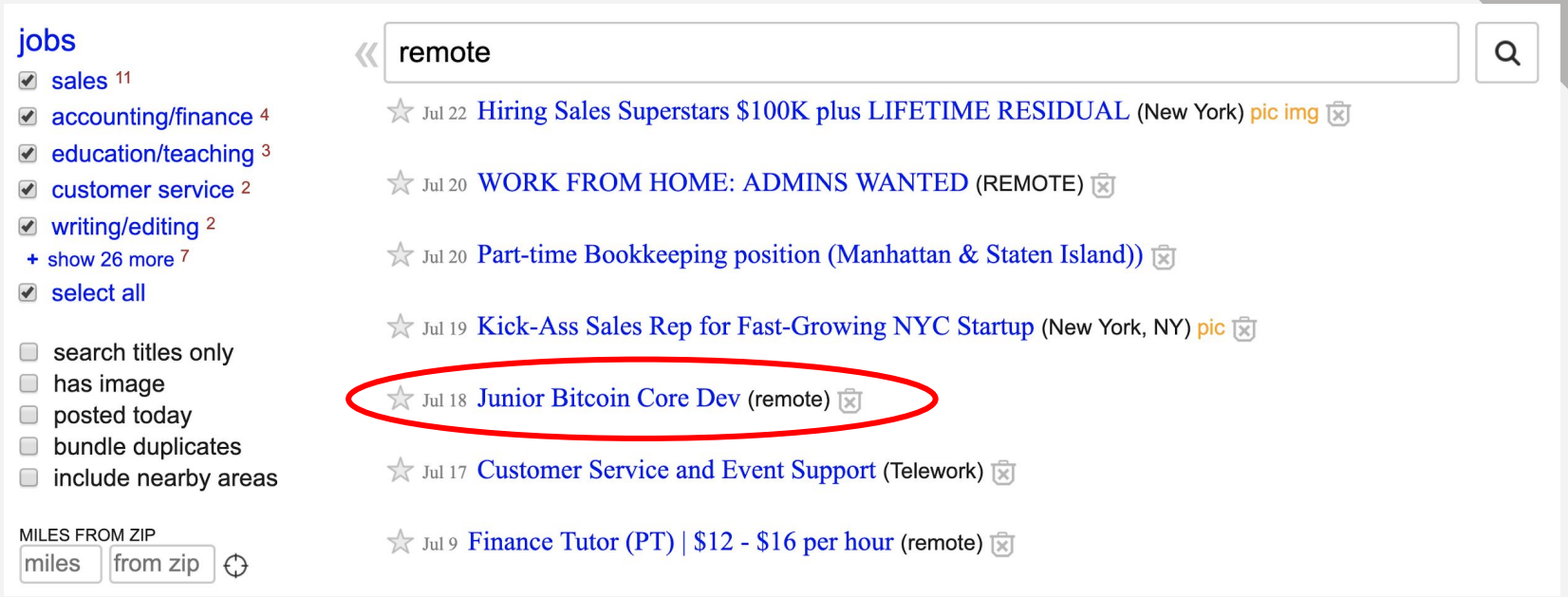
chaincode

Debugging Bitcoin

Welcome to Bitcoin



Content

1. Preparations
2. Logging
3. Using a debugger
4. Segfault tools

Part 1: Preparations

Install `ccache`

Deactivate optimization through compiler flags

```
./configure CXXFLAGS="-O0 -g" CFLAGS="-O0 -g"
```

Part 2: Logging

```
3
2     LogPrintf("@@@@@@@@@@@@");
1     std::cout << "###########" <<std::endl;
```

```
$ src/bitcoind -regtest
```

```
$ cat ~/Library/Application\ Support/Bitcoin/debug.log | grep @@@
```

wat?

Being in the right environment

```
$ bitcoin-cli -regtest getdifficulty
4.656542373906925e-10
```

std::out

```
2019-08-05T17:36:14Z Received a POST request for / from 127.0.0.1:65457
2019-08-05T17:36:14Z ThreadRPCServer method=getdifficulty user=satoshi
2019-08-05T17:36:14Z @@@@@@@@@@@@@@############
tor: Error connecting to Tor control socket
```

regtest/debug.log

```
$ sudo cat ~/Library/Application\ Support/Bitcoin/regtest/debug.log | grep @@@
2019-08-05T17:27:19Z @@@@@@@@@@@@@Adding fixed seed nodes as DNS doesn't seem to be available.
2019-08-05T17:34:47Z @@@@@@@@@@@@@tor: Error connecting to Tor control socket
2019-08-05T17:36:14Z @@@@@@@@@@@@@tor: Error connecting to Tor control socket
```

