A Thread-Aware Debugger with an Open Interface

Daniel Schulz

Qcentic Gmbh
Max-Planck-Str. 39A
50858 Cologne (Germany)

Frank Mueller

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

e-mail:

mueller@informatik.hu-berlin.de

WWW:

`http://www.informatik.hu-berlin.de/~mueller`

Overview

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

Motivation

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

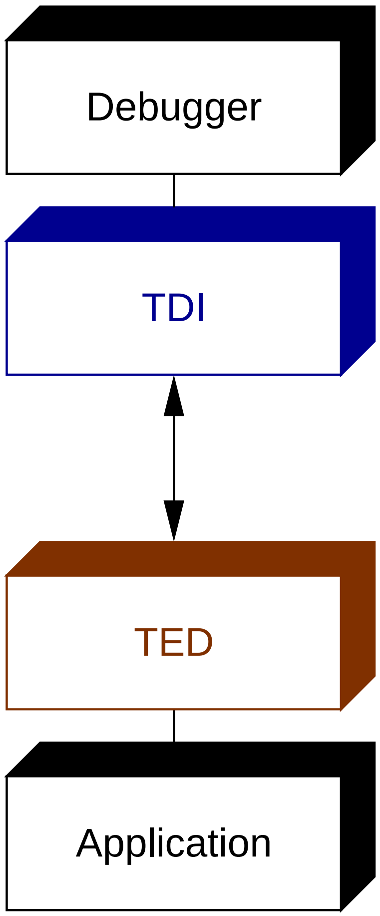
	traditionally	desirable
breakpoints	for all flows	thread-specific
synchronization	state invisible	inqueries
scheduling	implicit	optional control
breakpoints	explicit	at context switches

Design

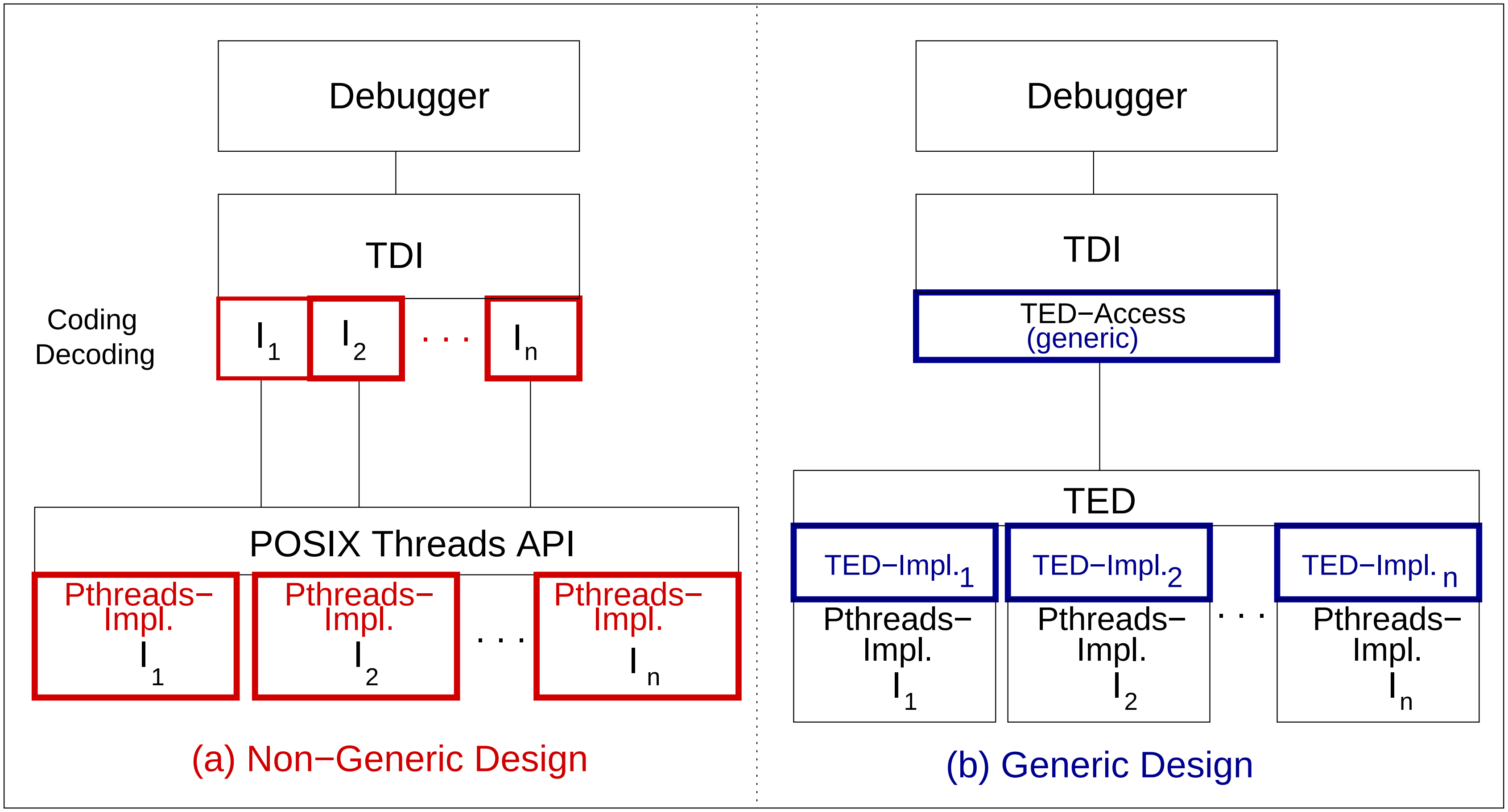
- portability: open interface for debugging threads
- extensibility: query API application $\leftrightarrow$ debugger
- flexibility: varying functionality
- activation: optional shared library

