Министерство образования и науки Российской Федерации

Московский авиационный институт  
(национальный исследовательский университет)

Кафедра №304

Отчет по лабораторной работе №3

«**Процессы в Linux. Командный интерпретатор**»

по дисциплине «Операционные системы»

Выполнил

Студент группы 3О-310Б

Сомов Д.Н.

Приняли

ассистент каф.304

Балян А.В.

ассистент каф.304

Аляутдин Р.Т.

Москва, 2017

**Задание:**

Ознакомиться с функциями, позволяющими работать с процессами в Linux: создавать их, заменять образ одного процесса на другой и расщеплять процессы. В качестве практического применения этих функций написать командный интерпретатор, который исполняет введенные команды с помощью расщепления и подмены образа процесса. Проверить написанную программу с помощью Memory Sanitizer, checkpatch.pl. Использовать систему контроля версий git и удаленный репозиторий на GitHub..

**Описание:**

Написанный мной командный интерпретатор использует типичную связку функций fork() + одну из функций из семейства exec(). После fork’a с помощью switch (и знания PID родительского и дочернего процессов) образ дочернего процесса подменяется на образ процесса, который нужно выполнить согласно введенной команде. Родительский процесс ожидает завершения дочернего с помощью функции wait() и после этого предлагает ввести команду снова. Сочетание Ctrl+C прервет выполнение программы, вынуждая терминал вновь напечатать приглашение к следующей команде.

Есть и другой режим работы: если перед командой набрать ключевое слово «\_\_silent» (его можно изменить в исходном коде, т.к. оно определено через #define), то она будет запущена «в фоне», а терминал будет свободен для дальнейшего выполнения команд. При этом посыл сигнала SIGINT вызовет прерывание терминала, а не запущенной программе.

Все простейшие команды, (которые, по сути, вызывают определенные программы из /bin/ ), можно выполнить, просто передав команду и ее аргументы в execvp(), чем я и воспользовался (а для cd и exit дописал обработку отдельно).

Также, чтобы не загромождать командный терминал, интерпретатор предлагает выводить содержимое потока stdout либо в лог-файл (лежит в той же папке, что и программа), либо в другой терминал (при этом нужно указать его PID). Можно также осуществлять работу с терминалом и в привычном виде, когда вывод появляется после введенной в терминале команды.

**Исходный код:**

/\*

\* Programming Language – C

\* Author – Dmitriy Somov, 3O-310B

\* Suprisingly Not Born Again SH

\*/

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <signal.h>

#include <fcntl.h>

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define BUFSIZE 512

#define SILENT\_ARG "\_\_silent"

int main(void)

{

char str[BUFSIZE];

char \*termPID;

char \*out;

char \*logto = NULL;

char logwd[BUFSIZE];

int exitCode;

int i;

int cnt;

int fd;

int silentMode = 0;

pid\_t pid;

pid\_t child\_pid;

// Print a greeting and receive PID to log into.

termPID = (char \*)calloc(BUFSIZE, 1);

out = "This is Surprisingly Not a \"Born Again Shell\", or snbash.\n";

write(1, out, strlen(out));

out = "Enter:\n\033[1mterminal PID\033[0m to send output to it,\n";

write(1, out, strlen(out));

out = "\033[1m'0'\033[0m to output to log file, or\n";

write(1, out, strlen(out));

out = "\033[1m'this'\033[0m to output to this terminal.\n";

write(1, out, strlen(out));

out = "\033[2m(for first option open another terminal\n";

write(1, out, strlen(out));

out = "and execute 'ps' to find out its PID)\033[0m\n: ";

write(1, out, strlen(out));

// If input here is incorrect, it will be catched later.

read(0, termPID, 10);

termPID[strlen(termPID)-1] = '\0';

if(strcmp(termPID, "this") == 0) {

logto = (char \*)calloc(BUFSIZE, 1);

sprintf(logto, "/proc/%d/fd/1", getpid());

out = "Sending output to this terminal.\n\n";

write(1, out, strlen(out));

