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: Гула Дмитрий Александрович

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.11.24

Москва, 2024

Постановка задачи

Вариант 7.

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<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- **`fork()`** — создаёт новый процесс, который является копией текущего (родительского) процесса. Возвращает PID дочернего процесса в родительском процессе и 0 в дочернем.
- **`pipe()`** — создаёт однонаправленный канал для обмена данными между процессами, возвращая два дескриптора: один для чтения, другой для записи.
- **`dup2()`** — дублирует дескриптор файла, перенаправляя один дескриптор на другой. Используется для перенаправления стандартного ввода/вывода, например, из файла или в канал.
- **`execl()`** — заменяет текущий процесс на новый, исполняя указанную программу. Если успешно выполняется, текущий код процесса завершается и выполняется новый код.
- **`open()`** — открывает файл и возвращает файловый дескриптор. Используется для открытия файла в определённом режиме (например, только для чтения).
- **`close()`** — закрывает дескриптор файла, освобождая ресурсы, связанные с файлом или каналом.
- **`waitpid()`** — приостанавливает выполнение родительского процесса до завершения дочернего, позволяя родителю дождаться завершения конкретного дочернего процесса и получить его статус.

В рамках лабораторной работы мы разработали две программы: **родительскую** и **дочернюю**, которые взаимодействуют между собой с использованием каналов и перенаправлением потоков. Целью программы было создать механизм, при котором дочерний процесс считывает команды из файла, производит вычисления и передает результаты родительскому процессу для отображения.

Родительский процесс

1. **Запрос имени файла**: Родительский процесс запрашивает у пользователя имя файла, в котором находятся команды для обработки.
2. **Создание канала**: Системный вызов `pipe()` используется для создания однонаправленного канала связи между процессами. Этот канал позволяет передавать данные от дочернего процесса к родительскому.
3. **Создание дочернего процесса**: Системный вызов `fork()` создаёт новый (дочерний) процесс. Родительский и дочерний процессы продолжают выполнять разные задачи.
4. **Перенаправление потоков для дочернего процесса**:
    - В дочернем процессе стандартный ввод перенаправляется из файла, выбранного пользователем, с помощью `dup2()`.
    - Стандартный вывод перенаправляется в канал, так что результаты вычислений дочернего процесса можно будет прочитать родителем.
5. **Запуск дочернего процесса**: Дочерний процесс заменяет себя с помощью `execl()`, чтобы начать выполнение другой программы, `child`.
6. **Чтение и вывод данных**: Родительский процесс считывает результаты из канала и выводит их на экран.

Дочерний процесс

1. **Чтение данных из файла**: После перенаправления стандартного ввода дочерний процесс считывает команды из файла.
2. **Обработка команд**: Дочерний процесс считывает числа, суммирует их и записывает результат в стандартный вывод (перенаправленный на канал).
3. **Передача результатов родительскому процессу**: Итоговая сумма отправляется через канал, после чего дочерний процесс завершает выполнение.

Принципы работы программы

1. **Перенаправление потоков**: Важно было перенаправить стандартный ввод и вывод дочернего процесса, чтобы он читал данные из файла, а результаты записывал в канал.
2. **Межпроцессное взаимодействие через каналы**: Использование канала `pipe()` позволило дочернему процессу передавать данные родительскому процессу. Это организовало одностороннюю связь, обеспечив безопасную передачу результатов вычислений.
3. **Использование `fork()` и `exec()` для создания и замены процессов**: `fork()` позволил создать параллельно выполняющийся дочерний процесс, а `execl()` заменил текущий код дочернего процесса новым, чтобы тот выполнял свою задачу.

Результат работы программы

1. Пользователь вводит имя файла с командами.
2. Родительский процесс создает канал и дочерний процесс.
3. Дочерний процесс читает числа из указанного файла, выполняет вычисления и передает результаты родителю.

4. Родительский процесс выводит результаты на экран.

Таким образом, программа продемонстрировала работу с системными вызовами для управления процессами, взаимодействие между ними и передачу данных через каналы. Это позволяет реализовать базовые функции IPC (межпроцессного взаимодействия) в UNIX-подобных системах.

Код программы

parent.cpp

```
#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <cstdlib>

#include <cstring>

#include <cstdio>



void create_pipe(int* pipe_fd) {

   if (pipe(pipe_fd) == -1) {

       perror("Pipe error!\n");

       exit(EXIT_FAILURE);

   }

}



int main() {

   char filename[256];



   write(STDOUT_FILENO, "Введите имя файла: ", 35);

   ssize_t bytesRead = read(STDIN_FILENO, filename, sizeof(filename) - 1);

   if (bytesRead > 0) {

       filename[bytesRead - 1] = '\0';

   } else {

       perror("Ошибка при вводе имени файла");
```

```
        exit(EXIT_FAILURE);
    }


    int pipe1_fd[2];
    create_pipe(pipe1_fd);


    pid_t pid = fork();
    if (pid == -1) {
        perror("Fork error!\n");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        close(pipe1_fd[0]);


        int file_fd = open(filename, O_RDONLY);
        if (file_fd == -1) {
            perror("File open error!\n");
            exit(EXIT_FAILURE);
        }


        dup2(file_fd, STDIN_FILENO);
        dup2(pipe1_fd[1], STDOUT_FILENO);


        close(file_fd);
        close(pipe1_fd[1]);


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


        perror("execl error!\n");
        exit(EXIT_FAILURE);
```

```
    } else {
        close(pipe1_fd[1]);

        char buffer[256];
        ssize_t bytesRead;
        while ((bytesRead = read(pipe1_fd[0], buffer, sizeof(buffer) - 1)) >
0) {
            buffer[bytesRead] = '\0';
            write(STDOUT_FILENO, buffer, bytesRead);
        }

        close(pipe1_fd[0]);

        int status;
        waitpid(pid, &status, 0);
    }

    return 0;
}
```

