Ptrace

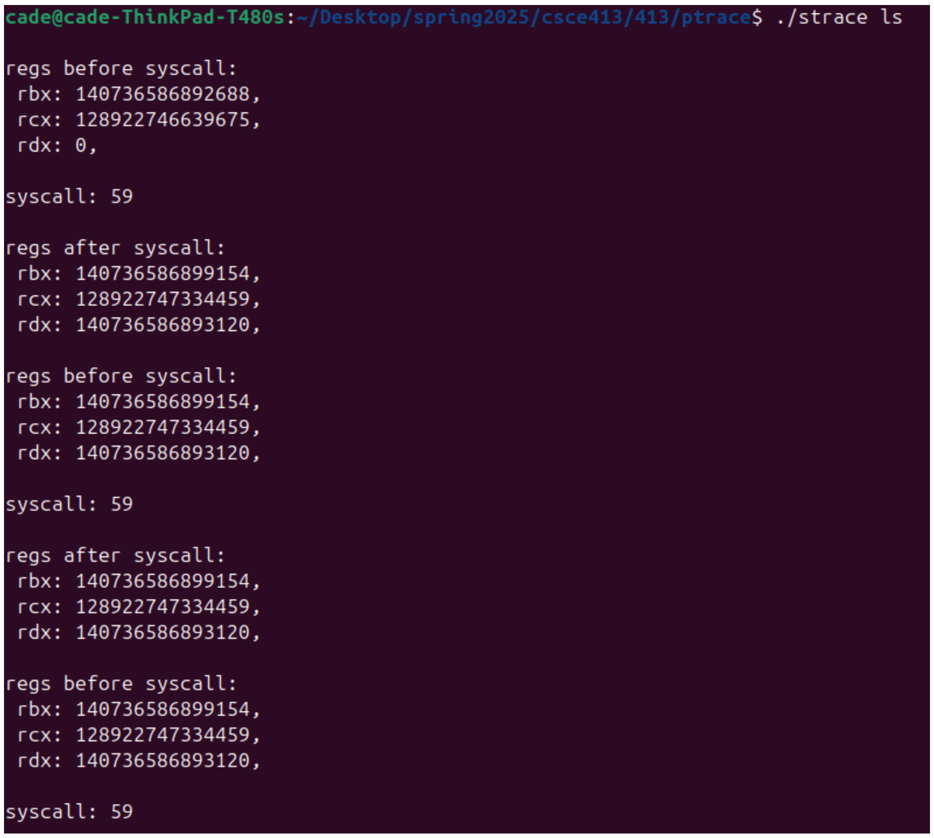
Code can be found here: https://github.com/c2003-tamu/413

Environment Setup

- Clone git repo
- Ensure gcc is installed

Running strace

- Navigate to 413/ptrace directory
- Run command: ./strace <binary>
- Should output all syscalls done by the binary with the registers both before and after the syscall (seen below)



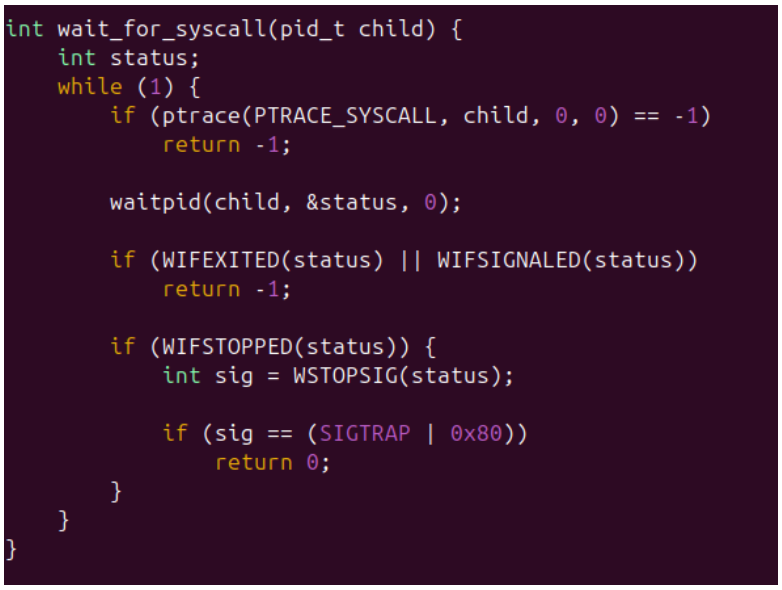
```
cade@cade-ThinkPad-T480s:~/Desktop/spring2025/csce413/413/ptrace$ ./strace ls

regs before syscall:
 rbx: 140736586892688,
 rcx: 128922746639675,
 rdx: 0,

syscall: 59

regs after syscall:
 rbx: 140736586899154,
 rcx: 128922747334459,
 rdx: 140736586893120,

regs before syscall:
 rbx: 140736586899154,
 rcx: 128922747334459,
 rdx: 140736586893120,

syscall: 59

regs after syscall:
 rbx: 140736586899154,
 rcx: 128922747334459,
 rdx: 140736586893120,

regs before syscall:
 rbx: 140736586899154,
 rcx: 128922747334459,
 rdx: 140736586893120,

syscall: 59
```