Logging from unit tests

Run `src/test/test_bitcoin` directly with `--log-level=all`

Can not use `LogPrintf()`

Use LibBoost functions, like `BOOST_TEST_MESSSAGE` or `BOOST_CHECK_MESSAGE`

From source files use `fprintf()` which prints ot `std::err`

Unit test logging in action

```
    delete block_index;
    BOOST_TEST_MESSAGE("FOO");

    RejectDifficultyMismatch(difficulty, expect
```

```
$ src/test/test_bitcoin --run_test=blockchain_tests --log_level=all
Running 5 test cases...
Entering test module "Bitcoin Core Test Suite"
test/blockchain_tests.cpp:46: Entering test suite "blockchain_tests"
test/blockchain_tests.cpp:48: Entering test case "get_difficulty_for
FOO
test/blockchain_tests.cpp:30: info: check 'Difficulty was 0.000001 b
```

Logging from functional tests

```
self.log.info("foo")
```

Need to run test directly (not through `test_runner.py`)

Part 3: Using a debugger

`gdb` or `lldb` on macOS

Start debugger with an executable

Set breakpoints

Run the executable from the debugger

Inspect variables, step through lines etc.

Debugger from own environment

```
$ lldb src/bitcoind

(lldb) b blockchain.cpp:123

(lldb) run -regtest
```

Also works for unit tests

```
$ lldb src/test/test_bitcoin
(lldb) target create "src/test/test_bitcoin"
Current executable set to 'src/test/test_bitcoin' (x86_64).
(lldb) b test/blockchain_tests.cpp:48
Breakpoint 1: 5 locations.
(lldb) run --run_test=blockchain_tests
Process 46577 launched: '/Users/FJ/projects/cpp/bitcoin/src/test/test_bi
test_bitcoin was compiled with optimization - stepping may behave oddly;
Process 46577 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1
    frame #0: 0x0000000100093d42 test_bitcoin`_GLOBAL__sub_I_blockchain_
   45
   46   BOOST_FIXTURE_TEST_SUITE(blockchain_tests, BasicTestingSetup)
   47
-> 48   BOOST_AUTO_TEST_CASE(get_difficulty_for_very_low_target)
   49   {
   50       TestDifficulty(0x1f111111, 0.000001);
   51   }
Target 0: (test_bitcoin) stopped.
(lldb)
```

Should be easy for functional tests...

Using Python

Debugging

- `import pdb; pdb.set_trace()`

But what about debugging the C++ code?

Where is the bitcoind process?

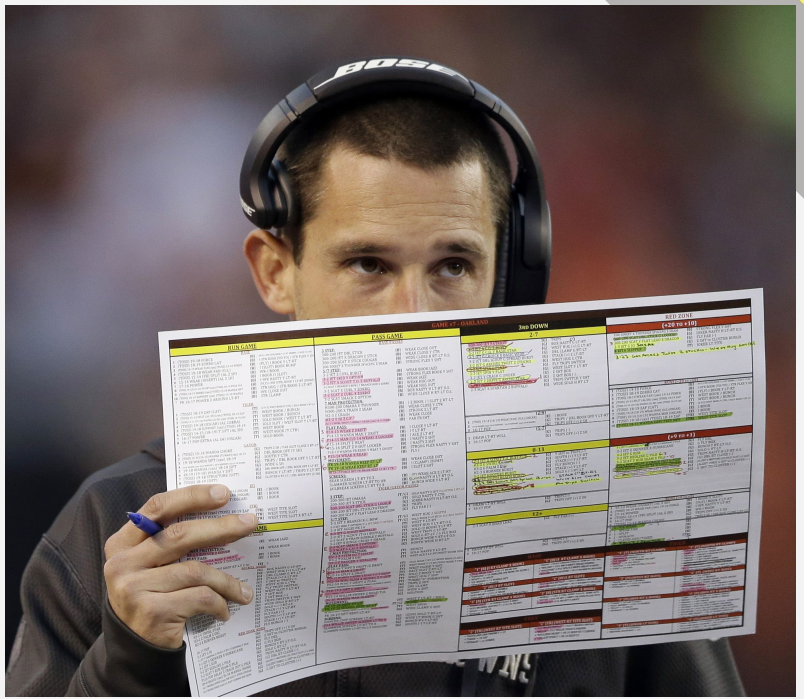
Functional tests launch our `bitcoind` themselves using a temp folder as datadir

