Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра №806 «Вычислительная математика и программирование»

Лабораторная работа №3 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:

- write(int fd, const void *buf, size_t count) - записывает данные в файл или в стандартный поток вывода, используя файловый дескриптор;
- open(const char *pathname, int flags, mode_t mode) - открывает файл с заданными правами доступа (для записи в файл);
- close(int fd) - закрывает файловый дескриптор, который ранее был открыт;
- 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);
- snprintf(char *str, size_t size, const char *format, ...) - форматирует строку с заданными параметрами и записывает её в буфер.
- shm_open - Открывает или создает объект общей памяти.
- ftruncate - Изменяет размер объекта общей памяти.
- Mmap - Отображает объект общей памяти в адресное пространство процесса.
- Munmap - Удаляет отображение объекта общей памяти из адресного пространства процесса
- shm_unlink- Удаляет объект общей памяти из файловой системы.
- sem_open - Создает или открывает именованный семафор
- sem_wait - Уменьшает значение семафора (захват). Если значение 0, блокирует процесс до освобождения.
- sem_post - Увеличивает значение семафора (освобождение). Разблокирует ожидающие процессы.
- sem_close - Закрывает дескриптор семафора в текущем процессе.
- sem_unlink - Закрывает дескриптор семафора в текущем процессе.

В данной лабораторной работе я написала программу, которая демонстрирует взаимодействие между родительским и дочерним процессами с использованием разделяемой памяти и семафоров. Родительский процесс создает область разделяемой памяти, запрашивает у пользователя ввод чисел, сохраняет их в эту память и передает управление дочернему процессу через семафор. Дочерний процесс считывает числа из разделяемой памяти, вычисляет их сумму и записывает результат в указанный файл. После

завершения работы дочернего процесса родительский процесс освобождает ресурсы, включая разделяемую память и семафор.

Код программы

Parent.c:

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

#define SHM_NAME "/my_shared_memory"
#define SEM_WRITE "/sem_write"
#define SEM_READ "/sem_read"
#define BUFFER_SIZE 100

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Необходимо указать имя файла в качестве
аргумента.\n");
        exit(EXIT_FAILURE);
    }

    // Удаляем старые ресурсы, если они существуют
    shm_unlink(SHM_NAME);
    sem_unlink(SEM_WRITE);
    sem_unlink(SEM_READ);

    // Создаем и настраиваем общую память
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0644);
    if (shm_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }
    if (ftruncate(shm_fd, BUFFER_SIZE) == -1) {
        perror("ftruncate");
        shm_unlink(SHM_NAME);
        exit(EXIT_FAILURE);
    }
    char *shared_memory = mmap(NULL, BUFFER_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shared_memory == MAP_FAILED) {
        perror("mmap");
        shm_unlink(SHM_NAME);
        exit(EXIT_FAILURE);
    }

    // Создаем семафоры
    sem_t *sem_write = sem_open(SEM_WRITE, O_CREAT | O_EXCL, 0644, 1);
    sem_t *sem_read = sem_open(SEM_READ, O_CREAT | O_EXCL, 0644, 0);
    if (sem_write == SEM_FAILED || sem_read == SEM_FAILED) {
        perror("sem_open");
        shm_unlink(SHM_NAME);
        sem_unlink(SEM_WRITE);
        sem_unlink(SEM_READ);
```

```
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid == -1) {
        perror("fork");
        munmap(shared_memory, BUFFER_SIZE);
        shm_unlink(SHM_NAME);
        sem_unlink(SEM_WRITE);
        sem_unlink(SEM_READ);
        exit(EXIT_FAILURE);
    } else if (pid == 0) { // Дочерний процесс
        char buffer[BUFFER_SIZE];
        while (1) {
            // Ожидание разрешения на чтение от родителя
            sem_wait(sem_read);

            // Проверяем, есть ли "end"
            if (strcmp(shared_memory, "end") == 0) {
                break;
            }

            // Чтение из общей памяти
            strncpy(buffer, shared_memory, BUFFER_SIZE);
            buffer[BUFFER_SIZE - 1] = '\0';

            // Вывод для проверки
            printf("[Child] Прочитано из общей памяти: %s\n", buffer);

            // Разрешаем родителю записывать
            sem_post(sem_write);
        }

        // Завершаем работу
        munmap(shared_memory, BUFFER_SIZE);
        sem_close(sem_write);
        sem_close(sem_read);
        exit(EXIT_SUCCESS);
    } else { // Родительский процесс
        char input[BUFFER_SIZE];
        while (1) {
            printf("Введите числа (или end для завершения): ");
            fgets(input, BUFFER_SIZE, stdin);
            input[strcspn(input, "\n")] = '\0'; // Убираем символ новой
строки

            // Ожидание разрешения на запись
            sem_wait(sem_write);

            // Пишем в общую память
            strncpy(shared_memory, input, BUFFER_SIZE);

            // Разрешаем дочернему процессу читать
            sem_post(sem_read);

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

        // Ожидаем завершения дочернего процесса
        wait(NULL);

