Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету

"Операционные системы"

№1

Студент: Филиппов А. М.

Преподаватель: Миронов Е.С.

Группа: М8О-201Б-24

Дата:

Оценка:

Подпись:

Москва, 2025

Оглавление

Цель работы

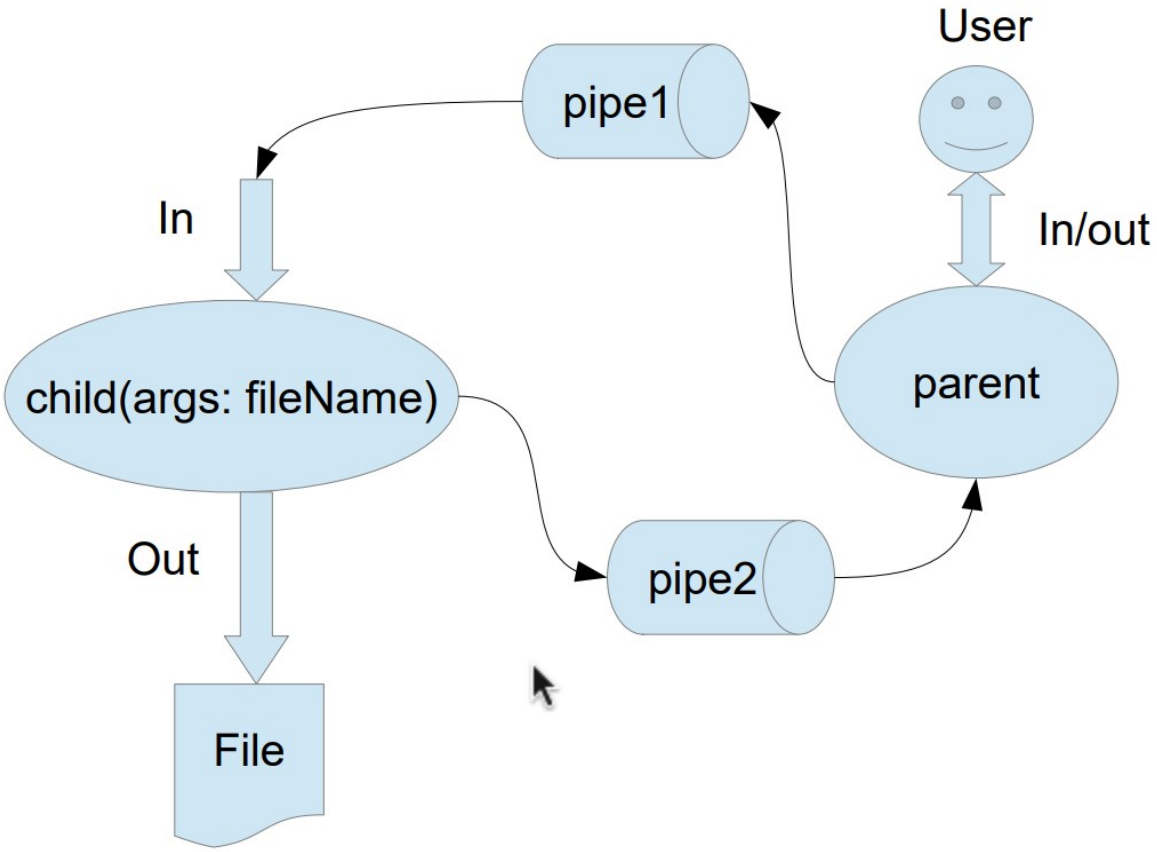
Приобретение практических навыков в:

- Управлении процессами в ОС
- Обеспечении обмена данных между процессами посредством каналов

Постановка задачи

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.



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.

5 вариант) Пользователь вводит команды вида: «число

Общие сведения о программе

Программа представлена двумя файлами – main.c и child1.c, child2

В программе используются следующие системные вызовы:

`pipe()` – создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами(конвейер)

`fork()` – создание дочернего процесса, в переменной id будет лежать «специальный код» процесса(-1 -ошибка, 0- дочерний процесс, >0- родительский)

`read()` – чтение из канала pipe()

`write()` – запись в канал pipe()

`dup2()` – перенаправление дескриптора

`execv()` – создание процесса с другой программой

`close()` – закрытие файлового дескриптора, который после этого не ссылается ни на один и файл и может быть использован повторно.

Общий алгоритм решения

Программа создает дочерний процесс child. Родительский процесс принимает числа от пользователя и отправляет их в pipe1. Child2 проверяет числа и записывает их в файл. Child2 отправляется обратно родительскому процессу флаг 1\0 которые регулирует завершение программы.