$\Rightarrow$ TDI: thread debug interface (generic)

$\Rightarrow$ TED: Thread extensions for debugging (implementation-dependent)



Design Options for Encapsulation



Active Debugging

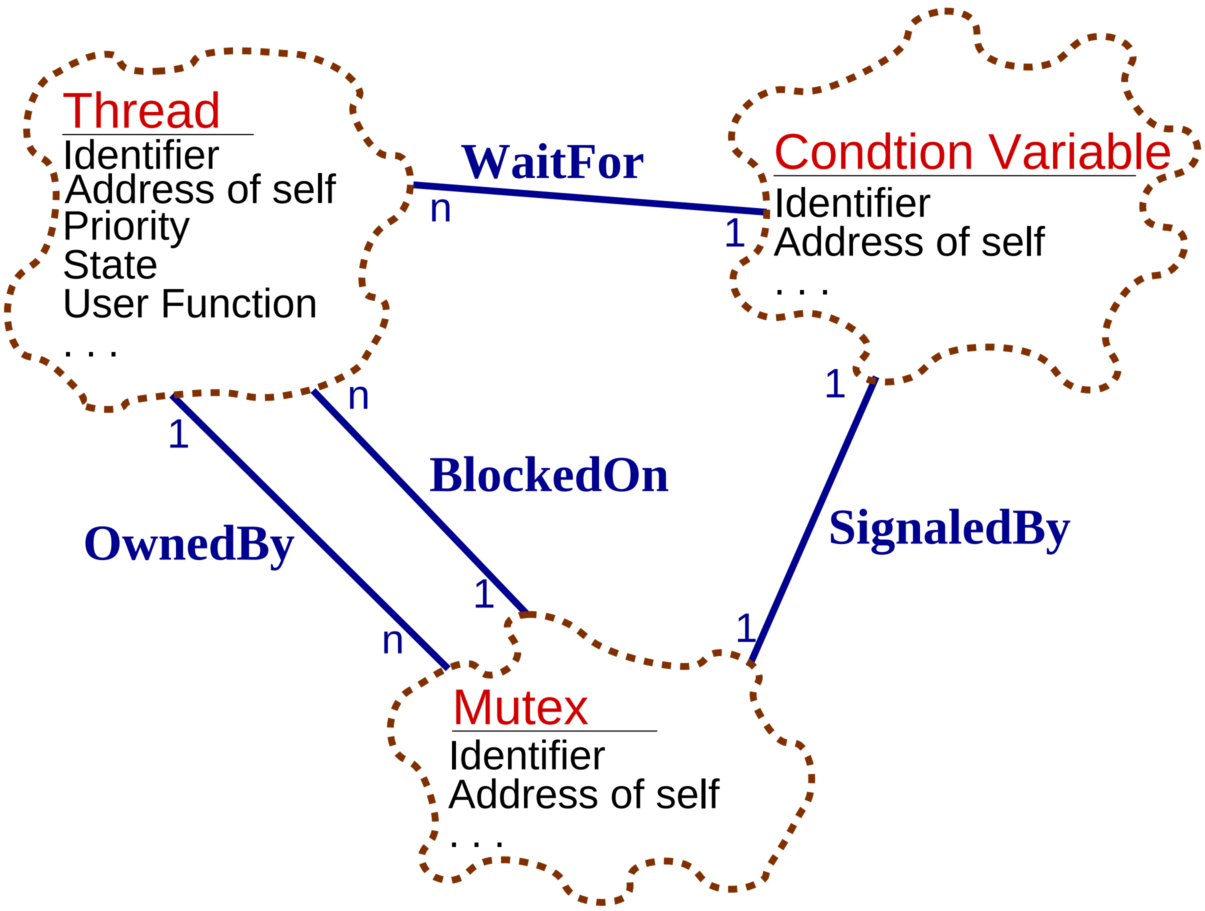
- Active Debugging
  - enhance application code
  - collect information / manipulate execution
- Passive Debugging (traditional)
  - enhance debugger
  - probe application

