Example scripts for User-space, Statically Defined Tracing (USDT)

This directory contains scripts showcasing User-space, Statically Defined Tracing (USDT) support for Bitcoin Core on Linux using. For more information on USDT support in Bitcoin Core see the USDT documentation.

Examples for the two main eBPF front-ends, bpftrace and BPF Compiler Collection (BCC), with support for USDT, are listed. BCC is used for complex tools and daemons and `bpftrace` is preferred for one-liners and shorter scripts.

To develop and run bpftrace and BCC scripts you need to install the corresponding packages. See installing bpftrace and installing BCC for more information. For development there exist a bpftrace Reference Guide, a BCC Reference Guide, and a bcc Python Developer Tutorial.

Examples

The bpftrace examples contain a relative path to the `bitcoind` binary. By default, the scripts should be run from the repository-root and assume a self-compiled `bitcoind` binary. The paths in the examples can be changed, for example, to point to release builds if needed. See the Bitcoin Core USDT documentation on how to list available tracepoints in your `bitcoind` binary.

WARNING: eBPF programs require root privileges to be loaded into a Linux kernel VM. This means the bpftrace and BCC examples must be executed with root privileges. Make sure to carefully review any scripts that you run with root privileges first!

log_p2p_traffic.bt

A bpftrace script logging information about inbound and outbound P2P network messages. Based on the `net:inbound_message` and `net:outbound_message` tracepoints.

By default, `bpftrace` limits strings to 64 bytes due to the limited stack size in the eBPF VM. For example, Tor v3 addresses exceed the string size limit which results in the port being cut off during logging. The string size limit can be increased with the `BPFTRACE_STRLEN` environment variable (`BPFTRACE_STRLEN=70` works fine).

```
$ bpftrace contrib/tracing/log_p2p_traffic.bt
```

Output

```
outbound 'ping' msg to peer 11 (outbound-full-relay, [2a02:b10c:f747:1:ef:fake:ipv6:addr]:83
inbound 'pong' msg from peer 11 (outbound-full-relay, [2a02:b10c:f747:1:ef:fake:ipv6:addr]:8
inbound 'inv' msg from peer 16 (outbound-full-relay, XX.XX.XXX.121:8333) with 37 bytes
outbound 'getdata' msg to peer 16 (outbound-full-relay, XX.XX.XXX.121:8333) with 37 bytes
```

```
inbound 'tx' msg from peer 16 (outbound-full-relay, XX.XX.XXX.121:8333) with 222 bytes
outbound 'inv' msg to peer 9 (outbound-full-relay, faketorv3addressa2ufa6odvoi3s77j4uegey0xb
outbound 'inv' msg to peer 7 (outbound-full-relay, XX.XX.XXX.242:8333) with 37 bytes
...
```

p2p_monitor.py

A BCC Python script using curses for an interactive P2P message monitor. Based on the `net:inbound_message` and `net:outbound_message` tracepoints.

Inbound and outbound traffic is listed for each peer together with information about the connection. Peers can be selected individually to view recent P2P messages.

```
$ python3 contrib/tracing/p2p_monitor.py ./src/bitcoind
```

Lists selectable peers and traffic and connection information.

```
 P2P Message Monitor
 Navigate with UP/DOWN or J/K and select a peer with ENTER or SPACE to see individual P2P me

 PEER  OUTBOUND              INBOUND               TYPE                   ADDR
    0  46          398 byte  61      1407590 byte  block-relay-only       XX.XX.XXX.196:8333
   11  1156     253570 byte  3431    2394924 byte  outbound-full-relay    XXX.X.XX.179:8333
   13  3425    1809620 byte  1236     305458 byte  inbound                XXX.X.X.X:60380
   16  1046     241633 byte  1589    1199220 byte  outbound-full-relay    4faketorv2pbfu7x.o
   19  577      181679 byte  390      148951 byte  outbound-full-relay    kfake4vctorjv2o2.o
   20  11         1248 byte  13         1283 byte  block-relay-only       [2600:fake:64d9:b1
   21  11         1248 byte  13         1299 byte  block-relay-only       XX.XXX.X.155:8333
   22  5           103 byte  1           102 byte  feeler                 XX.XX.XXX.173:8333
   23  11         1248 byte  12         1255 byte  block-relay-only       XX.XXX.XXX.220:833
   24  3           103 byte  1           102 byte  feeler                 XXX.XXX.XXX.64:833
...
```

