운영체제론 실습

- ptrace (1): 함수가 호출하는 시스템 콜 추적

목 차

- Ptrace
  - Tracer – Tracee 관계
  - ptrace 원형
  - Request의 종류
- 예제1: 각 함수는 어떤 시스템 콜을 호출할까요?
- 예제2: 프로세스 레지스터 값을 단번에 불러오기
- 예제3 : strace를 통해 시스템 콜 목록 추적하기
- 프로젝트: 함수가 호출하는 시스템 콜 추적하기

예제 코드 다운로드 경로

아래 명령어를 linux 환경에서 치면 다운받을 수 있음.

$ wget http://ce.hanyang.ac.kr/week7.zip

$ unzip week7.zip

Ptrace

ptrace (= process trace)

- 한 프로세스(tracer)가 다른 프로세스(tracee)의 실행을 관찰 및 제어할 수 있게 하는 시스템 콜
- 피추적자(tracee)의 **메모리**와 **레지스터**를 검사 및 변경할 수 있는 방법 제공

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

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

사용할 request 목록

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

request	설 명
PTRACE_TRACEME	프로세스가 부모에 의해 추적될 것임을 나타냄
PTRACE_PEEKUSER	피추적자의 USER 영역의 오프셋 addr에서 word를 읽음 (레지스터, 프로세스에 대한 기타 정보 포함)
PTRACE_GETREGS	각각 피추적자의 범용 레지스터들이나 부동 소수점 레지스터들을 추적자 내의 주소 data로 복사함
PTRACE_SYSCALL	정지된 피추적 프로세스를 재시작하되 다음번 시스템 호출 진입이나 퇴장에서 또는 한 인스트럭션 실행 후에 피추적자가 멈추도록 함

Ptrace 사용법 (1)

