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: Ю. М. Омаров
Группа: М8О-208БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

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 при необходимости 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.

Цель работы

Изучение принципов создания дочерних процессов и организации межпроцессного взаимодействия с использованием механизма pipes в операционной системе.

Задание

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. 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.

Вариант

4

Метод решения

Программа реализует многопроцессную архитектуру для выполнения последовательных делений чисел с обработкой ошибок. Основной алгоритм:

Архитектура программы

- **Родительский процесс** (division_parent.cpp): отвечает за взаимодействие с пользователем, ввод данных и управление дочерним процессом
- **Дочерний процесс** (division_child.cpp): выполняет математические операции и записывает результаты в файл
- **Общая библиотека** (division_os.h/cpp): содержит вспомогательные функции для работы с процессами и pipes

Алгоритм работы

1. Родительский процесс создает два pipe-канала для двусторонней связи

2. Создается дочерний процесс с перенаправлением стандартных потоков ввода/вывода
3. Пользователь вводит имя файла для результатов и последовательности чисел
4. Данные передаются через pipe в дочерний процесс
5. Дочерний процесс выполняет деление и проверяет на нулевой делитель
6. При обнаружении деления на ноль отправляется сигнал для завершения обоих процессов
7. Результаты операций записываются в указанный файл

Обработка ошибок

Программа использует механизм сигналов (SIGUSR1) для уведомления родительского процесса об ошибке деления на ноль. Это обеспечивает корректное завершение обоих процессов при возникновении критической ошибки.

Использованные источники

- Stevens W.R., Rago S.A. - Advanced Programming in the UNIX Environment
- Документация Linux man pages: pipe(2), fork(2), dup2(2), signal(7)

Описание программы

Структура файлов

- **division_os.h** - заголовочный файл с объявлениями функций и типов
- **division_os.cpp** - реализация системных функций-оберток
- **division_parent.cpp** - основной родительский процесс
- **division_child.cpp** - дочерний процесс для вычислений

Основные типы данных

- `ProcessRole` - перечисление для идентификации роли процесса
- `volatile sig_atomic_t` - атомарный тип для обработки сигналов

Ключевые функции

division_os.h/cpp

- `ProcessCreate()` - создание дочернего процесса с помощью `fork()`
- `pipeCreate()` - создание pipe с обработкой ошибок

- `pipeRead()/pipeWrite()` - чтение/запись в pipe
- `redirectFd()` - перенаправление файловых дескрипторов с помощью `dup2()`

division_parent.cpp

- `handle_signal()` - обработчик сигнала деления на ноль
- Основная логика взаимодействия с пользователем и управления дочерним процессом

division_child.cpp

- `handle_division_signal()` - обработчик сигнала
- Основная логика вычислений и записи в файл

Используемые системные вызовы

- **fork()** - создание нового процесса
- **pipe()** - создание канала межпроцессного взаимодействия
- **dup2()** - перенаправление файловых дескрипторов
- **read()/write()** - операции ввода/вывода
- **close()** - закрытие файловых дескрипторов
- **signal()** - установка обработчиков сигналов
- **execl()** - запуск исполняемого файла
- **wait()** - ожидание завершения дочернего процесса

Протокол взаимодействия

1. Родитель передает имя файла в дочерний процесс
2. Для каждой операции: родитель отправляет количество чисел и массив значений
3. Дочерний процесс выполняет вычисления и отправляет статус операции
4. При ошибке деления на ноль процессы завершаются

Результаты

В результате работы была разработана многопроцессная система для выполнения последовательных операций деления с обработкой ошибок. Программа успешно решает поставленную задачу и обладает следующими ключевыми особенностями:

Основные достижения

- Реализовано корректное межпроцессное взаимодействие через два pipe-канала
- Обеспечена двусторонняя связь между родительским и дочерним процессами
- Реализована надежная обработка ошибки деления на ноль с использованием механизма сигналов
- Организован корректный обмен данными произвольного количества чисел типа float

