МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №3
по курсу Операционные системы

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

Москва, 2025

Содержание

1 Условие задачи

Цель работы

Приобретение практических навыков в:

- Освоение принципов работы с файловыми системами
- Обеспечение обмена данных между процессами посредством технологии «File mapping»

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 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

2 Метод решения

Программа реализует взаимодействие между родительским и дочерним процессами с использованием разделяемой памяти (memory-mapped files). Основной алгоритм:

Архитектура программы

- **Родительский процесс**: управляет вводом данных от пользователя, созданием дочернего процесса и передачей данных через разделяемую память
- **Дочерний процесс**: выполняет операции деления и записывает результаты в файл
- **Разделяемая память**: используется для обмена данными между процессами

Алгоритм работы

1. Родительский процесс создает разделяемую память и инициализирует структуру данных
2. Создается дочерний процесс с помощью системного вызова `fork()`
3. Пользователь вводит числа через родительский процесс
4. Числа передаются в дочерний процесс через разделяемую память
5. Дочерний процесс выполняет деление первого числа на последующие
6. Результаты записываются в файл
7. При обнаружении деления на ноль оба процесса завершают работу

Взаимодействие процессов

Программа использует механизм разделяемой памяти (shared memory) для обмена данными между процессами. Синхронизация осуществляется с помощью атомарных флагов.

Обработка ошибок

- Проверка деления на ноль в дочернем процессе
- Обработка ошибок создания разделяемой памяти
- Обработка ошибок открытия файла для записи

3 Описание программы

Структура программы

- **division_parents.cpp** - родительский процесс
- **division_child.cpp** - дочерний процесс
- **division_os.h/cpp** - вспомогательные функции для работы с процессами и разделяемой памятью
- **shared_data.h** - структура данных для разделяемой памяти

Основные типы данных

- `SharedData` - структура для хранения данных в разделяемой памяти:
  - `data_ready` - флаг готовности данных для обработки
  - `processing_complete` - флаг завершения обработки
  - `division_by_zero` - флаг обнаружения деления на ноль
  - `terminate_process` - флаг завершения работы
  - `numbers_count` - количество чисел
  - `numbers[100]` - массив чисел
  - `filename` - имя файла для записи результатов
- `ProcessRole` - перечисление для идентификации роли процесса

Ключевые функции

Основные функции

- `ProcessCreate()` - создание дочернего процесса
- `create_shared_memory()` - создание разделяемой памяти
- `open_shared_memory()` - открытие существующей разделяемой памяти
- `close_shared_memory()` - закрытие разделяемой памяти

Логика процессов

- **Родительский процесс**: ввод данных, передача в разделяемую память, ожидание результатов
- **Дочерний процесс**: чтение данных из разделяемой памяти, выполнение деления, запись в файл

Используемые системные вызовы

- **fork()** - создание нового процесса
- **shm_open()** - создание/открытие разделяемой памяти
- **mmap()** - отображение разделяемой памяти в адресное пространство процесса
- **munmap()** - удаление отображения разделяемой памяти
- **shm_unlink()** - удаление разделяемой памяти
- **wait()** - ожидание завершения дочернего процесса

4 Исходный код программы

Заголовочные файлы

```cpp
#pragma once

#include <atomic>

struct SharedData {
    std::atomic<bool> data_ready;
    std::atomic<bool> processing_complete;
    std::atomic<bool> division_by_zero;
    std::atomic<bool> terminate_process;
    int numbers_count;
    float numbers[100];
    char filename[256];
};
```

Листинг 1: shared_data.h - структура разделяемой памяти

```cpp
#pragma once

#include <cstddef>

enum ProcessRole {
    IS_PARENT,
    IS_CHILD
};

//

void* create_shared_memory(const char* name, size_t size);
void* open_shared_memory(const char* name, size_t size);
void close_shared_memory(const char* name, void* ptr, size_t size);
ProcessRole ProcessCreate();
```

Листинг 2: division_os.h - вспомогательные функции

Реализация функций

```cpp
#include "division_os.h"

#include <iostream>
#include <unistd.h>
#include <cstdlib>
#include <cstring>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/stat.h>

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;
    }
}

void* create_shared_memory(const char* name, size_t size) {
    int shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        std::cerr << "Error creating shared memory: " << strerror(errno) << std::endl;
        exit(-1);
    }

    if (ftruncate(shm_fd, size) == -1) {
        std::cerr << "Error setting shared memory size: " << strerror(errno) << std::endl;
        exit(-1);
    }

    void* ptr = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (ptr == MAP_FAILED) {
        std::cerr << "Error mapping shared memory: " << strerror(errno) << std::endl;
        exit(-1);
    }

    close(shm_fd);
    return ptr;
}

void* open_shared_memory(const char* name, size_t size) {
    int shm_fd = shm_open(name, O_RDWR, 0666);
    if (shm_fd == -1) {
        std::cerr << "Error opening shared memory: " << strerror(errno) << std::endl;
        exit(-1);
    }

