Introducing dynamic features in uftrace

Progress Report Meeting

Clément Guidi

Polytechnique Montréal

June 4, 2021

Table of contents

Introduction

About myself:

- intern during the winter semester
- worked on uftrace dynamic features

Objective: interact with uftrace at runtime, dynamically instrument code
First step: build a framework to control uftrace at runtime

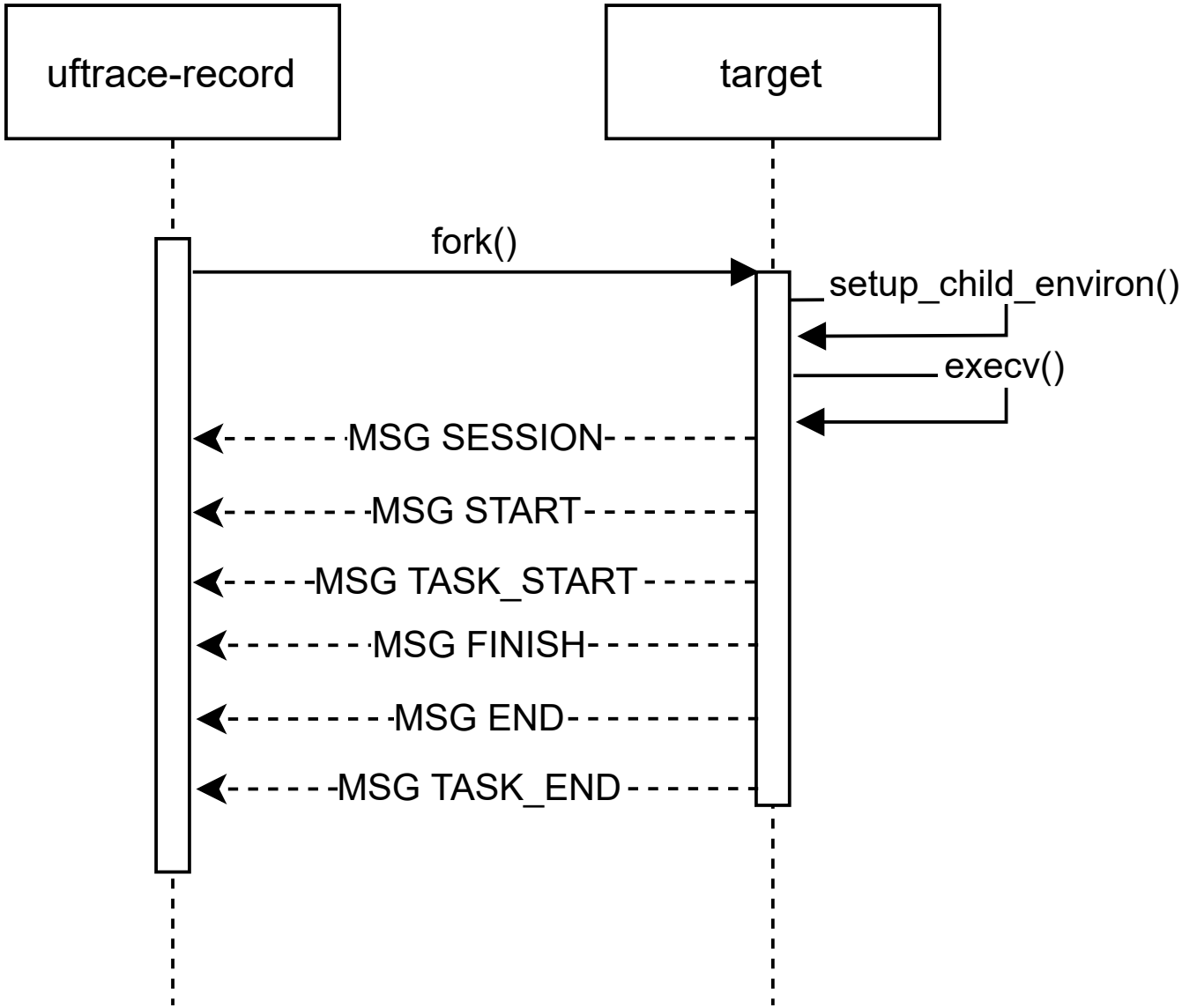
Progress status:

- pull request submitted to uftrace
- reviewed by the maintainers
- considered as interesting, but should be built as a separate library

Pull request link: `https://github.com/namhyung/uftrace/pull/1269`

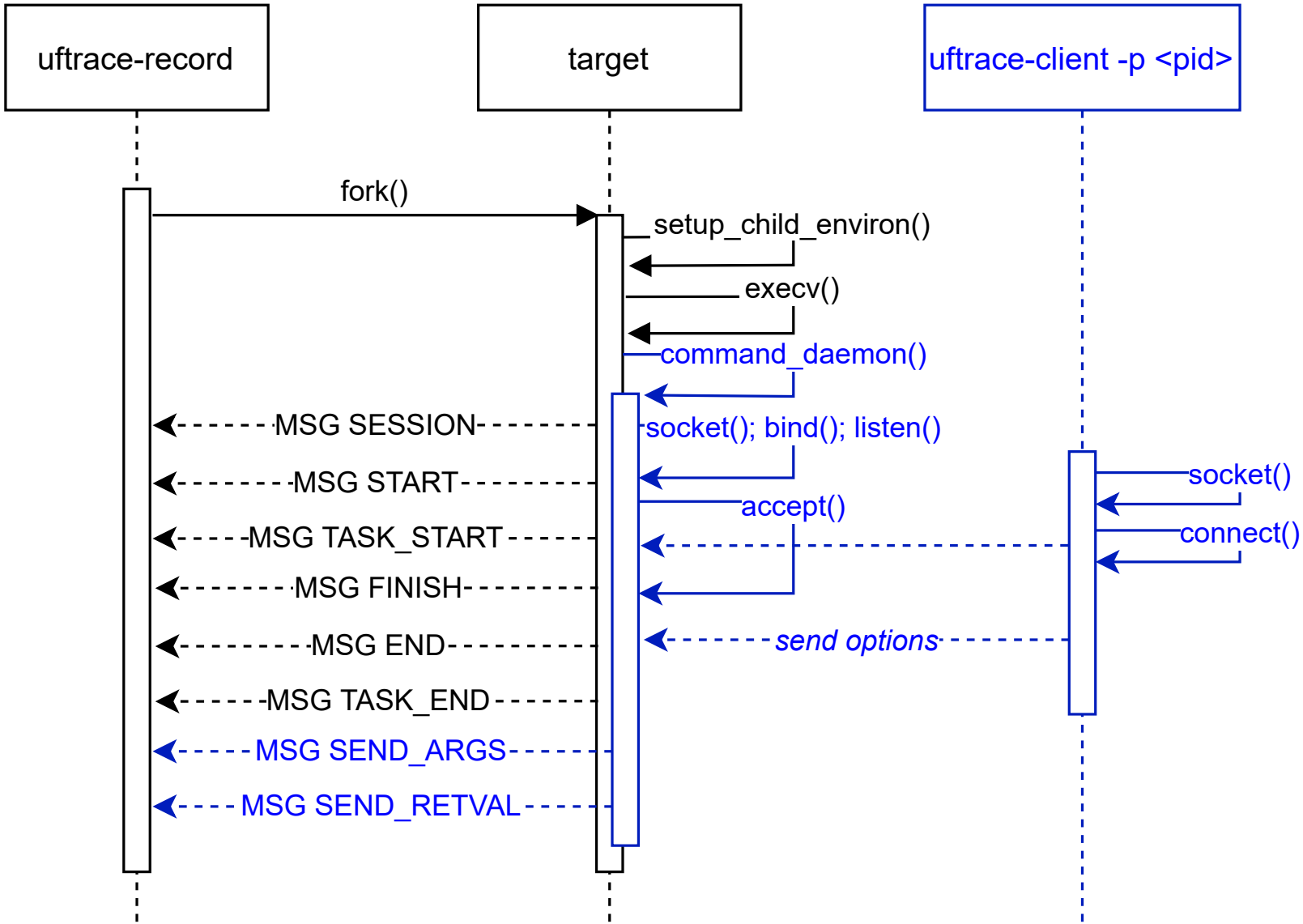
The `client` command

Original architecture



The `client` command

Client architecture



The `client` command

Internals and usage

A daemon thread is started in *libmcount*.
It listens to the `/var/run/user/$UID/uftrace/$PID.socket` socket.