child.cpp

```
#include <iostream>
#include <unistd.h>
#include <sstream>
#include <string>
#include <cstring>

int main() {
    char buffer[256];
```

```
    ssize_t bytesRead;

    while ((bytesRead = read(STDIN_FILENO, buffer, sizeof(buffer) - 1)) > 0)
{
        buffer[bytesRead] = '\0';

        char* line = strtok(buffer, "\n");
        while (line) {
            std::istringstream iss(line);
            float number, sum = 0;

            while (iss >> number) {
                sum += number;
            }

            char output[256];
            int output_len = snprintf(output, sizeof(output), "Сумма:
%.2f\n", sum);
            write(STDOUT_FILENO, output, output_len);

            line = strtok(nullptr, "\n");
        }
    }

    return 0;
}
```

Протокол работы программы

Тестирование:

```
➜  src git:(main) ./parent
Введите имя файла: data.txt
Сумма: 60.9
Сумма: 13
Сумма: 11
Сумма: 151
Сумма: 36.5
Сумма: 9.5
➜  src git:(main) cat < data.txt
10.5 20.3 30.1
5.5 3.2 4.3
1.1 2.2 3.3 4.4
100.25 50.75
7.7 8.8 9.9 10.10
7.7 8.8 9.9 10.10 -1.5 -25.5
```

Dtruss:

```
dmitry@dmitry-HP-Laptop-15-da3xxx:~/C/Labs-OS/lab1/src$ strace ./parent
execve("./parent", ["./parent"], 0x7ffc53524f10 /* 73 vars */) = 0
brk(NULL)                               = 0x6211b68e0000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x762310d54000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=56103, ...}) = 0
mmap(NULL, 56103, PROT_READ, MAP_PRIVATE, 3, 0) = 0x762310d46000
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) = 0x762310a00000
mmap(0x762310a28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x762310a28000
mmap(0x762310bb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x762310bb0000
mmap(0x762310bff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
```

```
3, 0x1fe000) = 0x762310bff000
mmap(0x762310c05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x762310c05000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x762310d43000
arch_prctl(ARCH_SET_FS, 0x762310d43740) = 0
set_tid_address(0x762310d43a10)         = 9364
set_robust_list(0x762310d43a20, 24)     = 0
rseq(0x762310d44060, 0x20, 0, 0x53053053) = 0
mprotect(0x762310bff000, 16384, PROT_READ) = 0
mprotect(0x6211b6290000, 4096, PROT_READ) = 0
mprotect(0x762310d8c000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x762310d46000, 56103)           = 0
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 35Введите имя
файла: ) = 35
read(0, data.txt
"data.txt\n", 255)              = 9
pipe2([3, 4], 0)                        = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x762310d43a10) = 9489
close(4)                                = 0
read(3, "\320\241\321\203\320\274\320\274\320\260:
60.9\n\320\241\321\203\320\274\320\274\320\260: 13\n", 255) = 32
write(1, "\320\241\321\203\320\274\320\274\320\260:
60.9\n\320\241\321\203\320\274\320\274\320\260: 13\n", 32Сумма: 60.9
Сумма: 13
) = 32
read(3, "\320\241\321\203\320\274\320\274\320\260:
11\n\320\241\321\203\320\274\320\274\320\260: 151\n\320"..., 255) = 64
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=9489, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
write(1, "\320\241\321\203\320\274\320\274\320\260:
11\n\320\241\321\203\320\274\320\274\320\260: 151\n\320"..., 64Сумма: 11
Сумма: 151
Сумма: 36.5
Сумма: 9.5
) = 64
read(3, "", 255)                        = 0
close(3)                                = 0
wait4(9489, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 9489
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В результате лабораторной работы была реализована программа, где родительский и дочерний процессы взаимодействуют через канал: родитель запрашивает у пользователя файл с командами, дочерний процесс читает из него числа, вычисляет их сумму и передает результат родителю для вывода на экран. Программа продемонстрировала успешное применение системных вызовов для организации межпроцессного взаимодействия и перенаправления потоков ввода-вывода.

Проблемы возникли при корректном перенаправлении стандартного ввода и вывода дочернего процесса на файл и канал, а также при обработке ошибок, связанных с неверными входными данными. Было бы полезно глубже изучить механизм каналов и особенности работы с `dup2` для более сложных сценариев перенаправления потоков.