B4 - Unix System Programming

B-PSU-402

Bootstrap

strace

2.0

STEP 1: ANALYZING A SYSCALL

The goal of this exercise is to find what corresponds to an *int 0x80* in op code.
You need a program only containing a *main* function, that makes a syscall with the help of the Int 0x80 instruction.
Once the program compiles, use **objdump** and find the corresponds to *int 0x80*.

objdump -D

You can use the following Makefile to compile your program:

```
NAME=test
ASM=nasm
LD=gcc
SRC=main.S
OBJ=$(SRC:.S=.o)
LDFLAGS=-fno-builtin
CFLAGS=-f elf

.S.o:
        $(ASM) $(CFLAGS) $< -c $@

$(NAME): $(OBJ)
        $(LD) $(OBJ) -o $(NAME) $(LDFLAGS)

all: $(NAME)

clean:
        rm -rf $(OBJ)

fclean: clean
        rm -rf $(NAME)
```

STEP 2: DISCOVERING PTRACE

Read ptrace's man(2) and try to understand how this syscall works.

Once you have thoroughly read this syscall's man, make a program that takes a binary name as parameter and:

```
* fork,
* in the child, execute a ptrace with the PT_TRACE_ME flag,
* in the child, excute the binary passed as parameter (using *execve*),
* in the parent, wait for the child using *wait4*,
* in the parent, trace the child with the help of PT_STEP_SINGLESTEP
```

Once the child has been traced, display “The child tracing is now done.”.

```
~/B-PSU-402> ./step2 ./test
The child tracing is now done.
```

STEP 3: STRACE WITH PT_SYSCALL

Write a program that displays a trace with each syscall.

Use **PTRACE_SYSCALL**, documented in ptrace's man(2) page.

Here's the expected output if we trace step 1 with the help of step 3:

Terminal

```
~/B-PSU-402> ./step3 ./test
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...  ret
syscall ...
The child tracing is now done.
```

STEP 4: FUNCTIONAL STRACE WITH PT_SYSCALL

Let's modify step 3 to display the syscalls' names and return values.
You're allowed to use ptrace(2)'s functionality, **PTRACE_GETREGS**.

To find out the syscall's name, you need to retrieve the old_eax field value once PTRACE_SYSCALL exits.

To find the return value, you need to retrieve eax while PTRACE_SYSCALL unblocks a second time.

To find out which of syscall's names corresponds to the old_eax field value, have a look at */usr/include/asm/unistd_32.h*:

```
/*
 * This file contains the system call numbers
 */

#define __NR_restart_syscall   0
#define __NR_exit              1
#define __NRfork               2
#define __NR_read              3
#define __NR_write             4
#define __NR_open              5
#define __NR_close             6
#define __NR_waitpid           7
#define __NR_creat             8
#define __NR_link              9
#define __NR_unlink            10
#define __NR_execve            11
#define __NR_chdir             12
```

Here is an output example:

Terminal

```
~/B-PSU-402> ./step4 ./test
syscall brk ret 0x9ac0000
syscall mmap2 ret 0xac2000
syscall access ret 0xfffffffe
syscall open ret 0x3
syscall fstat64 ret 0x0
syscall mmap2 ret 0xda5000
syscall close ret 0x0
syscall open ret 0x3
syscall read ret 0x200
syscall fstat64 ret 0x0
syscall mmap2 ret 0x2a6000
syscall mprotect ret 0x0
syscall mmap2 ret 0x42c000
syscall mmap2 ret 0x42f000
syscall close ret 0x0
syscall mmap2 ret 0x5f6000
syscall setthread_area ret 0x0
syscall mprotect ret 0x0
syscall mprotect ret 0x0
syscall munmap ret 0x0
syscall exit
The child tracing is now done.
```

STEP 5: THE END

The Bootstrap is over. These are the preliminaries of strace.
However, during your project, you will not be allowed to use *PTRACE_SYSCALL*; therefore you must use:

- PTRACE_SINGLESTEP (to scan your program, instruction by instruction.)
- PTRACE_GETREGS (to retrieve the eip value from each instruction.)
- PTRACE_PEEKTEXT (to retrieve the opcodes pointed by eip.)
- Decode the opcodes to see if there is a int 0x80 (0xcd 80) inside.
- If there is a syscall, retrieve the syscall's number, the arguments and the return value by trying to follow the real strace.

Good luck!