МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №0
по курсу «Операционные системы»

Выполнил: Г. А. Стрекаловский
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы:
Приобретение практических навыков в:

- Управлении процессами в ОС
- Обеспечении обмена данными 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) должен создать для решение 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.
Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. 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.
6 вариант) В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

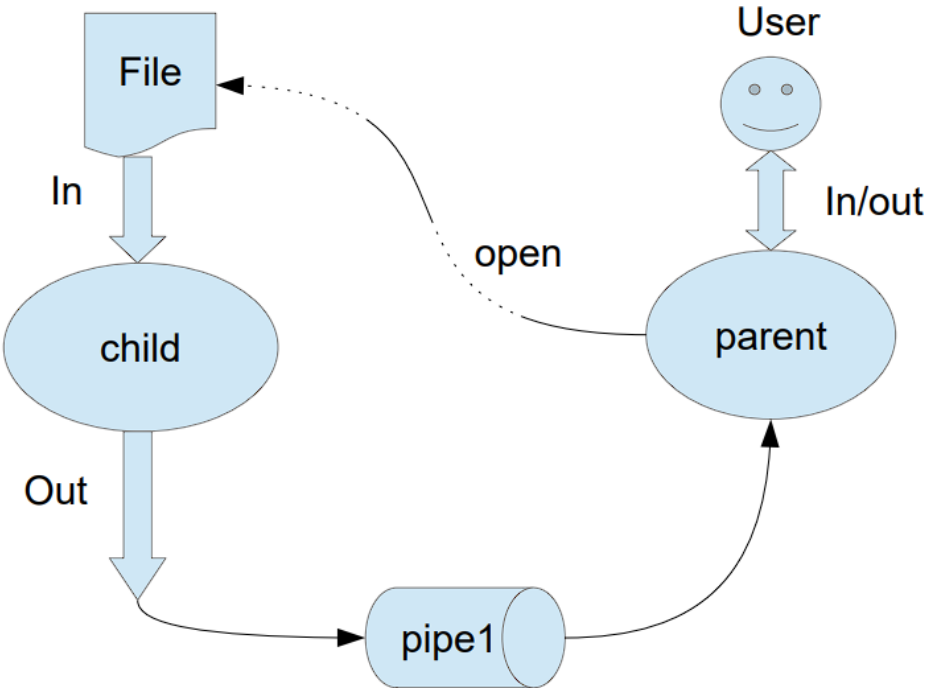

Рис. 1: Схема работы программы

Вариант: 6

Метод решения

В общих чертах алгоритм выглядит так:

1. Родительский процесс создаёт канал (pipe)
2. Затем происходит создание дочернего процесса.
3. После этого каждый из процессов закрывает ненужные концы канала (read или write), чтобы не было утечек.
4. Родитель открывает файл для ребенка. У ребенка стандартный поток ввода переопределяется этим файлом.
5. Ребенок выводит ответ в канал, который является для него стандартным потоком вывода.
6. Родитель читает из канала и выводит ответ в стандартный поток вывода (в консоль пользователю).

Все системные вызовы инкапсулированы, ими занимается специальный класс SysCall

Описание программы

Программа разделена на 6 файлов:

1. main.cpp - основная программа, ее пользователь запускате, чтобы начать работу
2. child.cpp - программа ребенка, там происходят все вычисления
3. sys_calls.cpp - файл с реализацией всех основных функций, которые прячут системные вызовы
4. sys_calls.hpp - заголовочный файл с описанием структуры класса SysCall
5. os.hpp - заголовочный для кроссплатформенности
6. os.cpp - реализация кросплатформенности

Описание используемых функций:

- StartProcesses() - запускает работу всей программы. В ней создается канал и ребенок.
- CreatePipe() - инкапсулирует системный вызов pipe(), создает канал
- CreareChild() - инкапсулирует системный вызов fork(), создает ребенка
- SetChildProc() - управляет входами и выходами в канал ребенка, переопределяет стандартный ввод и вывод ребенка, инкапсулирует системные вызовы close(), dup2(), execl()
- SetParentProc() - закрывает запись в канал для родителя, записывает ответ от ребенка и пишет его в стандартный поток вывода пользователю, инкапсулирует системные вызовы close(), waitpid().

- Clean() - проверяет закрыты ли все дескрипторы, если нет - закрывает.

Используемые системные вызовы:

- fork() - создание дочернего процесса
- close() - закрыть файл
- open() - открытие файла
- pipe() - создание неименованного канала для передачи данных между процессами
- dup2() - переназначение файлового дескриптора
- execl() - замена образа памяти процесса
- waitpid() - ожидание завершения дочернего процесса

Результаты

Результатом является одно число - сумма всех чисел в файле.

Выводы

Есть специальные системные вызовы fork() для создания дочернего процесса, pipe() для создания каналов общения между процессами. Важно держать в голове,что после форка создается полная копия процесса и если это не учесть то можно попасть в просак и запутаться в процессах, но само по себе разделение на процессы очень удобно тем, что появляется выполнять задачи параллельно. Отличить процессы можно по их id. Все каналы необходимо закрывать после окончания процесса. Изменить файловый дескриптор можно с помощью dup2(), так можно переопределять стандартные потоки, чтобы не возникало путаницы.

