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-209Б-23

Студент: Артонкин В.Н.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 06.10.25

Москва, 2025

Постановка задачи

Вариант 20.

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. Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void) - создает дочерний процесс.
- int pipe(int *fd) - создает канал и возвращает два дескриптора
- int close(int fd) - закрывает файловый дескриптор
- int execl(const char *path, const char *arg, ...) - загружает и запускает новую программу, полностью заменяя текущий процесс
- void free(void *ptr) - освобождает память.

Сначала были созданы все необходимые переменные, с учетом тех, которые потом использовались для работы с файловыми дескрипторами. Также были считаны названия файлов, в которые дочерние процессы писали строки. После дважды происходил форк родительского процесса, обеспечивалась корректная работа внутри родительских и дочерних процессов, закрывались ненужные файловые дескрипторы. Затем были добавлены: блок кода, отвечающий за считывание строк из стандартного потока ввода и распределение их по соответствующим дочерним процессам; блок кода, который инвертирует строки и записывает в файлы. Также была добавлена обработка ошибок

Код программы

parent.c

```
#include <string.h>

#include <stdio.h>

#include <unistd.h> // Системные вызовы

#include <stdlib.h> // Выход из программмы через exit, дин. пам. тут нет

#include <sys/wait.h> // Ожидание завершения дочернего процесса



#define MAX_CHAR 256

#define READ_END 0
```

```
#define WRITE_END 1

int main(){
   // Инициализация и считывание названий файлов
   char filename1[MAX_CHAR];
   char filename2[MAX_CHAR];

   printf("Введите имя 1-го файла\n");
   if (fgets(filename1, sizeof(filename1), stdin) == NULL) {
   perror("filename1 error");
   exit(1);
  }
   printf("Введите имя 2-го файла\n");
   if (fgets(filename2, sizeof(filename2), stdin) == NULL) {
   perror("filename2 error");
   exit(1);
  }
   // Обработка названий файлов
   filename1[strcspn(filename1, "\n")] = '\0';
   filename2[strcspn(filename2, "\n")] = '\0';


   int pipe1[2], pipe2[2];
   pid_t pid1, pid2;

   // Инициализация файловых дескрипторов для кажого из пайпов
   if (pipe(pipe1) == -1 || pipe(pipe2) == -1){
     perror("descriptors initialization error: pipe error");
     exit(1);
   }
```

```
// ----------------------
// 1 форк
// ----------------------

switch(pid1 = fork()){

// Fork не удался
  case -1:
    perror("fork 1 error");
    exit(1);
  case 0:
    close(pipe1[WRITE_END]);
    close(pipe2[WRITE_END]);
    close(pipe2[READ_END]);
  // Переназначение 0 дескриптора на pipe2[READ_END]
    if (dup2(pipe1[READ_END], STDIN_FILENO) == -1){
      perror("dup error");
      exit(1);
    }

    // Надо закрывать не только запись, но и чтение, потому что
    // чтение и так будет обеспечено из 0 канала, переназначенного на pipe1[READ_END]
    // и не нужно держать еще один дескриптор, ссылающийся на pipe1[READ_END], так как ядро будет думать,
    // что из него тоже может прийти какая-то информация, раз он не закрыт, и не будет закрывать процесссы,
    // завершающиеся через EOF, например getline.

    // Тогда, при закрытии pipe1[WRITE_END] у родителя будет автоматически отправлен EOF и функции завершатся
```

```
  close(pipe1[READ_END]);

  execl("./child1", "./child1", filename1, NULL);

  // Выполнится, если execl не отработает как надо
  perror("execl child 1 error");
  exit(1);

default:
  // ----------------------
  // 2 форк
  // ----------------------

  switch(pid2 = fork()){

  case -1:
    perror("fork 1 error");
    exit(1);
  case 0:
    close(pipe1[READ_END]);
    close(pipe1[WRITE_END]);
    close(pipe2[WRITE_END]);

    // Переназначение 0 дескриптора на pipe2[READ_END]
    if (dup2(pipe2[READ_END], STDIN_FILENO) == -1){
      perror("dup error");
      exit(1);
    }
```

```
  close(pipe2[READ_END]);
  execl("./child2", "./child2", filename2, NULL);

  // Выполнится, если execl не отработает как надо
  perror("execl child 2 error");
  exit(1);

default:
  close(pipe1[READ_END]);
  close(pipe2[READ_END]);

  // ----------------------
  // Считывание строк
  // ----------------------

  char *buffer = NULL;
  size_t size = 0;

  while (1) {
     printf("Введите строку или 'exit' для выхода:\n");
     fflush(stdout);
     ssize_t len = getline(&buffer, &size, stdin);
     if (len == -1) break;   // EOF или ошибка

     // buffer[strcspn(buffer, "\n")] = '\0';  // Если доавлена, то тогда 10, а не 11

     if (strcmp(buffer, "exit\n") == 0){
       close(pipe1[WRITE_END]);
       close(pipe2[WRITE_END]);
       break;
```

```
        }



        if (strlen(buffer) <= 10 + 1) {
            if (write(pipe1[WRITE_END], buffer, len) == -1){
              perror("writing to pipe1 error");
              exit(1);
            }
        } else {
            if (write(pipe2[WRITE_END], buffer, len) == -1){
              perror("writing to pipe2 error");
              exit(1);
            }
        }

        // printf("%s", buffer);
      }

      // ----------------------
      // Завершение
      // ----------------------

      free(buffer);
      wait(NULL);
      wait(NULL);
      return 0;
    }
  }
}
```

child1.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>


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

    if (argc < 2) {

        perror("No filename given\n");

        exit(1);