Showing recent P2P messages between our node and a selected peer.

```
    ----------------------------------------------------------------------
    |                PEER 16 (4faketorv2pbfu7x.onion:8333)               |
    | OUR NODE                outbound-full-relay                   PEER |
    |                                           <--- sendcmpct (9 bytes) |
    | inv (37 byte) --->                                                 |
    |                                                <--- ping (8 bytes) |
    | pong (8 byte) --->                                                 |
    | inv (37 byte) --->                                                 |
    |                                               <--- addr (31 bytes) |
    | inv (37 byte) --->                                                 |
    |                                       <--- getheaders (1029 bytes) |
    | headers (1 byte) --->                                              |
    |                                           <--- feefilter (8 bytes) |
    |                                                <--- pong (8 bytes) |
    |                                            <--- headers (82 bytes) |
    |                                            <--- addr (30003 bytes) |
    | inv (1261 byte) --->                                               |
    |                                 ...                                  |
```

log_raw_p2p_msgs.py

A BCC Python script showcasing eBPF and USDT limitations when passing data larger than about 32kb. Based on the `net:inbound_message` and `net:outbound_message` tracepoints.

Bitcoin P2P messages can be larger than 32kb (e.g. `tx`, `block`, …). The eBPF VM's stack is limited to 512 bytes, and we can't allocate more than about 32kb for a P2P message in the eBPF VM. The **message data is cut off** when the message is larger than MAX_MSG_DATA_LENGTH (see script). This can be detected in user-space by comparing the data length to the message length variable. The message is cut off when the data length is smaller than the message length. A warning is included with the printed message data.

Data is submitted to user-space (i.e. to this script) via a ring buffer. The throughput of the ring buffer is limited. Each p2p_message is about 32kb in size. In- or outbound messages submitted to the ring buffer in rapid succession fill the ring buffer faster than it can be read. Some messages are lost. BCC prints: `Possibly lost 2 samples` on lost messages.

```
$ python3 contrib/tracing/log_raw_p2p_msgs.py ./src/bitcoind
```

```
Logging raw P2P messages.
Messages larger that about 32kb will be cut off!
Some messages might be lost!
 outbound msg 'inv' from peer 4 (outbound-full-relay, XX.XXX.XX.4:8333) with 253 bytes: 0705
...
Warning: incomplete message (only 32568 out of 53552 bytes)! inbound msg 'tx' from peer 32 (
...
Possibly lost 2 samples
```

connectblock_benchmark.bt

A `bpftrace` script to benchmark the `ConnectBlock()` function during, for example, a blockchain re-index. Based on the `validation:block_connected` USDT tracepoint.

The script takes three positional arguments. The first two arguments, the start, and end height indicate between which blocks the benchmark should be run. The third acts as a duration threshold in milliseconds. When the `ConnectBlock()` function takes longer than the threshold, information about the block, is printed. For more details, see the header comment in the script.

The following command can be used to benchmark, for example, `ConnectBlock()` between height 20000 and 38000 on SigNet while logging all blocks that take longer than 25ms to connect.

```
$ bpftrace contrib/tracing/connectblock_benchmark.bt 20000 38000 25
```

In a different terminal, starting Bitcoin Core in SigNet mode and with re-indexing enabled.

```
$ ./src/bitcoind -signet -reindex
```

