

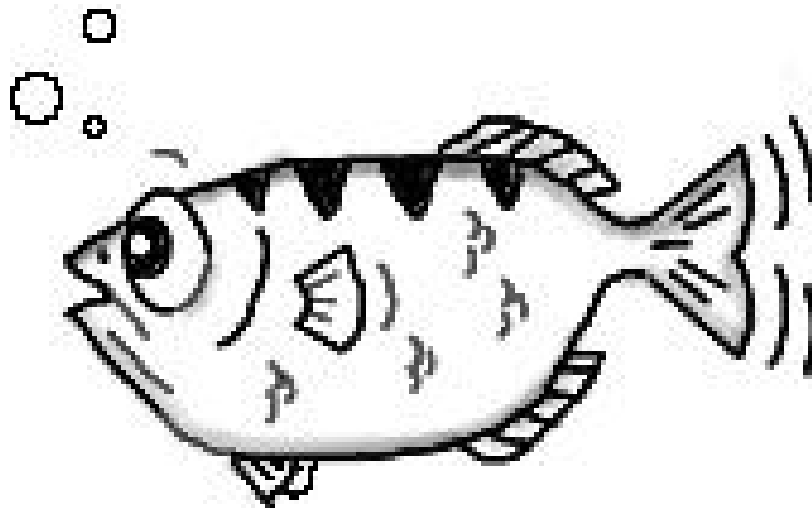
The Multicore GDB

BoF

The Multicore GDB BoF

Pedro Alves <palves@redhat.com>

GNU Cauldron 2018



Outline

- Multi target

Why?

- Combined host/accelerator|coprocessor debugging
 - GPGPU debugging? (CPU target + GPU target)
- Debug multiple embedded boards as multiple inferiors
 - requested on the gdb@ list a couple days ago! (errr, last year...)
- Combined Client+Server debugging
- Distributed computing (HPC debugging)
 - debug multiple nodes, potentially different archs
- Valgrind follow-fork/follow-exec
- Because it's cool?

GDB's evolution

- GDB starts single-process debugging only (1734)
 - Multi-threading (1830)
 - Multi-process (2010)
 - Multi-target (~~2017~~ 2018)

GDB's little brain

- inferior list is a global list
- thread list is a global list
- ptid_t is pervasive
- there's only one target stack
 - historical "current_target" squashed target
 - each target_ops instance has "beneath" pointer
 - targets > process_stratum skip "wrong" inferiors

GDB's little brain, after

- target_ops -> C++ class hierarchy + virtual methods [MERGED]
- inferior list is still a global list
- each inferior has its own thread list [SUBMITTING]
- target stack is now an array of target_ops pointers [MERGED]
- each inferior has its own target stack array
- squashed "current_target" is gone -> inf->m_stack.top (); [MERGED]
- "target_ops::beneath" pointer is gone m_stack.beneath (target); [MERGED]
- ptid_t remains the same
- ptid_t => thread_info * in many places [MERGED, most]
- ptid_t => 'target_ops *' + ptid_t in other places

target stack, after

```
class target_stack
{
public:
    void push_target (struct target_ops *);
    int unpush_target (struct target_ops *);

    target_ops *at (enum strata stratum) { return m_stack[stratum]; }
    target_ops *top () { return at (m_top); }
    target_ops *find_target_beneath (const target_ops *t);

private:
    enum strata m_top {};
    target_ops *m_stack[(int) debug_stratum] {};
};
```


Status

- Requires target_async-capable targets
- Non-stop native + gdbserver works
- Non-stop gdbserver (1) + gdbserver (2) works
- All-stop works, as long as all target backends are non-stop:
 - "maint set target-non-stop on"
- all-stop without all-stop-on-top-of-non-stop not attempted
- native + gdbserver + core works too, for fun
- Testsuite ~~not~~ regression-free. New tests pass, hurray!
 - required revamping target_terminal_inferior/target_terminal_ours...
- ~~Several hacks in place~~

User interface, threads

```
(gdb) info threads
```

Id	Target Id	Frame
1.1	Thread 8284.8284 "server"	main () at server.c:70
* 2.1	Thread 8287.8287 "client"	main () at client.c:66

User interface, inferiors

```
(gdb) info inferiors
```

Num	Description	Connection	Executable
1	process 8284	1 (extended-remote :20000)	/tmp/server
* 2	process 8287	2 (extended-remote :20001)	/tmp/client
3	process 11617	3 (core)	/tmp/threaded-core
4	<null>	2 (extended-remote :20001)	

User interface, "info connections"

```
(gdb) info connections
```

```
  Num  Description
  1    extended-remote :20000
*  2    extended-remote :20001
  3    core
```

User interface, new "connections"

```
(gdb) info inferiors
```

Num	Description	Connection	Executable
* 1	process 8284	1 (native)	/tmp/server
2	process 8287	2 (extended-remote :20001)	/tmp/client

```
(gdb) add-inferior
```

```
Added inferior 3 on target 1 (native)
```

```
(gdb) inferior 2
```

```
[Switching to inferior 2 [process 8287] (/tmp/client)]
```

```
(gdb) add-inferior
```

```
Added inferior 4 on target 2 (extended-remote :20001)
```

User interface, the new "connections"

```
(gdb) info inferiors
```

Num	Description	Connection	Executable
1	process 8284	1 (extended-remote :20000)	/tmp/server
* 2	process 8287	2 (extended-remote :20001)	/tmp/client
3	<null>	4 (native)	
4	<null>	2 (extended-remote :20001)	

```
(gdb)
```