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»

Выполнил: Д. А. Корнеева
Группа: М8О-208БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы: приобретение практических навыков в управлении процессами в ОС и обеспечении обмена данных между процессами посредством каналов

Задание: 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) должна создать для решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы

Вариант (9): в файле записаны команды вида:«число число число». Дочерний процесс производит деление первого числа команды, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

Метод решения

Программа представляет собой архитектуру с использованием межпроцессного взаимодействия через pipe. Состоит из двух компонентов: родительского и дочернего процесса.

Родительский процесс:
1) Получает имя файла с командами от пользователя
2) Создаёт pipe для межпроцессной коммуникации
3) Создаёт дочерний процесс с помощью fork()
4) Перенаправляет ввод/вывод через dup2()
5) Читает результаты вычислений из pipe
6) Ожидает завершения дочернего процесса с waitpid()

Дочерний процесс:
1) Читает строки из stdin (перенаправленного файла)
2) Парсит числа из каждой строки с помощью strtok()
3) Выполняет последовательное деление чисел
4) Обрабатывает ошибки (деление на ноль, недостаточное количество введённых чисел)
5) Выводит результаты в stdout (перенаправленный в pipe)

Родительский процесс управляет вводом/выводом, дочерний - вычислениями. Межпроцессная коммуникация через pipe. Проверка возвращаемых значений системных вызовов. Использование dup2() для перенаправления stdin/stdout.

Описание программы

Структура файлов проекта
1) general.h
Заголовочные файлы (stdio.h, stdlib.h, string.h, unistd.h, sys/wait.h, fcntl.h)
Макрос CHECK_ERROR для обработки ошибок

2) child.c (вычислитель)
Логика математических вычислений (последовательное деление)
Парсинг чисел из входных строк
Обработка ошибок (деление на ноль, недостаточно чисел)
Чтение из stdin, вывод в stdout

3) parent.c
Управление дочерним процессом
Работа с pipe для межпроцессного взаимодействия
Перенаправление потоков ввода/вывода
Чтение имени файла и запуск всей системы
Файл с командами (пользовательский)
Строки с числами для обработки

4) Исполняемые файлы (после компиляции)
parent - главная программа
child - дочерний процесс для вычислений

Общая схема: parent → [pipe] → child → вычисления → [pipe] → parent → вывод

Типы данных:
float numbers[100] - массив чисел для вычислений
char buffer[1024] - буферы для строковых операций
int pipefd[2] - дескрипторы pipe
pid_t child_pid - идентификатор процесса
int error - флаг ошибки в дочернем процессе
int child_status - статус завершения дочернего процесса

Управление процессами:
fork() - создание дочернего процесса
waitpid() - ожидание завершения дочернего процесса
exit() - завершение процесса
execl() - запуск исполняемого файла

Межпроцессное взаимодействие:
pipe() - создание канала связи
dup2() - перенаправление стандартных потоков

Файловая система:
open() - открытие файла
read() - чтение из файла/pipe
close() - закрытие дескриптора

Ввод-вывод:
fgets() - чтение строки из stdin
printf() - форматированный вывод
fflush() - сброс буфера вывода

Основные функции
child.c:
Парсинг строк: strtok(), atof()
Вычисления: последовательное деление чисел
Обработка ошибок: проверка деления на ноль

parent.c:
Управление процессами: создание и ожидание дочернего процесса
Перенаправление: настройка потоков ввода/вывода
Чтение результатов через pipe

Результаты
Программа реализует распределенные вычисления через механизм Unix-процессов. Родительский процесс управляет вводом/выводом, а дочерний - выполняет математические операции.