The client is a new uftrace instance.
It sends commands through the socket to the daemon. It forwards the supported options it gets.

Usage:
`uftrace [OPTION...] client -p <pid>`
where `<pid>` is the pid of the target.

The client command

Example code: abc2.c

```
#include <unistd.h>

void c(void) {
    /* do nothing */
}

void b(void) {
    c();
}

void a(void) {
    b();
}

int main(void) {
    a();
    sleep(2);
    a();
    return 0;
}
```

The client command

Example code: abc2.c

```
$ gcc -pg -o abc2 abc2.c
$ uftrace record ./abc2
$ uftrace replay
# DURATION     TID     FUNCTION
2.392 us [  1234] | __monstartup();
1.526 us [  1234] | __cxa_atexit();
         [  1234] | main() {
         [  1234] |   a() {
         [  1234] |     b() {
0.185 us [  1234] |       c();
0.909 us [  1234] |     } /* b */
1.366 us [  1234] |   } /* a */
2.000  s [  1234] |   sleep();
         [  1234] |   a() {
         [  1234] |     b() {
0.542 us [  1234] |       c();
2.135 us [  1234] |     } /* b */
3.308 us [  1234] |   } /* a */
2.000  s [  1234] | }
```

Depth and time filters

Depth filter

```
$ uftrace record ./abc2&
$ uftrace -D 2 client -p $(pidof abc2)
$ uftrace replay
# DURATION     TID     FUNCTION
1.989 us [  1234] | __monstartup();
1.192 us [  1234] | __cxa_atexit();
         [  1234] | main() {
         [  1234] |   a() {
         [  1234] |     b() {
0.166 us [  1234] |       c();
0.841 us [  1234] |     } /* b */
1.293 us [  1234] |   } /* a */
2.000  s [  1234] |   sleep();
1.413 us [  1234] |   a();
2.000  s [  1234] | }
```

Depth and time filters

Time filter

```
$ uftrace record ./abc2&
$ uftrace -t 2us client -p $(pidof abc2)
$ uftrace replay
# DURATION     TID     FUNCTION
1.997 us [  1234] | __monstartup();
1.218 us [  1234] | __cxa_atexit();
         [  1234] | main() {
         [  1234] |   a() {
         [  1234] |     b() {
0.174 us [  1234] |       c();
0.994 us [  1234] |     } /* b */
1.534 us [  1234] |   } /* a */
2.000  s [  1234] |   sleep();
2.109 us [  1234] |   a();
2.000  s [  1234] | }
```

Filters and triggers

```
$ uftrace -N b record ./abc2&
$ uftrace -F a -F b client -p $(pidof abc2)
$ uftrace replay
# DURATION     TID     FUNCTION
1.708 us [  1234] | __monstartup();
1.411 us [  1234] | __cxa_atexit();
         [  1234] | main() {
1.141 us [  1234] |   a();
2.000  s [  1234] |   sleep();
         [  1234] |   a() {
         [  1234] |     b() {
0.351 us [  1234] |       c();
1.689 us [  1234] |     } /* b */
3.443 us [  1234] |   } /* a */
2.000  s [  1234] | }
```

Arguments and return values

```
$ uftrace record ./abc2&
$ uftrace -A a@arg1 -R a@retval client -p $(pidof abc2)
$ uftrace replay
# DURATION     TID     FUNCTION
8.045 us [  1234] | __monstartup();
5.763 us [  1234] | __cxa_atexit();
         [  1234] | main() {
         [  1234] |   a() {
         [  1234] |     b() {
0.166 us [  1234] |       c();
0.944 us [  1234] |     } /* b */
2.109 us [  1234] |   } /* a */
2.000  s [  1234] |   sleep();
         [  1234] |   a(0) {
         [  1234] |     b() {
0.310 us [  1234] |       c();
1.734 us [  1234] |     } /* b */
4.376 us [  1234] |   } = 0; /* a */
2.000  s [  1234] | }
```

Conclusion

The patch is on its way to get accepted upstream.

Many options are supported dynamically.

This framework will allow dynamic instrumentation at runtime (patching and unpatching).

Questions?