Process Record Tutorial

sourceware.org/gdb/wiki/ProcessRecord/Tutorial

This is a short tutorial on how to use gdb's Process Record feature.

Process Record lets you:

- Record a program's execution and play it back.
- Debug a program in reverse (see Reverse Debugging).

目录

Quick Start

To get started using process record, all you need is gdb version 7.0 or later, and an x86 computer running Linux. And, of course, a program to debug. Here's a simple one:

```
int xyz;

int bar ()
{
  xyz = 2; /* break in bar */
  return 1;
}

int foo ()
{
  xyz = 1; /* break in foo */
  return bar ();
}

int main ()
{
  xyz = 0;      /* break in main */
  foo ();
  return (xyz == 2 ? 0 : 1);
}               /* end of main */
```

Compile this program (with -g of course), and load it into gdb, then do the following.

```
(gdb) break main
(gdb) run
(gdb) record
```

This will turn on process recording, which will now record all subsequent instructions executed by the program being debugged.

Note that you can start process recording at any point (not just at main). You may choose to start it later, or even earlier. The only restriction is that your program has to be running (so you have to type "run" before "record"). If you want to start recording from the very first instruction of your program, you can do it like this:

```
(gdb) break _start
(gdb) run
(gdb) record
```

run to the end of the program

Now let's make a common mistake -- say "continue" without having set any breakpoints. Your program will run, but process record will detect when it exits, and ask you whether you want to stop it before exiting. Say yes.

```
(gdb) continue
Continuing.
The next instruction is syscall exit_group.  It will make the program exit.  Do you
want to stop the program?([y] or n) yes
Process record: inferior program stopped.

Program received signal SIGTRAP, Trace/breakpoint trap.
0x005037a0 in _dl_sysinfo_int80 () from /lib/ld-linux.so.2
(gdb)
```

reverse-continue

You are now at the very last instruction before your program will exit. However, the execution of your program has been recorded, and you can now return to the beginning and replay it.

To return to the beginning, just use the new "reverse-continue" command. This will execute your program in reverse, and since there are still no breakpoints, it will run all the way back to the start of the recording, and then stop.

```
(gdb) reverse-continue
Continuing.

No more reverse-execution history.
main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:34
34        xyz = 0;      /* break in main */
(gdb)
```

forward replay

This time, let's set some breakpoints.

```
(gdb) break foo
(gdb) break bar
(gdb) continue
Continuing.

Breakpoint 2, foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:28
28        xyz = 1; /* break in foo */
(gdb) continue
Continuing.

Breakpoint 3, bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
22        xyz = 2; /* break in bar */
(gdb) backtrace
#0  bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
#1  0x08048366 in foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:29
#2  0x08048393 in main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:35
(gdb)
```

Looks just like an ordinary debugging session -- but you are now replaying the execution log that was made earlier when your program ran without any breakpoints.

backward replay

To finish out this quick start, let's play to the end and then run backwards.

```
(gdb) continue
Continuing.

No more reverse-execution history.
0x005037a0 in _dl_sysinfo_int80 () from /lib/ld-linux.so.2
(gdb) reverse-continue
Continuing.

Breakpoint 3, bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
22        xyz = 2; /* break in bar */
(gdb) reverse-continue
Continuing.

Breakpoint 2, foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:28
28        xyz = 1; /* break in foo */
(gdb) reverse-continue
Continuing.

No more reverse-execution history.
main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:34
34        xyz = 0;      /* break in main */
(gdb)
```

You'll notice that in replaying the program backwards, we hit the same breakpoints, but in the opposite order. That's the key observation for this quick start -- that you can replay the recorded execution log forward and backward, and that when replaying backward, the same events occur but in the opposite order.

Continue, reverse-continue, and breakpoints

If your gdb session from the "quick start" is still running, please quit and start again. We will use the same program to debug. This time, let's start debugging from the very first instruction (at the symbol "_start").

```
(gdb) break _start
(gdb) run
(gdb) record
```

This time, let's set our breakpoints before continuing, and then run up to the last breakpoint at "bar".

```
(gdb) break main
(gdb) break foo
(gdb) break bar
(gdb) continue
(gdb) continue
Continuing.

Breakpoint 2, main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:34
34        xyz = 0;      /* break in main */
(gdb) continue
Continuing.

Breakpoint 3, foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:28
28        xyz = 1; /* break in foo */
(gdb) continue
Continuing.

Breakpoint 4, bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
22        xyz = 2; /* break in bar */
(gdb) backtrace
#0  bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
#1  0x08048366 in foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:29
#2  0x08048393 in main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:35
```

We're now actually debugging the live program, just as we usually do, except that the program's execution is being recorded by process record.

Let's imagine that we get to this point, and then realize that we really were interested in what happens at the beginning of foo. Normally, we would have to start the whole debugging session over from the beginning -- but with process record and reverse debugging, we can simply "reverse-continue" the program back to the breakpoint at foo.

```
(gdb) reverse-continue
Continuing.

Breakpoint 3, foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:28
28        xyz = 1; /* break in foo */
(gdb)
```

Another "reverse-continue" would take us back to the next earlier breakpoint, at main. But rather than actually do that, let's imagine that we've finished debugging "foo" and are now ready to resume where we left off.