Особенности реализации

- **Модульная архитектура** - четкое разделение на родительский и дочерний процессы, представленные разными исполняемыми файлами
- **Надежная обработка ошибок** - при обнаружении деления на ноль оба процесса корректно завершают работу
- **Гибкий ввод данных** - поддержка произвольного количества чисел в одной операции

Протокол взаимодействия

Программа демонстрирует эффективный протокол обмена данными:

1. Передача имени файла для результатов
2. Последовательная отправка наборов чисел с указанием их количества
3. Получение подтверждения о успешном выполнении операции или ошибке
4. Корректное завершение при получении команды или критической ошибки

Обработка исключительных ситуаций

- Контроль деления на ноль на стороне дочернего процесса
- Проверка минимального количества чисел (не менее 2)
- Обработка ошибок открытия файла и работы с pipe-каналами
- Корректное освобождение ресурсов при завершении работы

Вывод

В ходе выполнения лабораторной работы были успешно освоены ключевые аспекты многопроцессного программирования и межпроцессного взаимодействия в операционных системах.

Исходная программа

Заголовочный файл division_os.h

```
#pragma once

#include <cstddef>

enum ProcessRole {
    IS_PARENT,
    IS_CHILD
};


ProcessRole ProcessCreate();
int pipeCreate(int fd[2]);
int pipeRead(int fd, void * buf, size_t count);
int pipeWrite(int fd, const void * buf, size_t count);
void pipeClose(int fd);
int redirectFd(int old_fd, int new_fd);
```

Листинг 1: Объявления функций для работы с процессами и pipe-каналами

Реализация системных функций division_os.cpp

```
#include "division_os.h"

#include <iostream>
#include <unistd.h>
#include <cstdlib>
#include <cstring>

ProcessRole ProcessCreate() {
    pid_t pid = fork();
    if (pid == -1) {
        std::cerr << "Fault of creating process" << std::endl;
        exit(-1);
    }

    if (pid > 0) {
        return IS_PARENT;
    } else {
        return IS_CHILD;
    }
}

int pipeCreate(int fd[2]) {
    int err = pipe(fd);
    if (err == -1) {
        std::cerr << "Fault of creating pipe" << std::endl;
        exit(-1);
    }
    return err;
}

```

```
int pipeRead(int fd, void * buf, size_t count) {
    int bytes = read(fd, buf, count);
    if (bytes == -1) {
        std::cerr << "Fault of reading from pipe" << std::endl;
        exit(-1);
    }
    return bytes;
}

int pipeWrite(int fd, const void * buf, size_t count) {
    int bytes = write(fd, buf, count);
    if (bytes == -1) {
        std::cerr << "Fault of writing in pipe" << std::endl;
        exit(-1);
    }
    return bytes;
}

void pipeClose(int fd) {
    close(fd);
}

int redirectFd(int old_fd, int new_fd) {
    int err = dup2(old_fd, new_fd);
    if (err == -1) {
        std::cerr << "Handle redirection error" << std::endl;
        exit(-1);
    }
    return err;
}
```

