Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу
«Операционные системы»

Студент: Кузьмичев Александр Николаевич
Группа: М80 – 306Б-18
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020

Содержание

Постановка задачи

Составить и отладить программу на языке Си, родительский процесс которой считывает стандартный входной поток, отдает его дочернему процессу, который удаляет «задвоенные» пробелы и выводит его в файл(имя файла также передается от родительского процесса.

Общие сведения о программе

Программа состоит из одного файла main.c. В данном файле используются заголовочные файлы stdio.h, stdlib.h, unistd.h, stdbool.h, fcntl.h, string.h, sys/wait.h.

Программа использует следующие системные вызовы:

1. **read –** для чтения данных из входного потока.
2. **write** – для записи данных в файл или выходной поток.
3. **pipe –** для создания канала, через который процессы смогут обмениваться информацией.
4. **fork –** для создания дочернего процесса.
5. **close –** для закрытия выходного файла.

Общий метод и алгоритм решения

- Проверить, задан ли файл для вывода как аргумент, создать дочерний процесс(с помощью **fork**) и **pipe**(для передачи данных из родительского процесса в дочерний) обработать возможные ошибки.
- Из родительского процесса: записать в **pipe** имя файла для вывода, переданное программе в качестве аргумента. Считать все символы, переданные во входной поток и передать их через **pipe** в дочерний. Считывается по 100 символов за раз, а так как **read** возвращает количество считанных символов, то когда символы во входном потоке закончатся, будет нетрудно это понять.
- Из дочернего процесса: считать из **pipe** названия выходного файла, открыть его с помощью **open**(при неудаче завершиться с кодом выхода 1). Далее считывать из **pipe** по 100 символов в **char buffer[100]**, и удалять из них лишние пробелы с помощью функции **parse_string**(на случай, если строка начинается с пробелов, а предыдущая строка закончилась на некоторое количество пробелов, предусмотрена булевская переменная space, передаваемая в функцию по указателю, которая будет сигнализировать о таких ситуациях). Далее полученная строка(без лишних пробелов) записывается в выходной файл с помощью функции **write**. Когда символы, переданные из родительского процесса закончатся, дочерний процесс закроет файл с помощью функции **close** и завершится с кодом выхода 0.

- После передачи символов входного потока в дочерний процесс, родительский процесс ожидает его завершения посредством функции **wait**. После проверяется код выхода дочернего процесса, и в стандартный вывод передается информация о том, был ли успешно завершен дочерний процесс.

Код программы

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>
#include <string.h>
#include <sys/wait.h>
#include <fcntl.h>

int parse_string(char buf[], int size, bool* prev) {
    int temp_size = 0;
    char copy[size];
    bool space = *prev;
    for (int i = 0; i < size; ++i) {
        if (buf[i] != ' ' || !space) {
            if (buf[i] == ' ') {
                space = true;
            } else {
                space = false;
            }
            copy[temp_size] = buf[i];
            temp_size++;
        }
    }
    for (int i = 0; i < temp_size; ++i) {
        buf[i] = copy[i];
    }
    *prev = space;
    return temp_size;
}

int main(int argc, char** argv) {
    int fd[2];
    const char* arg_fail = "This program needs a name file as argument\n";
    const char* pipe_fail = "Pipe failure. Terminate\n";
    const char* fork_fail = "Fork failure. Terminate\n";
    if (argc < 2) {
        write(STDOUT_FILENO, arg_fail, strlen(arg_fail));
        exit(1);
    }
    if (pipe(fd) < 0) {
        write(STDOUT_FILENO, pipe_fail, strlen(pipe_fail));
        exit(1);
    }
    int pid = fork();
    if (pid == -1) {
        write(STDOUT_FILENO, fork_fail, strlen(fork_fail));
        exit(1);
    }

    if (pid == 0) { // child code
        // обработать ввод, вывести в stdout
```

```
        char buffer[100];
        read(fd[0], buffer, 100);
        int output_fd = open(buffer, O_RDWR | O_TRUNC);
        if (output_fd == -1) {
            close(output_fd);
            exit(1);
        }

        int read_count = read(fd[0], buffer, 100);
        bool space = false;
        while (read_count == 100) {
            int parsed_size = parse_string(buffer, read_count, &space);
            write(output_fd, buffer, parsed_size);
            read_count = read(fd[0], buffer, 100);
        }
        if (read_count != 0) {
            int parsed_size = parse_string(buffer, read_count, &space);
            write(output_fd, buffer, parsed_size);
        }
        close(output_fd);
        exit(0);

