Systems Programming

Debugger and ptrace

Byoungyoung Lee
Seoul National University
byoungyoung@snu.ac.kr
https://lifeasageek.github.io

Debugger

- A debugger is a computer program that is used to test and debugger other programs (the “target” program)

  - https://en.wikipedia.org/wiki/Debugger

GDB workflow

$ gdb /bin/ls

(gdb) run

(gdb) breakpoint *ADDRESS

(gdb) stepi

(gdb) continue

(gdb) info registers

(gdb) set $REG = VALUE

(gdb) x ADDRESS

(gdb) set *ADDRESS = VALUE

ptrace

- **ptrace**
  - A system call to perform: **p**rocess **trace**
  - One process (i.e., a tracer) traces another process (i.e., tracee)
    - Observe and control memory and registers of tracee
  - Used to implement breakpoint, debugging, and system call tracing
  - Used by GDB, strace, etc.

```c
void trace_syscalls(pid_t child) {
    int status;
    struct user_regs_struct regs;

    waitpid(child, &status, 0);  // Wait for child to stop

    while (1) {
        // Let the child process run until the next syscall
        ptrace(PTRACE_SYSCALL, child, 0, 0);
        waitpid(child, &status, 0);
        if (WIFEXITED(status)) break;  // Exit loop if child exits

        // Get system call number
        ptrace(PTRACE_GETREGS, child, 0, &regs);
        // System call number (x86_64)

        printf("Syscall: %llu\n", regs.orig_rax);
        // Run until syscall exit
        ptrace(PTRACE_SYSCALL, child, 0, 0);
        waitpid(child, &status, 0);
        if (WIFEXITED(status)) break;  // Exit loop if child exits
    }
}
```

```c
int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <command>\n", argv[0]);
        return 1;
    }

    pid_t child = fork();
    if (child == 0) {
        // Child process: allow tracing
        ptrace(PTRACE_TRACEME, 0, 0, 0);
        // Stop itself so parent can attach
        kill(getpid(), SIGSTOP);
        // Execute the command
        execvp(argv[1], &argv[1]);
        perror("execvp");  // execvp only returns if an error occurs
        return 1;
    } else if (child > 0) {
        // Parent process: trace syscalls of the child
        trace_syscalls(child);
    } else {
        perror("fork");
        return 1;
    }

    return 0;
}
```

ptrace: request

```
#include <sys/ptrace.h>

long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

- **PTRACE_TRACE**
- **PTRACE_PEEKTEXT, PTRACE_PEEKDATA**
  - Read a word at addr (of tracee)
- **PTRACE_POKETEXT, PTRACE_POKEDATA**
  - Write a word at addr (of tracee)
- **PTRACE_GETREGS, PTRACE_SETREGS**
  - Copy/overwrite the tracee's registers
- **PTRACE_CONT**
  - Restart the stopped tracee process
- **PTRACE_SYSCALL, PTRACE_SINGLESTEP**
  - Restart the stopped tracee as for PTRACE_CONT
  - but arrange for the tracee to be stopped at the next entry to or exit from a system call, or after execution of a single instruction

Breakpoints

- **Software breakpoints**
  - x86 instruction: int 3 (0xcc)
  - If CPU executes the instruction “int 3”, CPU raises an exception
- **Hardware breakpoints**
  - CPU reserves the debug register, from DR0-DR7
  - DR0 and DR7 hold the address to be used as a breakpoint
- **Memory breakpoints**
  - Typically implemented by changing the page permissions

Next Assignment: snuDBG

```
└─$ ./snudbg /bin/ls
[*] Tracer with pid=595
[*] Tracee with pid=596
[*] Loading the executable [/bin/ls]
[*] [step 1] rip=7ffff7fd0103 child_status=1407
(snuDbg) help
[*] Available commands:
[*]      regs | get [REG] | set [REG] [value]
[*]      read [addr] [size] | write [addr] [value] [size]
[*]      step | continue | break [addr]
[*]      help
(snuDbg) regs
[*] HANDLE CMD: regs
        rax=0x0 rbx=0x0 rcx=0x0 rdx=0x0
        rbp=0x0 rsp=0x7fffffffdeb0 rsi=0x0 rdi=0x7fffffffdeb0
        r8=0x0 r9=0x0 r10=0x0 r11=0x0
        r12=0x0 r13=0x0 r14=0x0 r15=0x0
        rip=0x7ffff7fd0103 eflags=0x202
(snuDbg) stepi
[-] Not available commands
(snuDbg) step
[*] HANDLE CMD: step
[*] [step 2] rip=7ffff7fd0df0 child_status=1407
(snuDbg) step
[*] HANDLE CMD: step
[*] [step 3] rip=7ffff7fd0df4 child_status=1407
(snuDbg) step
[*] HANDLE CMD: step
[*] [step 4] rip=7ffff7fd0df5 child_status=1407
(snuDbg) continue
[*] HANDLE CMD: continue
LICENSE  Makefile  procmaps.c  procmaps.h  snudbg  snudbg.c  snudbg.h
[*] Exited in 5 steps with status=0
```

References

- ptrace(2) - Linux manual page: https://man7.org/linux/man-pages/man2/ptrace.2.html
- How do debuggers (really) work?: https://events.static.linuxfound.org/sites/events/files/slides/slides_16.pdf
- https://tldp.org/LDP/LG/issue81/sandeep.html
- GDB Internals Manual: https://sourceware.org/gdb/wiki/Internals