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: Малафеев И. Д.
Преподаватель: Бахарев В.Д.
Оценка: _____
Дата: 26.12.24
Москва, 2024

Постановка задачи

Вариант 17.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Общий метод и алгоритм решения

Использованные системные вызовы:

• pid_t fork(void); – создает дочерний процесс.

• int pipe(int *fd); – создаёт pipe и помещает дескрипторы в fd[0], fd[1], для чтения и записи.

• int write(int fd, const void* buffer, int count); – записывает по дескриптору fd count байт из buffer.

• int dup2(int fd1, int fd2); – перенаправляет дескрипторы.

• 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.

• void exit(int number); – вызывает нормальное завершение программы с кодом number.

В рамках лабораторной работы я реализовал 3 программы на C: Parent.c, Child1.c и Child2.c, которые работают совместно для редактирования строк. Программа Parent.c запрашивает у пользователя путь к файлам, создает pipe для связи с дочерними процессами и использует fork() для их создания. Дочерние процессы выполняют программу Child1/2, а родительский процесс перенаправляет строки по 2м разным pipe в зависимости от их длинны. Child1\2 считывает строки удаляют из них все гласные и выводят результаты в файлы. В случае ошибок, программа выводит соответствующие сообщения об ошибках. Также родительский процесс ожидает завершение дочерних процессов после вывода результатов.

Код программы

Parent.c

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

#define MAX_FILENAME 256
#define MAX_STRING 1024

int main() {
    int pipe1[2], pipe2[2];
    char filename1[MAX_FILENAME], filename2[MAX_FILENAME];
    ssize_t bytes_read;

    // Получаем имена файлов
    bytes_read = read(STDIN_FILENO, filename1, MAX_FILENAME);
    if (bytes_read <= 0) {
        write(STDERR_FILENO, "Error reading filename1\n", 23);
        exit(1);
    }
    filename1[strcspn(filename1, "\n")] = 0;

    bytes_read = read(STDIN_FILENO, filename2, MAX_FILENAME);
    if (bytes_read <= 0) {
        write(STDERR_FILENO, "Error reading filename2\n", 23);
        exit(1);
    }
    filename2[strcspn(filename2, "\n")] = 0;

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        write(STDERR_FILENO, "Pipe creation failed\n", 20);
        exit(1);
    }

    pid_t pid1 = fork();
    if (pid1 < 0) {
        write(STDERR_FILENO, "Fork failed\n", 12);
        exit(1);
    }

    if (pid1 == 0) {
        close(pipe1[1]);
        close(pipe2[0]);
        close(pipe2[1]);

        dup2(pipe1[0], STDIN_FILENO);

        char *args[] = {"./child1", filename1, NULL};
        execv(args[0], args);
        write(STDERR_FILENO, "Exec failed\n", 12);
        exit(1);
    }

    pid_t pid2 = fork();
    if (pid2 < 0) {
        write(STDERR_FILENO, "Fork failed\n", 12);
        exit(1);
    }
```

```
    close(pipe1[0]);
    close(pipe2[0]);

    char input[MAX_STRING];
    while ((bytes_read = read(STDIN_FILENO, input, MAX_STRING)) > 0) {
        if (input[bytes_read-1] == '\n') {
            input[bytes_read-1] = '\0';
            bytes_read--;
        }

        if (bytes_read > 10) {
            write(pipe2[1], input, bytes_read + 1);
        } else {
            write(pipe1[1], input, bytes_read + 1);
        }
    }

    close(pipe1[1]);
    close(pipe2[1]);

    wait(NULL);
    wait(NULL);

    return 0;
}
```

Child1/2:

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <ctype.h>

#define MAX_STRING 1024

int is_vowel(char c) {
    c = tolower(c);
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}

void remove_vowels(char *str) {
    int i, j;
    for (i = 0, j = 0; str[i] != '\0'; i++) {
        if (!is_vowel(str[i])) {
            str[j] = str[i];
            j++;
        }
    }
    str[j] = '\0';
}
```

```c
int main(int argc, char *argv[]) {
    if (argc != 2) {
        write(STDERR_FILENO, "Usage error\n", 11);
        exit(1);
    }

    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        write(STDERR_FILENO, "File opening failed\n", 19);
        exit(1);
    }

    char buffer[MAX_STRING];
    ssize_t bytes_read;

    while ((bytes_read = read(STDIN_FILENO, buffer, MAX_STRING)) > 0) {
        remove_vowels(buffer);
        write(STDOUT_FILENO, buffer, strlen(buffer));
        write(STDOUT_FILENO, "\n", 1);
        write(fd, buffer, strlen(buffer));
        write(fd, "\n", 1);
    }

    close(fd);
    return 0;
}
```

Протокол работы программы

Тестирование:

Strace:

```
execve("./parent", ["./parent", "-o", "log.txt"], 0x7ffdc10a4a50 /* 36 vars */) = 0
brk(NULL)                               = 0x5637626c7000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc1398e160) = -1 EINVAL (Invalid
argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f305ee5f000
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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=16299, ...},
AT_EMPTY_PATH) = 0
mmap(NULL, 16299, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f305ee5b000
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\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) = 0x7f305ec32000
mprotect(0x7f305ec5a000, 2023424, PROT_NONE) = 0
mmap(0x7f305ec5a000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) =
0x7f305ec5a000
mmap(0x7f305edef000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x7f305edef000
mmap(0x7f305ee48000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) =
0x7f305ee48000
mmap(0x7f305ee4e000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f305ee4e000
close(3)                                = 0
```

```
mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f305ec2f000
arch_prctl(ARCH_SET_FS, 0x7f305ec2f740) = 0
set_tid_address(0x7f305ec2fa10)         = 17076
set_robust_list(0x7f305ec2fa20, 24)     = 0
rseq(0x7f305ec300e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f305ee48000, 16384, PROT_READ) = 0
mprotect(0x563736774000, 4096, PROT_READ) = 0
mprotect(0x7f305ee99000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f305ee5b000, 16299)           = 0
read(0, out1
"out1\n", 256)                = 5
read(0, out2
"out2\n", 256)                = 5
pipe2([3, 4], 0)                    = 0
pipe2([5, 6], 0)                    = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f305ec2fa10) = 17363
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f305ec2fa10) = 17364
close(3)                            = 0
close(5)                            = 0
read(0, asdsdfsdf
"asdsdfsdf\n", 1024)           = 10
write(4, "asdsdfsdf\0", 10sdsdfsdf
)           = 10
read(0, asfeaasfdgdfgfg
"asfeaasfdgdfgfg\n", 1024)      = 16
write(6, "asfeaasfdgdfgfg\0", 16sfsfdgdfgfg
)      = 16
read(0, aaadd
"aaadd\n", 1024)              = 6
write(4, "aaadd\0", 6dd
)               = 6
read(0, "", 1024)                   = 0
close(4)                            = 0
close(6)                            = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17363,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17364,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
```

```
wait4(-1, NULL, 0, NULL)                = 17363
wait4(-1, NULL, 0, NULL)                = 17364
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Данная лабораторная работа оказалась необычной потому что мне до этого не приходилось использовать системные вызовы. Я научился использовать их, а также обмениваться данными между процессами используя каналы (pipe). Трудности возникли на моменте изучения и понимания материала, однако всё оказалось выполнимым.