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»

Группа: М8О-210Б-23

Студент: Григорян А.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 22.02.25

Москва, 2025

Постановка задачи

Вариант 6.

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.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создает канал и помещает дескрипторы файла для чтения и записи в fd[0] и fd[1].
- pid_t getpid(void); – возвращает ID вызывающего процесса.
- int open(const char *__file, int __oflag, …); – используется для открытия файла для чтения, записи или и того, и другого.
- ssize_t write(int __fd, const void *__buf, size_t __n); – Записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int close(int __fd); – сообщает операционной системе об окончании работы с файловым дескриптором, и закрывает файл(FD).
- int dup2(int __fd, int __fd2); – копирует FD в FD2, закрыв FD2 если это требуется.
- int execv(const char *__path, char *const *__argv); – заменяет образ текущего процесса на образ нового процесса, определённого в пути path.
- ssize_t read(int __fd, void *__buf, size_t __nbytes); – считывает указанное количество байт из файла(FD) в буфер(BUF).
- pid_t wait(int *__stat_loc); – используются для ожидания изменения состояния процесса-потомка вызвавшего процесса и получения информации о потомке, чьё состояние изменилось.

Для выполнения данной лабораторной работы я изучил указанные выше системные вызовы, а также

Программа parent.cpp получает и обрабатывает входные данные - название файла, а также создает pipe после чего происходит системный вызов fork. Родительский процесс переходит в стадию ожидания результаты работы дочернего процесса.

Дочерний процесс перенаправляет потоки ввода и вывода, после через с помощью execl запускает работу следующей программы – child.cpp.

Программа child.cpp происходит считывание данных из файла, их разделения и обработка для подсчета конечных сумм чисел в строках. После этого результат программы передается через pipe обратно в родительский процесс.

Родительский процесс же в свою очередь получает результат программы и выводит его, после чего дожидается полного окончания дочернего процесса и завершает свою работу.

Файл states.h хранит в себе исключительно перечисление типа enum для корректной обработки всех возможных ошибок в ходе выполнения программы.

Код программы

parent.cpp

```
#include <asm-generic/errno.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <cstdlib>

#include <cstring>

#include <cstdio>

#include "states.hpp"


int main(){


char fileName[250];

kState code = kS_OK;




write(STDOUT_FILENO, "enter file name\n", strlen("enter file name") + 1);


ssize_t readBytes = read(STDOUT_FILENO, fileName, sizeof(fileName) - 1 );


if (readBytes > 0){

   fileName[readBytes - 1] = '\0';
```

```
} else {
    code = kE_FILENAME_NOT_RECEIVED;
}

int pipe_fd[2];



if (pipe(pipe_fd) == -1){
    code = kE_PIPE_ERROR;
}

pid_t pid = fork();

if (pid == -1){
    code = kE_FORK_ERROR;
} else if (pid == 0) {
    close(pipe_fd[0]);

     int file_fd = open(fileName, O_RDONLY);

    if (file_fd == -1) {
        code = kE_FILENAME_NOT_RECEIVED;
    }

    dup2(file_fd, STDIN_FILENO);
    dup2(pipe_fd[1], STDOUT_FILENO);

    close(file_fd);
    close(pipe_fd[1]);
```

```
    execl("./child", "./child", nullptr);

    code = kE_EXECL_ERROR;
} else {
    close(pipe_fd[1]);

    char buffer[256];
    ssize_t bytesRead;
    while ((bytesRead = read(pipe_fd[0], buffer, sizeof(buffer) - 1)) > 0) {
            buffer[bytesRead] = '\0';
            write(STDOUT_FILENO, buffer, bytesRead);
        }

        close(pipe_fd[0]);

        int status;

        waitpid(pid, &status, 0);
}
return 0;
}
```

