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-213Б-23

Студент: Петров М.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.12.24

Москва, 2024

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:

```
markvolkov@MacBook-Air-Mark-2 LAB1 % ./parent 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:

strace -f ./p output.txt
execve("./p", ["./p", "output.txt"], 0x7ffcfcec30f0 /* 46 vars */) = 0
brk(NULL) = 0x62ed785af000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe9a18f780) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73b8c3f16000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58047, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x73b8c3f07000
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"..., 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) = 0x73b8c3c00000
mprotect(0x73b8c3c28000, 2023424, PROT_NONE) = 0
mmap(0x73b8c3c28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x73b8c3c28000
mmap(0x73b8c3dbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x73b8c3dbd000
mmap(0x73b8c3e16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x73b8c3e16000
mmap(0x73b8c3e1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x73b8c3e1c000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73b8c3f04000
arch_prctl(ARCH_SET_FS, 0x73b8c3f04740) = 0
set_tid_address(0x73b8c3f04a10) = 3840
set_robust_list(0x73b8c3f04a20, 24) = 0
rseq(0x73b8c3f050e0, 0x20, 0, 0x53053053) = 0
mprotect(0x73b8c3e16000, 16384, PROT_READ) = 0

```
mprotect(0x62ed5d11c000, 4096, PROT_READ) = 0
mprotect(0x73b8c3f50000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x73b8c3f07000, 58047)           = 0
pipe2([3, 4], 0)                        = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 3841 attached
, child_tidptr=0x73b8c3f04a10) = 3841
[pid  3841] set_robust_list(0x73b8c3f04a20, 24 <unfinished ...>
[pid  3840] close(3)                    = 0
[pid  3840] newfstatat(1, "",  <unfinished ...>
[pid  3841] <... set_robust_list resumed>) = 0
[pid  3841] close(4 <unfinished ...>
[pid  3840] <... newfstatat resumed>{st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
[pid  3840] getrandom( <unfinished ...>
[pid  3841] <... close resumed>)        = 0
[pid  3840] <... getrandom resumed>"\x77\xec\x85\x35\x86\x81\x46\x43", 8, GRND_NONBLOCK) = 8
[pid  3841] read(3,  <unfinished ...>
[pid  3840] brk(NULL)                   = 0x62ed785af000
[pid  3840] brk(0x62ed785d0000)         = 0x62ed785d0000
[pid  3840] newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
[pid  3840] 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  3840] read(0, 3121
"3121\n", 1024)     = 5
[pid  3840] write(4, "3121\0", 5)       = 5
[pid  3841] <... read resumed>"3121\0", 1024) = 5
[pid  3840] 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  3841] getrandom(Введите числа (или 'exit' для выхода):  <unfinished ...>
[pid  3840] <... write resumed>)        = 63
[pid  3840] read(0,  <unfinished ...>
[pid  3841] <... getrandom resumed>"\x9b\x4e\x82\xdd\x9f\x96\x3b\xcb", 8, GRND_NONBLOCK) = 8
[pid  3841] brk(NULL)                   = 0x62ed785af000
[pid  3841] brk(0x62ed785d0000)         = 0x62ed785d0000
[pid  3841] openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_APPEND, 0666) = 4
[pid  3841] lseek(4, 0, SEEK_END)       = 169
[pid  3841] newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=169, ...}, AT_EMPTY_PATH) = 0
[pid  3841] write(4, "\320\241\321\203\320\274\320\274\320\260: 3121.00\n", 20) = 20
[pid  3841] close(4)                    = 0
[pid  3841] read(3, exit
 <unfinished ...>
[pid  3840] <... read resumed>"exit\n", 1024) = 5
[pid  3840] write(4, "exit\0", 5)       = 5
[pid  3841] <... read resumed>"exit\0", 1024) = 5
```

```
[pid  3840] close(4)                = 0
[pid  3841] close(3 <unfinished ...>
[pid  3840] wait4(-1,  <unfinished ...>
[pid  3841] <... close resumed>)    = 0
[pid  3841] exit_group(0)           = ?
[pid  3841] +++ exited with 0 +++
<... wait4 resumed>NULL, 0, NULL)   = 3841
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3841, 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.