Issue	Active Debugging	Passive Debugging
details of thread implementation known to debugger?	No (transparent)	Yes
change/add new thread impl.	no changes	enhance debugger
extract info from application	declarative	procedural
query overhead	lower, no redundancies	higher, redundant requests
post-mortem thread debugging	not possible	always

TED: Thread Extensions

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

Booch Class Diagram of Object Classes



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

  $\Rightarrow$ all of these or subset
- supports persistent identifiers (e.g., thread IDs)
- communicates consistent state to debugger

$\Rightarrow$ uniform, extensible database query language
  - selections of relations with values
  - projections of relations with assignments

  $\Rightarrow$ queries clustered $\rightarrow$ remove redundancies
  - responses reduced $\rightarrow$ 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

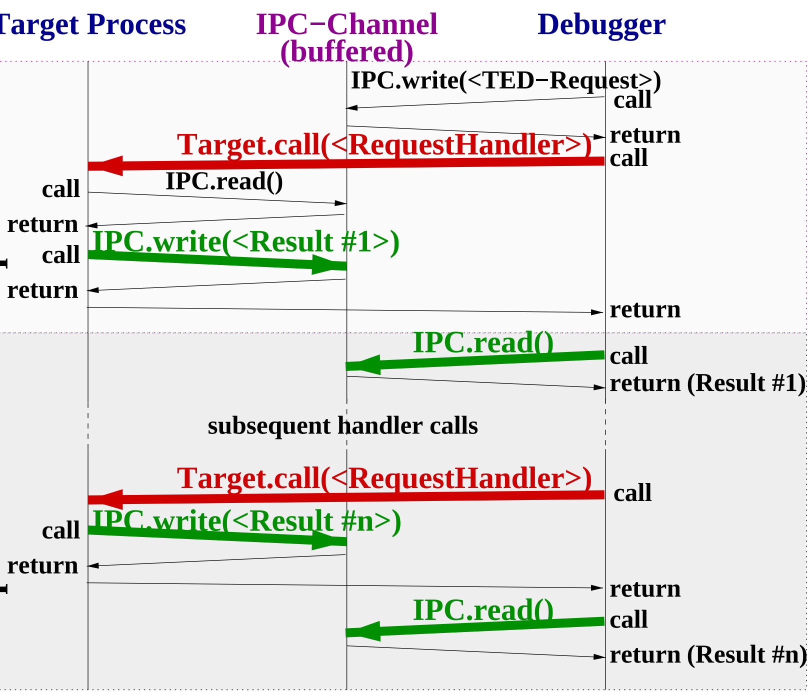
⇒ debugger calls handler function in application

