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-209БВ-24

Студентка: Полевая А.О.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 02.10.24

Москва 2024

**Постановка задачи**

Необходимо разработать программный комплекс, состоящий из родительского и двух дочерних процессов. Родительский процесс должен запросить у пользователя имена двух файлов, а затем принимать строки произвольной длины с консоли. В зависимости от длины строки осуществляется фильтрация:

* если строка содержит 10 или менее символов, она направляется через первый канал первому дочернему процессу;
* если строка содержит более 10 символов, она направляется через второй канал второму дочернему процессу.

Каждый дочерний процесс выполняет обработку полученных данных: удаляет все гласные символы из строк и сохраняет результат в соответствующий файл, имя которого было введено пользователем.

**Общий метод и алгоритм решения**

В лабораторной работе используются системные вызовы POSIX:

* **pid\_t fork(void)** — создание дочернего процесса.
* **int pipe(int fd[2])** — создание неименованного канала для обмена данными.
* **int dup2(int oldfd, int newfd)** — перенаправление стандартного ввода/вывода.
* **int execl(const char \*path, const char \*arg, …)** — запуск нового исполняемого файла в процессе.
* **ssize\_t write(int fd, const void \*buf, size\_t count)** — запись данных в файл или канал.
* **ssize\_t read(int fd, void \*buf, size\_t count)** — чтение данных из файла или канала.
* **int wait(int \*status)** — ожидание завершения дочернего процесса.

Алгоритм:  
1. Родительский процесс создаёт два канала и два дочерних процесса.  
2. Каждый дочерний процесс перенаправляет свой stdin на соответствующий канал и запускает отдельную программу (child1 или child2).  
3. Родитель принимает строки от пользователя и направляет их либо в pipe1, либо в pipe2 в зависимости от длины строки.  
4. Дочерние процессы удаляют гласные и записывают результат в указанные пользователем файлы.

**Код программы**

**Parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <string.h>

#define MAX\_STR 256

int main() {

int pipe1[2], pipe2[2];

char file1[128], file2[128];

// создаём два канала

if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {

perror("pipe");

exit(1);

}

printf("Введите имя файла для child1: ");

scanf("%127s", file1);

printf("Введите имя файла для child2: ");

scanf("%127s", file2);

getchar(); // убрать \n после scanf

pid\_t pid1 = fork();

if (pid1 == -1) {

perror("fork");

exit(1);

}

if (pid1 == 0) {

// child1

dup2(pipe1[0], STDIN\_FILENO); // читаем из pipe1

close(pipe1[1]);

close(pipe2[0]); close(pipe2[1]);

execl("./child1", "child1", file1, NULL);

perror("execl child1");

exit(1);

}

pid\_t pid2 = fork();

if (pid2 == -1) {

perror("fork");

exit(1);

}

if (pid2 == 0) {

// child2

dup2(pipe2[0], STDIN\_FILENO); // читаем из pipe2

close(pipe2[1]);

close(pipe1[0]); close(pipe1[1]);

execl("./child2", "child2", file2, NULL);

perror("execl child2");

exit(1);

}

// Родительский процесс

close(pipe1[0]);

close(pipe2[0]);

char buffer[MAX\_STR];

printf("Введите строки (exit для завершения):\n");

while (1) {

if (fgets(buffer, MAX\_STR, stdin) == NULL) break;

buffer[strcspn(buffer, "\n")] = 0; // убрать \n

if (strcmp(buffer, "exit") == 0) break;

size\_t len = strlen(buffer);

strcat(buffer, "\n"); // добавляем перевод строки для детей

if (len <= 10) {

write(pipe1[1], buffer, strlen(buffer));

} else {

write(pipe2[1], buffer, strlen(buffer));

}

}

close(pipe1[1]);

close(pipe2[1]);

wait(NULL);

wait(NULL);

printf("Родитель завершил работу.\n");

return 0;

}

**Child1.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STR 256

// удаляем гласные

void remove\_vowels(char \*str) {

char \*src = str, \*dst = str;

while (\*src) {

if (strchr("aeiouyAEIOUY", \*src) == NULL) {

\*dst++ = \*src;

}

src++;

}

\*dst = '\0';

}

int main(int argc, char \*argv[]) {

if (argc < 2) {

fprintf(stderr, "Не передано имя файла\n");

exit(1);

}

FILE \*f = fopen(argv[1], "w");

if (!f) {

perror("fopen");

exit(1);

}

char buffer[MAX\_STR];

while (fgets(buffer, MAX\_STR, stdin)) {

buffer[strcspn(buffer, "\n")] = 0;

remove\_vowels(buffer);

fprintf(f, "%s\n", buffer);

}

fclose(f);

return 0;

}

**Child2.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STR 256

// удаляем гласные

void remove\_vowels(char \*str) {

char \*src = str, \*dst = str;

while (\*src) {

if (strchr("aeiouyAEIOUY", \*src) == NULL) {

\*dst++ = \*src;