Листинг 2: Реализация функций-оберток для системных вызовов

Родительский процесс division_parent.cpp

```
#include "division_os.h"

#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <unistd.h>
#include <string>
#include <vector>
#include <sstream>
#include <signal.h>
#include <sys/wait.h>

volatile sig_atomic_t division_by_zero = 0;

void handle_signal(int sig) {
    division_by_zero = 1;
}

int main() {
    int pipe_to_child[2];
    int pipe_from_child[2];
```

```
22      pipeCreate(pipe_to_child);
23      pipeCreate(pipe_from_child);
24
25      ProcessRole role = ProcessCreate();
26
27      if (role == IS_CHILD) {
28          pipeClose(pipe_to_child[1]);
29          redirectFd(pipe_to_child[0], 0);
30          pipeClose(pipe_to_child[0]);
31
32          pipeClose(pipe_from_child[0]);
33          redirectFd(pipe_from_child[1], 1);
34          pipeClose(pipe_from_child[1]);
35
36          execl("./division_child", "child", NULL);
37          std::cerr << "Fault execl" << std::endl;
38          exit(-1);
39      }
40
41      if (role == IS_PARENT) {
42          pipeClose(pipe_to_child[0]);
43          pipeClose(pipe_from_child[1]);
44
45          signal(SIGUSR1, handle_signal);
46
47          std::string file_name;
48          std::cout << "Type file's name: ";
49          std::cin >> file_name;
50
51          std::string data = file_name + "\n";
52          pipeWrite(pipe_to_child[1], data.c_str(), data.size());
53
54          std::cout << "Write numbers thought space(first divides by others):" << std::
                endl;
55          std::cout << "For exit write 'q'" << std::endl;
56
57          std::string input;
58          std::cin.ignore();
59
60          int signal_from_child;
61
62          while (!division_by_zero) {
63              std::cout << "Write numbers: ";
64              std::getline(std::cin, input);
65
66              if (input == "q" || input == "quit") {
67                  break;
68              }
69
70              if (input.empty()) {
71                  continue;
72              }
73
74              std::vector<float> numbers;
75              std::stringstream ss(input);
76              float num;
77
78              while (ss >> num) {
```

```
79                  numbers.push_back(num);
80              }
81
82              if (numbers.size() > 0) {
83                  int count = numbers.size();
84                  pipeWrite(pipe_to_child[1], &count, sizeof(count));
85
86                  pipeWrite(pipe_to_child[1], numbers.data(), count * sizeof(float));
87
88                  pipeRead(pipe_from_child[0], &signal_from_child, sizeof(
                        signal_from_child));
89
90                  if (signal_from_child == 1) {
91                      std::cout << "Divide by zero. Program is closing" << std::endl;
92                      division_by_zero = true;
93                  }
94              } else {
95                  std::cout << "Error: false format" << std::endl;
96              }
97          }
98
99          if (!division_by_zero) {
100             int count = 0;
101             pipeWrite(pipe_to_child[1], &count, sizeof(count));
102         }
103
104         pipeClose(pipe_to_child[1]);
105         pipeClose(pipe_from_child[0]);
106
107         wait(NULL);
108         std::cout << "Program finished" << std::endl;
109     }
110
111     return 0;
112 }
```

Листинг 3: Основной процесс

Дочерний процесс division_child.cpp

```
1  #include "division_os.h"
2
3  #include <iostream>
4  #include <fstream>
5  #include <string>
6  #include <vector>
7  #include <unistd.h>
8  #include <signal.h>
9
10 volatile sig_atomic_t division_by_zero = 0;
11
12 void handle_division_signal(int sig) {
13     division_by_zero = 1;
14 }
15
16 int main() {
```

```
17      signal(SIGUSR1, handle_division_signal);
18
19      std::string filename;
20      std::cin >> filename;
21
22      std::ofstream out(filename, std::ios::app);
23      if (!out.is_open()) {
24          std::cerr << "Fault of opening file" << std::endl;
25          return 1;
26      }
27
28      int count;
29      int signal_to_parent;
30
31      while (pipeRead(0, &count, sizeof(count))) {
32          if (count == 0) break;
33
34          std::vector<float> numbers(count);
35          pipeRead(0, numbers.data(), count * sizeof(float));
36
37          if (count < 2) {
38              out << "Fault: program is needed minimum 2 numbers" << std::endl;
39              signal_to_parent = 0;
40          } else {
41
42              float result = numbers[0];
43              bool error_occurred = false;
44
45              for (int i = 1; i < count; i++) {
46                  if (numbers[i] == 0.0f) {
47                      out << "Fault: division by zero" << std::endl;
48                      error_occurred = true;
49                      signal_to_parent = 1;
50                      break;
51                  }
52                  result /= numbers[i];
53              }
54
55              if (!error_occurred) {
56                  out << numbers[0];
57                  for (int i = 1; i < count; i++) {
58                      out << " / " << numbers[i];
59                  }
60                  out << " = " << result << std::endl;
61                  signal_to_parent = 0;
62              }
63          }
64
65          out.flush();
66
67          pipeWrite(1, &signal_to_parent, sizeof(signal_to_parent));
68
69          if (signal_to_parent == 1) {
70              break;
71          }
72      }
73
74      out.close();
```

```
75 ||     return 0;
76 || }
```

