GDB Tracepoints for the Linux kernel

Jim Blandy
CodeSourcery, LLC
Why can't I use GDB to debug the Linux kernel?
Why can't I use GDB to debug the kernel?

It is morally wrong to use a debugger. Use printk.
Why can't I use GDB to debug the kernel?

Debuggers facilitate observation.
Why can't I use GDB to debug the kernel?

You need a second machine.
What are tracepoints?
What are tracepoints?

GDB-based source-level debugging
What are tracepoints?

- GDB-based source-level debugging
- Minimally intrusive
What are tracepoints?

- GDB-based source-level debugging
- Minimally intrusive
- Can debug the kernel GDB itself is running under
Breakpoints vs. Tracepoints

Breakpoints stop the program, while you inspect its state.
Breakpoints vs. Tracepoints

皆さん Breakpoints が、プログラムを停止し、それの状態を検査することを許可します。

皆さん Tracepoints は、プログラムを一時停止し、情報ログを記録し、その後再始動することを許可します。
Breakpoints vs. Tracepoints

Breakpoints stop the program, while you inspect its state.

Tracepoints pause the program, log information, and then continue.

In GDB, a selected log hit becomes “the current state of the program”.
Breakpoints vs. Tracepoints

Breakpoints stop the program, while you inspect its state.

Tracepoints pause the program, log information, and then continue.

In GDB, a selected log hit becomes “the current state of the program”.

You choose the information to log ahead of time.
Demo #1
How does it work?
Tracepoint Implementation

GDB compiles source-language expressions to bytecode
Tracepoint Bytecode

(gdb) maintenance agent file->f_dentry->d_iname
  0  reg 0
  3  zero_ext 32
  5  const8 8
  7  add
  8  trace_quick 4
 10  ref32
 11  const8 108
 13  add
 14  trace_quick 36
 16  pop
 17  end
(gdb)
Tracepoint probes

kprobes makes it easy to patch tracepoint handlers into code stream
Tracepoint probes

- kprobes makes it easy to patch tracepoint handlers into code stream
- Passes registers to handler as a struct pt_regs
Tracepoint probes

.kafka takes a function `kprobes_add_kprobe` to construct a probe on any function.

- The probe adds a function that `kprobes_add_kprobe` inserts before the function.
- The function provides a `struct pt_regs` that allows your code to inspect the function's register values.

Note: This is not always true due to implementation details.
Tracepoint Hit Log
Tracepoint Hit Log

In kernel memory
Tracepoint Hit Log

- In kernel memory
- Each entry records:
Tracepoint Hit Log

- In kernel memory
- Each entry records:
  - Which tracepoint was hit
Tracepoint Hit Log

- In kernel memory
- Each entry records:
  - Which tracepoint was hit
  - Register values
Tracepoint Hit Log

- In kernel memory
- Each entry records:
  - Which tracepoint was hit
  - Register values
  - Contents of all memory touched by tracepoint's bytecode expressions
Tracepoint Hit Log

🔍 In kernel memory
🧐 Each entry records:
   🔍 Which tracepoint was hit
   🔍 Register values
   🔍 Contents of *all* memory touched by tracepoint's bytecode expressions
🔍 SMP-safe
Bad /proc interface

Essentially passes GDB remote protocol packets via write calls, responses via read calls on /proc/gdb-tracepoints
Bad /proc interface

Essentially passes GDB remote protocol packets via write calls, responses via read calls on /proc/gdb-tracepoints

Can be controlled by shell scripts (Python!)
Bad /proc interface

Essentially passes GDB remote protocol packets via write calls, responses via read calls on /proc/gdb-tracepoints

Can be controlled by shell scripts (Python!)

Ought to be sysfs/kobject-based
Cute Hack #1

(Due to the inimitable Michael Snyder)
Cute Hack #1

Log holds raw memory, not expression results
Cute Hack #1

Log holds raw memory, not expression results

Selecting a hit makes those regs and memory contents 'current' to GDB
Cute Hack #1

- Log holds raw memory, not expression results
- Selecting a hit makes those regs and memory contents 'current' to GDB
- So they can be reinterpreted in more helpful ways
Cute Hack #2

(Also due to the inimitable Michael Snyder)
Cute Hack #2

```c
struct gtp_hit
{
    spinlock_t lock;
    int number;
    struct gtp_tracepoint *tracepoint;
    size_t entries_used;
    int error;
    struct pt_regs regs;
    size_t num_bytes;
    unsigned char bytes[];
};
```
Cute Hack #2

One tracepoint hit structure (with tail) holds all the memory logged for a given tracepoint hit.
Cute Hack #2

One tracepoint hit structure (with tail) holds all the memory logged for a given tracepoint hit.

A hit may hold any number of blocks of memory, each possibly from a different address, and of a different length.
Cute Hack #2

```c
struct gtp_hit
{
    spinlock_t lock;
    int number;
    struct gtp_tracepoint *tracepoint;
    size_t entries_used;
    int error;
    struct pt_regs regs;
    size_t num_bytes;
    unsigned char bytes[];
};
```
Cute Hack #2

When we log a hit, we log *all* the bytes it refers to, traced or not, in the order the interpreter requests them.
Cute Hack #2

☞ When we log a hit, we log *all* the bytes it refers to, traced or not, in the order the interpreter requests them.

☞ When we query a hit, we re-evaluate the expression, handing out the next block of bytes as the interpreter requests them.
Cute Hack #2

 сохраним все байты, которые оно ссылается, указывая или нет, в порядке, в котором они были запрослены интерпретатором.

 Когда мы стараемся получить значение по умолчанию, мы пересчитываем выражение, выдавая следующую порцию байтов, как требует интерпретатор.

 Эта два интерпретатора синхронизированы, поэтому они требуют одинаковых блоков.
Credits

Michael Snyder
Nicholas McGuire
Thank you!

http://www.red-bean.com/jimb