Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС. Обеспечение обмена данных между процессами посредством каналов.**

Студент: Епанешников Владислав Сергеевич

Группа: М80 – 206Б-19

Вариант: 1

Преподаватель: Миронов Евгений Сергеевич

Дата: 22.04.2021

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

1. **Постановка задачи**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами.Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

1 вариант) Пользователь вводит команды вида: «число числочисло<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int.

1. **Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, ctype.h. В программе используются следующие системные вызовы:

fork - создает копию текущего процесса, который является дочерним процессом для текущего процесса

pipe - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.

fflush - если поток связан с файлом, открытым для записи, то вызов приводит к физической записи содержимого буфера в файл. Если же поток указывает на вводимый файл, то очищается входной буфер.

open - открывает файл, аргументом можно задать, открыть файл на запись, чтение.

close - закрывает файл.

read - читает количество байт(третий аргумент) из файла с файловым дескриптором(первый аргумент) в область памяти(второй агрумент).

write - записывает в файл с файловым дескриптором(первый аргумент) из области памяти(второй аргумент) количество байт(третий аргумент).

perror – вывод сообщения об ошибке.

1. **Общий метод и алгоритм решения**

Для реализации поставленной задачи необходимо:

* Изучить принципы работы fork, pipe, open, close, read, write
* Написать программу, которая будет работать с 2-мя процессами: один родительский и один дочерний, процессы связываются между собой при помощи pipe-ов.
* Организовать работу с выделением памяти под строку неопределенной длины.

1. **Основные файлы программы**

**main.c**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <ctype.h>

#include <stdbool.h>

#define STDIN 0

#define STDOUT 1

#define MIN\_CAP 4

char\* read\_string(int fd) {

size\_t str\_size = 0;

size\_t cap = MIN\_CAP;

char \*str = (char\*) malloc(sizeof(char) \* cap);

if (str == NULL) {

perror("Malloc error");

exit(-1);

}

char c;

while (read(fd, &c, sizeof(char)) == 1) {

if (c == '\n') {

break;

}

str[(str\_size)++] = c;

if (str\_size == cap) {

str = (char\*) realloc(str, sizeof(char) \* cap \* 3 / 2);

cap = cap \* 3 / 2;

if (str == NULL) {

perror("Realloc error");

exit(-2);

}

}

}

str[str\_size] = '\0';

return str;

}

int str\_length(char \*str) {

int length = 0;

for (int i = 0; str[i] != '\0'; ++i) {

++length;

}

return length;

}

char\* int\_to\_string(int sum) {

int size = 0;

int temp\_sum = sum;

while (temp\_sum != 0) {

temp\_sum /= 10;

++size;

}

bool is\_positive = true;

if (sum < 0) {

is\_positive = false;

sum \*= -1;

size++;

}

char\* str = (char\*) malloc(sizeof(char) \* (size + 1));

if (str == NULL) {

perror("Malloc error");

exit(-20);

}

str[size] = '\0';

for (int i = size - 1; i >= 0; --i) {

if (!is\_positive && i == 0) {

str[0] = '-';

} else {

str[i] = sum % 10 + '0';

sum /= 10;

}

}

return str;

}

void child\_work(char str[], char path[]) {

mode\_t mode = S\_IRUSR | S\_IWUSR;

int flags = O\_WRONLY | O\_CREAT | O\_TRUNC;

int file = open(path, flags, mode);

if (file == -1) {

perror("Open error");

exit(7);

}

int sum = 0, number = 0, is\_positive = 1;

for (size\_t i = 0; str[i] != '\0'; ++i) {

if (str[i] == '-') {

is\_positive = -1;

continue;

}

if (isdigit(str[i])) {

number = number \* 10 + str[i] - '0';

}

else if (str[i] == ' ') {

sum += number \* is\_positive;

number = 0;

is\_positive = 1;

}

}

sum += number \* is\_positive;

char\* answer = int\_to\_string(sum);

int answer\_size = str\_length(answer);

printf("Child: ");

if (fflush(stdout) != 0) {

perror("Fflush error");

exit(24);

}

answer[answer\_size] = '\n';

if (write(STDOUT, answer, sizeof(char) \* answer\_size + 1) != answer\_size + 1) {

perror("Write error");

exit(25);

}

if (write(file, answer, sizeof(char) \* answer\_size) != answer\_size) {

perror("Write error");

exit(26);

}

if (close(file) == -1) {

perror("Close error");

exit(27);

}

free(answer);

}

int main() {

int fd[2];

if (pipe(fd) < 0){

perror("Can\'t create pipe");

exit(-3);

}

char \*path = read\_string(STDIN);

int id = fork();

if (id == -1) {

perror("Can\'t fork child");

exit(-4);

} else if (id > 0) {

close(fd[0]);

char\* str = read\_string(STDIN);

size\_t str\_size = str\_length(str);

if (write(fd[1], &str\_size, sizeof(size\_t)) != sizeof(size\_t)) {

perror("Write error");

exit(3);

}

if (write(fd[1], str, sizeof(char) \* str\_size) != str\_size) {

perror("write error");

exit(4);

}

if (close(fd[1]) == -1) {

perror("Close error");

exit(6);

}

int child\_status;

if (wait(&child\_status) == -1) {

perror("wait error");

exit(5);

}

int exit\_code = WEXITSTATUS(child\_status);

if (!exit\_code) {

printf("Parent: The child process exited normally with exit code %d\n", exit\_code);

} else {

printf("Parent: The child process exited abnormally with exit code %d\n", exit\_code);

}

free(str);

printf("Parent exit\n");

}

else {

close(fd[1]);

size\_t str\_size;

if (read(fd[0], &str\_size, sizeof(size\_t)) != sizeof(size\_t)) {

perror("Read error");

exit(7);

}

char\* str = (char\*) malloc(sizeof(char) \* str\_size);

if (str == NULL) {

perror("Malloc error");

exit(8);

}

if (read(fd[0], str, sizeof(char) \* str\_size) != str\_size) {

perror("Read error");

exit(9);

}

if (close(fd[0]) == -1) {

perror("Close error");

exit(10);

}

child\_work(str, path);

free(path);

free(str);

printf("Child exit\n");

}

return 0;

}

1. **Демонстрация работы программы**

MacBook:src vladislove$ cat ../tests/test1

file

1 2 3 4 5 6 7 8 9 10

MacBook:src vladislove$ ./a.out < ../tests/test1

Child: 55

Child exit

Parent: The child process exited normally with exit code 0

Parent exit

MacBook:src vladislove$ cat ../tests/test2

file

-80 40 -20 10 0 1 -1

MacBook:src vladislove$ ./a.out < ../tests/test2

Child: -50

Child exit

Parent: The child process exited normally with exit code 0

Parent exit

MacBook:src vladislove$ cat ../tests/test3

file3

-1

MacBook:src vladislove$ ./a.out < ../tests/test3

Child: -1

Child exit

Parent: The child process exited normally with exit code 0

Parent exit

MacBook:src vladislove$ cat file

-50MacBook:src vladislove$ cat file3

-1MacBook:src vladislove$

1. **Выводы**

Выполнив данную лабораторную, я изучил базовые системные вызовы. Узнал, что существуют специальные системные вызовы (fork) для создания процессов, а также существуют каналы pipe, которые позволяют связать процессы, чтобы можно было обмениваться данными с помощью них. Каналы очень удобны для таких маленьких задач, так как они не требуют сложных ухищрений при их создании. При использовании fork важно помнить, что создаётся копия вашего текущего процесса, иначе неправильная работа может привести к ошибкам.