That means we can not simply start it ourselves

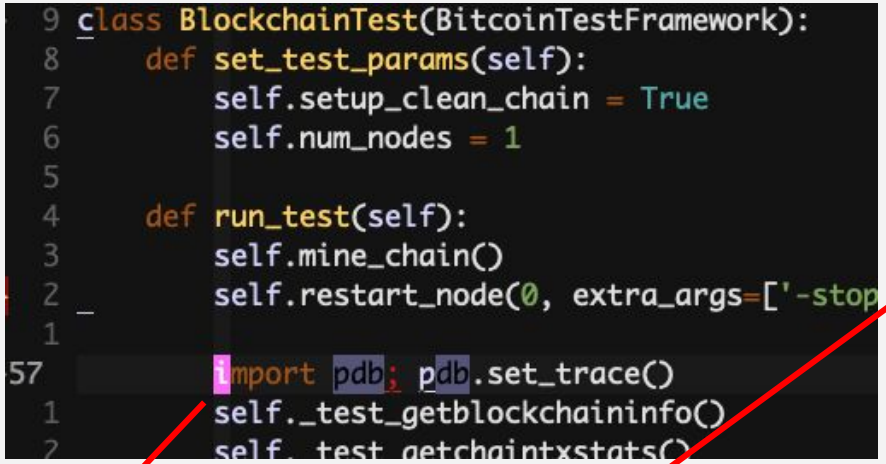
We need a gameplan!

Gameplan

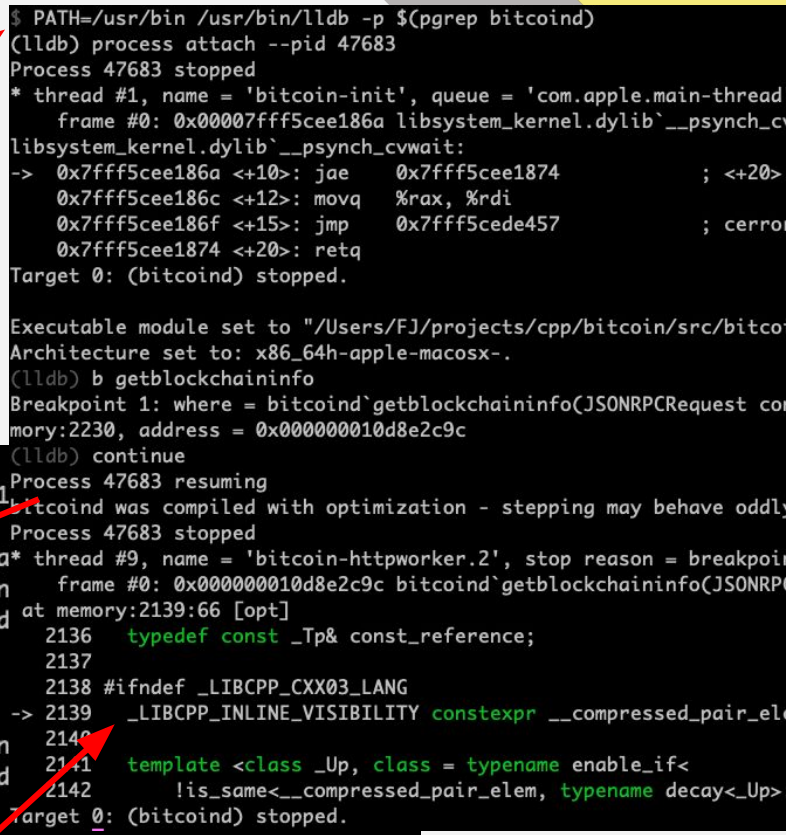
1. Start the functional test directly (not using `test_runner.py`) and let them start the `bitcoind` process
2. Pause the functional tests with `pdb.set_trace()`
3. Find the running `bitcoind` process, attach to it using `lldb` and setting breakpoints
4. Then let the test continue (`continue` in `pdb`) and let it run into our `lldb` breakpoints
5. Optional: May want to remove 60s timeout



Demo time!


