Лабораторная работа №1 по курсу «Операционные системы»

**\*\*Московский Авиационный Институт\*\***

(Национальный Исследовательский Университет)

**Факультет:** Информационных технологий и прикладной математики

**Кафедра:** Вычислительной математики и программирования

**Студент:** Подгурский Александр Владимирович

**Группа:** М8О-203Б-24

**Вариант:** 8

**Преподаватель:** Соколов Андрей Алексеевич

**Оценка:** \_\_\_\_\_\_\_\_\_\_

**Дата:** \_\_\_\_\_\_\_\_\_\_

**Подпись:** \_\_\_\_\_\_\_\_\_\_

Москва, 2025

Постановка задачи

## Цель работы:

Приобретение практических навыков в:

- Управление процессами в ОС

- Обеспечение обмена данных между процессами посредством каналов

## Задание:

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними. Основной процесс должен создавать 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.

**Спецификация варианта 8:**

- Родительский процесс создает один дочерний процесс

- Родительский процесс читает команды из файла и передает их дочернему процессу через pipe

- Дочерний процесс выполняет арифметические операции деления над числами из команд

- При обнаружении деления на ноль дочерний процесс отправляет сигнал SIGUSR1 родительскому процессу и завершается

- Родительский процесс обрабатывает сигнал и корректно завершает работу

Общие сведения о программе

## Состав проекта и компоновка:

- **Исполняемые файлы:** `main` (родительский процесс), `child` (дочерний процесс)

- **Исходные файлы:**

- `main.c` - родительский процесс

- `child.c` - дочерний процесс

- `child.h` - заголовочный файл с объявлениями функций дочернего процесса

## Используемые системные вызовы и функции:

**В родительском процессе:**

- `fork()` - создание дочернего процесса

- `pipe()` - создание канала для межпроцессного взаимодействия

- `dup2()` - перенаправление стандартных потоков ввода-вывода

- `signal()` - установка обработчика сигналов

- `waitpid()` - ожидание завершения дочернего процесса

- `kill()` - отправка сигналов

**В дочернем процессе:**

- `getline()` - чтение строк из стандартного ввода

- `strtol()` - преобразование строк в числа

- `kill()` - отправка сигнала родительскому процессу

Архитектура программы

## Родительский процесс (main.c):

1. **Инициализация:** проверка аргументов командной строки, открытие файла с командами

2. **Создание канала:** инициализация pipe для связи с дочерним процессом

3. **Обработка сигналов:** установка обработчика для SIGUSR1

4. **Создание процесса:** вызов fork() для создания дочернего процесса

5. **Перенаправление потоков:** настройка stdin дочернего процесса на чтение из pipe

6. **Передача команд:** чтение команд из файла и запись в pipe

7. **Обработка ошибок:** реагирование на сигналы от дочернего процесса

8. **Завершение:** корректное закрытие ресурсов и ожидание дочернего процесса

## Дочерний процесс (child.c):

1. **Перенаправление ввода:** настройка stdin на чтение из pipe

2. **Чтение команд:** построчное чтение и парсинг чисел

3. **Выполнение операций:** последовательное деление чисел

4. **Проверка ошибок:** обнаружение деления на ноль

5. **Уведомление:** отправка сигнала родителю при ошибке

6. **Завершение:** освобождение ресурсов и выход

Ключевые функции

## Родительский процесс:

1. void handle\_child\_signal(int sig); // Обработчик сигнала SIGUSR1

2. void main\_handle\_error(const char \*msg); // Обработка ошибок

**Дочерний процесс:**

1. void child\_handle\_error(const char \*msg); *// Обработка ошибок*
2. void notify\_parent(void); *// Отправка сигнала родителю*
3. int parse\_numbers(char \*line, int \*\*numbers); *// Парсинг чисел из строки*
4. void run\_child\_process(void); *// Основная логика дочернего процесса*
5. char \*safe\_strdup(const char \*str); *// Безопасное копирование строк*

**Алгоритм работы**

1. **Запуск программы:** родительский процесс получает имя файла с командами как аргумент
2. **Инициализация канала:** создается pipe для связи процессов
3. **Создание дочернего процесса:** вызов fork() и настройка потоков
4. **Передача данных:** родитель читает команды из файла и пишет в pipe
5. **Обработка команд:** дочерний процесс читает из pipe, парсит числа и выполняет деление
6. **Обработка ошибок:** при делении на ноль дочерний процесс уведомляет родителя и завершается
7. **Корректное завершение:** родительский процесс закрывает каналы и ожидает дочерний процесс

**Особенности реализации**

**Обработка ошибок:**

* Проверка возвращаемых значений всех системных вызовов
* Обработка ошибок выделения памяти
* Валидация входных данных и чисел
* Корректная обработка сигналов

**Безопасность:**

* Использование strtol() вместо atoi() для безопасного парсинга чисел
* Проверка границ чисел (INT\_MAX, INT\_MIN)
* Проверка деления на ноль
* Безопасное освобождение ресурсов

**Межпроцессное взаимодействие:**

* Использование pipe для передачи данных
* Сигналы для асинхронного уведомления об ошибках
* Корректное закрытие файловых дескрипторов

**Пример работы программы**

**Входной файл commands.txt:**

text

100 2 5

50 5 2

30 0 5

40 2 2

**Вывод программы:**

text

Родительский процесс PID: 1234

Дочерний процесс PID: 1235

Чтение команд из файла: commands.txt

Родитель: Отправка команды 1: 100 2 5

Дочерний: Получена команда 1: 100 2 5

Дочерний: Делимое: 100

Дочерний: Делитель 1: 2

Дочерний: Промежуточный результат: 50

Дочерний: Делитель 2: 5

Дочерний: Промежуточный результат: 10

Дочерний: ИТОГОВЫЙ РЕЗУЛЬТАТ: 10

Родитель: Отправка команды 2: 50 5 2

Дочерний: Получена команда 2: 50 5 2

...

Родитель: Отправка команды 3: 30 0 5

Дочерний: Получена команда 3: 30 0 5

Дочерний: Делимое: 30

Дочерний: Делитель 1: 0

Дочерний: ОШИБКА: деление на ноль!

Дочерний: Отправка сигнала родителю и завершение

Родитель: Получен сигнал о ошибке в дочернем процессе. Завершение работы.

Родитель: Прекращаем отправку команд из-за ошибки

Родитель: Дочерний процесс завершился с кодом: 1

Родитель: Завершение работы

**Вывод**

В ходе лабораторной работы были успешно реализованы:

1. Создание и управление процессами с использованием fork()
2. Межпроцессное взаимодействие через анонимные каналы (pipe)
3. Обработка системных сигналов для асинхронного уведомления
4. Корректная обработка ошибок системных вызовов
5. Безопасный парсинг и валидация входных данных

Программа демонстрирует эффективное взаимодействие между процессами, обработку исключительных ситуаций и корректное освобождение ресурсов. Все требования задания выполнены в полном объеме.