Реализация

parent.c

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

int main(void) {
int pipe1[2];
int pipe2[2];

// pipe[0] - read
// pipe[1] - write

char filename[256];
pid_t child_pid;

if (pipe(pipe1) == -1) {
perror("pipe1");
return 1;
}
if (pipe(pipe2) == -1) {
perror("pipe2");
close(pipe1[0]);
close(pipe1[1]);
return 1;
}

printf("Введите имя файла: ");
if (fgets(filename, sizeof(filename), stdin) == NULL) {
perror("Ошибка чтения имени файла");
return 1;
```

```
}
filename[strcspn(filename, "\n")] = 0; // 0 = \0 in ASCII

child_pid = fork(); // вилка хихи

if (child_pid == -1) {
perror("fork");
close(pipe1[0]);
close(pipe1[1]);
close(pipe2[0]);
close(pipe2[1]);
return 1;
}

// for child, preparing to execlp()
if (child_pid == 0) {
close(pipe1[1]);
close(pipe2[0]);

char pipe1_read_fd_str[10];
char pipe1_write_fd_str[10];
char pipe2_read_fd_str[10];
char pipe2_write_fd_str[10];

sprintf(pipe1_read_fd_str, "%d", pipe1[0]);
sprintf(pipe1_write_fd_str, "%d", pipe1[1]);
sprintf(pipe2_read_fd_str, "%d", pipe2[0]);
sprintf(pipe2_write_fd_str, "%d", pipe2[1]);

execlp("./child", "child", pipe1_read_fd_str, pipe1_write_fd_str,
pipe2_read_fd_str, pipe2_write_fd_str, filename, NULL); // 6 args

// if execlp returned controll
perror("execlp failed");
close(pipe1[0]);
close(pipe2[1]);
exit(1);
}

// parent logic
else {
close(pipe1[0]);
close(pipe2[1]);

printf("Введите числа для проверки (число, затем Enter).\n");
printf("Ввод отрицательного или простого числа завершит программу.\n");

int number;
```

```
char number_is_invalid;

while (1) {

printf("Введите число: ");
int ret = scanf("%d", &number);
if (ret == EOF) {
perror("scanf EOF");
break;
} else if (ret < 0) {
perror("scanf < 0");
break;
}

if (write(pipe1[1], &number, sizeof(number)) == -1) {
perror("write to pipe1");
break;
}

// wait for child feedback
if (read(pipe2[0], &number_is_invalid, sizeof(char)) == -1) {
perror("read from pipe2");
break;
}
if (number_is_invalid) {
break;
}
}
}

close(pipe1[1]);
close(pipe2[0]);

// wait for child
int child_status;
pid_t terminated_pid = waitpid(child_pid, &child_status, 0);
if (terminated_pid == -1) {
perror("waitpid");
return 0;
}

printf("Родительский процесс завершен.\n");
return 0;
}
```

child.c

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

int is_prime(int number) {
int divisors_count = 0;

if (number <= 1) {
return 0;
}

else {
for (int i = 2; i * i <= number; i++) {
if (number % i == 0)
divisors_count++;
}
if (divisors_count > 0) {
return 0;
} else {
return 1;
}
}
}

int main(int argc, char *argv[]) {
if (argc != 5) {
fprintf(stderr, "Wrong arg count in child");
return 1;
}

int pipe1_read_fd = atoi(argv[1]);
int pipe1_write_fd = atoi(argv[2]);
int pipe2_read_fd = atoi(argv[3]);
int pipe2_write_fd = atoi(argv[4]);
const char *filename = argv[5];

close(pipe1_write_fd);
close(pipe2_read_fd);

dup2(pipe1_read_fd, 0);

close(pipe1_read_fd);

FILE *output_file = fopen(filename, "w");
if (output_file == NULL) {
perror("child: fopen");
return 1;
}
```

```
int number = 0;

char flag_yes = 1;
char flag_no = 0;

// child logic cycle
while (1) {
if (read(0, &number, sizeof(number)) == -1) {
perror("child: pipe1 read");
return 1;
}

if (number < 0) {
printf("Число отрицательное, выход...\n");
if (write(pipe2_write_fd, &flag_yes, sizeof(char)) == -1) {
perror("child: pipe2 write");
}
break;
} else if (is_prime(number)) {
printf("Число простое, выход...\n");
if (write(pipe2_write_fd, &flag_yes, sizeof(char)) == -1) {
perror("child: pipe2 write");
}
break;
} else {
if (write(pipe2_write_fd, &flag_no, sizeof(char)) == -1) {
perror("child: pipe2 write");
}
if (fprintf(output_file, "%d\n", number) < 0) {
perror("child: fprintf");
break;
}
}
}

fclose(output_file);
close(pipe1_read_fd);
close(pipe2_write_fd);

printf("Дочерний процесс завершен.\n");
return 0;
}
```

Пример работы

```
…/lab_01/build master □ ? ❯ ./lab_01_osi_app
Введите имя файла: out.txt
Введите числа для проверки (число, затем Enter).
Ввод отрицательного или простого числа завершит программу.
Введите число: 14
Введите число: 66
Введите число: 55
Введите число: 13
Число простое, выход...
Дочерний процесс завершен.
Родительский процесс завершен.

…/lab_01/build master □ ? ❯ cat out.txt
14
66
55
```

Вывод

Существуют специальные системные вызовы(fork) для создания процессов, также существуют специальные каналы pipe, которые позволяют связать процессы и обмениваться данными при помощи этих pipe-ов. При использовании fork важно помнить, что фактически создается копию вашего текущего процесса и неправильная работа может привести к неожиданным результатам и последствиям, однако создание процессов очень удобно, когда вам нужно выполнять несколько действий параллельно. Также у каждого процесса есть свой id, по которому его можно определить. Также важно работать с чтением и записью из канала, помня что read, write возвращает количество успешно считанных/записанных байт и оно не обязательно равно тому значению, которое вы указали. Также важно не забывать закрывать pipe после завершения работы.