Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-210Б-23

Студент: Кудаева В.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 31.10.24

Москва, 2024

Постановка задачи

Вариант 16:

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода. Правило проверки: строка должна оканчиваться на «.» или «;».

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создаёт пайп и помещает дескрипторы в fd[0], fd[1], для чтения и записи.
- int write(int fd, const void* buff, int count); – записывает по дескриптору fd count байт из buff.
- void exit(int number); – вызывает нормальное завершение программы с кодом number.
- int dup2(int fd1, int fd2); – делает эквиваентными дескрипторы fd1 и fd2.
- int exec(char* path, const char* argc); – заменяет текущий процесс на процесс path, с аргументами argc;
- int close(int fd); – закрывает дескриптор fd.
- pid_t wait(int status) — функция, которая приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится.

В файле parent.c создается родительский и дочерний процессы, которые обмениваются данными через два канала (pipe1 и pipe2). Родительский процесс запрашивает имя файла и отправляет строки на запись в этот файл дочернему процессу, который принимает строки через канал, проверяет их на валидность (оканчиваются ли они на . или ;) и записывает в файл, если строка валидна. Дочерний процесс отвечает родителю сообщением "OK" или об ошибке, а родитель выводит это сообщение или завершает передачу, если пользователь вводит "exit".

В файле child.c дочерний процесс принимает имя файла, которое передается как аргумент командной строки, и открывает файл для записи. После получения строки через стандартный ввод, процесс проверяет её на корректное окончание и записывает в файл, если строка валидна, отправляя родительскому процессу подтверждение "OK" или сообщение об ошибке.

Код программы

parent.c

```c
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>

#define BUFFER_SIZE 1024

int main() {
    // Массивы для каналов, где 0 - конец для чтения, 1 - для записи
    int pipe1[2], pipe2[2];
    pid_t pid;
    char buffer[BUFFER_SIZE];
    char resp_buffer[BUFFER_SIZE];

    // Создание каналов для связи между процессами
    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        perror("Pipe creation failed");
        exit(EXIT_FAILURE);
    }

    // Ввод и чтение имени файла
    write(STDOUT_FILENO, "Enter the file name to record: ", 32);
    char filename[256];
    ssize_t len = read(STDIN_FILENO, filename, sizeof(filename));
    if (filename[len - 1] == '\n') {
        filename[len - 1] = '\0';
    }
```

```c
    // Создание дочернего процесса
    pid = fork();

    if (pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        dup2(pipe1[STDIN_FILENO], STDIN_FILENO);
        dup2(pipe2[STDOUT_FILENO], STDOUT_FILENO);
        close(pipe1[STDOUT_FILENO]);
        close(pipe2[STDIN_FILENO]);

        execl("./child", "child", filename, NULL);
        perror("execl failed");
        exit(EXIT_FAILURE);
    } else {
```

```
        close(pipe1[STDIN_FILENO]);
        close(pipe2[STDOUT_FILENO]);

        // Ввод строки и отправка дочернему процессу
        write(STDOUT_FILENO, "Enter a line (or 'exit' to exit): ", 35);
        while ((len = read(STDIN_FILENO, buffer, BUFFER_SIZE)) > 0) {

            if (buffer[len - 1] == '\n') {
                buffer[len - 1] = '\0';
            }
            if (strcmp(buffer, "exit") == 0) {
                break;
            }

            // Отправка строки дочернему процессу, используя дескриптор pipe[1]
            write(pipe1[STDOUT_FILENO], buffer, strlen(buffer) + 1);

            // Проверка ошибок
            ssize_t bytes_read = read(pipe2[STDIN_FILENO], resp_buffer, sizeof(resp_buffer) - 1);
            if (bytes_read > 0) {
                resp_buffer[bytes_read] = '\0';
                if (strcmp(resp_buffer, "OK\n") != 0) {
                    write(STDOUT_FILENO, "Error: ", 8);
                    write(STDOUT_FILENO, resp_buffer, bytes_read);
                }
            }
        }

        close(pipe1[STDOUT_FILENO]);
        close(pipe2[STDIN_FILENO]);
        // Ожидание завершения дочернего процесса
        wait(NULL);
        exit(EXIT_SUCCESS);
    }

    return 0;
}
```

child.c

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

#define BUFFER_SIZE 1024

bool validate_string(const char *str) {
    size_t len = strlen(str);
    if (len == 0) return false;
    return (str[len - 1] == '.' || str[len - 1] == ';');
}

int main(int argc, char *argv[]) {
    if (argc < 2){
        exit(EXIT_FAILURE);
    }
    char buffer[BUFFER_SIZE];
    char error_message[] = "Invalid string\n";
```