    }


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

    if (!f) {

        perror("fopen error");

        exit(1);

    }


    char *buffer = NULL;

    size_t len = 0;

    ssize_t read;


    while ((read = getline(&buffer, &len, stdin)) != -1) {

        // printf("Передана строка: %s", buffer);  // Уже есть \n

        if (read > 0){

            for (int i=read-2; i >= 0; i--){

                fputc(buffer[i], f);

            }

            // Складывается в буфер

            fputc('\n', f);
```

```
        }
        // Пушится в файл
        fflush(f);
}

    fflush(stdout);
    free(buffer);
    fclose(f);
    // exit(0);
    return 0;
}
```

child2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        perror("No filename given\n");
        exit(1);
    }

    FILE *f = fopen(argv[1], "w");
    if (!f) {
        perror("fopen error");
        exit(1);
    }

    char *buffer = NULL;
```

```
    size_t len = 0;
    ssize_t read;

    while ((read = getline(&buffer, &len, stdin)) != -1) {
        // printf("Передана строка: %s", buffer);  // Уже есть \n
        if (read > 0){
            for (int i=read-2; i >= 0; i--){
                fputc(buffer[i], f);
            }
            // Складывается в буфер
            fputc('\n', f);
        }
        // Пушится в файл
        fflush(f);
}

    fflush(stdout);
    free(buffer);
    fclose(f);
    // exit(0);
    return 0;
}
```

Протокол работы программы

```
$ strace -o trace.log ./parent
Введите имя 1-го файла
fedya.txt
Введите имя 2-го файла
vasya.txt
Введите строку или 'exit' для выхода:
1234567890
Введите строку или 'exit' для выхода:
12345678901234567890
Введите строку или 'exit' для выхода:
```

```
argentina manit negra
Введите строку или 'exit' для выхода:
mama mila ramu
Введите строку или 'exit' для выхода:
abc123deg
Введите строку или 'exit' для выхода:
exit
$ cat < fedya.txt
0987654321
ged321cba
$ cat < vasya.txt
09876543210987654321
argen tinam anitnegra
umar alim amam
```

execve("./parent", ["./parent"], 0x7ffe6dc176f0 /* 33 vars */) = 0

brk(NULL) = 0x55b979fae000

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=21784, ...}) = 0

mmap(NULL, 21784, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc030da8000

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\260>\2\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0755, st_size=1901536, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc030da6000

mmap(NULL, 1914496, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fc030bd2000

mmap(0x7fc030bf4000, 1413120, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7fc030bf4000

mmap(0x7fc030d4d000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17b000) = 0x7fc030d4d000

mmap(0x7fc030d9c000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c9000) = 0x7fc030d9c000

mmap(0x7fc030da2000, 13952, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fc030da2000

close(3) = 0

arch_prctl(ARCH_SET_FS, 0x7fc030da7540) = 0

mprotect(0x7fc030d9c000, 16384, PROT_READ) = 0

mprotect(0x55b953760000, 4096, PROT_READ) = 0

mprotect(0x7fc030dd8000, 4096, PROT_READ) = 0

munmap(0x7fc030da8000, 21784) = 0

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0

brk(NULL) = 0x55b979fae000

brk(0x55b979fcf000) = 0x55b979fcf000

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\2171-\320\263\320\276 \321\204\320"..., 40) = 40

fstat(0, {st_mode=S_IFREG|0644, st_size=104, ...}) = 0

read(0, "fedya.txt\nvasya.txt\n1234567890\n1"..., 4096) = 104

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\2172-\320\263\320\276 \321\204\320"..., 40) = 40

pipe([3, 4]) = 0

pipe([5, 6]) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fc030da7810) = 17062

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fc030da7810) = 17063

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\321\203 \320\270\320\273"..., 63) = 63

write(4, "1234567890\n", 11) = 11

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\321\203 \320\270\320\273"..., 63) = 63

write(6, "123456789012345678901\n", 21) = 21

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\321\203 \320\270\320\273"..., 63) = 63

write(6, "argentina manit negra\n", 22) = 22

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\321\203 \320\270\320\273"..., 63) = 63

write(6, "mama mila ramu\n", 15) = 15

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\321\203 \320\270\320\273"..., 63) = 63

```
write(4, "abc123deg\n", 10)           = 10
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\321\203 \320\270\320\273"..., 63) = 63
close(4)                            = 0
close(6)                            = 0
wait4(-1, NULL, 0, NULL)            = 17062
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17062, si_uid=1000,
si_status=0, si_utime=0, si_stime=1} ---
wait4(-1, NULL, 0, NULL)            = 17063
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17063, si_uid=1000,
si_status=0, si_utime=0, si_stime=1} ---
exit_group(0)                       = ?
+++ exited with 0 +++
```

Вывод

Я научился создавать пайпы, процессы. Понял, как программа взаимодействует с процессами, научился более грамотно работать с тем, что возвращают функции, чтобы четко контролировать поведение программы. Было не очень очевидно, почему дочерние процессы не завершаются (проблема решалась закрытием дескрипторов на чтение). Все понравилось, пришлось погрузиться в изучение “матчасти”, чтобы выполнить задание.