}

else if (strcmp(termPID, "0") != 0) {

logto = (char \*)calloc(BUFSIZE, 1);

sprintf(logto, "/proc/%s/fd/1", termPID);

out = "Sending output to terminal with PID ";

write(1, out, strlen(out));

write(1, termPID, strlen(termPID));

write(1, "\n\n", 2);

} else {

getcwd(logwd, BUFSIZE);

strcat(logwd, "/outlog.txt");

out = "Sending output to logfile 'outlog.txt'.\n\n";

write(1, out, strlen(out));

}

free(termPID);

// Execute commands while not receiving 'exit'.

while (1) {

(void) signal(SIGINT, SIG\_DFL);

// Print current working directory.

i = 0;

out = "\033[38;5;166m\033[1m";

getcwd(str, BUFSIZE);

strcat(str, ":\033[0m ");

write(1, out, strlen(out));

write(1, str, strlen(str));

for (i = 0; i < BUFSIZE; i++)

str[i] = '\0';

// Read a command and process 'exit' and empty commands.

// If command is incorrect, it will be catched later.

i = read(0, str, BUFSIZE);

if (i == 0 || strcmp(str, "\n") == 0)

continue;

else if (strcmp(str, "exit\n") == 0)

break;

// Delete junk symbols from the end of string.

for (i = strlen(str)-1; i >= 0; i--) {

if (isalnum(str[i]) || ispunct(str[i]))

break;

if (str[i] == ' ')

str[i] = '\0';

else if (str[i] == '\n')

str[i] = '\0';

}

// Count arguments and split string into args array.

cnt = 0;

i = 0;

do {

if (str[i] == ' ')

cnt++;

i++;

} while (str[i-1] != '\0');

char \*args[cnt+2];

i = 0;

args[i] = strtok(str, " ");

i++;

if (strcmp(args[0], SILENT\_ARG) == 0) {

silentMode = 1;

i--;

} else {

silentMode = 0;

}

while (args[i - 1 + silentMode] != NULL) {

args[i] = strtok(NULL, " ");

i++;

}

// Special processing for 'cd' command.

if (strcmp(args[0], "cd") == 0) {

if (chdir(args[1]) == -1)

perror("Failed to perform 'cd'");

continue;

}

// Fork and execute command in the child process.

pid = fork();

switch (pid) {

case -1:

printf("Forking failed.\n");

perror(NULL);

break;

case 0:

if (silentMode == 0)

(void) signal(SIGINT, SIG\_DFL);

else

(void) signal(SIGINT, SIG\_IGN);

// Open log file and redirect output to it.

if (logto == NULL) {

fd = open(logwd, O\_RDWR|O\_APPEND|O\_CREAT, 0600);

} else {

fd = open(logto, O\_RDWR|O\_CREAT);

}

if (fd == -1) {

out = "Failed to open log stream. Stopping.\n";

write(1, out, strlen(out));

perror("Open stream error");

exit(3);

}

dup2(fd, 1);

close(fd);

write(1, "\n", 1);

if (logto != NULL) {

out = "\033[93m\033[1m";

write(1, out, strlen(out));

}

if (silentMode == 1) {

out = SILENT\_ARG;

write(1, out, strlen(out));

write(1, " ", 1);

}

for (i = 0; i < cnt+1-silentMode; i++) {

write(1, args[i], strlen(args[i]));

write(1, " ", 1);

}

if (logto != NULL)

write(1, ">>\033[0m\n", 7);

else

write(1, ">>\n", 3);

if (execvp(args[0], args) == -1) {

exitCode = 2;

perror("Error executing command");

exit(exitCode);

}

break;

default:

if (silentMode == 0) {

(void) signal(SIGINT, SIG\_IGN);

// Parent waits for child's command completion.

child\_pid = wait(&i);

if (WIFEXITED(i)) {

printf("\033[2m--- Operation finished, ");

printf("code %d ---\033[0m\n\n", WEXITSTATUS(i));

} else {

printf("Child was interrupted.\n\n");

exitCode = 1;

}

} else {

(void) signal(SIGINT, SIG\_DFL);

printf("\033[2m--- Silently started new process, ");

printf("PID = %d ---\033[0m\n\n", pid);

sleep(1);

}

break;