This produces the following output.

```
Attaching 5 probes...
ConnectBlock Benchmark between height 20000 and 38000 inclusive
Logging blocks taking longer than 25 ms to connect.
Starting Connect Block Benchmark between height 20000 and 38000.
BENCH   39 blk/s     59 tx/s      59 inputs/s       20 sigops/s (height 20038)
Block 20492 (000000f555653bb05e2f3c6e79925e01a20dd57033f4dc7c354b46e34735d32b)    20 tx   23
BENCH 1840 blk/s   2117 tx/s    4478 inputs/s     2471 sigops/s (height 21879)
BENCH 1816 blk/s   4972 tx/s    4982 inputs/s      125 sigops/s (height 23695)
BENCH 2095 blk/s   2890 tx/s    2910 inputs/s      152 sigops/s (height 25790)
BENCH 1684 blk/s   3979 tx/s    4053 inputs/s      288 sigops/s (height 27474)
BENCH 1155 blk/s   3216 tx/s    3252 inputs/s      115 sigops/s (height 28629)
BENCH 1797 blk/s   2488 tx/s    2503 inputs/s      111 sigops/s (height 30426)
BENCH 1849 blk/s   6318 tx/s    6569 inputs/s    12189 sigops/s (height 32275)
BENCH  946 blk/s  20209 tx/s   20775 inputs/s    83809 sigops/s (height 33221)
Block 33406 (0000002adfe4a15cfcd53bd890a89bbae836e5bb7f38bac566f61ad4548c87f6)    25 tx   20
Block 33687 (00000073231307a9828e5607ceb8156b402efe56747271a4442e75eb5b77cd36)    52 tx   17
BENCH  582 blk/s  21581 tx/s   27673 inputs/s    60345 sigops/s (height 33803)
BENCH 1035 blk/s  19735 tx/s   19776 inputs/s    51355 sigops/s (height 34838)
Block 35625 (0000006b00b347390c4768ea9df2655e9ff4b120f29d78594a2a702f8a02c997)    20 tx   33
BENCH  887 blk/s  17857 tx/s   22191 inputs/s    24404 sigops/s (height 35725)
Block 35937 (000000d816d13d6e39b471cd4368db60463a764ba1f29168606b04a22b81ea57)    75 tx   39
BENCH  823 blk/s  16298 tx/s   21031 inputs/s    18440 sigops/s (height 36548)
Block 36583 (000000c3e260556dbf42968aae3f904dba8b8c1ff96a6f6e3aa5365d2e3ad317)    24 tx   21
Block 36700 (000000b3b173de9e65a3cfa738d976af6347aaf83fa17ab3f2a4d2ede3ddfac4)    73 tx   16
Block 36832 (0000007859578c02c1ac37dabd1b9ec19b98f350b56935f5dd3a41e9f79f836e)    34 tx   14
BENCH  613 blk/s  16718 tx/s   25074 inputs/s    23022 sigops/s (height 37161)
Block 37870 (000000f5c1086291ba2d943fb0c3bc82e71c5ee341ee117681d1456fbf6c6c38)    25 tx   15
BENCH  811 blk/s  16031 tx/s   20921 inputs/s    18696 sigops/s (height 37972)

Took 14055 ms to connect the blocks between height 20000 and 38000.

Histogram of block connection times in milliseconds (ms).
@durations:
[0]                16838 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[1]                  882 |@@                                                  |
[2, 4)               236 |                                                    |
[4, 8)                23 |                                                    |
[8, 16)                9 |                                                    |
[16, 32)               9 |                                                    |
[32, 64)               4 |                                                    |
```

log_utxocache_flush.py

A BCC Python script to log the UTXO cache flushes. Based on the `utxocache:flush` tracepoint.

```
$ python3 contrib/tracing/log_utxocache_flush.py ./src/bitcoind
```

```
Logging utxocache flushes. Ctrl-C to end...
Duration (µs)   Mode       Coins Count     Memory Usage    Prune
730451          IF_NEEDED  22990           3323.54 kB      True
637657          ALWAYS     122320          17124.80 kB     False
81349           ALWAYS     0               1383.49 kB      False
```

log_utxos.bt

A `bpftrace` script to log information about the coins that are added, spent, or uncached from the UTXO set. Based on the `utxocache:add`, `utxocache:spend` and `utxocache:uncache` tracepoints.

```
$ bpftrace contrib/tracing/log_utxos.bt
```

This should produce an output similar to the following. If you see bpftrace warnings like `Lost 24 events`, the eBPF perf ring-buffer is filled faster than it is being read. You can increase the ring-buffer size by setting the ENV variable `BPFTRACE_PERF_RB_PAGES` (default 64) at a cost of higher memory usage. See the bpftrace reference guide for more information.

```
Attaching 4 probes...
OP      Outpoint                                                                           V
Added   6ba9ad857e1ef2eb2a2c94f06813c414c7ab273e3d6bd7ad64e000315a887e7c:1                 1
Spent   fa7dc4db56637a151f6649d8f26732956d1c5424c82aae400a83d02b2cc2c87b:0             18226
Added   eeb2f099b1af6a2a12e6ddd2eeb16fc5968582241d7f08ba202d28b60ac264c7:0                 1
Added   eeb2f099b1af6a2a12e6ddd2eeb16fc5968582241d7f08ba202d28b60ac264c7:1             18225
Added   a0c7f4ec9cccef2d89672a624a4e6c8237a17572efdd4679eea9e9ee70d2db04:0              1007
Spent   25e0df5cc1aeb1b78e6056bf403e5e8b7e41f138060ca0a50a50134df0549a5e:2
Spent   42f383c04e09c26a2378272ec33aa0c1bf4883ca5ab739e8b7e06be5a5787d61:1               384
Added   f85e3b4b89270863a389395cc9a4123e417ab19384cef96533c6649abd6b0561:0               378
Added   f85e3b4b89270863a389395cc9a4123e417ab19384cef96533c6649abd6b0561:2
Spent   a05880b8c77971ed0b9f73062c7c4cdb0ff3856ab14cbf8bc481ed571cd34b83:1           559128
Added   eb689865f7d957938978d6207918748f74e6aa074f47874724327089445b0960:0           558969
Added   eb689865f7d957938978d6207918748f74e6aa074f47874724327089445b0960:1               156
```