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-215Б-23

Студент: Агафонов А. С.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 18.09.24

Москва, 2024

**Постановка задачи**

**Вариант 20.**

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 два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t pid1 = fork();– создает дочерний процесс.
* int pipe(fd1);– создание канала.
* close(fd1[…]);– закрытия файлового дескриптора.
* int file1 = open(file1\_name.c\_str(), O\_WRONLY | O\_CREAT | O\_TRUNC); – открывает файл.
* int dup2(int oldfd, int newfd); – переназначение файлового дескриптора.
* execl("./child", "./child", NULL); – заменяет изображение памяти текущего процесса на новый исполняемый файл.
* write();– записывает данные в канал.
* read();– читает данные из канала.

При запуске программы пользователю предлагается ввести имена файлов для первого и второго дочерних процессов. Эти файлы будут созданы, если не существуют, и в них будет записываться вывод соответствующих процессов.

После получения имен файлов программа создает два канала (fd1 и fd2) для межпроцессорной коммуникации и два дочерних процесса. Родительский процесс считывает строки с консольного ввода с помощью функции get\_string(), которая позволяет вводить строки произвольной длины.

Для каждой строки происходит фильтрация по длине. Строки длины больше 10 символов отправляются в pipe2, иначе в pipe1.

Передача строки в дочерний процесс осуществляется функцией to\_pipe, которая записывает строку посимвольно в указанный канал связи. Строки передаются до тех пор, пока не встретится null-терминатор, который служит признаком конца строки.

Дочерние процессы закрывают ненужные каналы связи и перенаправляют свой стандартный вывод в созданный файл, а стандартный ввод – через соответствующий pipe. Затем они заменяют свой образ памяти и выполняют программу child.

Каждый дочерний процесс затем заменяет своё изображение памяти на исполняемый файл child, который считывает строку, выполняет её инверсию с помощью функции reverse\_string и выводит результат в файл. Функция reverse\_string инвертирует строку, используя технику "двух указателей".

Если пользователь нажимает Ctrl+D, родительский процесс отправляет сигнал завершения обоим дочерним процессам и завершает свою работу.

**Код программы**

**main.cpp**

#include <iostream>

#include <string>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

// Функция для записи строки в pipe

void to\_pipe(int\* fd, const std::string& str) {

if (write(fd[1], str.c\_str(), str.size()) < 0) {

perror("Can't write to the pipe");

exit(4);

}

char null\_terminator = '\0'; // передаём null-терминатор

if (write(fd[1], &null\_terminator, sizeof(char)) < 0) {

perror("Can't write to the pipe");

exit(4);

}

}

int main() {

std::cout << "Enter file's name for child process 1: ";

std::string file1\_name;

std::getline(std::cin, file1\_name);

std::cout << "Enter file's name for child process 2: ";

std::string file2\_name;

std::getline(std::cin, file2\_name);

// Открытие файлов для дочерних процессов

int file1 = open(file1\_name.c\_str(), O\_WRONLY | O\_CREAT | O\_TRUNC);

int file2 = open(file2\_name.c\_str(), O\_WRONLY | O\_CREAT | O\_TRUNC);

if (file1 < 0 || file2 < 0) {

perror("Can't open file");

exit(1);

}

// Создание двух каналов

int fd1[2], fd2[2];

if (pipe(fd1) < 0 || pipe(fd2) < 0) {

perror("Can't create pipe");

exit(2);

}

pid\_t pid1 = fork();

if (pid1 < 0) {

perror("Can't create child process");

exit(3);

}

if (pid1 > 0) { // Родительский процесс

pid\_t pid2 = fork();

if (pid2 < 0) {

perror("Can't create child process");

exit(3);

}

if (pid2 > 0) { // Родительский процесс

close(fd1[0]);

close(fd2[0]);

while (true) {

std::string s;

std::getline(std::cin, s);

if (s.empty()) { // Завершение работы при пустой строке

to\_pipe(fd1, s);

to\_pipe(fd2, s);

break;

}

// Если длина строки больше 10 символов, отправляем её в pipe2, иначе — в pipe1

if (s.size() > 10) {

to\_pipe(fd2, s);

