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: Стаценко В.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 7.11.24

Москва, 2024

Постановка задачи

Вариант 1.

Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. 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 pipefd[2]) - создает наименованный канал для передачи данных между процессами
- void exit(int status) - завершает выполнение процесса и возвращение статуса
- int dup2(int oldfd, int newfd) - переназначает файловый дескриптор
- int open(const char* pathname, int flags, mode_t mode) - открывает\создает файл
- int close(int fd) - закртывает файл
- int execv(const char *path, char* const argv[]) - заменяет текущий исполняемый файл на path
- int write(int fd, void* buffer, int count) - записывает данные в файл, связанный с файловым дескриптором
- int read(int fd, void* buffer, int count) - читает данные из файла, связанного с файловым дескриптором
- pid_t wait(int status) - ожидает завершение дочернего процесса

В данной лабораторной работе я написала программу, состоящую из двух процессов: родительского и дочернего, которые взаимодействуют друг с другом с помощью канала (pipe). Родительский процесс запрашивает ввод чисел и передает их дочернему процессу для обработки. Дочерний процесс читает данные из канала, вычисляет сумму введенных чисел каждой новой строки, пока не встретит end, и записывает результаты в указанный файл. Программа включает в себя обработку ошибок, таких как отсутствие аргументов командной строки и сбои при создании процессов и открытии файлов.

Код программы

parent.c:

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
    int pipe1[2];
    pid_t pid;
    char buffer[100];

    if (argc < 2) {
        char error_msg[100];
        snprintf(error_msg, sizeof(error_msg), "Необходимо указать имя файла в качестве аргумента.\n");
        write(STDERR_FILENO, error_msg, strlen(error_msg));
        exit(EXIT_FAILURE);
    }

    if (pipe(pipe1) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    pid = fork();

    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);

    } else if (pid == 0) {
        close(pipe1[1]);
        dup2(pipe1[0], STDIN_FILENO);
        close(pipe1[0]);
        char *args[] = {"./child", argv[1], NULL};
        execv(args[0], args);
        perror("execv");
        exit(EXIT_FAILURE);
```

```c
    } else {
        close(pipe1[0]);
        while (1) {
            write(STDOUT_FILENO, "Введите числа (или end для завершения): ", strlen("Введите числа (или end для завершения): "));
            read(STDIN_FILENO, buffer, sizeof(buffer));
            buffer[strcspn(buffer, "\n")] = 0;

            if (strcmp(buffer, "end") == 0) {
                break;
            }

            write(pipe1[1], buffer, strlen(buffer));
            write(pipe1[1], "\n", 1);
        }

        close(pipe1[1]);
        wait(NULL);
    }

    return 0;
}
```

child.c:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
    char buffer[4096];
    char *token;
    int sum;

    if (argc < 2) {
        char error_msg[128];
        snprintf(error_msg, sizeof(error_msg), "Необходимо указать имя файла в качестве аргумента.\n");
        write(STDERR_FILENO, error_msg, strlen(error_msg));
        exit(EXIT_FAILURE);
    }

    char *filename = argv[1];

    while (fgets(buffer, sizeof(buffer), stdin) != NULL) {
        buffer[strcspn(buffer, "\n")] = 0;
        sum = 0;
        token = strtok(buffer, " ");
        while (token != NULL) {
            sum += atoi(token);
            token = strtok(NULL, " ");
        }

        int fd = open(filename, O_WRONLY | O_CREAT | O_APPEND, 0644);
        if (fd == -1) {
            perror("open");
            exit(1);
        }
        char sum_str[20];
        snprintf(sum_str, sizeof(sum_str), "%d\n", sum);
        write(fd, sum_str, strlen(sum_str));
        close(fd);
    }
    return 0;
}
```

Протокол работы программы

Тестирование

```
victoria@victoria:~/laba/os/OSlabs/laba1$ gcc parent.c -o parent
victoria@victoria:~/laba/os/OSlabs/laba1$ gcc child.c -o child
victoria@victoria:~/laba/os/OSlabs/laba1$ ./parent output.txt
Введите числа (или end для завершения): 12 34 67 1
Введите числа (или end для завершения): 47983 8 29
Введите числа (или end для завершения): 1 2 3 4 5 6 7 8 9
Введите числа (или end для завершения): end
```

os > OSlabs > laba1 > output.txt

```
114
48020
45

```

Strace

```
$ strace ./parent output.txt
execve("./parent", ["./parent", "output.txt"], 0x7ffe4c2bf3d8 /* 36 vars */) = 0
brk(NULL)                               = 0x55d94613a000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc6e41e5b0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f4a16662000
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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=17839, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 17839, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f4a1665d000
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\0P\237\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
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"...,
68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f4a16434000
mprotect(0x7f4a1645c000, 2023424, PROT_NONE) = 0
mmap(0x7f4a1645c000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f4a1645c000
mmap(0x7f4a165f1000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f4a165f1000
mmap(0x7f4a1664a000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f4a1664a000
mmap(0x7f4a16650000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f4a16650000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f4a16431000
arch_prctl(ARCH_SET_FS, 0x7f4a16431740) = 0
set_tid_address(0x7f4a16431a10)         = 673526
set_robust_list(0x7f4a16431a20, 24)     = 0
rseq(0x7f4a164320e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f4a1664a000, 16384, PROT_READ) = 0
mprotect(0x55d944218000, 4096, PROT_READ) = 0
mprotect(0x7f4a1669c000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
munmap(0x7f4a1665d000, 17839)           = 0
pipe2([3, 4], 0)                        = 0
```

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f4a16431a10) = 673527
close(3) = 0
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\207\320\270\321\201\320\273\320\260 (\320\270\320\273\320"..., 68Введите числа (или end
для завершения):) = 68
read(0, 2 3 4
"2 3 4\n", 100) **= 6**
write(4, "2 3 4", 5) **= 5**
write(4, "\n", 1) **= 1**
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\207\320\270\321\201\320\273\320\260 (\320\270\320\273\320"..., 68Введите числа (или end
для завершения):) = 68
read(0, 5 5 5
"5 5 5\n", 100) **= 6**
write(4, "5 5 5", 5) **= 5**
write(4, "\n", 1) **= 1**
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\207\320\270\321\201\320\273\320\260 (\320\270\320\273\320"..., 68Введите числа (или end
для завершения):) = 68
read(0, end
"end\n", 100) **= 4**
close(4) **= 0**
wait4(-1, NULL, 0, NULL) **= 673527**
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=673527, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
exit_group(0) = ?
+++ exited with 0 +++

Вывод

В ходе лабораторной работы я узнала о некоторых системных вызовах и научилась их использовать. Впервые мне пришлось создавать каналы, чтобы с их помощью обменивать данные между процессами. Сложность возникла в том, что было слишком много новой информации, и ушло долгое время на их понимание.