- repeated calls for large responses ⇒ many context switches

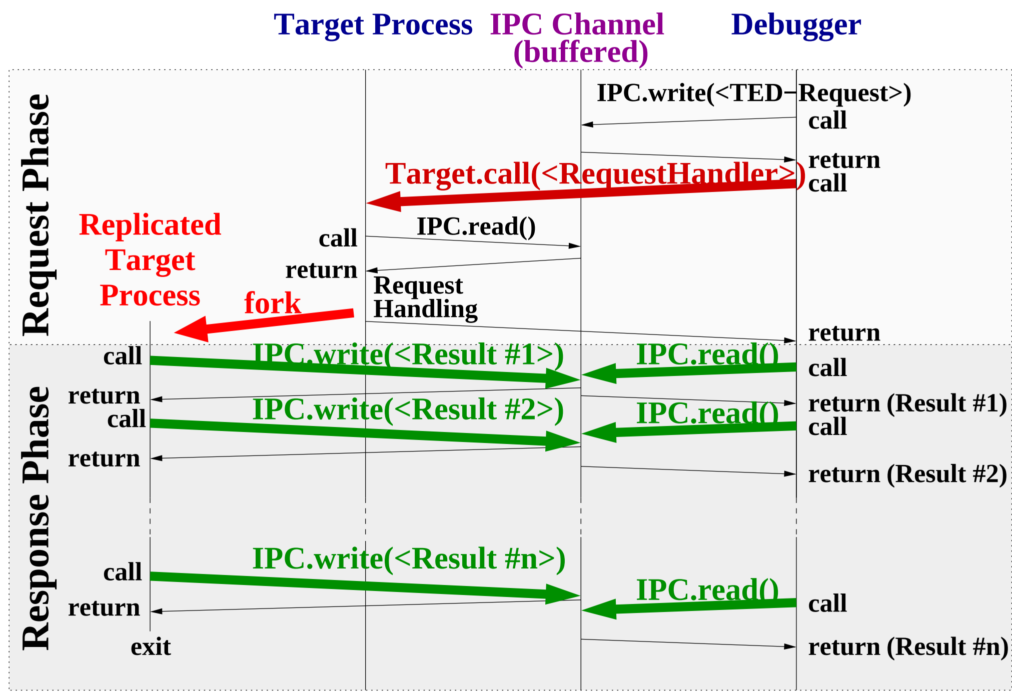
⇒ fork child in application

- child fills IPC buffers ⇒ avoids consecutive calls

Communication between Debugger and Application



) mutually exclusive execution



(b) parallel execution

Debugger Extensions

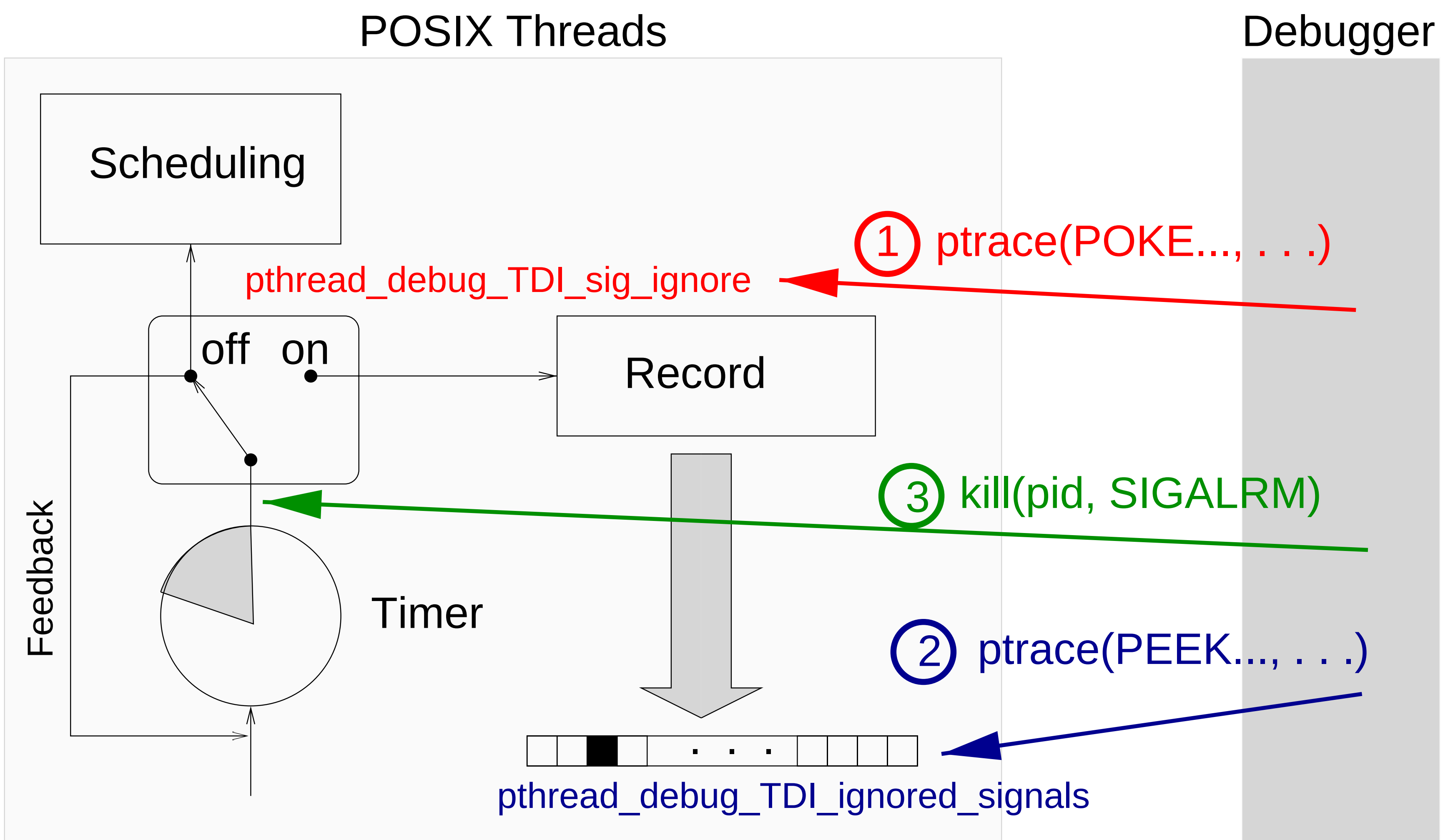
- IPC interface to TDI client
- new user commands
- query / response handshake:
  - issue query
  - then call TDI server handler
  - TDI server parses query
  - updates state using TED
  - formats and sends repsonse