} else {

to\_pipe(fd1, s);

}

}

close(fd1[1]);

close(fd2[1]);

} else { // Дочерний процесс 2

close(fd1[0]);

close(fd1[1]);

close(fd2[1]);

if (dup2(fd2[0], STDIN\_FILENO) < 0) {

perror("Can't redirect stdin for child process");

exit(5);

}

if (dup2(file2, STDOUT\_FILENO) < 0) {

perror("Can't redirect stdout for child process");

exit(5);

}

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

perror("Can't execute child process");

exit(6);

}

} else { // Дочерний процесс 1

close(fd1[1]);

close(fd2[0]);

close(fd2[1]);

if (dup2(fd1[0], STDIN\_FILENO) < 0) {

perror("Can't redirect stdin for child process");

exit(5);

}

if (dup2(file1, STDOUT\_FILENO) < 0) {

perror("Can't redirect stdout for child process");

exit(5);

}

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

perror("Can't execute child process");

exit(6);

}

return 0;

}

**child.cpp**

#include <iostream>

#include <string>

#include <algorithm>

#include <unistd.h>

// Функция для реверса строки

void reverse\_string(std::string& str) {

std::reverse(str.begin(), str.end());

}

int main() {

while (true) {

std::string str;

char c;

// Чтение строки до null-терминатора

while (read(STDIN\_FILENO, &c, 1) > 0 && c != '\0') {

str += c;

}

if (str.empty()) { // Завершаем работу, если пустая строка

break;

}

reverse\_string(str);

std::cout << str << std::endl; // Записываем реверсированную строку в файл

std::cout.flush(); // Очищаем буфер вывода

}

return 0;

}

**Протокол работы программы**

**Тестирование:**

root@bdc60f1003f5:/workspaces/os\_base/lab1# ./main

Enter file's name for child process 1: ch1

Enter file's name for child process 2: ch2

0123456789

0123456789a

root@bdc60f1003f5:/workspaces/os\_base/lab1# cat ch1

9876543210

root@bdc60f1003f5:/workspaces/os\_base/lab1# cat ch2

a9876543210

**Strace:**

38485 **execve**("./main", ["./main"], 0x7ffc849a5ae8 /\* 30 vars \*/) = 0

38485 **close**(3) = 0

38485 **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

38485 **close**(3) = 0

38485 **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

38485 **close**(3) = 0

38485 **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

38485 **close**(3) = 0

38485 **read**(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832

38485 **close**(3) = 0

38485 **write**(1, "Enter file's name for child proc"..., 39) = 39

38485 **read**(0, "ch1\n", 1024) = 4

38485 **write**(1, "Enter file's name for child proc"..., 39) = 39

38485 **read**(0, "ch2\n", 1024) = 4

38485 **clone**(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f5b96080750) = 38521

38485 **clone**(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD <unfinished ...>

38521 **close**(6 <unfinished ...>

38485 <... clone resumed>, child\_tidptr=0x7f5b96080750) = 38522

38521 <... close resumed>) = 0

38485 **close**(5) = 0

38521 **close**(7 <unfinished ...>

38485 **close**(7) = 0

38521 <... close resumed>) = 0

38485 read(0, <unfinished ...>

38522 **close**(5 <unfinished ...>

38521 **close**(8 <unfinished ...>

38522 <... close resumed>) = 0

38521 <... close resumed>) = 0

38522 **close**(6 <unfinished ...>

38521 **dup2**(5, 0 <unfinished ...>

38522 <... close resumed>) = 0

38521 <... dup2 resumed>) = 0

38522 **close**(8 <unfinished ...>

38521 **dup2**(3, 1 <unfinished ...>

38522 <... close resumed>) = 0

38521 <... dup2 resumed>) = 1

38522 **dup2**(7, 0 <unfinished ...>

38521 **execve**("./child", ["./child"], 0x7fff0a011c28 /\* 30 vars \*/ <unfinished ...>

