Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу

«Операционные системы»

ВЗАИМОДЕЙСТВИЕ МЕЖДУ ПРОЦЕССАМИ

Студент: Забелкин Андрей Алексеевич
Группа: М8О–206Б–20
Вариант: 6
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023.

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

- Организации взаимодействий процессов
- Создание программ, создающих во время исполнения новые процессы

Задание

Требуется написать программу, которая исполняет определенную задачу в дочернем процессе и передает результат в родительский.

Родительский процесс создает дочерний процесс. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В конечном итоге, программа должна состоять из следующих частей:

- Программа, отвечающая за родительский процесс;
- Программа, отвечающая за дочерний процесс

Общие сведения о программе

Программа компилируется из с помощью Makefile, сгенерированным cmake.

Также используется заголовочные файлы: stdio.h, stdlib.h, unistd.h, sys/types.h, sys/wait.h, fcntl.h . В программе используются следующие системные вызовы:

1. **pipe()** - эта команда создает анонимный канал для обмена данными между родительским и дочерним процессами. Она создает два файловых дескриптора pipefd[0] (для чтения из канала) и pipefd[1] (для записи в канал).
2. **fork()** - этот вызов создает новый дочерний процесс, который является копией родительского процесса.
3. **execl()** - этот системный вызов заменяет текущий процесс новым процессом, который указан в аргументах.
4. **dup2()** - этот вызов создает копию одного файлового дескриптора и связывает ее с другим файловым дескриптором.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pipe(), fork(), execl(), dup2().
2. Написать программу для дочернего процесса child_process.c.
3. Скомпилировать программы child_process.c и lab1.c
4. Придумать тесты и ответы к этим тестам.
5. Написать bash-скрипт, который запускает и проверяет программу на тестах.

Основные файлы программы

child_process.c:

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

int main() {

   int sum = 0;
   char buffer[1024];
   int num = 0;
   int num_started = 0; // Флаг, указывающий, что число началось
   int is_negative = 0; //Флаг для отрицательных чисел
   ssize_t bytes_read;

   // Считываем данные из файла в виде строк
   while ((bytes_read = read(STDIN_FILENO, buffer, sizeof(buffer))) > 0) {
       for (ssize_t i = 0; i < bytes_read; i++) {
           char c = buffer[i];
           if (c == '-') {
               // Если обнаружен символ '-', устанавливаем флаг отрицательного числа
               is_negative = 1;
           } else if (c >= '0' && c <= '9') {
               num = num * 10 + (c - '0');
               num_started = 1;
           } else if (c == ' ' || c == '\n') {
               // Если обнаружен пробел или символ новой строки, значит число
закончилось
               if (num_started) {
                   // Если число отрицательное, вычитаем его модуль из суммы
                   if (is_negative) {
                       sum -= num;
                       is_negative = 0;
                   } else {
                       // Иначе прибавляем число к сумме
                       sum += num;
                   }
                   num = 0;
                   num_started = 0;
               }
           }
       }
   }
   char sum_chars[32];
   int len = 0;
   int temp_sum = sum;

   if (sum == 0) {
```

```
        sum_chars[0] = '0';
        write(STDOUT_FILENO, sum_chars, sizeof(char));
        return 0;
    }

    if (num_started) {
        if (is_negative) {
            sum -= num;
        } else {
            sum += num;
        }
    }

    // Вычисляем длину суммы
    if (temp_sum < 0) {//число отрицательное
        char minus = '-';
        write(STDOUT_FILENO, &minus, sizeof(char));
        temp_sum = sum * -1;
        sum *= -1;
    }
    while (temp_sum > 0) {
        len++;
        temp_sum /= 10;
    }
    if (sum == 0) {
        sum_chars[0] = '0';
        write(STDOUT_FILENO, sum_chars, len*sizeof(char));
        return 0;
    }
    // Преобразуем сумму в символы, начиная с конца массива
    for (int i = len - 1; i >= 0; i--) {
        sum_chars[i] = '0' + (sum % 10);
        sum /= 10;
    }

    write(STDOUT_FILENO, sum_chars, len*sizeof(char));

    return 0;
}
```

lab1.c

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

int main(int argc, char *argv[]) {
    int pipefd[2];
    int sum;
    // Создаем канал для обмена данными между процессами
    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    pid_t pid;
    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
```

```
    }
    if (pid == 0) {
        // Дочерний процесс
        pid = getpid();
        // Закрываем конец канала, который не используется дочерним процессом
        close(pipefd[0]);
        // Открываем файл для чтения
        int file_fd = open(argv[1], O_RDONLY);
        //где то после этого портятся данные
        if (file_fd == -1) {
            perror("open");
            close(pipefd[1]); // Закрываем запись в канал
            exit(EXIT_FAILURE);
        }
        // Перенаправляем стандартный вывод в канал
        dup2(file_fd, STDIN_FILENO);
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[1]);
        execl("../src/build/child", "child", NULL);
        // Этот код выполняется только в случае неудачи запуска execl
        perror("execl");
        exit(EXIT_FAILURE);
    } else {
        // Родительский процесс
        pid = getpid();
        close(pipefd[1]);
        // Ждем завершения дочернего процесса
        wait(NULL);
        // Читаем число из канала
        char buffer; // Буфер для чтения данных
        ssize_t bytes_read;
        while ((bytes_read = read(pipefd[0], &buffer, sizeof(buffer))) > 0) {
            write(STDOUT_FILENO, &buffer, bytes_read);
        }
        char endline = '\n';
        write(STDOUT_FILENO, &endline, 1);
        close(pipefd[0]);
        exit(EXIT_SUCCESS);
    }

    return 0;
}
```

Пример работы

Запуск программы для файла test1.txt:

-4

Пройдено

Запуск программы для файла test2.txt:

60

Пройдено

Запуск программы для файла test3.txt:

297

Пройдено

Запуск программы для файла test4.txt:

0

Пройдено

Вывод

Для выполнения это лабораторной работы, необходимо было изучить работу команд pipe(), fork(), dup2(), execl(). Также важно было разобраться в потоках ввода и вывода данных, узнать об основных дескрипторах. Научиться пользоваться вводом через команду read() и выводом через команду write(), а также грамотно обрабатывать полученные данные.