Исходная программа

```
#include <cstddef>
#include <cstdio>
#include <unistd.h>
#include <sys/wait.h>

namespace os {
    int Pipe(int pipefd[2]);
    pid_t Fork();
    int Dup2(int fd1, int fd2);
    int Close(int fd);
    int Execl(const char* proc_path, const char* proc_name);
    pid_t Wait(pid_t pid);
    ssize_t Write(int fd, char* buf, size_t cnt);
    void Perror(const char* s);
}
```

Листинг 1: *Добавляем кроссплатформенность*

```
#include "../include/os.hpp"
#include <cstdio>
#include <sys/wait.h>
#include <unistd.h>

namespace os {
    int Pipe(int pipefd[2]) {return pipe(pipefd);}

    pid_t Fork() {return fork();}

    int Dup2(int fd1, int fd2) {return dup2(fd1, fd2);}

    int Close(int fd) {return close(fd);}

    int Execl(const char *proc_path, const char *proc_name) {
        return execl(proc_path, proc_name, nullptr);
    }

    pid_t Wait(pid_t pid) {
        return waitpid(pid, nullptr, 0);
    }

    ssize_t Write(int fd, char* buf, size_t cnt) {
        return write(fd, buf, cnt);
    }

    void Perror(const char* s) {perror(s);}
}
```

Листинг 2: *Добавляем кроссплатформенность*

```
#include <string>


namespace sys_call {
```

```
    class SysCall {
      private:
        int pipe_rd_;
        int pipe_wr_;
        int child_pid_;

        void CreatePipe();
        void CreateChild();
        void SetChildProc(const std::string& filename);
        void SetParentProc();
        void Clean();

      public:
        SysCall();
        void StartProcesses(const std::string& filename);
        ~SysCall() = default;
    };
}
```

Листинг 3: *Сигнатура для класса SysCall*

```
#include "../include/sys_calls.hpp"
#include "../include/os.hpp"
#include <iostream>
#include <sched.h>
#include <string>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>

namespace sys_call {
    SysCall::SysCall(): pipe_rd_(-1), pipe_wr_(-1), child_pid_(-1) {}

    void SysCall::CreatePipe() {
        int pipe_fd[2];
        if (os::Pipe(pipe_fd) == -1) {
            os::Perror("pipe");
        }
        pipe_rd_ = pipe_fd[0];
        pipe_wr_ = pipe_fd[1];
    }

    void SysCall::CreateChild() {
        child_pid_ = os::Fork();
        if (child_pid_ == -1) {
            os::Perror("fork");
        }
    }

    void SysCall::SetChildProc(const std::string& filename) {
        os::Close(pipe_rd_);
        os::Dup2(pipe_wr_, STDOUT_FILENO);
        os::Close(pipe_wr_);

        int file_fd = open(filename.c_str(), O_RDONLY);
        if (file_fd == -1) {
            os::Perror("open");
```

```
        }
        os::Dup2(file_fd, STDIN_FILENO);
        os::Close(file_fd);

        os::Execl("./child", "./child");
        os::Perror("exec");
    }

    void SysCall::SetParentProc() {
        os::Close(pipe_wr_);

        char buffer[1024];
        ssize_t cnt;
        while ((cnt = read(pipe_rd_, buffer, sizeof(buffer))) > 0) {
            os::Write(STDOUT_FILENO, buffer, cnt);
        }
        os::Close(pipe_rd_);
        os::Wait(child_pid_);
    }

    void SysCall::StartProcesses(const std::string& filename) {
        if (access(filename.c_str(), F_OK) == -1) {
            std::cerr << "Error: File '" << filename << "' does not exist!" << std::
                endl;
            return;
        }
        CreatePipe();
        CreateChild();
        if (child_pid_ == 0) {
            SetChildProc(filename);
        } else {
            SetParentProc();
        }
    }
}
```

Листинг 4: *Реализация класса SysCall*

```
#include <iostream>
#include <sstream>
#include <string>

int main() {
    std::string line;

    while (std::getline(std::cin, line)) {
        std::stringstream ss(line);
        int num;
        int sum = 0;
        bool is_has_number = false;

        while (ss >> num) {
            sum += num;
            is_has_number = true;
        }

        if (is_has_number) {
```

```
            std::cout << sum << std::endl;
        }
    }

    return 0;
}
```

Листинг 5: *Реализация дочернего процесса*

```
#include <iostream>
#include <string>
#include "include/sys_calls.hpp"

int main() {
    std::string filename;
    std::cout << "Enter filename with numbers" << std::endl;
    std::cin >> filename;
    if (filename.empty()) {
        std::cerr << "File shouldn‘t be empty" << std::endl;
        return 1;
    }

    sys_call::SysCall oper;
    oper.StartProcesses(filename);
    return 0;
}
```