38522 <... dup2 resumed>) = 0

38522 **dup2**(4, 1) = 1

38522 **execve**("./child", ["./child"], 0x7fff0a011c28 /\* 30 vars \*/) = 0

38521 <... execve resumed>) = 0

38522 **close**(5) = 0

38522 read(5, "\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

38521 **close**(6) = 0

38521 read(6, "\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

38522 **close**(5) = 0

38522 read(5, "\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

38521 **close**(6) = 0

38521 read(6, "\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

38522 **close**(5) = 0

38522 read(5, "\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

38521 **close**(6) = 0

38521 read(6, "\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

38522 **close**(5) = 0

38522 read(5, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832

38521 **close**(6) = 0

38521 read(6, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832

38522 **close**(5) = 0

38521 **close**(6) = 0

38522 read(0, <unfinished ...>

38521 read(0, <unfinished ...>

38485 <... read resumed>"0123456789\n", 1024) = 11

38485 **write**(6, "0123456789", 10) = 10

38521 <... read resumed>"0", 1) = 1

38485 **write**(6, "\0", 1 <unfinished ...>

38521 read(0, <unfinished ...>

38485 <... write resumed>) = 1

38521 <... read resumed>"1", 1) = 1

38485 read(0, <unfinished ...>

38521 read(0, "2", 1) = 1

38521 read(0, "3", 1) = 1

38521 read(0, "4", 1) = 1

38521 read(0, "5", 1) = 1

38521 read(0, "6", 1) = 1

38521 read(0, "7", 1) = 1

38521 read(0, "8", 1) = 1

38521 read(0, "9", 1) = 1

38521 read(0, "\0", 1) = 1

38521 **write**(1, "9876543210\n", 11) = 11

38521 read(0, <unfinished ...>

38485 <... read resumed>"0123456789a\n", 1024) = 12

38485 **write**(8, "0123456789a", 11) = 11

38522 <... read resumed>"0", 1) = 1

38485 **write**(8, "\0", 1 <unfinished ...>

38522 read(0, <unfinished ...>

38485 <... write resumed>) = 1

38522 <... read resumed>"1", 1) = 1

38485 read(0, <unfinished ...>

38522 read(0, "2", 1) = 1

38522 read(0, "3", 1) = 1

38522 read(0, "4", 1) = 1

38522 read(0, "5", 1) = 1

38522 read(0, "6", 1) = 1

38522 read(0, "7", 1) = 1

38522 read(0, "8", 1) = 1

38522 read(0, "9", 1) = 1

38522 read(0, "a", 1) = 1

38522 read(0, "\0", 1) = 1

38522 **write**(1, "a9876543210\n", 12) = 12

38522 read(0, <unfinished ...>

38485 <... read resumed>"\n", 1024) = 1

38485 **write**(6, "", 0) = 0

38485 **write**(6, "\0", 1) = 1

38521 <... read resumed>"\0", 1) = 1

38485 **write**(8, "", 0 <unfinished ...>

38521 exit\_group(0 <unfinished ...>

38485 <... write resumed>) = 0

38521 <... exit\_group resumed>) = ?

38485 **write**(8, "\0", 1) = 1

38522 <... read resumed>"\0", 1) = 1

38485 **close**(6 <unfinished ...>

38521 +++ exited with 0 +++

38485 <... close resumed>) = 0

38522 exit\_group(0 <unfinished ...>

38485 --- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=38521, si\_uid=0, si\_status=0, si\_utime=0, si\_stime=0} ---

38522 <... exit\_group resumed>) = ?

38485 **close**(8) = 0

38522 +++ exited with 0 +++

38485 --- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=38522, si\_uid=0, si\_status=0, si\_utime=0, si\_stime=0} ---

38485 exit\_group(0) = ?

38485 +++ exited with 0 +++

**Вывод**

Создание скрипта для управления процессами в C++ — это увлекательный опыт, позволяющий вам создавать дочерние процессы с помощью fork и управлять их выполнением параллельно с основным процессом. Использование функций из unistd.h позволяет устанавливать связи между процессами через каналы, созданные pipe. Также я научился отслеживать системные вызовы с помощью strace.