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-212БВ-24

Студент: Брежнев Г. О.

Преподаватель: Бахарев В.Д.

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

Дата: 17.10.25

Москва, 2025

**Постановка задачи**

**Группа вариантов 5:** 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.

**Вариант 21.**

Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* int pipe(int pipefd[2]); – создание каналов.
* pid\_t fork(void); – создание процессов.
* int open(char \*pathname, int flags, mode\_t mode); - открытие файла
* ssize\_t read(int fd, void \*buf, size\_t n); - чтение из stdin
* ssize\_t write(int fd, void \*buf, size\_t n); - запись в каналы
* int dup2(int oldfd, int newfd); - перенаправление ввода
* int execlp(char \*file, char \*arg, ...); - запуск программ
* pid\_t waitpid(pid\_t pid, int \*status, int options); - ожидание процессов
* int close(int fd); - закрытие дескрипторов
* void \_exit(int status); - аварийный выход

**Родительский процесс:**

1. Читает два имени файлов из STDIN для результатов дочерних процессов

2. Создает два канала pipe1 и pipe2 для межпроцессного взаимодействия

3. Запускает первый дочерний процесс через fork()

- Дочерний процесс перенаправляет STDIN на pipe1

- Запускает программу child1 с первым именем файла

4. Запускает второй дочерний процесс через fork()

- Дочерний процесс перенаправляет STDIN на pipe2

- Запускает программу child2 со вторым именем файла

5. Закрывает неиспользуемые концы каналов

6. Читает входные данные из STDIN построчно

7. Распределяет строки между процессами:

- Нечетные строки отправляет в pipe1 первому процессу

- Четные строки отправляет в pipe2 второму процессу

8. После завершения ввода закрывает каналы

9. Ожидает завершения обоих дочерних процессов

**Дочерний процесс (child1/child2):**

1. Открывает указанный файл для записи

2. Читает данные из STDIN (перенаправленного из канала)

3. Для каждой полученной строки:

- Переворачивает символы в строке задом-наперед

- Сохраняет перевернутую строку в файл

4. Закрывает файл и завершает работу

**Алгоритм переворота строки:**

- Игнорирует символ новой строки \n при перевороте

- Переставляет симметричные символы относительно центра строки

- Сохраняет оригинальное форматирование строк

**Код программы**

**parent.c**

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

int read\_line(int fd, char \*buf, int maxlen) {

int i = 0;

char c;

while (i < maxlen - 1) {

ssize\_t n = read(fd, &c, 1);

if (n == 0) break;

buf[i++] = c;

if (c == '\n') break;

}

buf[i] = '\0';

if (i > 0 && buf[i - 1] == '\n') {

buf[i - 1] = '\0';

}

return i;

}

int main() {

int pipe1[2];

int pipe2[2];

pid\_t pid1;

pid\_t pid2;

char input[256];

char file1[256], file2[256];

int line = 1;

if (read\_line(STDIN\_FILENO, file1, 256) == 0) \_exit(1);

if (read\_line(STDIN\_FILENO, file2, 256) == 0) \_exit(1);

pipe(pipe1);

pipe(pipe2);

pid1 = fork();

if (pid1 == 0) {

close(pipe1[1]);

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

close(pipe1[0]);

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

\_exit(1);

}

pid2 = fork();

if (pid2 == 0) {

close(pipe2[1]);

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

close(pipe2[0]);

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

\_exit(1);

}

close(pipe1[0]);

close(pipe2[0]);

char c;

int i = 0;

while (1) {

ssize\_t n = read(STDIN\_FILENO, &c, 1);

if (n == 0) break;

input[i++] = c;

if (c == '\n'){

if (line % 2 == 1) {

write(pipe1[1], input, i);

} else {

write(pipe2[1], input, i);

}

i = 0;

line++;

}

}

if (i > 0) {

if (line % 2 == 1) {

write(pipe1[1], input, i);

} else {

write(pipe2[1], input, i);

}

}

close(pipe1[1]);

close(pipe2[1]);

waitpid(pid1, NULL, 0);

waitpid(pid2, NULL, 0);

return 0;

}

**child1.c**

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

void reverse(char \*s, int len){

if (len <= 0) return;

int end = len - 1;

if (s[end] == '\n') {

end--;

}

if (end < 0) return;

int swaps = (end + 1) / 2;

for (int i = 0; i < swaps; i++) {

char tmp = s[i];

s[i] = s[end - i];

s[end - i] = tmp;

}

}

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

if (argc < 2) \_exit(1);

int fd = open(argv[1], O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd < 0) \_exit(2);

char buf[256];

int i = 0;

char c;

while (read(STDIN\_FILENO, &c, 1) > 0) {

buf[i++] = c;

if (c == '\n'){

reverse(buf, i);

write(fd, buf, i);

i = 0;

}

}

if (i > 0) {

reverse(buf, i);

write(fd, buf, i);

}

close(fd);

return 0;

}

**child2.c**

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

void reverse(char \*s, int len){

if (len <= 0) return;

int end = len - 1;

if (s[end] == '\n') {

end--;

}

if (end < 0) return;

int swaps = (end + 1) / 2;

for (int i = 0; i < swaps; i++) {

char tmp = s[i];

s[i] = s[end - i];

s[end - i] = tmp;

}

}

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

if (argc < 2) \_exit(1);

int fd = open(argv[1], O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd < 0) \_exit(2);

char buf[256];

int i = 0;

char c;

while (read(STDIN\_FILENO, &c, 1) > 0) {

buf[i++] = c;

if (c == '\n'){

reverse(buf, i);

write(fd, buf, i);

i = 0;

}

}

if (i > 0) {

reverse(buf, i);

write(fd, buf, i);

}

close(fd);

return 0;

}

**Протокол работы программы**

~/Projects/mai\_os\_lab/lab1 main gcc parent.c -o parent ✔

~/Projects/mai\_os\_lab/lab1 main gcc child1.c -o child1 ✔

~/Projects/mai\_os\_lab/lab1 main gcc child2.c -o child2 ✔

~/Projects/mai\_os\_lab/lab1 main ./parent ✔

child1\_out.txt

child2\_out.txt

hello

bye

beard

dead

deed

indeed

butter

goodbye

^C

~/Projects/mai\_os\_lab/lab1 main cat child1\_out.txt INT ✘ 32s

olleh

draeb

deed

rettub

~/Projects/mai\_os\_lab/lab1 main cat child2\_out.txt ✔

eyb

daed

deedni

eybdoog

**Вывод**

При выполнении данной работы я научился основам работы с системными вызовами Unix для создания многопроцессных приложений. Я освоил механизмы межпроцессного взаимодействия через каналы (pipes) и управление дочерними процессами. Также я приобрел практические навыки работы с файловыми дескрипторами, перенаправлением ввода-вывода и синхронизацией процессов.