Implementation

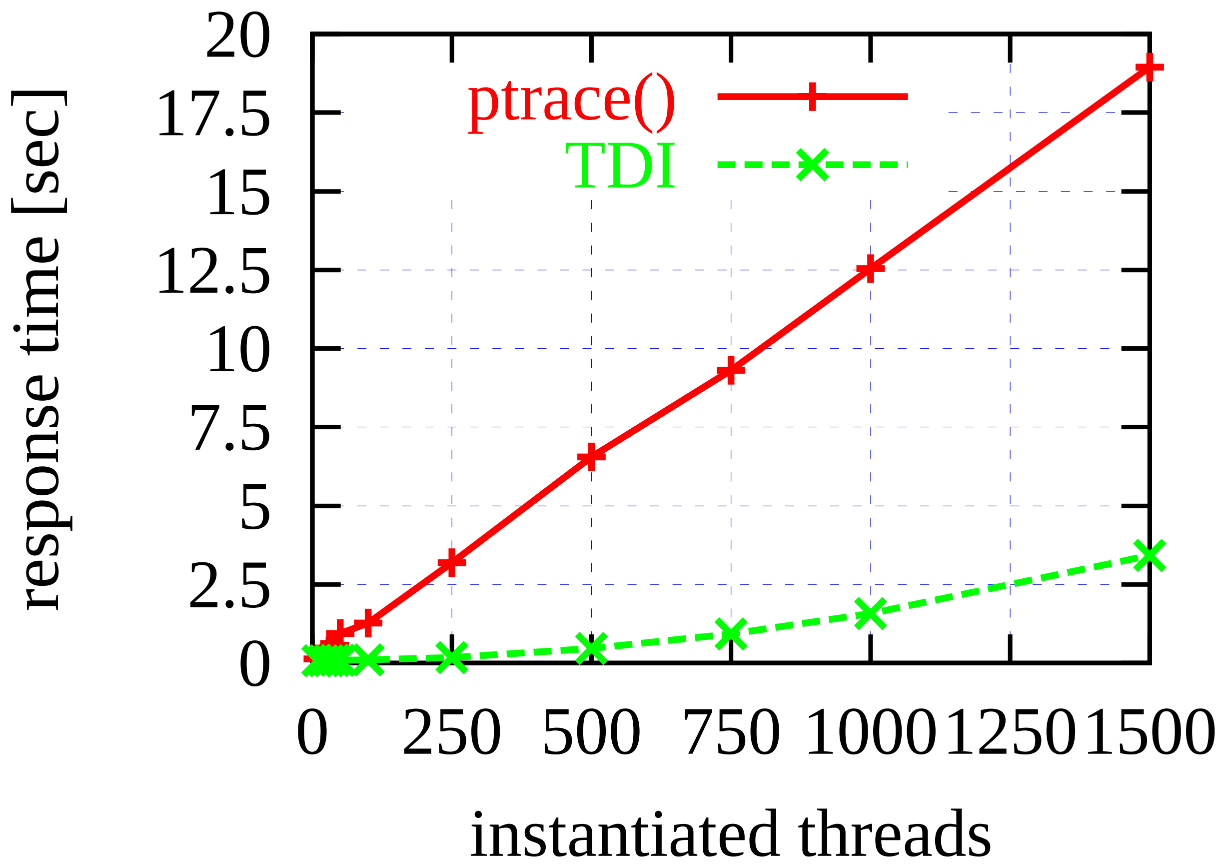
- GDB 4.18
- LinuxThreads, Solaris Threads, FSU Pthreads, MIT Pthreads
- application bound to support dynamic linking
- TDI server as DLL, only activated if debugged (flag in TED)
- Problem: assure consistent state
  - skew of TDI execution ⇒ deal with probe effect
  - event notification ⇒ postpone signals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

