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-209БВ-24

Студент: Крысанов А. Ю.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 10.12.25

Москва 2025

Постановка задачи

Вариант 12.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода. Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

Общий метод и алгоритм решения

Использованные системные вызовы

shm_open — создаёт или открывает объект разделяемой памяти
ftruncate — задаёт размер объекта shared memory
mmap — отображает участок памяти в адресное пространство процесса
fork — создаёт дочерний процесс
sem_init — инициализирует семафор
sem_wait — блокирует процесс до освобождения семафора
sem_post — освобождает семафор
waitpid — ожидает завершение дочернего процесса
munmap — снимает отображение разделяемой памяти
close — закрывает файловый дескриптор
shm_unlink — удаляет объект shared memory

Алгоритм работы:

Родительский процесс создаёт объект разделяемой памяти с помощью системного вызова shm_open, задаёт его размер через ftruncate и отображает область памяти в своё адресное пространство с помощью mmap. В общей структуре инициализируются три семафора parent_to_child_1, child_1_to_child_2 и child_2_to_parent, обеспечивающие синхронизацию между процессами. После этого родитель считывает строку, введённую пользователем, и помещает её в первый буфер в разделяемой памяти. Затем он подаёт сигнал семафором parent_to_child_1, разрешая первому дочернему процессу начать работу.

Первый дочерний процесс создаётся с помощью fork. После запуска он блокируется на семафоре parent_to_child_1, ожидая, пока родитель передаст строку. Получив разрешение, процесс копирует строку из первого буфера и преобразует все её символы в верхний регистр. Результат он помещает во второй буфер разделяемой памяти и сигнализирует второму дочернему процессу через семафор child_1_to_child_2.

Второй дочерний процесс также создаётся вызовом fork. Он ожидает поступления данных от первого ребёнка, блокируясь на семафоре child_1_to_child_2. После получения строки из второго буфера процесс удаляет повторяющиеся пробелы, оставляя между словами только один, и помещает итоговый результат в третий буфер. После завершения обработки он освобождает семафор child_2_to_parent, передавая управление обратно родителю.

Родительский процесс ожидает сигнал child_2_to_parent, после чего получает из третьего буфера итоговую строку и выводит её пользователю. Затем он вызывает waitpid для ожидания завершения обоих дочерних процессов, освобождает разделяемую память вызовом munmap, закрывает файловый дескриптор и удаляет объект разделяемой памяти через shm_unlink. На этом выполнение программы завершается.

Код программы

main.c

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

#define BUFFER_SIZE 256

struct MemoryShared
{
    sem_t parent_to_child_1;
    sem_t child_1_to_child_2;
    sem_t child_2_to_parent;
```

```
    char buffer1[BUFFER_SIZE];
    char buffer2[BUFFER_SIZE];
    char buffer3[BUFFER_SIZE];
};

int main() {
    int fd = shm_open("/my_shared_memory", O_CREAT | O_RDWR, 0666);
        if (fd == -1){
            perror("shm_open ошибка");
            exit(EXIT_FAILURE);
        }
    ftruncate(fd, sizeof(struct MemoryShared));
    struct MemoryShared *shared = mmap(NULL, sizeof(struct MemoryShared), PROT_READ |
PROT_WRITE, MAP_SHARED, fd, 0);
    if (shared == MAP_FAILED) {
        perror("mmap ошибка");
        exit(EXIT_FAILURE);
    }
    sem_init(&shared->parent_to_child_1, 1, 0);
    sem_init(&shared->child_1_to_child_2, 1, 0);
    sem_init(&shared->child_2_to_parent, 1, 0);
    char buffer[BUFFER_SIZE];
    printf("Введите строку для обработки: ");
    if (fgets(buffer, BUFFER_SIZE, stdin) != NULL) {
        buffer[strcspn(buffer, "\n")] = '\0';
    } else {
        perror("fgets ошибка");
        exit(EXIT_FAILURE);
    }
    strncpy(shared->buffer1, buffer, BUFFER_SIZE - 1);
    shared->buffer1[BUFFER_SIZE - 1] = '\0';
    sem_post(&shared->parent_to_child_1);