Листинг 4: Процесс выполнения математических операций и записи результатов

Системные вызовы программы

Системные вызовы

```
22:16:45.809239 execve("./main",["./main"],0x7ffe80dd98c8 /* 65 vars */) =
0
22:16:45.810684 brk(NULL)               = 0x593b26cb4000
22:16:45.810915 mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0)
= 0x70a91c6a9000
22:16:45.811138 access("/etc/ld.so.preload",R_OK) = -1 ENOENT (No such file
or directory)
22:16:45.811317 openat(AT_FDCWD,"/etc/ld.so.cache",O_RDONLY|O_CLOEXEC) = 3
22:16:45.811362 fstat(3,{st_mode=S_IFREG|0644,st_size=56283,...}) = 0
22:16:45.811390 mmap(NULL,56283,PROT_READ,MAP_PRIVATE,3,0) = 0x70a91c69b000
22:16:45.811546 close(3)                = 0
22:16:45.811574 openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libstdc++.so.6",O_RDONLY|O_CLO
= 3
22:16:45.811602 read(3,"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0
= 832
22:16:45.811624 fstat(3,{st_mode=S_IFREG|0644,st_size=2592224,...}) = 0
22:16:45.811644 mmap(NULL,2609472,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0)
= 0x70a91c400000
22:16:45.811667 mmap(0x70a91c49d000,1343488,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED
= 0x70a91c49d000
22:16:45.811693 mmap(0x70a91c5e5000,552960,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWR
= 0x70a91c5e5000
22:16:45.811715 mmap(0x70a91c66c000,57344,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
= 0x70a91c66c000
22:16:45.811742 mmap(0x70a91c67a000,12608,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
= 0x70a91c67a000
22:16:45.811774 close(3)                = 0
22:16:45.811797 openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libc.so.6",O_RDONLY|O_CLOEXEC)
= 3
22:16:45.811821 read(3,"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0
= 832
22:16:45.811840 pread64(3,"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0
= 784
22:16:45.811859 fstat(3,{st_mode=S_IFREG|0755,st_size=2125328,...}) = 0
22:16:45.811879 pread64(3,"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0
= 784
22:16:45.811896 mmap(NULL,2170256,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0)
= 0x70a91c000000
22:16:45.811923 mmap(0x70a91c028000,1605632,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED
= 0x70a91c028000
```

```
22:16:45.811949 mmap(0x70a91c1b0000,323584,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWR
= 0x70a91c1b0000
22:16:45.811972 mmap(0x70a91c1ff000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
= 0x70a91c1ff000
22:16:45.811999 mmap(0x70a91c205000,52624,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
= 0x70a91c205000
22:16:45.812030 close(3)                = 0
22:16:45.812054 openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libm.so.6",O_RDONLY|O_CLOEXEC)
= 3
22:16:45.812077 read(3,"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0
= 832
22:16:45.812097 fstat(3,{st_mode=S_IFREG|0644,st_size=952616,...}) = 0
22:16:45.812117 mmap(NULL,950296,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) =
0x70a91c317000
22:16:45.812140 mmap(0x70a91c327000,520192,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|
= 0x70a91c327000
22:16:45.812271 mmap(0x70a91c3a6000,360448,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWR
= 0x70a91c3a6000
22:16:45.812700 mmap(0x70a91c3fe000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|M