}

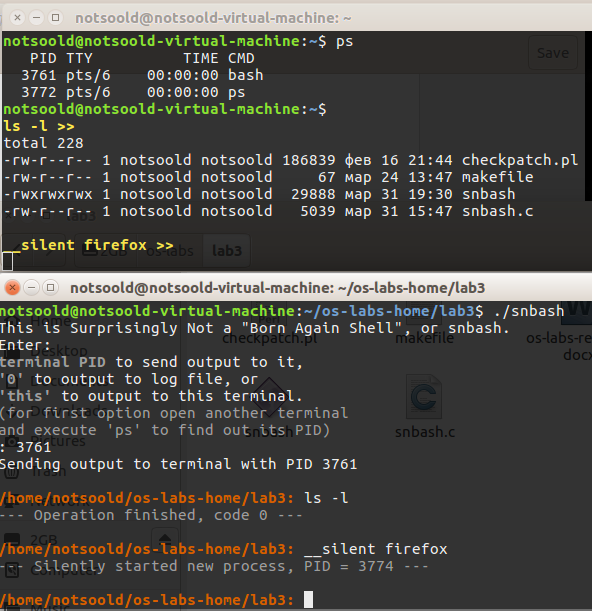
}

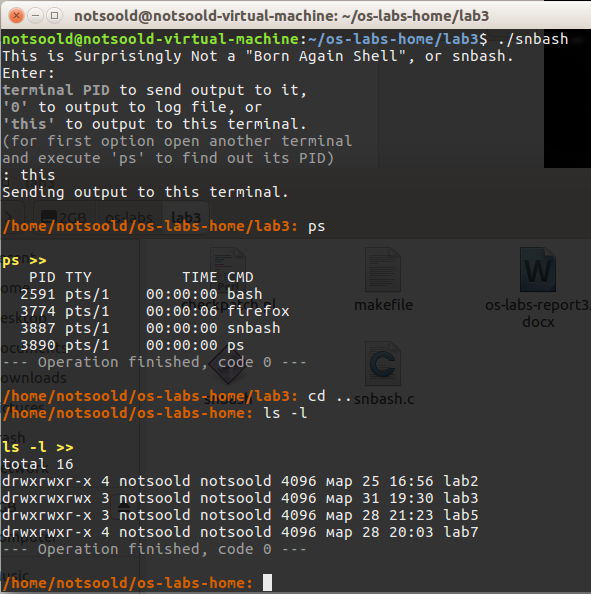
exitCode = 0;

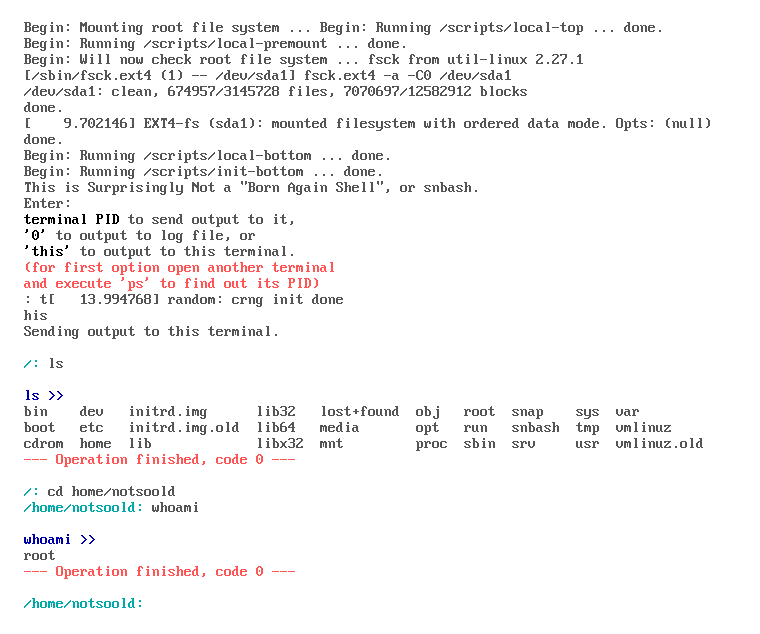
exit(exitCode);

}

**Скриншоты:**

****

****

****