Листинг 6: *Запуск работы программы*

Вывод strace:

```
execve("./main",["./main"],0x7ffdcb4c7490 /* 76 vars */) = 0
brk(NULL)                                       = 0x583f3d484000
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x78e0b2e82000
access("/etc/ld.so.preload",R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD,"/etc/ld.so.cache",O_RDONLY|O_CLOEXEC) = 3
fstat(3,{st_mode=S_IFREG|0644,st_size=74755,...}) = 0
mmap(NULL,74755,PROT_READ,MAP_PRIVATE,3,0) = 0x78e0b2e6f000
close(3)                                        = 0
openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libstdc++.so.6",O_RDONLY|O_CLOEXEC)
= 3
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"...,832)
= 832
fstat(3,{st_mode=S_IFREG|0644,st_size=2592224,...}) = 0
mmap(NULL,2609472,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x78e0b2a00000
mmap(0x78e0b2a9d000,1343488,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3
= 0x78e0b2a9d000
mmap(0x78e0b2be5000,552960,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1e5000)
= 0x78e0b2be5000
mmap(0x78e0b2c6c000,57344,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x78e0b2c6c000
```

```
mmap(0x78e0b2c7a000,12608,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1
= 0x78e0b2c7a000
close(3)                                = 0
openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libgcc_s.so.1",O_RDONLY|O_CLOEXEC) =
3
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\0"...,832)
= 832
fstat(3,{st_mode=S_IFREG|0644,st_size=183024,...}) = 0
mmap(NULL,185256,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x78e0b2e41000
mmap(0x78e0b2e45000,147456,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x78e0b2e45000
mmap(0x78e0b2e69000,16384,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x28000)
= 0x78e0b2e69000
mmap(0x78e0b2e6d000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x78e0b2e6d000
close(3)                                = 0
openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libc.so.6",O_RDONLY|O_CLOEXEC) = 3
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\0\0"...,832)
= 832
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"...,784,64)
= 784
fstat(3,{st_mode=S_IFREG|0755,st_size=2125328,...}) = 0
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"...,784,64)
= 784
mmap(NULL,2170256,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x78e0b2600000
mmap(0x78e0b2628000,1605632,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3
= 0x78e0b2628000
mmap(0x78e0b27b0000,323584,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1b0000)
= 0x78e0b27b0000
mmap(0x78e0b27ff000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x78e0b27ff000
mmap(0x78e0b2805000,52624,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1
= 0x78e0b2805000
close(3)                                = 0
openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libm.so.6",O_RDONLY|O_CLOEXEC) = 3
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"...,832)
= 832
fstat(3,{st_mode=S_IFREG|0644,st_size=952616,...}) = 0
mmap(NULL,950296,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x78e0b2d58000
mmap(0x78e0b2d68000,520192,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x78e0b2d68000
mmap(0x78e0b2de7000,360448,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x8f000)
= 0x78e0b2de7000
mmap(0x78e0b2e3f000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x78e0b2e3f000
close(3)                                = 0
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x78e0b2d56000
mmap(NULL,12288,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x78e0b2d53000
```

```
arch_prctl(ARCH_SET_FS,0x78e0b2d53740) = 0
set_tid_address(0x78e0b2d53a10)         = 11493
set_robust_list(0x78e0b2d53a20,24)     = 0
rseq(0x78e0b2d54060,0x20,0,0x53053053) = 0
mprotect(0x78e0b27ff000,16384,PROT_READ) = 0
mprotect(0x78e0b2e3f000,4096,PROT_READ) = 0
mprotect(0x78e0b2e6d000,4096,PROT_READ) = 0
mprotect(0x78e0b2c6c000,45056,PROT_READ) = 0
mprotect(0x583f2043a000,4096,PROT_READ) = 0
mprotect(0x78e0b2eba000,8192,PROT_READ) = 0
prlimit64(0,RLIMIT_STACK,NULL,{rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY})
= 0
munmap(0x78e0b2e6f000,74755)           = 0
futex(0x78e0b2c7a7bc,FUTEX_WAKE_PRIVATE,2147483647) = 0
getrandom("\xdc\xdd\x33\x78\x25\xe5\x4a\xd4",8,GRND_NONBLOCK) = 8
brk(NULL)                               = 0x583f3d484000
brk(0x583f3d4a5000)                     = 0x583f3d4a5000
fstat(1,{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0x1),...}) = 0
write(1,"Enter filename with numbers\n",28) = 28
fstat(0,{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0x1),...}) = 0
read(0,"numbers.txt\n",1024)          = 12
access("numbers.txt",F_OK)             = 0
pipe2([3,4],0)                         = 0
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_ti
= 11929
close(4)                                = 0
read(3,"793\n",1024)                   = 4
write(1,"793\n",4)                     = 4
read(3,"",1024)                        = 0
---SIGCHLD {si_signo=SIGCHLD,si_code=CLD_EXITED,si_pid=11929,si_uid=1000,si_status=0,
---
close(3)                                = 0
wait4(11929,NULL,0,NULL)              = 11929
lseek(0,-1,SEEK_CUR)                    = -1 ESPIPE (Недопустимая операция смещения)
exit_group(0)                           = ?
+++ exited with 0 +++
```