= 0x70a91c3fe000
22:16:45.812935 close(3)                = 0
22:16:45.813274 openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libgcc_s.so.1",O_RDONLY|O_CLOE
= 3
22:16:45.813672 read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0
= 832
22:16:45.814014 fstat(3,{st_mode=S_IFREG|0644,st_size=183024,...}) = 0
22:16:45.814378 mmap(NULL,185256,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) =
0x70a91c2e9000
22:16:45.814670 mmap(0x70a91c2ed000,147456,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|
= 0x70a91c2ed000
22:16:45.814881 mmap(0x70a91c311000,16384,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRI
= 0x70a91c311000
22:16:45.815085 mmap(0x70a91c315000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|M
= 0x70a91c315000
22:16:45.815267 close(3)                = 0
22:16:45.815425 mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0)
= 0x70a91c699000
22:16:45.815660 arch_prctl(ARCH_SET_FS,0x70a91c69a500) = 0
22:16:45.815788 set_tid_address(0x70a91c69a7d0) = 25039
22:16:45.815933 set_robust_list(0x70a91c69a7e0,24) = 0
22:16:45.816020 rseq(0x70a91c69ae20,0x20,0,0x53053053) = 0
22:16:45.816304 mprotect(0x70a91c1ff000,16384,PROT_READ) = 0
22:16:45.816533 mprotect(0x70a91c315000,4096,PROT_READ) = 0
22:16:45.816943 mprotect(0x70a91c3fe000,4096,PROT_READ) = 0
22:16:45.818517 mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0)
= 0x70a91c697000
22:16:45.819014 mprotect(0x70a91c66c000,45056,PROT_READ) = 0
22:16:45.819202 mprotect(0x593b0e043000,4096,PROT_READ) = 0
```

```
22:16:45.820309 mprotect(0x70a91c6e7000,8192,PROT_READ) = 0
22:16:45.820449 prlimit64(0,RLIMIT_STACK,NULL,{rlim_cur=8192*1024,rlim_max=RLIM64_INF
= 0
22:16:45.820661 munmap(0x70a91c69b000,56283) = 0
22:16:45.820887 futex(0x70a91c67a7bc,FUTEX_WAKE_PRIVATE,2147483647) = 0
22:16:45.821102 getrandom("\x44\x2d\xa4\xa2\x40\xd3\x32\xf7",8,GRND_NONBLOCK)
= 8
22:16:45.821358 brk(NULL)               = 0x593b26cb4000
22:16:45.821435 brk(0x593b26cd5000)     = 0x593b26cd5000
22:16:45.821598 fstat(1,{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0),...})
= 0
22:16:45.821770 write(1,"Starting the program...\n",24) = 24
22:16:45.821998 rt_sigaction(SIGINT,{sa_handler=SIG_IGN,sa_mask=[],sa_flags=SA_RESTOR
= 0
22:16:45.822296 rt_sigaction(SIGQUIT,{sa_handler=SIG_IGN,sa_mask=[],sa_flags=SA_RESTO
= 0
22:16:45.822544 rt_sigprocmask(SIG_BLOCK,[CHLD],[],8) = 0
22:16:45.822741 mmap(NULL,36864,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_ST
= 0x70a91c6a0000
22:16:45.823052 rt_sigprocmask(SIG_BLOCK,~[],[CHLD],8) = 0
22:16:45.823273 clone3({flags=CLONE_VM|CLONE_VFORK|CLONE_CLEAR_SIGHAND,exit_signal=SI
= 25040
22:16:45.825216 munmap(0x70a91c6a0000,36864) = 0
22:16:45.825450 rt_sigprocmask(SIG_SETMASK,[CHLD],NULL,8) = 0
22:16:45.826410 wait4(25040,[{WIFEXITED(s) && WEXITSTATUS(s) == 0}],0,NULL)
= 25040
22:17:21.139618 rt_sigaction(SIGINT,{sa_handler=SIG_DFL,sa_mask=[],sa_flags=SA_RESTOR
= 0
22:17:21.139708 rt_sigaction(SIGQUIT,{sa_handler=SIG_DFL,sa_mask=[],sa_flags=SA_RESTO
= 0
22:17:21.139731 rt_sigprocmask(SIG_SETMASK,[],NULL,8) = 0
22:17:21.140250 ---SIGCHLD {si_signo=SIGCHLD,si_code=CLD_EXITED,si_pid=25040,si_uid=1
---
22:17:21.140387 write(1,"The program ended with the code:"...,35) = 35
22:17:21.140829 exit_group(0)           = ?
22:17:21.141520 +++ exited with 0 +++
```