NOTE: You can translate the syscall numbers to actual syscalls with this table

Creation Process

When creating this program, I took inspiration from the code presented in class and then asked chatGPT when I ran into a wall. This wall was essentially my program would quit whenever the exec() syscall returned 0. Prompt: “This code terminates its strace-like behavior when exec() returns 0. I believe it has something to do with the wait_for_syscall function. why is that and how do i fix it: #include <sys/ptrace.h>#include <sys/user.h>#include <sys/reg.h>#include <sys/wait.h>#include <sys/types.h>#include <unistd.h>#include <stdlib.h>#include <stdio.h>#include <string.h>int child_func(int argc, char **argv);int parent_func(pid_t child);int wait_for_syscall(pid_t child);int main(int argc, char **argv) {if (argc < 2){printf("usage: %s prog args \n", argv[0]);exit(1);}pid_t child_pid = fork();if(child_pid == 0){return child_func(argc-1, argv+1);} else {return parent_func(child_pid);}}int child_func(int argc, char **argv) {char *args [argc+1];memcpy(args, argv, argc * sizeof(char*));args[argc] = NULL;ptrace(PTRACE_TRACEME);kill(getpid(), SIGSTOP);return execvp(args[0], args);}int parent_func(pid_t child) {int status, syscall_number, retval;struct user_regs_struct registers;waitpid(child, &status, 0);ptrace(PTRACE_SETOPTIONS, child, 0, PTRACE_O_TRACESYSGOOD);while(1) {ptrace(PTRACE_GETREGS, child, NULL, &registers);printf("\nregs before syscall:\n rbx: %llu,\n rcx: %llu,\n rdx: %llu, \n", registers.rbx, registers.rcx, registers.rdx);if (wait_for_syscall(child) != 0) break;syscall_number = ptrace(PTRACE_PEEKUSER, child, sizeof(long)*ORIG_RAX);printf("\nsyscall: %i\n", syscall_number);ptrace(PTRACE_GETREGS, child, NULL, &registers);printf("\nregs after syscall:\n rbx: %llu,\n rcx: %llu,\n rdx: %llu, \n", registers.rbx, registers.rcx, registers.rdx);}return 0;} nt wait_for_syscall(pid_t child) { int status; while(1) { ptrace(PTRACE_SYSCALL, child, 0, 0); waitpid(child, &status, 0); if (WIFSTOPPED(status) && WSTOPSIG(status) & 0x80) { return 0; } else { return 1; } } } ” (the format got messed up on copy/paste)

The response from this prompt helped me change my wait_for_syscall function to the following:



```
int wait_for_syscall(pid_t child) {
    int status;
    while (1) {
        if (ptrace(PTRACE_SYSCALL, child, 0, 0) == -1)
            return -1;

        waitpid(child, &status, 0);

        if (WIFEXITED(status) || WIFSIGNALED(status))
            return -1;

        if (WIFSTOPPED(status)) {
            int sig = WSTOPSIG(status);

            if (sig == (SIGTRAP | 0x80))
                return 0;
        }
    }
}
```

This code essentially is waiting for a syscall and returning 0 when one is detected. If there is an error in setting the ptrace or if the program exited for any reason, it returns -1 and the do_parent function will also exit.