child.cpp

```
#include <asm-generic/errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <cstdlib>
#include <cstring>
#include <cstdio>
#include "states.hpp"
```

```
int main(){

char fileName[250];
kState code = kS_OK;


write(STDOUT_FILENO, "enter file name\n", strlen("enter file name") + 1);

ssize_t readBytes = read(STDOUT_FILENO, fileName, sizeof(fileName) - 1 );

if (readBytes > 0){
    fileName[readBytes - 1] = '\0';
} else {
    code = kE_FILENAME_NOT_RECEIVED;
}

int pipe_fd[2];


if (pipe(pipe_fd) == -1){
    code = kE_PIPE_ERROR;
}

pid_t pid = fork();

if (pid == -1){
    code = kE_FORK_ERROR;
} else if (pid == 0) {
    close(pipe_fd[0]);
```

```
    int file_fd = open(fileName, O_RDONLY);

    if (file_fd == -1) {
        code = kE_FILENAME_NOT_RECEIVED;
    }

    dup2(file_fd, STDIN_FILENO);
    dup2(pipe_fd[1], STDOUT_FILENO);

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

    execl("./child", "./child", nullptr);

    code = kE_EXECL_ERROR;
} else {
    close(pipe_fd[1]);

    char buffer[256];
    ssize_t bytesRead;
    while ((bytesRead = read(pipe_fd[0], buffer, sizeof(buffer) - 1)) > 0) {
        buffer[bytesRead] = '\0';
        write(STDOUT_FILENO, buffer, bytesRead);
    }

    close(pipe_fd[0]);

    int status;
```

```
        waitpid(pid, &status, 0);


}




return 0;
}
```

Протокол работы программы

[arcsenius@ars-nbdewxx9 src]$ strace ./parent ./child

execve("./parent", ["./parent", "./child"], 0x7ffd64643928 /* 66 vars */) = 0

brk(NULL) = 0x60791f2bd000

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=241615, ...}) = 0

mmap(NULL, 241615, PROT_READ, MAP_PRIVATE, 3, 0) = 0x70279cafe000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/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"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0755, st_size=22040176, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x70279cafc000

mmap(NULL, 2641984, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x70279c800000

mmap(0x70279c897000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x97000) = 0x70279c897000

mmap(0x70279c9e4000, 589824, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e4000) = 0x70279c9e4000

mmap(0x70279ca74000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x274000) = 0x70279ca74000

mmap(0x70279ca82000, 12352, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x70279ca82000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/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"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0755, st_size=973144, ...}) = 0

mmap(NULL, 975176, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x70279c711000

mmap(0x70279c71f000, 536576, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000) = 0x70279c71f000

mmap(0x70279c7a2000, 376832, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x91000) = 0x70279c7a2000

mmap(0x70279c7fe000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xec000) = 0x70279c7fe000

close(3)

mprotect(0x70279c703000, 16384, PROT_READ) = 0

mprotect(0x70279cafa000, 4096, PROT_READ) = 0

mprotect(0x70279c7fe000, 4096, PROT_READ) = 0

mprotect(0x70279ca74000, 53248, PROT_READ) = 0

mprotect(0x60790dc40000, 4096, PROT_READ) = 0

mprotect(0x70279cb73000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x70279cafe000, 241615) = 0

futex(0x70279ca826bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0

getrandom("\x1e\xbe\x57\x3f\xd4\x74\x4a\x7d", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x60791f2bd000

brk(0x60791f2de000) = 0x60791f2de000

write(1, "enter file name\n", 16enter file name

) = 16

read(1, test.txt

"test.txt\n", 249) = 9

pipe2([3, 4], 0) = 0

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x70279cacd890) = 15235

close(4) = 0

read(3, "\320\241\321\203\320\274\320\274\320\260: 10\n\320\241\321\203\320\274\320\274\320\260: 1\n\320\241\321"..., 255) = 43

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=15235, si_uid=1001, si_status=0, si_utime=0, si_stime=0} ---

write(1, "\320\241\321\203\320\274\320\274\320\260: 10\n\320\241\321\203\320\274\320\274\320\260: 1\n\320\241\321"..., 43Сумма: 10

Сумма: 1

Сумма: 2

) = 43

read(3, "", 255) = 0

close(3) = 0

wait4(15235, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 15235

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе написания данной лабораторной работы я научился работать с системными вызовами в СИ. Научился создавать программы, состоящие из нескольких процессов, и передавать данные между процессами по каналам. Во время отладки программы я познакомился с утилитой strace, она оказалась достаточно удобной для получения информации о работе многопоточных программ. Лабораторная работа была довольно интересна, так как я раньше не создавал программы на СИ, которые запускают несколько процессов параллельно.