    pid_t child_1 = fork();
```

```
if (child_1 == 0) {
    sem_wait(&shared->parent_to_child_1);
    char child_1_buffer[BUFFER_SIZE];
    strncpy(child_1_buffer, shared->buffer1, BUFFER_SIZE - 1);
    child_1_buffer[BUFFER_SIZE - 1] = '\0';
    for (int i = 0; i < BUFFER_SIZE; i++) {
        child_1_buffer[i] = toupper(child_1_buffer[i]);
    }
    strncpy(shared->buffer2, child_1_buffer, BUFFER_SIZE - 1);
    shared->buffer2[BUFFER_SIZE - 1] = '\0';
    sem_post(&shared->child_1_to_child_2);
    exit(0);
}
pid_t child_2 = fork();
if (child_2 == 0) {
    sem_wait(&shared->child_1_to_child_2);
    char child_2_buffer[BUFFER_SIZE];
    strncpy(child_2_buffer, shared->buffer2, BUFFER_SIZE - 1);
    child_2_buffer[BUFFER_SIZE - 1] = '\0';
    int j = 0;
    int prev_space = 0;
    for (int i = 0; child_2_buffer[i] != '\0'; ++i) {
        if (isspace(buffer[i])) {
            if (!prev_space) {
                shared->buffer3[j++] = ' ';
                prev_space = 1;
            }
        } else {
            shared->buffer3[j++] = child_2_buffer[i];
            prev_space = 0;
        }
    }
    shared->buffer3[j] = '\0';
    sem_post(&shared->child_2_to_parent);
```

