redhat.

Debugging Hung Python Processes With GDB

John Schwarz
Senior Software Engineer, Red Hat.
May 2, 2016

(Based on work by Brian Bouterse and David Malcolm, Red Hat)

Why are we here?

There are (roughly) 3 type of programs to debug:

1. “Program doesn't do what I want!”
   - Debug with prints/pdb...
2. “Program crashed!”
   - Inspect traceback, goto 1.
3. “My program seems stuck and **I don't know** what it's doing!”
   - ???

What will we learn?

- Tools to help you know **what** the program is doing?
- Won't teach you **how** to debug
- Still need to think about the **why**

Who Am I?

- Python user since 2008
- Working at Red Hat since 2014
- Before that – Intelligence Core, IDF
- Contributes for OpenStack (Neutron)
- Coding Enthusiastic
- Problem Solver (race conditions, deadlocks...)

pdb

- Allows for easy, gdb-like interface with code

- Requirement: put a breakpoint & restart program
  - Cannot attach!

```
import pdb; pdb.set_trace()
```

- This is also a problem

pdb commands

- list — Print surrounding source code
- bt — Print program's backtrace
- print — Print variable or function’s return value
- up, down — Move up and down the stack
- Can also run code!

rpdb (as in “remote pdb”)

- Same interface as pdb, installable with pip
- Requirement: put a breakpoint & restart program:

```
import rpdb
rpdb.Rpdb(port=1337).set_trace()
```

- Connect with ‘telnet’

Trigger rpdb.set_trace() with SIGTRAP

```
# Trigger rpdb.set_trace() on SIGTRAP with
# specified IP/port
rpdb.handle_trap("0.0.0.0", 1337)
```

- Recent versions already have it build in
- Problem?

strace (syscall tracer)

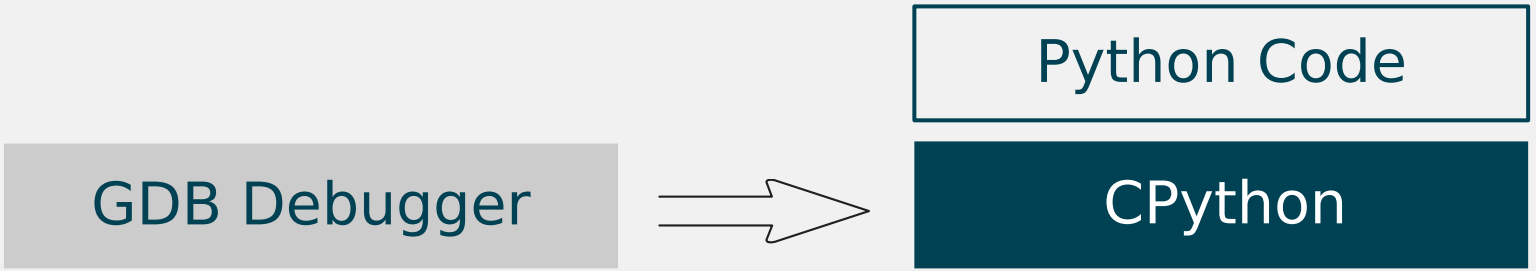
- Success

  open("/dev/null", O_RDONLY) = 3
- Failure

  open("/foo/bar", O_RDONLY) = -1 ENOENT (No such file or directory)
- Blocking

  select(1, [0], NULL, NULL, NULL

Conceptual Model



Why use GDB for Python?

- Production application where pdb can't go
- Remote applications where rpdb isn't available
- “My program seems stuck and **I don't know** what it's doing!”
  - Solution: use GDB!

GDB Basics

- Connect to a running process: `gdb -p <pid>`
- `c` to continue
- Ctrl+C to stop execution again
- Ctrl+D to detach (which continues)

GDB Commands

- list — Print surrounding source code
- bt — Print program’s backtrace
- print — Print variable or function’s return value
- up, down — Move up and down the stack
- Problem?