⇒ must prevent preemption / suspension during TDI activity

Signal Handling during Active Debugging



Response Times: IPC vs. Ptrace



Query Language

- 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 contex switch)

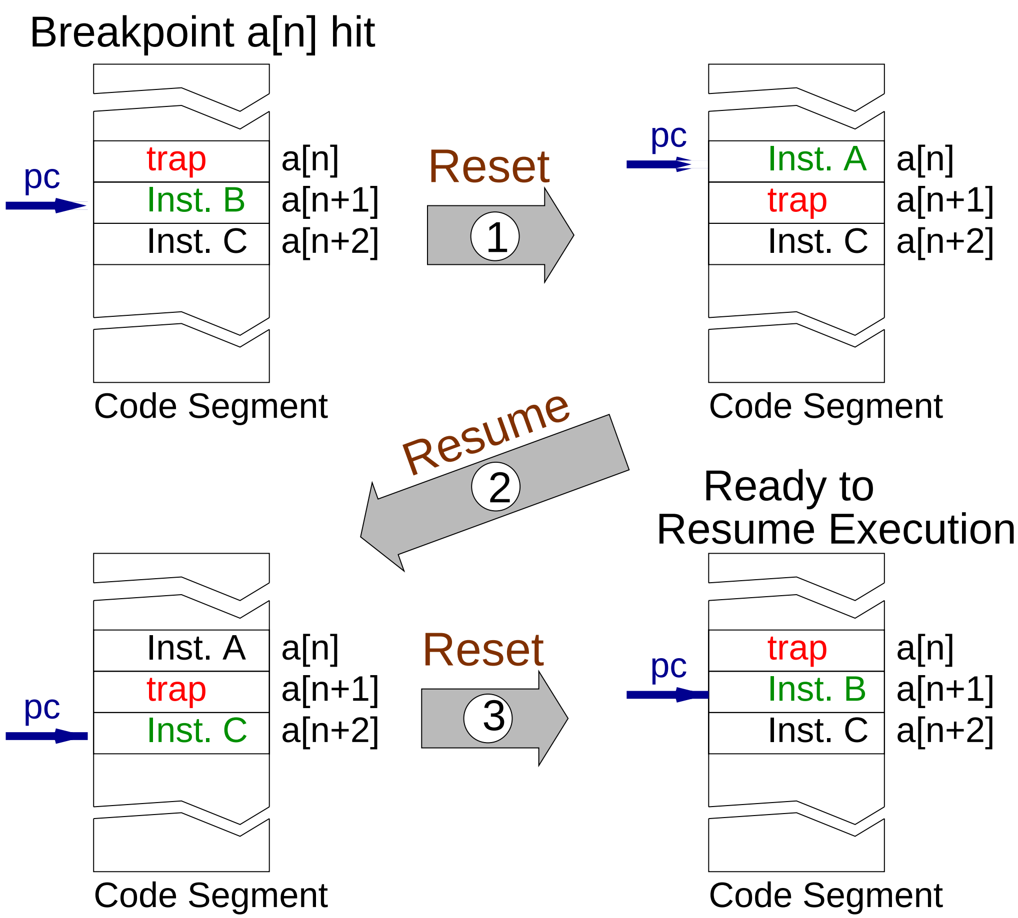
$\Rightarrow$ disable signals during (2)

- breakpoint at context switch

$\Rightarrow$ trap at nextPC of all suspended threads

- forced suspension
- signal application $\Rightarrow$ invoke scheduler

Resetting a Breakpoint



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

Program	No Debugging	GDB-TDI	Overhead
fft	14 sec	16 sec	**12.5%**
barnes	33 sec	40 sec	**17.5%**

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`