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: Стаценко В.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 26.12.24

Москва, 2024

Постановка задачи

Вариант 1.

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 int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void) - создает дочерний процесс;
- int write(int fd, const void* buffer, int count) - записывает по дескриптору fd count байт из buffer;
- int execl(const char *path, const char *arg, ... /* (char *) NULL */) - выполняет программу, указанную в path, с аргументами, переданными в виде списка;
- pid_t wait(int status) - ожидает завершения родительского процесса;
- void exit(int number) - вызывает нормальное завершение программы с кодом number;
- sem_t * sem_open(const char *name, int oflag, mode_t mode, unsigned int value) - открывает или создает именованный семафор;
- int shmget(key_t key, size_t size, int shmflg) - создает новый сегмент разделяемой памяти или получает доступ к существующему;
- void * shmat(int shmid, const void *shmaddr, int shmflg) - присоединяет сегмент разделяемой памяти к адресному пространству процесса;
- int shmdt(const void *shmaddr) - отключает сегмент разделяемой памяти от адресного пространства процесса;
- int shmctl(int shmid, int cmd, struct shmid_ds *buf) - выполняет операции управления над сегментом разделяемой памяти;
- int sem_close(sem_t *sem) - закрывает семафор;
- int sem_unlink(const char *name) - удаляет именованный семафор;
- int sem_post(sem_t *sem) - увеличивает значение семафора sem. Если семафор был заблокирован (т.е. его значение было 0), то один из процессов, ожидающих на этом семафоре, будет разбужен.
- int sem_wait(sem_t *sem) - ожидает, пока семафор не станет положительным, уменьшая его значение;

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:

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

#define SHM_SIZE 1024
#define SEM_NAME "/my_semaphore"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Использование: %s <имя_файла>\n", argv[0]);
        exit(1);
    }

    int shm_id = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }

    sem_t *semaphore = sem_open(SEM_NAME, O_CREAT, 0644, 0);
    if (semaphore == SEM_FAILED) {
        perror("sem_open");
        shmctl(shm_id, IPC_RMID, NULL);
        exit(1);
    }

    pid_t pid = fork();
    if (pid < 0) {
        perror("fork");
        shmctl(shm_id, IPC_RMID, NULL);
        sem_close(semaphore);
        sem_unlink(SEM_NAME);
        exit(1);
    }
```

```c
    if (pid == 0) {
        char shm_id_str[10];
        sprintf(shm_id_str, "%d", shm_id);
        execl("./child", "child", shm_id_str, argv[1], NULL);
        perror("execl");
        shmctl(shm_id, IPC_RMID, NULL);
        sem_close(semaphore);
        sem_unlink(SEM_NAME);
        exit(1);
    } else {
        char *shm_ptr = shmat(shm_id, NULL, 0);
        if (shm_ptr == (char *) -1) {
            perror("shmat");
            shmctl(shm_id, IPC_RMID, NULL);
            sem_close(semaphore);
            sem_unlink(SEM_NAME);
            exit(1);
        }

        printf("Введите числа через пробел: ");
        fgets(shm_ptr, SHM_SIZE, stdin);

        if (sem_post(semaphore) == -1) {
            perror("sem_post");
        }

        wait(NULL);

        shmdt(shm_ptr);
        shmctl(shm_id, IPC_RMID, NULL);

        sem_close(semaphore);
        sem_unlink(SEM_NAME);
    }

    return 0;
}
```

child.c:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <semaphore.h>
#include <unistd.h>
#include <unistd.h>
#include <fcntl.h>

#define SEM_NAME "/my_semaphore"

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Использование: %s <shm_id> <имя_файла>\n", argv[0]);
        exit(1);
    }

    int shm_id = atoi(argv[1]);
    char *filename = argv[2];

    sem_t *semaphore = sem_open(SEM_NAME, 0);
    if (semaphore == SEM_FAILED) {
        perror("sem_open");
        exit(1);
    }

    char *shm_ptr = shmat(shm_id, NULL, 0);
    if (shm_ptr == (char *) -1) {
        perror("shmat");
        sem_close(semaphore);
        exit(1);
    }

    if (sem_wait(semaphore) == -1) {
        perror("sem_wait");
    }

    char *input = shm_ptr;
    int sum = 0;
    char *token = strtok(input, " ");
```

```c
    while (token != NULL) {
        sum += atoi(token);
        token = strtok(NULL, " ");
    }

    int fd = open(filename, O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (fd == -1) {
        perror("open");
        shmdt(shm_ptr);
        sem_close(semaphore);
        exit(1);
    }
    char sum_str[20];
    snprintf(sum_str, sizeof(sum_str), "%d\n", sum);
    write(fd, sum_str, strlen(sum_str));
    close(fd);

    shmdt(shm_ptr);
    sem_close(semaphore);
    return 0;
}
```