```

```
    void* ptr = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (ptr == MAP_FAILED) {
        std::cerr << "Error mapping shared memory: " << strerror(errno) << std::endl;
        exit(-1);
    }

    close(shm_fd);
    return ptr;
}

void close_shared_memory(const char* name, void* ptr, size_t size) {
    if (munmap(ptr, size) == -1) {
        std::cerr << "Error unmapping shared memory: " << strerror(errno) << std::endl;
    }

    if (shm_unlink(name) == -1 && errno != ENOENT) {
        std::cerr << "Error unlinking shared memory: " << strerror(errno) << std::endl;
    }
}
```

Листинг 3: division_os.cpp - работа с процессами и памятью

Основные процессы

```
#include "division_os.h"
#include "shared_data.h"

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

int main() {
    const char* shm_name = "/division_shm";
    SharedData* shared_data = static_cast<SharedData*>(create_shared_memory(shm_name, sizeof(SharedData)));

    shared_data->data_ready = false;
    shared_data->processing_complete = false;
    shared_data->division_by_zero = false;
    shared_data->terminate_process = false;
    shared_data->numbers_count = 0;

    ProcessRole role = ProcessCreate();

    if (role == IS_CHILD) {
        close_shared_memory(shm_name, shared_data, sizeof(SharedData));
        execl("./division_child", "division_child", NULL);
        std::cerr << "Fault execl" << std::endl;
        exit(-1);
    }

    std::string file_name;
```

```
    std::cout << "Type file's name: ";
    std::cin >> file_name;

    strncpy(shared_data->filename, file_name.c_str(), sizeof(shared_data->filename) - 1);
    shared_data->filename[sizeof(shared_data->filename) - 1] = '\0';

    std::cout << "Write numbers separated by space (first divides by others):" << std::endl;
    std::cout << "For exit write 'q'" << std::endl;

    std::string input;
    std::cin.ignore();

    while (!shared_data->division_by_zero && !shared_data->terminate_process) {
        std::cout << "Write numbers: ";
        std::getline(std::cin, input);

        if (input == "q" || input == "quit") {
            shared_data->terminate_process = true;
            shared_data->data_ready = true;
            break;
        }

        if (input.empty()) {
            continue;
        }

        std::vector<float> numbers;
        std::stringstream ss(input);
        float num;

        while (ss >> num) {
            numbers.push_back(num);
        }

        if (numbers.size() > 0) {
            shared_data->numbers_count = numbers.size();
            for (size_t i = 0; i < numbers.size() && i < 100; i++) {
                shared_data->numbers[i] = numbers[i];
            }

            shared_data->data_ready = true;
            shared_data->processing_complete = false;

            while (!shared_data->processing_complete && !shared_data->division_by_zero) {
                usleep(1000);
            }

            if (shared_data->division_by_zero) {
                std::cout << "Divide by zero. Program is closing" << std::endl;
                break;
            }
        } else {
            std::cout << "Error: false format" << std::endl;
        }
    }
```

```
88
89      sleep(1);
90
91      close_shared_memory(shm_name, shared_data, sizeof(SharedData));
92      wait(NULL);
93      std::cout << "Program finished" << std::endl;
94
95      return 0;
96  }
```

Листинг 4: division_parents.cpp - родительский процесс

```
1   #include "division_os.h"
2   #include "shared_data.h"
3
4   #include <iostream>
5   #include <fstream>
6   #include <string>
7   #include <unistd.h>
8
9   int main() {
10      const char* shm_name = "/division_shm";
11      SharedData* shared_data = static_cast<SharedData*>(open_shared_memory(
    shm_name, sizeof(SharedData)));
12
13      std::ofstream out(shared_data->filename, std::ios::app);
14      if (!out.is_open()) {
15          std::cerr << "Fault of opening file: " << shared_data->filename <<
    std::endl;
16          close_shared_memory(shm_name, shared_data, sizeof(SharedData));
17          return 1;
18      }
19
20      while (!shared_data->terminate_process) {
21          if (shared_data->data_ready) {
22              if (shared_data->numbers_count == 0) {
23                  break;
24              }
25
26              if (shared_data->numbers_count < 2) {
27                  out << "Fault: program needs minimum 2 numbers" << std::endl
    ;
28              } else {
29                  float result = shared_data->numbers[0];
30                  bool error_occurred = false;
31
32                  for (int i = 1; i < shared_data->numbers_count; i++) {
33                      if (shared_data->numbers[i] == 0.0f) {
34                          out << "Fault: division by zero" << std::endl;
35                          error_occurred = true;
36                          shared_data->division_by_zero = true;
37                          break;
38                      }
39                      result /= shared_data->numbers[i];
40                  }
41
42                  if (!error_occurred) {
43                      out << shared_data->numbers[0];
44                      for (int i = 1; i < shared_data->numbers_count; i++) {
45                          out << " / " << shared_data->numbers[i];
46                      }
```

```
                    out << " = " << result << std::endl;
                }
            }

            out.flush();
            shared_data->processing_complete = true;
            shared_data->data_ready = false;
        }

        usleep(1000);
    }

    out.close();
    close_shared_memory(shm_name, shared_data, sizeof(SharedData));
    return 0;
}
```

