread-Aware Debugging

- extensive thread info
- state of synchronization
- thread-specific breakpoints
- explicit suspension/resumption
- thread-specific stack trace
- breakpoint on next context switch
- thread-specific step/next

- performance overhead

<table>
<thead>
<tr>
<th>Program</th>
<th>No Debugging</th>
<th>GDB-TDI</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>fft</td>
<td>14 sec</td>
<td>16 sec</td>
<td>12.5%</td>
</tr>
<tr>
<td>barnes</td>
<td>33 sec</td>
<td>40 sec</td>
<td>17.5%</td>
</tr>
</tbody>
</table>
Related Work

- Mach debugger
- SmartGDB
- GDB 4.18
- Solaris
- Partop
- path expressions/actions
- HPDF: debug command interface
- MPI message display / TotalView
- Panorama
- KDB
- Fast breakpoints
- Relational query debugging

⇒ Our work: active debugging + relational queries for debugging threads, functionality, portability
**Conclusion**

- open interface for debugging
- thin layer (extension to thread impl.)
- thread-aware debugging facilities → new features
- implemented in GDB
- paradigm of active debugging
- language-independent protocols for communication
- relational query model
- supports partial or complete TEDs
- sample impl. for variety of thread impl. types
- improved efficiency and portability
- download: [http://www.informatik.hu-berlin.de/~mueller/TDI](http://www.informatik.hu-berlin.de/~mueller/TDI)