    } else { // parent code
        char buffer[100];
        write(fd[1], argv[1], 100);
        int read_count = read(STDIN_FILENO, buffer, 100);
        while (read_count == 100) {
            write(fd[1], buffer, 100);
            read_count = read(STDIN_FILENO, buffer, 100);
        }
        if (read_count != 0) {
            write(fd[1], buffer, read_count);
        }
        int statlock = 0;
        wait(&statlock);
        const char* success = "Process completed successfully\n";
        const char* fail = "Process failed\n";
        if (statlock == 0) {
            write(STDOUT_FILENO, success, strlen(success));
        } else {
            write(STDOUT_FILENO, fail, strlen(fail));
        }
    }

    return 0;
}
```

Демонстрация работы программы

```
alex@alex-lenovo:~/CLionProjects/OS_lab_2/cmake-build-debug$ cat input_file
Lorem ipsum dolor sit amet consectetur
adipiscing elit, sed do eiusmod tempor incididunt ut labore
et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat.
```

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore

ue fugiat nulla pariatur.

alex@alex-lenovo:~/CLionProjects/OS_lab_2/cmake-build-debug$ hexdump input_only_spaces

0000000 2020 2020 2020 000a

0000007

alex@alex-lenovo:~/CLionProjects/OS_lab_2/cmake-build-debug$ ls

CMakeCache.txt CMakeFiles cmake_install.cmake input_file input_only_spaces maga Makefile OS_lab OS_lab.cbp test2.txt testout testout2

alex@alex-lenovo:~/CLionProjects/OS_lab_2/cmake-build-debug$ touch result

alex@alex-lenovo:~/CLionProjects/OS_lab_2/cmake-build-debug$ $ ./OS_lab result < input_file

Process completed successfully

alex@alex-lenovo:~/CLionProjects/OS_lab_2/cmake-build-debug$ cat result

Lorem ipsum dolor sit amet consectetur

adipiscing elit, sed do eiusmod tempor incididunt ut labore

et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud

exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore

ue fugiat nulla pariatur.

alex@alex-lenovo:~/CLionProjects/OS_lab_2/cmake-build-debug$./OS_lab result < input_only_spaces

Process completed successfully

alex@alex-lenovo:~/CLionProjects/OS_lab_2/cmake-build-debug$ hexdump result

0000000 0a20

0000002

alex@alex-lenovo:~/CLionProjects/OS_lab_2/cmake-build-debug$./OS_lab not_exist < input_only_spaces

Process failed

вывод strace:

execve("./OS_lab", ["./OS_lab", "result"], 0x7ffe869ced18 /* 52 vars */) = 0

brk(NULL) = 0x562106f55000

access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=144444, ...}) = 0

mmap(NULL, 144444, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe399268000

close(3) = 0

access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe399266000

mmap(NULL, 4131552, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fe398c74000

mprotect(0x7fe398e5b000, 2097152, PROT_NONE) = 0

mmap(0x7fe39905b000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fe39905b000

mmap(0x7fe399061000, 15072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fe399061000

close(3) = 0

arch_prctl(ARCH_SET_FS, 0x7fe3992674c0) = 0

mprotect(0x7fe39905b000, 16384, PROT_READ) = 0

mprotect(0x562106694000, 4096, PROT_READ) = 0

mprotect(0x7fe39928c000, 4096, PROT_READ) = 0

munmap(0x7fe399268000, 144444) = 0

pipe([3, 4])   = 0

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fe399267790) = 632

write(4, "result\0CLUTTER_IM_MODULE=xim\0LS_"..., 100) = 100

read(0, "Lorem   ipsum   "..., 100) = 100

write(4, "Lorem   ipsum   "..., 100) = 100

read(0, ", sed do eiusmod tempor incididu"..., 100) = 100

write(4, ", sed do eiusmod tempor incididu"..., 100) = 100

read(0, " minim veniam, quis nostrud\nexer"..., 100) = 100

write(4, " minim veniam, quis nostrud\nexer"..., 100) = 100

read(0, "   "..., 100) = 100

write(4, "   "..., 100) = 100

read(0, "   ut aliquip ex ea commod"..., 100) = 100

write(4, "   ut aliquip ex ea commod"..., 100) = 100

read(0, "tate velit esse cillum dolore\nue"..., 100) = 75

write(4, "tate velit esse cillum dolore\nue"..., 75) = 75

wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 632

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=632, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---

write(1, "Process completed successfully\n", 31Process completed successfully

) = 31

exit_group(0)   = ?

+++ exited with 0 +++

Вывод

В результате данной лабораторной работы я научился работать с процессами, реализовать обмен информацией между дочерним и родительским процессом, а так же работать с strace.