- **PTRACE_TRACEME**

```
ptrace(PTRACE_TRACEME, 0, 0, 0);
```

- 이 프로세스가 부모에 의해 추적될 것임을 나타냄

- **PTRACE_PEEKUSER**

```
ptrace(PTRACE_PEEKUSER, pid, addr, 0);
```

- 피추적자 USER 영역의 오프셋 addr에서 워드를 읽음

- **TRACE_GETREGS**

```
ptrace(PTRACE_GETREGS, pid, 0, &regs);
```

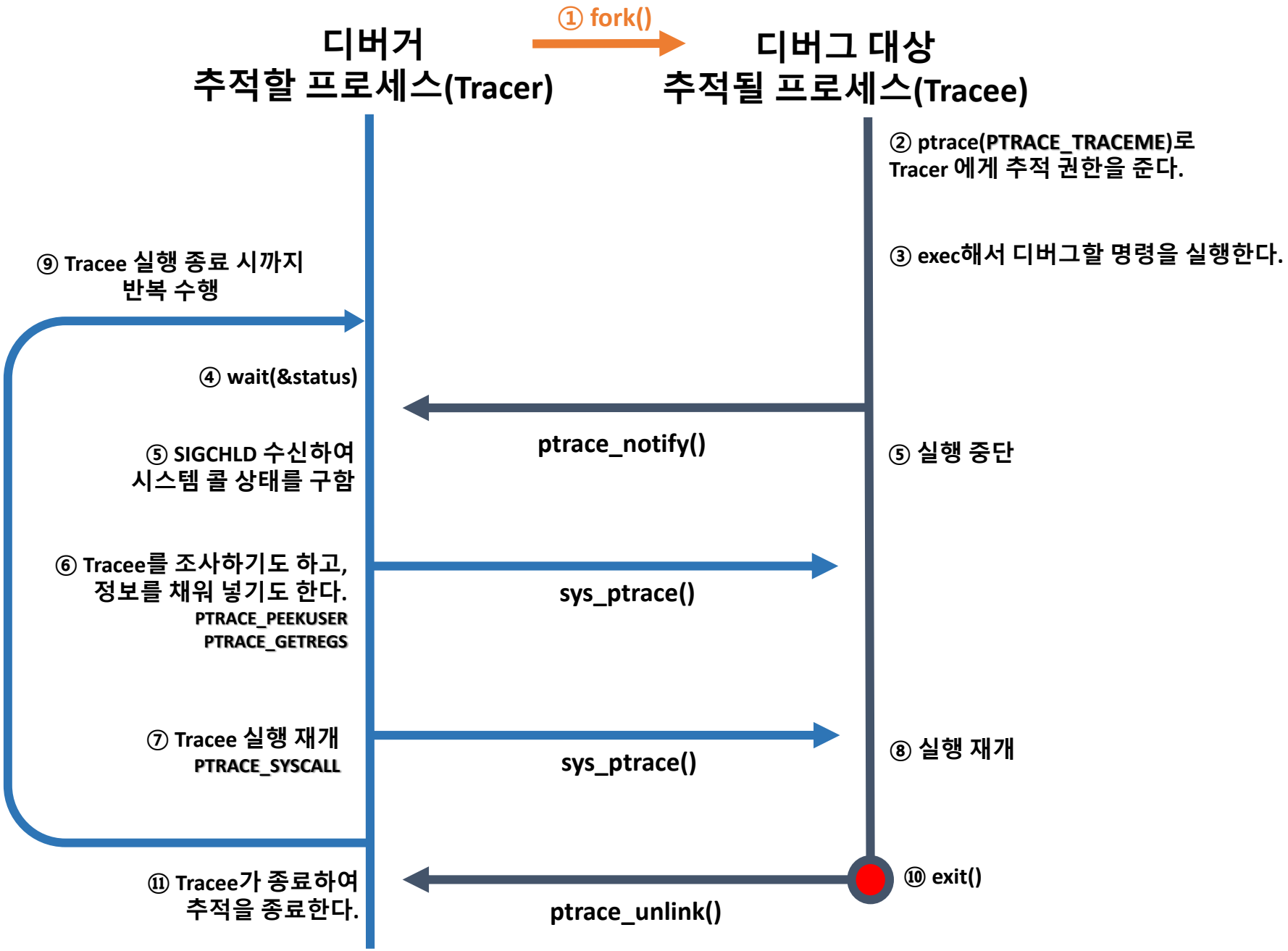
- 피추적자의 범용 레지스터들이나 부동 소수점 레지스터들을 추적자 내의 주소 regs(data)에 복사함 register 에 대한 정보는 <sys/user.h> 참고

Ptrace 사용법 (2)

- **PTRACE_SYSCALL**

```
ptrace(PTRACE_SYSCALL, pid, 0, sig);
```

- 매 시스템호출 진입/퇴장/인스트럭션 실행에 피추적자가 멈추도록 설정됨
- sig(data)가 0이 아니면 피 추적자에게 보낼 시그널 번호로 해석
- 0 이면 시그널을 보내지 않음


① fork()
디버거
추적할 프로세스(Tracer)
디버그 대상
추적될 프로세스(Tracee)
② ptrace(PTRACE_TRACEME)로
Tracer 에게 추적 권한을 준다.
③ exec해서 디버그할 명령을 실행한다.
⑨ Tracee 실행 종료 시까지
반복 수행
④ wait(&status)
ptrace_notify()
⑤ 실행 중단
⑤ SIGCHLD 수신하여
시스템 콜 상태를 구함
⑥ Tracee를 조사하기도 하고,
정보를 채워 넣기도 한다.
PTRACE_PEEKUSER
PTRACE_GETREGS
sys_ptrace()
⑦ Tracee 실행 재개
PTRACE_SYSCALL
sys_ptrace()
⑧ 실행 재개
⑩ exit()
⑪ Tracee가 종료하여
추적을 종료한다.
ptrace_unlink()

예제 1) 각 함수는 어떤 시스템 콜을 호출할까요?

