System Call Tracing using ptrace

Introduction

- **Synopsis**[1]

  #include <sys/ptrace.h>

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

- **Description**[2]

  The ptrace system call provides a means by which a parent process may observe and control the execution of another process, and examine and change its core image and registers. It is primarily used to implement breakpoint debugging and system call tracing.

* (1), (2) from ptrace man page

Introduction

- **More on Description**

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

request: The value request determines what action needs to perform
pid: The PID of the process to be traced
addr: The address in the USER SPACE of the traced process
(1) to where the ***data*** may be written when instructed to do so, or
(2) from where a word is read and returned as the result of the ptrace system call

Select of ptrace Request
(extracted from ptrace Man Page)

- **PTRACE_TRACEME**

  Called when the child is to be traced by the parent, used only in the child process. Any signal (except SIGKILL) delivered to the process causes it to stop and the parent can be notified using **wait**. Subsequent calls to **exec** (if successful) by this process will cause a **SIGTRAP** to be sent to it.
- **PTRACE_SYSCALL**

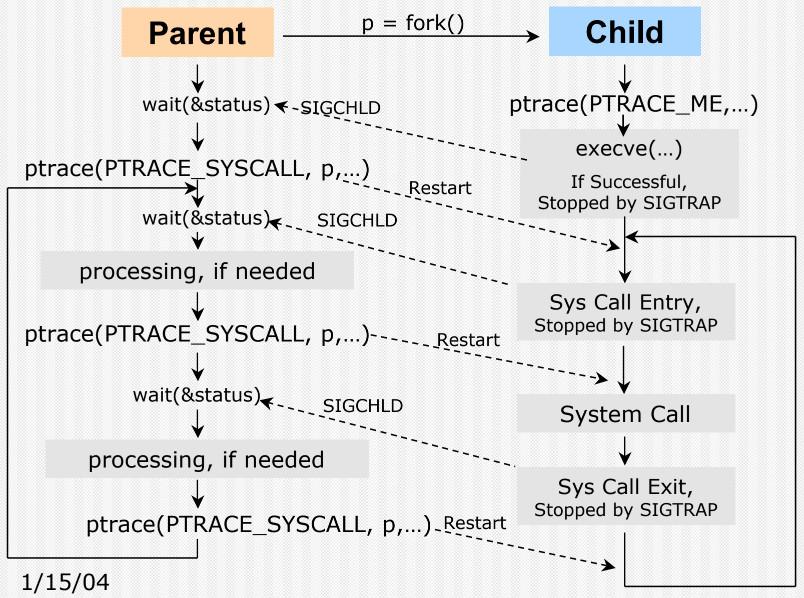
  Restart the stopped child and arranges the child to be stopped at the next ENTRY to or EXIT from a system call. From the parent's perspective, the child will appear to have been stopped by a **SIGTRAP**.
- **PTRACE_PEEKDATA**

  Reads a word at the location ***addr*** in the child's memory, returning it as the result of the ptrace system call.
- **PTRACE_POKEDATA**

  Copies a word from location ***data*** in the parent's memory to location ***addr*** in the child's memory.
- **PTRACE_GETREGS (More OS or Architecture Dependant)**

  Read general purpose registers of the child process into the location ***data*** in the parent.

System Call Tracing



Case Study
Modify the path parameter of the open system call

- **Step by step**

1. Use PTRACE_SYSCALL request to trace into the entry to a system call in the child process.
2. Use PTRACE_GETREGS request to get the general purpose (gp) registers of the child process.
3. Retrieve the system call number from the gp register to make sure it is an **open** system call.
4. Retrieve the address of the path parameter from the gp register and use PTRACE_PEEKDATA request to get the path at location ***addr*** of the child's memory.
5. Modify the path and use PTRACE_POKEDATA to write the path (***data***) back to the location ***addr*** of the child's memory.
6. Restart the stopped child process.

Implementation Detail

- System Call convention

As with the Unix convention, for a system call, before the interruption is raised to transfer the call into kernel mode, the function number is placed in general purpose register EAX and the parameters are passed into EBX, ECX, EDX, ESI, EDI and EBP. For example, the **open** system call has a function number 5 and it has up to three parameters: path, flags and mode. The assembly routine may be simplified as:

```
open:
        mov eax, 5         }
        mov ebx, path      }  calling stack frame
        mov ecx, flags     }
        mov edx, mode      }
        int 80h               // system call entry, transfer to kernel
```

By checking the register value of the child process before system call entry, we are able to get the system call number. Furthermore, we can retrieve and modify the system call parameters.

Implementation Detail

- User Register Struct

```
#include <linux/user.h>
struct user_regs_struct                          Address of Path
{
        long ebx, ecx, edx, esi, edi, ebp, eax ;
        unsigned short ds, __ds, es, __es;
        unsigned short fs, __fs, gs, __gs;
        long orig_eax, eip;
        unsigned short cs, __cs;                 System Call Number
        long eflags, esp;
        unsigned short ss, __ss;
}
```

Implementation Detail

- Sample Code – Start up the tracing

```
…
p = fork();
if (p ==  -1) {
     exit(-1);
}else if (p == 0) {                /* In Child */
     ptrace(PTRACE_TRACEME, 0, 0, 0);
     /* Execute the given process */
     argv[argc] = 0;
     execvp(argv[1], argv+1);
     /* The success of execve will cause a SIGTRAP to be sent to this child process. */
}
/* In parent */
/* Wait for execve to finish*/
wait(&status);
/* Start to trace system calls */
ptrace(PTRACE_SYSCALL, p, 0, 0);
…
```

Implementation Detail

- Sample Code – Get open system call parameters

```
…
/* Start to trace system calls */
ptrace(PTRACE_SYSCALL, p, 0, 0);
/* Wait until the entry to a sys call */
wait(&status);
/* Check the GP register and get the system call number*/
int syscall;
struct user_regs_struct u_in;                        /* #include <linux/user.h> */
ptrace(PTRACE_GETREGS, p, 0, &u_in);
syscall = u_in.orig_eax;
if(syscall == __NR_open) {
     printf("%s", syscall_names[syscall-1]);       /* System call name */
     printf("%08lx ", u_in.ebx);                    /* Address of the path */
     printf("%08lx ", u_in.ecx);                    /* Flag */
     printf("%08lx\n ", u_in.edx);                  /* Mode */
}
…
```

Questions?