B4 - Unix System Programming

B-PSU-402

Bootstrap

strace

2.0

{EPITECH.}

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)
```

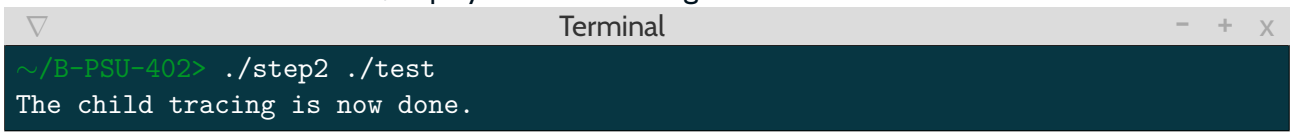
STEP 2: DISCOVERING PTRACE

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.”.



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 ... 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.