Листинг 5: division_child.cpp - дочерний процесс

5 Системные вызовы программы

```
22:16:45.809239 execve("./division_parents", ["./division_parents"], 0x7ffe80dd98c8 /* 6
22:16:45.810684 brk(NULL)               = 0x593b26cb4000
22:16:45.810915 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
22:16:45.811138 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or director
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_CLOE
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\0"
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) = 0x70a9
22:16:45.811667 mmap(0x70a91c49d000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED
22:16:45.811693 mmap(0x70a91c5e5000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWR
22:16:45.811715 mmap(0x70a91c66c000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
22:16:45.811742 mmap(0x70a91c67a000, 12608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
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)
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\0\
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\0\0\0\
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\0\0\0\
22:16:45.811896 mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x70a9
22:16:45.811923 mmap(0x70a91c028000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED
22:16:45.811949 mmap(0x70a91c1b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWR
22:16:45.811972 mmap(0x70a91c1ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
22:16:45.811999 mmap(0x70a91c205000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
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)
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\0"
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) = 0x70a91
22:16:45.812140 mmap(0x70a91c327000, 520192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
```

```
22:16:45.812271 mmap(0x70a91c3a6000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWR
22:16:45.812700 mmap(0x70a91c3fe000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|M
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_CLOEX
22:16:45.813672 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"
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) = 0x70a91
22:16:45.814670 mmap(0x70a91c2ed000, 147456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
22:16:45.814881 mmap(0x70a91c311000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRI
22:16:45.815085 mmap(0x70a91c315000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|M
22:16:45.815267 close(3)                = 0
22:16:45.815425 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
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)
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_IN
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, "Type file's name: ", 18) = 18
22:16:45.821998 rt_sigaction(SIGINT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTOR
22:16:45.822296 rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTO
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
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=SIGC
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
22:17:21.139708 rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTO
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=
22:17:21.140387 write(1, "Program finished", 16) = 16
22:17:21.140829 exit_group(0)           = ?
22:17:21.141520 +++ exited with 0 +++
```

6 Пример работы программы

Ввод данных

```
Type file's name: results.txt
Write numbers separated by space (first divides by others):
For exit write 'q'
Write numbers: 100 2 5
Write numbers: 50 2 2.5
Write numbers: 10 0 5
Divide by zero. Program is closing
Program finished
```

Содержимое файла results.txt

```
100 / 2 / 5 = 10
50 / 2 / 2.5 = 10
Fault: division by zero
```

Вывод

В ходе выполнения лабораторной работы была успешно реализована программа для выполнения операций деления с использованием взаимодействия процессов через разделяемую память. Основные достижения и наблюдения:

1. **Взаимодействие процессов**: Программа корректно реализует взаимодействие между родительским и дочерним процессами с использованием механизма разделяемой памяти (shared memory).

2. **Обработка ошибок**: Реализована надежная обработка ошибок, включая проверку деления на ноль в дочернем процессе и корректное завершение работы обоих процессов при обнаружении ошибки.

3. **Синхронизация**: Использование атомарных флагов обеспечивает правильную синхронизацию между процессами без необходимости использования дополнительных механизмов синхронизации.

4. **Работа с файловой системой**: Программа успешно записывает результаты операций в файл, демонстрируя практические навыки работы с файловыми системами.

5. **Обработка пользовательского ввода**: Реализован удобный интерфейс для ввода данных с возможностью многократного ввода чисел и корректного завершения работы.

6. **Системные вызовы**: Программа эффективно использует системные вызовы для работы с процессами и разделяемой памятью, включая `fork()`, `shm_open()`, `mmap()` и другие.

Программа демонстрирует практическое применение технологии «File mapping» для обмена данными между процессами и может служить основой для разработки более сложных многопроцессных приложений.