

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: Болдинова В.В.
Преподаватель: Бахарев В.Д.
Оценка: _____
Дата: 01.01.25

Москва, 2024

Постановка задачи

Вариант 2.

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 float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `read(int fd, void *buf, size_t count)` - читает данные из файла или стандартного ввода;
- `strtok(char *str, const char *delim)` - разбивает строку на токены (части), используя разделители;
- `strcspn(const char *str1, const char *str2)` - вычисляет индекс первого символа из `str1`, который есть в `str2`;
- `strtof(const char *nptr, char **endptr)` - преобразует строку в число с плавающей точкой (float);
- `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 создаст файл
- `int close(int fd)` - закрывает файл
- `int exece(const char *file, char* const arg....)` - заменит образ текущего процесса на образ нового процесса `file`
- `int write(int pipe, char* buffer, int size)` - записывает данные в файл, связанный с файловым дескриптором
- `int fgets(char* buffer, int size, stdin)` - читает данные из файла(stdin), связанного с файловым дескриптором
- `pid_t wait(int status)` - ожидает завершения дочернего процесса
- `snprintf(char *str, size_t size, const char *format, ...)` - форматирует строку с заданными параметрами и записывает её в буфер.

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:

```
#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) { // Дочерний процесс
        close(pipe1[1]); // Закрываем сторону для записи
        dup2(pipe1[0], STDIN_FILENO); // Дублируем pipe в стандартный ввод
        close(pipe1[0]); // Закрываем исходный дескриптор

        char *args[] = {"/mnt/d/si/OSI/Lab1/L1/child", argv[1], NULL};
        execv(args[0], args);
        perror("execv"); // Если execv не сработал
        exit(EXIT_FAILURE);
    } else { // Родительский процесс
        close(pipe1[0]); // Закрываем сторону для чтения
        while (1) {
            const char newline[] = "Введите числа (или end для завершения):
";
            write(STDOUT_FILENO, newline, strlen(newline));
            read(STDIN_FILENO, buffer, sizeof(buffer));
            buffer[strcspn(buffer, "\n")] = 0; // Убираем символ новой
строки

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

            write(pipe1[1], buffer, strlen(buffer)); // Отправляем данные в
pipe
            write(pipe1[1], "\n", 1); // Разделяем числа новой строкой
        }

        close(pipe1[1]); // Закрываем сторону для записи
    }
}
```

```

        wait(NULL); // Ожидаем завершения дочернего процесса
    }

    return 0;
}

```

Child.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;
    float 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; // Убираем символ новой строки в
конце строки
        sum = 0.0f;
        token = strtok(buffer, " ");
        while (token != NULL) {
            sum += atof(token); // Преобразуем строку в float
            token = strtok(NULL, " ");
        }

        int fd = open(filename, O_WRONLY | O_CREAT | O_APPEND, 0644);
        if (fd == -1) {
            perror("open");
            exit(1);
        }

        char sum_str[50]; // Увеличим размер буфера для записи float
        snprintf(sum_str, sizeof(sum_str), "%.2f\n", sum); // Форматируем
как float
        write(fd, sum_str, strlen(sum_str));
        close(fd);
    }

    return 0;
}

```

Протокол работы программы

Тестирование:

PS D:\si\OSI\Lab1\L1> .\parent.exe output.txt

Enter numbers (or 'end' to finish): 1.0

Enter numbers (or 'end' to finish): 1.5 1.5 12.0

Enter numbers (or 'end' to finish): 1.0 1.2 1.3 5.5

Enter numbers (or 'end' to finish): end

1	1.00
2	15.00
3	9.00
4	

Strace:

```
execve("/mnt/d/si/OSI/Lab1/L1/parent", ["/mnt/d/si/OSI/Lab1/L1/parent",  
"/mnt/d/si/OSI/Lab1/L1/output.txt"], 0x7ffc0aca5258 /* 26 vars */) = 0  
brk(NULL) = 0x564afeb5b000  
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,  
0) = 0x7ff9324cd000  
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=20115, ...}) = 0  
mmap(NULL, 20115, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff9324c8000  
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\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"..., 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"..., 784, 64) = 784  
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =  
0x7ff9322b6000  
mmap(0x7ff9322de000, 1605632, PROT_READ|PROT_EXEC,  
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7ff9322de000  
mmap(0x7ff932466000, 323584, PROT_READ,  
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7ff932466000  
mmap(0x7ff9324b5000, 24576, PROT_READ|PROT_WRITE,  
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7ff9324b5000  
mmap(0x7ff9324bb000, 52624, PROT_READ|PROT_WRITE,  
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ff9324bb000  
close(3) = 0  
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -  
1, 0) = 0x7ff9322b3000  
arch_prctl(ARCH_SET_FS, 0x7ff9322b3740) = 0  
set_tid_address(0x7ff9322b3a10) = 59117  
set_robust_list(0x7ff9322b3a20, 24) = 0
```

```

rseq(0x7ff9322b4060, 0x20, 0, 0x53053053) = 0
mprotect(0x7ff9324b5000, 16384, PROT_READ) = 0
mprotect(0x564acdacd000, 4096, PROT_READ) = 0
mprotect(0x7ff932505000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
munmap(0x7ff9324c8000, 20115) = 0
pipe2([3, 4], 0) = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7ff9322b3a10) = 59118
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"..., 68) = 68
read(0, "1\n", 100) = 2
write(4, "1", 1) = 1
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"..., 68) = 68
read(0, "1.2\n", 100) = 4
write(4, "1.2", 3) = 3
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"..., 68) = 68
read(0, "1.3 1.3\n", 100) = 8
write(4, "1.3 1.3", 7) = 7
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"..., 68) = 68
read(0, "1/8\n", 100) = 4
write(4, "1/8", 3) = 3
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"..., 68) = 68
read(0, "1.0\n", 100) = 4
write(4, "1.0", 3) = 3
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"..., 68) = 68
read(0, "end\n", 100) = 4
close(4) = 0
wait4(-1, NULL, 0, NULL) = 59118
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=59118, si_uid=0,
si_status=0, si_utime=0, si_stime=1 /* 0.01 s */} ---
exit_group(0) = ?
+++ exited with 0 +++

```

Вывод

В ходе лабораторной работы я узнала о некоторых системных вызовах и научилась их использовать. Впервые в ходе работы мне пришлось создавать каналы, чтобы с их помощью обменивать данные между процессами. Сложность возникла в том, что было слишком много новой информации, и некоторая сложная для понимания, из-за чего ушло много времени.