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-211Б-23

Студент: Соболин Т.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 12.01.25

Москва, 2025

Постановка задачи

Вариант 2.

Пользователь вводит команды вида: «число число число<endline>». 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:

- fork() — создает новый процесс. ○ pipe() — создает анонимный канал для межпроцессного взаимодействия. ○ read() — читает данные из файлового дескриптора. ○ write() — записывает данные в файловый дескриптор.
- strtok() — разбивает строку на лексемы. ○ atof() — преобразует строку в число с плавающей точкой.
- fopen() — открывает файл для чтения или записи. ○ fprintf() — записывает форматированные данные в файл. ○ fclose() — закрывает открытый файл.
- perror() — выводит сообщение об ошибке. ○ strcpy() — копирует строку в буфер. ○ strcmp() — сравнивает две строки. ○ strlen() — возвращает длину строки. ○ strcspn() — возвращает длину части строки до первого появления любого символа из заданного набора.

Программа создает два процесса: родительский и дочерний. Родительский процесс принимает от пользователя числа, разделенные пробелами, и отправляет их дочернему процессу через анонимный канал (пайп). Дочерний процесс получает эти числа, суммирует их и записывает результат в конец указанного файла. Если пользователь вводит команду "exit", программа завершает работу.

Код программы

parent.c

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

#define BUFFER_SIZE 1024

int main(int argc, char *argv[]) {
    int pipe1[2]; // pipe для передачи данных от родителя к дочернему     pid_t pid;

    if (argc != 2) {
        fprintf(stderr, "Использование: %s имя_файла\n", argv[0]);         exit(EXIT_FAILURE);
    }

    // Создаем pipes
```

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

    // Создаем дочерний процесс
pid = fork();      if (pid < 0)
{          perror("fork");
exit(EXIT_FAILURE);
    }
    if (pid == 0) { // Дочерний процесс
        close(pipe1[1]); // Закрываем запись в pipe1

        // Передаем имя файла дочернему процессу
char filename[256];          strcpy(filename,
argv[1]);

        // Читаем данные от родителя
char buffer[BUFFER_SIZE];          while (1) {
read(pipe1[0], buffer, BUFFER_SIZE);
if (strcmp(buffer, "exit") == 0) {
                break; // Выход при получении команды exit
}

            // Обработка чисел
float sum = 0.0;
            char *token = strtok(buffer, " ");
while (token != NULL) {                  sum +=
atof(token);                  token =
strtok(NULL, " ");
            }

            // Записываем результат в файл
FILE *file = fopen(filename, "a");              if
(file != NULL) {                  fprintf(file,
"Сумма: %.2f\n", sum);                  fclose(file);
            } else {
                perror("fopen");
            }
        }

        close(pipe1[0]);
exit(EXIT_SUCCESS);
    } else { // Родительский процесс
        close(pipe1[0]); // Закрываем чтение из pipe1

        char input[BUFFER_SIZE];
while (1) {
            printf("Введите числа (или 'exit' для выхода): ");
fgets(input, BUFFER_SIZE, stdin);              input[strcspn(input,
"\n")] = 0;

            // Отправляем данные дочернему процессу
```

```
            write(pipe1[1], input, strlen(input) + 1);

            if (strcmp(input, "exit") == 0) {
break;
            }
        }
         close(pipe1[1]);
wait(NULL);
exit(EXIT_SUCCESS);
    }
}
```

Child.c

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

#define BUFFER_SIZE 1024

int main() {     int pipe1[2]; // pipe для передачи данных от
родителя к дочернему
    // Читаем имя файла от аргумента (это делается в родительском процессе)
char filename[256];

    // Читаем данные от родителя     char
buffer[BUFFER_SIZE];     while (1) {
read(pipe1[0], buffer, BUFFER_SIZE);
if (strcmp(buffer, "exit") == 0) {
            break; // Выход при получении команды exit
}

        // Обработка чисел
float sum = 0.0;
        char *token = strtok(buffer, " ");
while (token != NULL) {             sum +=
atof(token);             token =
strtok(NULL, " ");
        }

        // Записываем результат в файл
FILE *file = fopen(filename, "a");         if
(file != NULL) {             fprintf(file,
"Сумма: %.2f\n", sum);             fclose(file);
        } else {
            perror("fopen");
        }
    }
close(pipe1[0]);
exit(EXIT_SUCCESS);



}
```

Протокол работы программы

Тестирование:

```
kotlasboy@kotlasboy-Modern-15-B12M:~/Programming/Projects/OS/lab_1$ ./a.out output.txt
Введите числа (или 'exit' для выхода): 10.2 Введите
числа (или 'exit' для выхода): 10.2 10.2  Введите
числа (или 'exit' для выхода):
Введите числа (или 'exit' для выхода): exit
```

Strace:

kotlasboy@kotlasboy-Modern-15-B12M:~/Programming/Projects/OS/lab_1$ strace -f ./a.out output.txt

execve("./a.out", ["./a.out", "output.txt"], 0x7ffd6c64d8b0 /* 60 vars */) = 0

brk(NULL) = 0x613773c3e000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x745ea0868000

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=68847, ...}) = 0

mmap(NULL, 68847, PROT_READ, MAP_PRIVATE, 3, 0) = 0x745ea0857000

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) = 0x745ea0600000

mmap(0x745ea0628000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x745ea0628000

mmap(0x745ea07b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x745ea07b0000

mmap(0x745ea07ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x745ea07ff000

mmap(0x745ea0805000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x745ea0805000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x745ea0854000

arch_prctl(ARCH_SET_FS, 0x745ea0854740) = 0

set_tid_address(0x745ea0854a10) = 62842

set_robust_list(0x745ea0854a20, 24) = 0

rseq(0x745ea0855060, 0x20, 0, 0x53053053) = 0

```
mprotect(0x745ea07ff000, 16384, PROT_READ) = 0
mprotect(0x613771e19000, 4096, PROT_READ) = 0
mprotect(0x745ea08a0000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x745ea0857000, 68847)           = 0
pipe2([3, 4], 0)                        = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 62843
attached
, child_tidptr=0x745ea0854a10) = 62843
[pid 62843] set_robust_list(0x745ea0854a20, 24 <unfinished ...>
[pid 62842] close(3 <unfinished ...>
[pid 62843] <... set_robust_list resumed>) = 0
[pid 62842] <... close resumed>)        = 0
[pid 62842] 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"..., 63Введите числа (или 'exit'
для выхода): ) = 63
[pid 62843] close(4 <unfinished ...>
[pid 62842] read(0,  <unfinished ...>
[pid 62843] <... close resumed>)        = 0
[pid 62843] read(3, 10.2
 <unfinished ...>
[pid 62842] <... read resumed>"10.2\n", 1024) = 5
[pid 62842] write(4, "10.2\0", 5)       = 5
[pid 62843] <... read resumed>"10.2\0", 1024) = 5
[pid 62842] 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"..., 63Введите числа (или 'exit'
для выхода): ) = 63
[pid 62843] openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_APPEND, 0666
<unfinished ...>
[pid 62842] read(0,  <unfinished ...>
[pid 62843] <... openat resumed>)       = 4
[pid 62843] write(4, "10.19\n", 6)      = 6
[pid 62843] close(4)                    = 0
[pid 62843] read(3, 10.2 10.2
 <unfinished ...>
[pid 62842] <... read resumed>"10.2 10.2\n", 1024) = 10
[pid 62842] write(4, "10.2 10.2\0", 10) = 10
[pid 62843] <... read resumed>"10.2 10.2\0", 1024) = 10
[pid 62842] 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"..., 63 <unfinished ...>
Введите числа (или 'exit' для выхода): [pid 62843] openat(AT_FDCWD, "output.txt",
O_WRONLY|O_CREAT|O_APPEND, 0666 <unfinished ...>
```

```
[pid 62842] <... write resumed>)        = 63
[pid 62842] read(0,  <unfinished ...>
[pid 62843] <... openat resumed>)       = 4
[pid 62843] write(4, "20.39\n", 6)      = 6
[pid 62843] close(4)                    = 0
[pid 62843] read(3,
 <unfinished ...>
[pid 62842] <... read resumed>"\n", 1024) = 1
[pid 62842] write(4, "\0", 1)           = 1
[pid 62843] <... read resumed>"\0", 1024) = 1
[pid 62842] 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"..., 63 <unfinished ...>
[pid 62843] openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_APPEND, 0666Введите
числа (или 'exit' для выхода):  <unfinished ...>
[pid 62842] <... write resumed>)        = 63
[pid 62843] <... openat resumed>)       = 4
[pid 62842] read(0,  <unfinished ...>
[pid 62843] write(4, "0.00\n", 5)       = 5
[pid 62843] close(4)                    = 0
[pid 62843] read(3, exit
 <unfinished ...>
[pid 62842] <... read resumed>"exit\n", 1024) = 5
[pid 62842] write(4, "exit\0", 5)       = 5
[pid 62843] <... read resumed>"exit\0", 1024) = 5
[pid 62842] close(4 <unfinished ...>
[pid 62843] close(3 <unfinished ...>
[pid 62842] <... close resumed>)        = 0
[pid 62843] <... close resumed>)        = 0
[pid 62842] wait4(-1,  <unfinished ...>
[pid 62843] exit_group(0)               = ?
[pid 62843] +++ exited with 0 +++
<... wait4 resumed>NULL, 0, NULL)       = 62843
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=62843, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Программа, использующая общую память для взаимодействия между родительским и дочерним процессами, демонстрирует эффективный способ передачи данных в многопоточных или многопроцессорных системах. Родительский процесс считывает ввод пользователя и записывает его в сегмент общей памяти, в то время как дочерний процесс обрабатывает эти данные и записывает результаты в файл. Использование общей памяти позволяет избежать накладных расходов на межпроцессное взаимодействие, обеспечивая более быструю и эффективную передачу информации. В конечном итоге, программа иллюстрирует важность управления памятью и синхронизации процессов для достижения оптимальной производительности.