class BlockchainTest(BitcoinTestFramework):
def set_test_params(self):
self.setup_clean_chain = True
self.num_nodes = 1
def run_test(self):
self.mine_chain()
self.restart_node(0, extra_args=['-stop
import pdb; pdb.set_trace()
self._test_getblockchaininfo()


$ ./test/functional/rpc_blockchain.py --loglevel=debug
2019-08-05T19:49:54.334000Z TestFramework (DEBUG): PRNG seed is: 7714090491127031
2019-08-05T19:49:54.334000Z TestFramework (DEBUG): Setting up network thread
2019-08-05T19:49:54.335000Z TestFramework (INFO): Initializing test directory /va
2019-08-05T19:49:54.340000Z TestFramework.node0 (DEBUG): bitcoind started, waitin
2019-08-05T19:49:54.859000Z TestFramework.node0 (DEBUG): RPC successfully started
2019-08-05T19:49:54.925000Z TestFramework (INFO): Create some old blocks
2019-08-05T19:49:55.458000Z TestFramework.node0 (DEBUG): Stopping node
2019-08-05T19:49:55.783000Z TestFramework.node0 (DEBUG): Node stopped
2019-08-05T19:49:55.790000Z TestFramework.node0 (DEBUG): bitcoind started, waitin
2019-08-05T19:49:56.303000Z TestFramework.node0 (DEBUG): RPC successfully started
> /Users/FJ/projects/cpp/bitcoin/test/functional/rpc_blockchain.py(58)run_test()
-> self._test_getblockchaininfo()
(Pdb) continue
2019-08-05T19:50:23.047000Z TestFramework (INFO): Test getblockchaininfo


$ PATH=/usr/bin /usr/bin/lldb -p $(pgrep bitcoind)
(lldb) process attach --pid 47683
Process 47683 stopped
* thread #1, name = 'bitcoin-init', queue = 'com.apple.main-thread
frame #0: 0x00007fff5cee186a libsystem_kernel.dylib`__psynch_cv
libsystem_kernel.dylib`__psynch_cvwait:
-> 0x7fff5cee186a <+10>: jae 0x7fff5cee1874 ; <+20>
0x7fff5cee186c <+12>: movq %rax, %rdi
0x7fff5cee186f <+15>: jmp 0x7fff5cede457 ; cerro
0x7fff5cee1874 <+20>: retq
Target 0: (bitcoind) stopped.
Executable module set to "/Users/FJ/projects/cpp/bitcoin/src/bitco
Architecture set to: x86_64h-apple-macosx-.
(lldb) b getblockchaininfo
Breakpoint 1: where = bitcoind`getblockchaininfo(JSONRPCRequest co
mory:2230, address = 0x000000010d8e2c9c
(lldb) continue
Process 47683 resuming
itcoind was compiled with optimization - stepping may behave oddly
Process 47683 stopped
* thread #9, name = 'bitcoin-httpworker.2', stop reason = breakpoi
frame #0: 0x000000010d8e2c9c bitcoind`getblockchaininfo(JSONRP
at memory:2139:66 [opt]
2136 typedef const _Tp& const_reference;
2137
2138 #ifndef _LIBCPP_CXX03_LANG
-> 2139 _LIBCPP_INLINE_VISIBILITY constexpr __compressed_pair_el
2140
2141 template <class _Up, class = typename enable_if<
2142 !is_same<__compressed_pair_elem, typename decay<_Up>
Target 0: (bitcoind) stopped.

Debugging contexts

	Test driver	Bitcoind context
Manual	- `bitcoin-cli`/RPC	- Path: your own bitcoin path - Log: `ENV/debug.log` - Debug: run `bitcoind` with `lldb`
Unit tests	- `src/test/test_bitcoin`	- Path: `/var/` - Log: to `std::out` with LibBoost - Debug: Run `test_bitcoin` with `lldb`
Functional tests	- `test/functional/test_runner.py` (or the test directly) - Log: `self.log.print()` - Debug: `pdb`	- Path: `/var/` with `--no-cleanup` - Log: temporary `debug.log` with consolidation tool - Debug: `pdb` + `lldb`

Part 4: Segfault tools

Core dumps

- Need to activate with `ulimit -c unlimited` and then run in same terminal session
- Run program with segfault
- Find core dump in `/cores/*`
- Make sure to clean up afterwards

`valgrind`

- Inspections, used similar to `lldb`
- `valgrind --leak-check=yes src/bitcoind -regtest`

http://bit.ly/debugbitcoin

Thank you and questions?

chaincode