GDB commands

- list — Print surrounding source code
- bt — Print program’s backtrace
- print — Print variable or function’s return value
- up, down — Move up and down the stack
- Problem?

example1.py

```
import os
import time

def foobar(amount):
    time.sleep(amount)

def main():
    print "Hello, World! My pid is %d" % os.getpid()
    foobar(amount=1337)
    print "Bye, World!"

if __name__ == '__main__':
    main()
```

A function call in CPython

```
#8  0x00007ff43137e666 in fast_function (nk=<optimized
    out>, na=0, n=0, pp_stack=0x7ffd25b961a0,
    afunc=<function at remote 0x7ff43172d6e0>)
    at /usr/src/debug/Python-2.7.10/Python/ceval.c:4196

#9  call_function (oparg=<optimized out>,
    pp_stack=0x7ffd25b961a0)
    at /usr/src/debug/Python-2.7.10/Python/ceval.c:4131

#10 PyEval_EvalFrameEx (f=f@entry=Frame 0x7ff43185fc20
    for file example1.py, line 10, in <module> (),
    throwflag=throwflag@entry=0)
    at /usr/src/debug/Python-2.7.10/Python/ceval.c:2753
```

Calling into the kernel

```
#0  0x00007ff4306add43 in __select_nocancel ()
    from /lib64/libc.so.6
#1  0x00007ff42fe2ffc0 in floatsleep (secs=<optimizedout>) at
    /usr/src/debug/Python2.7.10/Modules/timemodule.c:948
#2  time_sleep (self=<optimized out>, args=<optimized out>) at
    /usr/src/debug/Python-2.7.10/Modules/timemodule.c:206
#3  0x00007ff43137e8be in call_function
    (oparg=<optimized out>, pp_stack=0x7ffd25b95f40) at
    /usr/src/debug/Python-2.7.10/Python/ceval.c:4110
#4  PyEval_EvalFrameEx (f=f@entry=Frame 0x7ff431738050,
    for file example1.py, line 6, in bar (),
    throwflag=throwflag@entry=0)
    at /usr/src/debug/Python-2.7.10/Python/ceval.c:2753
```

Python extensions for GDB

Python extensions for GDB

- py-list — Print surrounding *python* source code
- py-bt — Print *python* stack trace
- py-print — Print *python* variables
- py-up, py-down — Move up and down the *python* stack
- py-locals — Print all *python* locals

`py-list` output of example1.py

```
(gdb) py-list

   1    #!/usr/bin/env python
   2    import os
   3    import time
   4
   5    def foobar(amount):
  >6        time.sleep(amount)
   7
   8    def main():
   9        print "Hello, World! My pid is %d" % os.getpid()
  10        foobar(amount=1337)
  11        print "Bye, World!"
```

`py-bt` output of example1.py

```
(gdb) py-bt

#4 Frame 0x7f547357d3c0, for file ./example1.py,
        line 6, in foobar (amount=1337)
    time.sleep(amount)

#8 Frame 0x7f547357d050, for file ./example1.py,
        line 10, in main ()
    foobar(amount=1337)

#11 Frame 0x7f54735bedd0, for file ./example1.py,
        line 14, in <module> ()
    main()
```

Demo

GDB and threads

- `info threads`
  - Current thread is marked with *
- Switch: `thread <id>`
- `thread apply all <COMMAND>`
  - `thread apply all py-bt`
  - `thread apply all py-list`

Working with Core Dumps

- Also works with core dumps.
- Generate coredump: `` `gcore <pid>` ``
- `` `gdb /path/to/program <core_file>` ``

Gotchas

- You need debuginfo libraries installed
  - GDB will tell you what you need
- Optimized out Python code = bad GDB
- Root is required for GDB's attach

What’s next?

- A lot

- Example: `py-print` can’t traverse namespaces
- Example: `py-print` can’t call functions
- Example: run pdb.set_trace() from GDB?

- Code in Python’s HG, please contribute!

References

- https://wiki.python.org/moin/DebuggingWithGdb
- https://fedoraproject.org/wiki/Features/EasierPythonDebugging
- https://sourceware.org/gdb/current/onlinedocs/gdb/Threads.html
- https://github.com/tamentis/rpdb#trigger-rpdb-with-signal
- https://hg.python.org/cpython/rev/6de3de3ab71f/

Questions?

Thank You

John Schwarz
jschwarz@redhat.com
jschwarz at FreeNode