```c

    // Открытие файла для записи строк
    int fd = open(argv[1], O_WRONLY | O_TRUNC | O_CREAT | O_APPEND, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        perror("Failed to open file");
        exit(EXIT_FAILURE);
    }
    ssize_t bytes_read;
    while ((bytes_read = read(STDIN_FILENO, buffer, sizeof(buffer) - 1)) > 0) {
        buffer[bytes_read] = '\0';
        printf("Child: received string '%s', %zd bytes read\n", buffer, bytes_read);
        // Проверка строки на валидность
        if (validate_string(buffer)) {
            write(fd, buffer, strlen(buffer));
            write(fd, "\n", 1);
            {
                const char msg[] = "OK\n";
                write(STDOUT_FILENO, msg, sizeof(msg) - 1);
            }
        } else {
            write(STDOUT_FILENO, error_message, sizeof(error_message) - 1);
        }
    }
    if (bytes_read == -1) {
        perror("Failed to read from stdin");
    }
    close(fd);
    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

Тестирование

```
Enter the file name to record: file1.txt
Enter a line (or 'exit' to exit): ewgrv;
474j8GH%-=^
Error: Invalid string
sfg84jdifh;
+==3de.
afca00000adca7yh3JJJJKN
Error: Invalid string
exit
```

file1.txt

```
ewgrv;
sfg84jdifh;
+==3de.
```

strace ./parent

execve("./parent", ["./parent"], 0x7ffcf4df51b0 /* 72 vars */) = 0

brk(NULL) = 0x558ec6415000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fff7293bdd0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f1a0177d000

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

openat(AT_FDCWD,
"/home/ksenoox/OpenFOAM/ksenoox-v2312/platforms/linux64GccDPInt32Opt/lib/
glibc-hwcaps/x86-64-v4/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such
file or directory)

openat(AT_FDCWD,
"/usr/lib/openfoam/openfoam2312/platforms/linux64GccDPInt32Opt/lib/dummy/
libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT_FDCWD,
"/usr/lib/openfoam/openfoam2312/platforms/linux64GccDPInt32Opt/lib/dummy",
{st_mode=S_IFDIR|0755, st_size=4096, ...}, 0) = 0

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

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=27211, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 27211, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1a01776000

close(3)   = 0

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\0P\237\2\0\0\0\0"..., 832) =
832**

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) =
784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) =
48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\
324\224\323\236S"..., 68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH)
= 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) =
784

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f1a0154d000

mprotect(0x7f1a01575000, 2023424, PROT_NONE) = 0

mmap(0x7f1a01575000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f1a01575000

mmap(0x7f1a0170a000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1bd000) = 0x7f1a0170a000

mmap(0x7f1a01763000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f1a01763000

mmap(0x7f1a01769000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1a01769000

close(3)   = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f1a0154a000

arch_prctl(ARCH_SET_FS, 0x7f1a0154a740) = 0

set_tid_address(0x7f1a0154aa10)   = 334966

set_robust_list(0x7f1a0154aa20, 24)   = 0

rseq(0x7f1a0154b0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f1a01763000, 16384, PROT_READ) = 0

mprotect(0x558ec5867000, 4096, PROT_READ) = 0

mprotect(0x7f1a017b7000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7f1a01776000, 27211)   = 0

pipe2([3, 4], 0)   = 0

pipe2([5, 6], 0)   = 0

write(1, "Enter the file name to record: \0", 32Enter the file name to record:) = 32

read(0, svsf;

"svsf;\n", 256)   = 6

**clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|
CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f1a0154aa10) = 335084**

close(3) = 0

close(6) = 0

write(1, "Enter a line (or 'exit' to exit)"..., 35Enter a line (or 'exit' to exit):) = 35

read(0, dfb

"dfb\n", 1024) = 4

write(4, "dfb\0", 4) = 4

read(5, "Invalid string\n", 1023) = 15

write(1, "Error: \0", 8Error:) = 8

write(1, "Invalid string\n", 15Invalid string

) = 15

read(0, exit

"exit\n", 1024) = 5

close(4) = 0

close(5) = 0

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=335084, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

wait4(-1, NULL, 0, NULL) = 335084

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе лабораторной работы мне удалось реализовать взаимодействие между родительским и дочерним процессами с использованием каналов pipe и системных вызовов. Я узнала, что такое процесс и канал, а также научилась пользоваться некоторыми системными вызовами.