        // Освобождаем ресурсы
```

```
        munmap(shared_memory, BUFFER_SIZE);
        shm_unlink(SHM_NAME);
        sem_close(sem_write);
        sem_close(sem_read);
        sem_unlink(SEM_WRITE);
        sem_unlink(SEM_READ);
    }

    return 0;
}
```

Child.c:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#define SEM_NAME "/my_semaphore"
#define SHM_NAME "/my_shared_memory"

int main(int argc, char *argv[]) {
    char buffer[4096];
    char *token;
    float sum = 0.0f;

    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];

    // Открываем семафор
    sem_t *sem = sem_open(SEM_NAME, 0);
    if (sem == SEM_FAILED) {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }

    // Открываем общую память
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0644);
    if (shm_fd == -1) {
        perror("shm_open");
        sem_close(sem);
        exit(EXIT_FAILURE);
    }

    int *shared_memory = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shared_memory == MAP_FAILED) {
        perror("mmap");
        close(shm_fd);
        sem_close(sem);
        exit(EXIT_FAILURE);
    }

    while (fgets(buffer, sizeof(buffer), stdin) != NULL) {
```

```
        buffer[strcspn(buffer, "\n")] = 0; // Убираем символ новой строки
        sum = 0.0f;
        token = strtok(buffer, " ");
        while (token != NULL) {
            sum += atof(token);
            token = strtok(NULL, " ");
        }

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

        char sum_str[50];
        snprintf(sum_str, sizeof(sum_str), "%.2f\n", sum);
        write(fd, sum_str, strlen(sum_str));
        close(fd);

        // Обновляем сумму в общей памяти
        sem_wait(sem);
        *shared_memory += (int)sum; // Сохраняем целую часть суммы
        sem_post(sem);
    }

    sem_close(sem);
    munmap(shared_memory, sizeof(int));
    close(shm_fd);

    return 0;
}
```

Протокол работы программы

Тестирование:

```
root@LAPTOP-6G05B5VT:~# strace -o /mnt/d/si/OSI/Lab3/
/Lab3/L3/parent /mnt/d/si/OSI/Lab3/L3/output.txt
Введите числа (или end для завершения): 1.1
Введите числа (или end для завершения): 1.1 1.1 1.1
Введите числа (или end для завершения): end
```

```
1.10
3.30
```

Strace:

execve("/mnt/d/si/OSI/Lab3/L3/parent", ["/mnt/d/si/OSI/Lab3/L3/parent", "/mnt/d/si/OSI/Lab3/L3/output.txt"], 0x7ffc30bd69f8 /* 26 vars */) = 0
brk(NULL) = 0x56306ea29000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd475dd9000
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) = 0x7fd475dd4000
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) =
0x7fd475bc2000
mmap(0x7fd475bea000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fd475bea000
mmap(0x7fd475d72000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7fd475d72000
mmap(0x7fd475dc1000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7fd475dc1000
mmap(0x7fd475dc7000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd475dc7000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
1, 0) = 0x7fd475bbf000
arch_prctl(ARCH_SET_FS, 0x7fd475bbf740) = 0
set_tid_address(0x7fd475bbfa10)         = 31896
set_robust_list(0x7fd475bbfa20, 24)     = 0
rseq(0x7fd475bc0060, 0x20, 0, 0x53053053) = 0
mprotect(0x7fd475dc1000, 16384, PROT_READ) = 0
mprotect(0x563066a50000, 4096, PROT_READ) = 0
mprotect(0x7fd475e11000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
munmap(0x7fd475dd4000, 20115)           = 0
unlink("/dev/shm/sem.my_semaphore")     = -1 ENOENT (No such file or directory)
unlink("/dev/shm/my_shared_memory")     = -1 ENOENT (No such file or directory)
getrandom("\x26\x9f\x9f\x75\x8d\x83\xb0\xe7", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.ufBeVh", 0x7ffe3d9db750,
AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.ufBeVh",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0644) = 5
write(5, "\1\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7fd475dd8000
link("/dev/shm/sem.ufBeVh", "/dev/shm/sem.my_semaphore") = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
getrandom("\x64\x18\x71\x1e\x61\xd3\x07\x38", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x56306ea29000
brk(0x56306ea4a000)                     = 0x56306ea4a000
unlink("/dev/shm/sem.ufBeVh")           = 0
close(5)                                = 0
```

openat(AT_FDCWD, "/dev/shm/my_shared_memory",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0644) = 5
ftruncate(5, 4) = 0
mmap(NULL, 4, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7fd475dd7000
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fd475bbfa10) = 31897
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\321.1\n", 100) = 5
write(4, "1\321.1", 4) = 4
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, "1.1 1.1 1.1\n", 100) = 12
write(4, "1.1 1.1 1.1", 11) = 11
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
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=31897, si_uid=0,
si_status=0, si_utime=0, si_stime=0} ---
wait4(-1, NULL, 0, NULL) = 31897
munmap(0x7fd475dd8000, 32) = 0
unlink("/dev/shm/sem.my_semaphore") = 0
munmap(0x7fd475dd7000, 4) = 0
close(5) = 0
unlink("/dev/shm/my_shared_memory") = 0
exit_group(0) = ?
+++ exited with 0 +++

Вывод

В процессе выполнения лабораторной работы я углубила свои знания о механизмах межпроцессного взаимодействия (IPC) в Unix-системах, таких как разделяемая память и семафоры. Я научилась использовать ключевые функции, такие как shmget, shmat, shmdt, sem_open, sem_wait, sem_post, а также овладела методами эффективного управления системными ресурсами (памятью и семафорами) для предотвращения утечек и сбоев. Этот опыт позволил мне глубже понять, как организовать синхронизацию процессов и обмен данными, обеспечивая при этом безопасность и стабильность работы програм.