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ptrace.h>
#include <sys/reg.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <sys/user.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
  long orig_rax;
  pid_t child;
  int status;
  int input;
  child = fork();
  if (child == 0) {
    ptrace(PTRACE_TRACEME, 0, NULL, NULL);
    raise(SIGSTOP);
    scanf("%d", &input);
    printf("INPUT: %d\n", input);
  } else {
    while (1) {
      wait(&status);
      if (WIFEXITED(status))
        break;
      orig_rax = ptrace(PTRACE_PEEKUSER, child, 8 * ORIG_RAX, NULL);
      printf("SYSCALL NO: %ld\n", orig_rax);
      ptrace(PTRACE_SYSCALL, child, NULL, NULL);
    }
  }
  return 0;
}
```

예제 1) 각 함수는 어떤 시스템 콜을 호출할까요?

```
jsbaik@jsbaik:~/OS2019/week7/1_which_syscall$ ./which_syscall
SYSCALL NO: 234
SYSCALL NO: 14
SYSCALL NO: 14
SYSCALL NO: 5
SYSCALL NO: 5
SYSCALL NO: 12
SYSCALL NO: 12
SYSCALL NO: 12
SYSCALL NO: 12
SYSCALL NO: 0
0
SYSCALL NO: 0
SYSCALL NO: 5
SYSCALL NO: 5
SYSCALL NO: 1
INPUT: 0
SYSCALL NO: 1
SYSCALL NO: 8
SYSCALL NO: 8
SYSCALL NO: 231
```

scanf 함수가 호출하는 system call 중 하나

printf 함수가 호출하는 system call 중 하나

해당 번호에 해당되는 시스템 콜이 무엇인지 table에서 찾아보자

예제 1) 시스템 콜 번호가 등록되어 있는 위치

위치: /usr/src/linux-$(uname –r)/arch/x86/entry/syscalls/syscall_64.tbl

시스템 콜 번호

시스템 콜 이름

```
0   common  read            __x64_sys_read
1   common  write           __x64_sys_write
2   common  open            __x64_sys_open
3   common  close           __x64_sys_close
4   common  stat            __x64_sys_newstat
5   common  fstat           __x64_sys_newfstat
6   common  lstat           __x64_sys_newlstat
7   common  poll            __x64_sys_poll
8   common  lseek           __x64_sys_lseek
9   common  mmap            __x64_sys_mmap
10  common  mprotect        __x64_sys_mprotect
11  common  munmap          __x64_sys_munmap
12  common  brk         __x64_sys_brk
13  64  rt_sigaction       __x64_sys_rt_sigaction
14  common  rt_sigprocmask      __x64_sys_rt_sigprocmask
15  64  rt_sigreturn        __x64_sys_rt_sigreturn/ptregs
16  64  ioctl           __x64_sys_ioctl
17  common  pread64         __x64_sys_pread64
18  common  pwrite64        __x64_sys_pwrite64
19  64  readv           __x64_sys_readv
20  64  writev          __x64_sys_writev
21  common  access          __x64_sys_access
22  common  pipe            __x64_sys_pipe
23  common  select          __x64_sys_select
24  common  sched_yield     __x64_sys_sched_yield
25  common  mremap          __x64_sys_mremap
26  common  msync           __x64_sys_msync
```

예제 1) user_regs_struct 구조체

위치: /usr/src/linux-$(uname –r)/arch/이하 각 경로

x86/include/asm/user_64.h

0번째