We can go directly to the point at which the recording currently ends, by disabling all our breakpoints and then telling gdb to continue. Process record will replay until it reaches the current end of the recording, and then stop.

```
(gdb) disable breakpoints
(gdb) continue
Continuing.

No more reverse-execution history.
bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
22        xyz = 2; /* break in bar */
(gdb)
```

Now we are back where we were, at the beginning of "bar". We can simply continue debugging, and process record will append the new execution log to the end of the old one.

```
(gdb) finish
Run till exit from #0  bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:30
30      }
Value returned is $1 = 1
(gdb) finish
Run till exit from #0  foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:30
main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:36
36        return (xyz == 2 ? 0 : 1);
Value returned is $2 = 1
(gdb) finish
"finish" not meaningful in the outermost frame.
(gdb) step
37      }               /* end of main */
(gdb) step
0x00531de3 in __libc_start_main () from /lib/tls/libc.so.6
(gdb)
```

And now, just to show that our execution log is still there and is complete:

```
(gdb) reverse-continue
Continuing.

Breakpoint 4, bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
22        xyz = 2; /* break in bar */
(gdb)
Continuing.

Breakpoint 3, foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:28
28        xyz = 1; /* break in foo */
(gdb)
Continuing.

Breakpoint 2, main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:34
34        xyz = 0;      /* break in main */
(gdb)
Continuing.

No more reverse-execution history.
0x08048298 in _start ()
(gdb)
```

Step and reverse-step

Let's quit gdb and start a new session:

```
(gdb) break _start
(gdb) run
(gdb) record
(gdb) break main
(gdb) continue

Breakpoint 2, main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:34
34        xyz = 0;      /* break in main */
(gdb)
```

Now let's do some normal single stepping.

```
(gdb) step
foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:28
28        xyz = 1; /* break in foo */
(gdb) step
29        return bar ();
(gdb) step
bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
22        xyz = 2; /* break in bar */
(gdb) step
23        return 1;
(gdb) step
24      }
(gdb) step
foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:30
30      }
(gdb) step
main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:36
36        return (xyz == 2 ? 0 : 1);
(gdb) step
37      }               /* end of main */
```

And now let's just do that in reverse.

```
37      }               /* end of main */
(gdb) reverse-step
36        return (xyz == 2 ? 0 : 1);
(gdb) reverse-step
foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:30
30      }
(gdb) reverse-step
bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:24
24      }
(gdb) reverse-step
23        return 1;
(gdb) reverse-step
22        xyz = 2; /* break in bar */
(gdb) reverse-step
foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:29
29        return bar ();
(gdb) reverse-step
28        xyz = 1; /* break in foo */
(gdb) reverse-step
main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:35
35        foo ();
(gdb) reverse-step

Breakpoint 2, main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:34
34        xyz = 0;      /* break in main */
```

Just as we saw when using breakpoints, we see the exact same events going backward as going forward, except that the events occur in reverse order.

next and reverse-next

These work analogously to step and reverse-step:

```
35        foo ();
(gdb) next
36        return (xyz == 2 ? 0 : 1);
(gdb) reverse-next
35        foo ();
(gdb)
```

finish and reverse-finish

Whereas "finish" takes you forward to just after the function returns, "reverse-finish" takes you backward to just before the function was called

```
(gdb) step
foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:28
28        xyz = 1; /* break in foo */
(gdb) step
29        return bar ();
(gdb) step
bar () at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
22        xyz = 2; /* break in bar */
(gdb) finish
Run till exit from #0  bar ()
30      }
Value returned is $3 = 1
(gdb) finish
Run till exit from #0  foo ()
36        return (xyz == 2 ? 0 : 1);
Value returned is $4 = 1
(gdb) reverse-step
foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:30
30      }
(gdb) reverse-step
bar ()
23        return 1;
(gdb) reverse-finish
Run back to call of #0  bar ()
29        return bar ();
(gdb) reverse-finish
Run back to call of #0  foo ()
35        foo ();
(gdb)
```

set exec-direction

To simplify long debugging sessions, gdb has a modal command "set exec-direction".

After "set exec direction reverse", all gdb execution commands such as continue, step, next, and finish will behave like their reverse-execution equivalents.

After "set exec direction forward', gdb's behavior will return to the norm (default).

```
(gdb) set exec-direction reverse
(gdb) show exec-direction
Reverse.
(gdb) set exec-direction forward
```

Advanced Options

These commands affect how process record behaves internally.

set record insn-number-max

This command controls the limit (default 200000) for how many instructions gdb will store in its execution log.

```
(gdb) show record insn-number-max
Record/replay buffer limit is 200000.
(gdb) set record insn-number-max 400000
(gdb)
```

set record stop-at-limit

The internal memory buffer in which gdb stores the execution log can behave either as a linear buffer, or as a circular buffer.

In the default linear mode, when the buffer reaches its maximum limit, the recording will stop.

In the circular buffer mode, when the buffer is full, gdb will begin deleting the oldest records in the buffer to make room for new records. In this way, you can continue to record the execution indefinitely, but the recording will not go all the way back to the beginning.

```
(gdb) show record stop-at-limit
Whether record/replay stops when record/replay buffer becomes full is on.
(gdb) set record stop-at-limit off
(gdb)
```

info record insn-number

This command will show how many instructions are currently saved in the execution log.

```
(gdb) info record insn-number
Record instruction number is 1287.
(gdb)
```

None: ProcessRecord/Tutorial (2009-10-17 23:01:13由AS-40816编辑)