Выводы
Успешно применен системный вызов fork() для создания дочерних процессов
Освоено использование waitpid() для контроля выполнения процессов-потомков
Реализована корректная обработка статусов завершения процессов
Реализовано перенаправление стандартных потоков через dup2()
Освоены ключевые системные вызовы Unix: pipe(), fork(), dup2(), waitpid()
Усвоены принципы работы с файловыми дескрипторами

Исходная программа

C child.c

```c
#include "general.h"

int main() {
  float numbers[100];
  char buffer[1024];
  int count_num;
  float result;
  int error = 0;

  while (fgets(buffer, sizeof(buffer), stdin) != NULL && !error) {
    buffer[strcspn(buffer, "\n")] = 0;

    count_num = 0;
    char *token = strtok(buffer, " ");

    while (token != NULL && count_num < 100) {
      numbers[count_num] = atof(token);
      count_num++;
      token = strtok(NULL, " ");
    }

    if (count_num < 2) {
      printf("Недостаточно чисел в строке\n");
      fflush(stdout);
      continue;
    }

    result = numbers[0];

    for (int i = 1; i < count_num; i++) {
      if (numbers[i] == 0.0f) {
        printf("Ошибка: деление на ноль\n");
        fflush(stdout);
        error = 1;
        break;
      }
      result /= numbers[i];
    }

    if (!error) {
      printf("%.2f\n", result);
      fflush(stdout);
    }
  }
  return error ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

Листинг 1: чтение чисел из stdin и выполнение их последовательного деления с проверкой ошибок.

C general.h

```c
#ifndef GENERAL_H
#define GENERAL_H

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

#define CHECK_ERROR(condition, message)                                 \
  if (condition) {                                                      \
    perror(message);                                                    \
    exit(EXIT_FAILURE);                                                 \
  }

#endif
```

Листинг 2: заголовочный файл с общими библиотеками и макросом для обработки ошибок

M Makefile

```
CC = gcc
CFLAGS = -std=c11 -Wall -Wextra -Werror

all: parent child

parent: parent.c general.h
    $(CC) $(CFLAGS) -o parent parent.c

child: child.c general.h
    $(CC) $(CFLAGS) -o child child.c

clean:
    rm -f parent child

test: all
    ./parent

.PHONY: all clean test
```

Листинг 3: сборка программ (parent и child) с флагами компиляции и целями для очистки и тестирования

C parent.c

```c
#include "general.h"

int main() {
    char name_file[100];
    int pipefd[2];
    pid_t child_pid;
    int file_fd;
    char buffer[1024];
    int child_status = 0;

    printf("Введите имя файла с командами (родительский процесс): ");
    scanf("%s", name_file);

    file_fd = open(name_file, O_RDONLY);
    CHECK_ERROR(file_fd == -1, "Ошибка открытия файла")

    CHECK_ERROR(pipe(pipefd) == -1, "Ошибка создания pipe")

    child_pid = fork();
    CHECK_ERROR(child_pid == -1, "Ошибка создания процесса")

    if (child_pid == 0) {
        close(pipefd[0]);

        CHECK_ERROR(dup2(file_fd, STDIN_FILENO) == -1,
                    "Ошибка перенаправления ввода")

        CHECK_ERROR(dup2(pipefd[1], STDOUT_FILENO) == -1,
                    "Ошибка перенаправления вывода")

        close(file_fd);
        close(pipefd[1]);

        execl("./child", "child", (char *)NULL);

        perror("Ошибка запуска дочерней программы");
        exit(EXIT_FAILURE);

    } else {
        printf("Родительский процесс создал дочерний процесс\n");

        close(file_fd);
        close(pipefd[1]);

```

```
45      printf("Результаты вычислений: \n");
46      ssize_t bytes_read;
47      while ((bytes_read = read(pipefd[0], buffer, sizeof(buffer) - 1)) > 0) {
48        buffer[bytes_read] = '\0';
49        printf("%s", buffer);
50        fflush(stdout);
51      }
52
53      int status;
54      waitpid(child_pid, &status, 0);
55
56      if (WIFEXITED(status)) {
57        child_status = WEXITSTATUS(status);
58        if (child_status == 0) {
59          printf("Дочерний процесс завершился успешно (родительский процесс)\n");
60        } else {
61          printf("Дочерний процесс завершился с кодом ошибки %d (родительский "
62                 "процесс)\n",
63                 child_status);
64          close(pipefd[0]);
65          exit(EXIT_FAILURE);
66        }
67      }
68      close(pipefd[0]);
69      printf("Работа родительского процесса завершена\n");
70    }
71    return 0;
72  }
```

Листинг 4: родительский процесс создаёт дочерний процесс, перенаправляет ему ввод/вывод через pipe и читает результаты вычислений