}

src++;

}

\*dst = '\0';

}

int main(int argc, char \*argv[]) {

if (argc < 2) {

fprintf(stderr, "Не передано имя файла\n");

exit(1);

}

FILE \*f = fopen(argv[1], "w");

if (!f) {

perror("fopen");

exit(1);

}

char buffer[MAX\_STR];

while (fgets(buffer, MAX\_STR, stdin)) {

buffer[strcspn(buffer, "\n")] = 0;

remove\_vowels(buffer);

fprintf(f, "%s\n", buffer);

}

fclose(f);

return 0;

}

**Протокол работы программы**

Пример работы:

$ ./parent

Введите имя файла для child1: short.txt

Введите имя файла для child2: long.txt

Введите строки (exit для завершения):

cat

meowmeowmeowmeow

dogdog

exit

Файлы после выполнения:

short.txt:

ct

dgdg

long.txt:

mwmwmwmwmwmwmw

**Strace:**

execve("./main", ["./main"], 0x7ffd996ab720 /\* 26 vars \*/) = 0

brk(NULL) = 0x5f911eaee000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x79f62ae8f000

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=20155, ...}) = 0

mmap(NULL, 20155, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x79f62ae8a000

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\0\220\243\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

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 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, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x79f62ac00000

mmap(0x79f62ac28000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x79f62ac28000

mmap(0x79f62adb0000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x79f62adb0000

mmap(0x79f62adff000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x79f62adff000

mmap(0x79f62ae05000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x79f62ae05000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x79f62ae87000

arch\_prctl(ARCH\_SET\_FS, 0x79f62ae87740) = 0

set\_tid\_address(0x79f62ae87a10) = 535

set\_robust\_list(0x79f62ae87a20, 24) = 0

rseq(0x79f62ae88060, 0x20, 0, 0x53053053) = 0

mprotect(0x79f62adff000, 16384, PROT\_READ) = 0

mprotect(0x5f90ea454000, 4096, PROT\_READ) = 0

mprotect(0x79f62aec7000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x79f62ae8a000, 20155) = 0

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

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

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

getrandom("\x4a\x96\x3c\x2a\xc0\x3e\xb8\x69", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5f911eaee000

brk(0x5f911eb0f000) = 0x5f911eb0f000

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 48Введите имя файла для child1: ) = 48

read(0, short.txt

"short.txt\n", 1024) = 10

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 48Введите имя файла для child2: ) = 48

read(0, long.txt

"long.txt\n", 1024) = 9

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x79f62ae87a10) = 539

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x79f62ae87a10) = 540

close(3) = 0

close(5) = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\320\270 (exi"..., 64Введите строки (exit для завершения):

) = 64

read(0, cat

"cat\n", 1024) = 4

write(4, "cat\n", 4) = 4

read(0, meowmeowmeowmeow

"meowmeowmeowmeow\n", 1024) = 17

write(6, "meowmeowmeowmeow\n", 17) = 17

read(0, dogdog

"dogdog\n", 1024) = 7

write(4, "dogdog\n", 7) = 7

read(0, exit

"exit\n", 1024) = 5

close(4) = 0

close(6) = 0

wait4(-1, NULL, 0, NULL) = 539

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=539, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

wait4(-1, NULL, 0, NULL) = 540

write(1, "\320\240\320\276\320\264\320\270\321\202\320\265\320\273\321\214 \320\267\320\260\320\262\320\265\321\200\321\210\320\270\320"..., 48Родитель завершил работу.

) = 48

exit\_group(0) = ?

+++ exited with 0 +++

**Фрагмент вывода:**

pipe2([3, 4], 0) = 0 <создание первого канала>

pipe2([5, 6], 0) = 0 <создание второго канала>

clone(...) = 539 <создание первого дочернего процесса>

clone(...) = 540 <создание второго дочернего процесса>

write(4, "cat\n", 4) = 4 <запись короткой строки в pipe1>

write(6, "meowmeowmeowmeow\n", 17) = 17 <запись длинной строки в pipe2>

wait4(-1, NULL, 0, NULL) = 539 <ожидание завершения первого дочернего процесса>

wait4(-1, NULL, 0, NULL) = 540 <ожидание завершения второго дочернего процесса>

**Вывод**

В данной лабораторной работе я освоил работу с процессами и межпроцессным взаимодействием в ОС Linux.

Были использованы системные вызовы fork, pipe, dup2, execl, read, write, wait.

В результате работы программы строки корректно распределяются между двумя дочерними процессами по длине и записываются в соответствующие файлы после фильтрации.

Проблема, с которой я столкнулся: сначала в файл записывалась только первая строка, так как строки передавались без перевода строки. После добавления символа новой строки ('\n') к строкам в родительском процессе программа заработала корректно.