Протокол работы программы

Тестирование

```
victoria@victoria:~/laba/os/OSlabs/laba3$ gcc parent.c -o parent
victoria@victoria:~/laba/os/OSlabs/laba3$ gcc child.c -o child
victoria@victoria:~/laba/os/OSlabs/laba3$ ./parent w.txt
Введите числа через пробел: 5 6 8 9
```

```
os > OSlabs > laba3 > w.txt
1    28
2
```

Strace

victoria@victoria:~/laba/os/OSlabs/laba3$ strace ./parent w.txt
execve("./parent", ["./parent", "w.txt"], 0x7ffd75d523a8 /* 36 vars */) = 0
brk(NULL) = 0x55ee77ec6000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff771119e0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7faeef8cc000
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

```
        newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=17839, ...}, AT_EMPTY_PATH) = 0
        mmap(NULL, 17839, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faeef8c7000
        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\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) =
0x7faeef69e000
        mprotect(0x7faeef6c6000, 2023424, PROT_NONE) = 0
        mmap(0x7faeef6c6000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7faeef6c6000
        mmap(0x7faeef85b000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7faeef85b000
        mmap(0x7faeef8b4000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7faeef8b4000
        mmap(0x7faeef8ba000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7faeef8ba000
        close(3)                        = 0
        mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7faeef69b000
        arch_prctl(ARCH_SET_FS, 0x7faeef69b740) = 0
        set_tid_address(0x7faeef69ba10)         = 498702
        set_robust_list(0x7faeef69ba20, 24)     = 0
        rseq(0x7faeef69c0e0, 0x20, 0, 0x53053053) = 0
        mprotect(0x7faeef8b4000, 16384, PROT_READ) = 0
        mprotect(0x55ee77e35000, 4096, PROT_READ) = 0
        mprotect(0x7faeef906000, 8192, PROT_READ) = 0
        prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
        munmap(0x7faeef8c7000, 17839)           = 0
        shmget(IPC_PRIVATE, 1024, IPC_CREAT|0666) = 20
        openat(AT_FDCWD, "/dev/shm/sem.my_semaphore", O_RDWR|O_NOFOLLOW) = -1
ENOENT (No such file or directory)
        getrandom("\xc1\xb7\x59\x77\xe1\xb8\x97\x09", 8, GRND_NONBLOCK) = 8
        newfstatat(AT_FDCWD, "/dev/shm/sem.JX2rfp", 0x7fff771116f0,
AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)
        openat(AT_FDCWD, "/dev/shm/sem.JX2rfp", O_RDWR|O_CREAT|O_EXCL, 0644) = 3
        write(3, "\0\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, 3, 0) = 0x7faeef905000
        link("/dev/shm/sem.JX2rfp", "/dev/shm/sem.my_semaphore") = 0
        newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0
        getrandom("\xf5\xbe\xea\x6a\x50\x6c\x81\xb7", 8, GRND_NONBLOCK) = 8
        brk(NULL)                               = 0x55ee77ec6000
        brk(0x55ee77ee7000)                     = 0x55ee77ee7000
        unlink("/dev/shm/sem.JX2rfp")           = 0
        close(3)                        = 0
        clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7faeef69ba10) = 498703
        shmat(20, NULL, 0)                      = 0x7faeef8cb000
```

newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}, AT_EMPTY_PATH) = 0

newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}, AT_EMPTY_PATH) = 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 \321\207\320\265\321\200"..., 51Введите числа через пробел:) = 51

read(0, 7 11 3

"7 11 3\n", 1024) = 7

futex(0x7faeef905000, FUTEX_WAKE, 1) = 1

wait4(-1, NULL, 0, NULL) = 498703

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=498703, si_uid=1000, si_status=0, si_utime=0, si_stime=1} ---

shmdt(0x7faeef8cb000) = 0

shmctl(20, IPC_RMID, NULL) = 0

munmap(0x7faeef905000, 32) = 0

unlink("/dev/shm/sem.my_semaphore") = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы я научилась работать с механизмами межпроцессного взаимодействия (IPC) в Unix-системах, такими как разделяемая память и семафоры. В процессе работы я освоила ключевые функции, такие как shmget, shmat, shmdt, sem_open, sem_wait, sem_post, а также научилась корректно управлять ресурсами (памятью и семафорами) для предотвращения утечек. Эта лабораторная помогла мне понять, как организовывать синхронизацию и обмен данными между процессами.