```
69 struct user_regs_struct {
70     unsigned long    r15;
71     unsigned long    r14;
72     unsigned long    r13;
73     unsigned long    r12;
74     unsigned long    bp;
75     unsigned long    bx;
76     unsigned long    r11;
77     unsigned long    r10;
78     unsigned long    r9;
79     unsigned long    r8;
80     unsigned long    ax;
81     unsigned long    cx;
82     unsigned long    dx;
83     unsigned long    si;
84     unsigned long    di;
85     unsigned long    orig_ax;
86     unsigned long    ip;
87     unsigned long    cs;
88     unsigned long    flags;
89     unsigned long    sp;
90     unsigned long    ss;
91     unsigned long    fs_base;
92     unsigned long    gs_base;
93     unsigned long    ds;
94     unsigned long    es;
95     unsigned long    fs;
96     unsigned long    gs;
97 };
```

15번째

x86/entry/calling.h

```
58 #ifdef CONFIG_X86_64
59
60 /*
61  * 64-bit system call stack frame layout defines and helpers,
62  * for assembly code:
63  */
64
65 /* The layout forms the "struct pt_regs" on the stack: */
66 /*
67  * C ABI says these regs are callee-preserved. They aren't saved on kernel entry
68  * unless syscall needs a complete, fully filled "struct pt_regs".
69  */
70 #define R15     0*8
71 #define R14     1*8
72 #define R13     2*8
73 #define R12     3*8
74 #define RBP     4*8
75 #define RBX     5*8
76 /* These regs are callee-clobbered. Always saved on kernel entry. */
77 #define R11     6*8
78 #define R10     7*8
79 #define R9      8*8
80 #define R8      9*8
81 #define RAX     10*8
82 #define RCX     11*8
83 #define RDX     12*8
84 #define RSI     13*8
85 #define RDI     14*8
86 /*
87  * On syscall entry, this is syscall#. On CPU exception, this is error code.
88  * On hw interrupt, it's IRQ number:
89  */
90 #define ORIG_RAX    15*8
91 /* Return frame for iretq */
92 #define RIP     16*8
93 #define CS      17*8
94 #define EFLAGS      18*8
95 #define RSP     19*8
```

구조체 내에서 위치(offset)를 나타내는 매크로 상수

예제 2) 프로세스 레지스터 값을 단번에 불러오기

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <sys/ptrace.h>
 4 #include <sys/reg.h>
 5 #include <sys/syscall.h>
 6 #include <sys/types.h>
 7 #include <sys/user.h>
 8 #include <sys/wait.h>
 9 #include <unistd.h>
10
11 int main() {
12   pid_t child;
13   long orig_rax, rax;
14   long params[3];
15   int status;
16   int insyscall = 0;
17   struct user_regs_struct regs;
18   child = fork();
19   if (child == 0) {
20     ptrace(PTRACE_TRACEME, 0, NULL, NULL);
21     execl("/bin/ls", "ls", NULL);
```

예제 2) 프로세스 레지스터 값을 단번에 불러오기

```
22    } else {
23      while (1) {
24        wait(&status);
25        if (WIFEXITED(status))
26          break;
27        orig_rax = ptrace(PTRACE_PEEKUSER, child, 8 * ORIG_RAX, NULL);
28        if (orig_rax == SYS_write) {
29          if (insyscall == 0) {
30            insyscall = 1;
31            ptrace(PTRACE_GETREGS, child, NULL, &regs);
32            printf("Write called with "
33                   "%lld %lld %lld\n",
34                   regs.rdi, regs.rsi, regs.rdx);
35          } else {
36            rax = ptrace(PTRACE_PEEKUSER, child, 8 * RAX, NULL);
37            printf("Write returned "
38                   "with %ld\n",
39                   rax);
40            insyscall = 0;
41          }
42        }
43        ptrace(PTRACE_SYSCALL, child, NULL, NULL);
44      }
45    }
46    return 0;
47  }
```

예제 2) 프로세스 레지스터 값... (결과화면)

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

```
jsbaik@jsbaik:~/OS2019/week7/3_get_regs$ make
gcc -g --save-temps -o ptrace_regs ptrace_regs.c
jsbaik@jsbaik:~/OS2019/week7/3_get_regs$ ./ptrace_regs
Write called with 1 94903925744912 82
Makefile  ptrace_regs  ptrace_regs.c  ptrace_regs.i  ptrace_regs.o  ptrace_regs.s
Write returned with 82
```

