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:

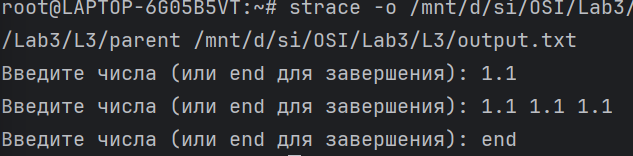
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <sys/mman.h>  
#include <fcntl.h>  
#include <semaphore.h>  
#include <sys/types.h>  
#include <sys/wait.h>  
  
#define SHM\_NAME "/my\_shared\_memory"  
#define SEM\_WRITE "/sem\_write"  
#define SEM\_READ "/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:**

****

****

**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\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", 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.