쓰여진 문자열 길이가 82

쓰여진 바이트 단위
길이가 반환됨

예제 3) strace를 통해 시스템 콜 확인

- **strace**는 진단, 디버깅 및 교육용 사용자 공간 유틸리티
- **system call**, **signal** 및 프로세스 상태 변경 및 프로세스와 Linux 커널 간의 상호 작용을 모니터하고 변경하는 데 사용
- strace의 작동은 **ptrace**로 알려진 커널 기능으로 가능

- **Strace 실습 대상 : print_something.c**

```
1 #include <stdio.h>
2 void main(){
3       char c = 'A';
4       putchar(c);
5       return;
6 }
```

예제 3) strace를 통해 시스템 콜 확인 (결과화면)

```
jsbaik@jsbaik:~/OS2019/week7/2_strace$ make
gcc -g --save-temps -o print_something print_something.c
jsbaik@jsbaik:~/OS2019/week7/2_strace$ strace ./print_something
execve("./print_something", ["./print_something"], 0x7fff49acc410 /* 63 vars */) = 0
brk(NULL)                               = 0x56288a002000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=108147, ...}) = 0
mmap(NULL, 108147, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fdc9b2cc000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdc9b2ca000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fdc9accf000
mprotect(0x7fdc9aeb6000, 2097152, PROT_NONE) = 0
mmap(0x7fdc9b0b6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7
000) = 0x7fdc9b0b6000
mmap(0x7fdc9b0bc000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
 0x7fdc9b0bc000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fdc9b2cb4c0) = 0
mprotect(0x7fdc9b0b6000, 16384, PROT_READ) = 0
mprotect(0x5628885a5000, 4096, PROT_READ) = 0
mprotect(0x7fdc9b2e7000, 4096, PROT_READ) = 0
munmap(0x7fdc9b2cc000, 108147)          = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
brk(NULL)                               = 0x56288a002000
brk(0x56288a023000)                     = 0x56288a02
write(1, "A", 1A)                        = 1
exit_group(65)                          = ?
+++ exited with 65 +++
```

호출되는 시스템 콜 목록

본 프로젝트는 strace와 비슷하나, 호출되는 System call 들의 이름을 모두 출력하도록 구현하면 됨

프로젝트:
함수가 호출하는 시스템 콜 추적

결과화면

```
jsbaik@jsbaik:~/OS2019/week7/4_ptrace$ make
gcc -g --save-temps -o my_ptrace my_ptrace.c
jsbaik@jsbaik:~/OS2019/week7/4_ptrace$ ./my_ptrace
[SYSCALL]:SYS_execve                59
[SYSCALL]:SYS_brk                   12
[SYSCALL]:SYS_brk                   12
[SYSCALL]:SYS_access                21
[SYSCALL]:SYS_access                21
[SYSCALL]:SYS_access                21
[SYSCALL]:SYS_access                21
[SYSCALL]:SYS_openat               257
[SYSCALL]:SYS_openat               257
```

... 생략 ...

```
[SYSCALL]:SYS_close                 3
[SYSCALL]:SYS_fstat                 5
[SYSCALL]:SYS_fstat                 5
[SYSCALL]:SYS_write                 1
Makefile  my_ptrace  my_ptrace.c  my_ptrace.i  my_ptrace.o  my_ptrace.s
[SYSCALL]:SYS_write                 1
[SYSCALL]:SYS_close                 3
[SYSCALL]:SYS_close                 3
[SYSCALL]:SYS_close                 3
[SYSCALL]:SYS_close                 3
[SYSCALL]:SYS_exit_group          231
jsbaik@jsbaik:~/OS2019/week7/4_ptrace$
```

수고하셨습니다.

- 다음 시간:

  ptrace(2): 프로세스 해킹하기