```
            exit(0);
        }
        sem_wait(&shared->child_2_to_parent);
        waitpid(child_1, NULL, 0);
        waitpid(child_2, NULL, 0);
        printf("Результат: %s\n", shared->buffer3);
        munmap(shared, sizeof(struct MemoryShared));
        close(fd);
        shm_unlink("/my_shared_memory");
    }
```

Протокол работы программы

Тестирование:

root ➜ /workspace/laba3 $ ./shared_memory_example

Введите строку для обработки: egor likes FOOTBALL

Результат: EGOR LIKES FOOTBALL

Strace(через docker):

root ➜ /workspace/laba3 $ strace -f ./shared_memory_example

execve("./shared_memory_example", ["./shared_memory_example"], 0x7ffc051e60b8 /* 12 vars */) = 0

brk(NULL)   = 0x5ec559066000

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=21784, ...}) = 0

mmap(NULL, 21784, PROT_READ, MAP_PRIVATE, 3, 0) = 0x79563f7bd000

close(3)   = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/librt.so.1", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260#\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0644, st_size=35808, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x79563f7bb000

mmap(NULL, 39904, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x79563f7b1000

mmap(0x79563f7b3000, 16384, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x79563f7b3000

mmap(0x79563f7b7000, 8192, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x79563f7b7000

mmap(0x79563f7b9000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x79563f7b9000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 l\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0755, st_size=149520, ...}) = 0

mmap(NULL, 136304, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x79563f78f000

mmap(0x79563f795000, 65536, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x79563f795000

mmap(0x79563f7a5000, 24576, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16000) = 0x79563f7a5000

mmap(0x79563f7ab000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b000) = 0x79563f7ab000

mmap(0x79563f7ad000, 13424, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x79563f7ad000

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\260>\2\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0755, st_size=1901536, ...}) = 0

mmap(NULL, 1914496, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x79563f5bb000

mmap(0x79563f5dd000, 1413120, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x79563f5dd000

mmap(0x79563f736000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17b000) = 0x79563f736000

mmap(0x79563f785000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c9000) = 0x79563f785000

mmap(0x79563f78b000, 13952, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x79563f78b000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x79563f5b8000

arch_prctl(ARCH_SET_FS, 0x79563f5b8740) = 0

mprotect(0x79563f785000, 16384, PROT_READ) = 0

mprotect(0x79563f7ab000, 4096, PROT_READ) = 0

mprotect(0x79563f7b9000, 4096, PROT_READ) = 0

mprotect(0x5ec5365e7000, 4096, PROT_READ) = 0

mprotect(0x79563f7ed000, 4096, PROT_READ) = 0

munmap(0x79563f7bd000, 21784) = 0

set_tid_address(0x79563f5b8a10) = 145

set_robust_list(0x79563f5b8a20, 24) = 0

rt_sigaction(SIGRTMIN, {sa_handler=0x79563f795690, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x79563f7a2140}, NULL, 8) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x79563f795730, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x79563f7a2140}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

statfs("/dev/shm/", {f_type=TMPFS_MAGIC, f_bsize=4096, f_blocks=16384, f_bfree=16384, f_bavail=16384, f_files=1001495, f_ffree=1001494, f_fsid={val=[3884477280, 1800598335]}, f_namelen=255, f_frsize=4096, f_flags=ST_VALID|ST_NOSUID|ST_NODEV|ST_NOEXEC|ST_RELATIME}) = 0

futex(0x79563f7b0410, FUTEX_WAKE_PRIVATE, 2147483647) = 0

openat(AT_FDCWD, "/dev/shm/my_shared_memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3

ftruncate(3, 864) = 0

mmap(NULL, 864, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x79563f7ec000

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0

brk(NULL) = 0x5ec559066000

brk(0x5ec559087000) = 0x5ec559087000

fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\321\203 \320\264\320\273"..., 55Введите строку для обработки:) = 55

read(0, egor likes FOOTBALL

"egor likes FOOTBALL\n", 1024) = 22

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 146 attached

, child_tidptr=0x79563f5b8a10) = 146

[pid 145] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD <unfinished ...>

[pid 146] set_robust_list(0x79563f5b8a20, 24) = 0

strace: Process 147 attached

[pid 145] <... clone resumed>, child_tidptr=0x79563f5b8a10) = 147

[pid 147] set_robust_list(0x79563f5b8a20, 24 <unfinished ...>

[pid 146] exit_group(0 <unfinished ...>

[pid 145] futex(0x79563f7ec040, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 147] <... set_robust_list resumed>) = 0

[pid 146] <... exit_group resumed>) = ?

[pid 147] futex(0x79563f7ec040, FUTEX_WAKE, 1 <unfinished ...>

[pid 146] +++ exited with 0 +++

[pid 145] <... futex resumed>) = ? ERESTARTSYS (To be restarted if SA_RESTART is set)

[pid 147] <... futex resumed>) = 0

[pid 145] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=146, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---

[pid 145] futex(0x79563f7ec040, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 147] exit_group(0 <unfinished ...>

[pid 145] <... futex resumed>) = -1 EAGAIN (Resource temporarily unavailable)

[pid 147] <... exit_group resumed>) = ?

[pid 145] wait4(146, NULL, 0, NULL) = 146

[pid 145] wait4(147, <unfinished ...>

[pid 147] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL) = 147

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=147, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202: EGOR LIKES F"..., 40Результат: EGOR LIKES FOOTBALL

) = 40

munmap(0x79563f7ec000, 864) = 0

close(3) = 0

unlink("/dev/shm/my_shared_memory") = 0

exit_group(0)

Вывод

В ходе работы программы была реализована конвейерная обработка строки с использованием разделяемой памяти и семафоров. Родительский процесс и два дочерних процесса последовательно выполняют преобразование входной строки, обеспечивая синхронизацию с помощью POSIX-семафоров. Итоговая строка корректно передаётся через общую память и возвращается родителю после обработки, что подтверждает правильность работы механизмов межпроцессного взаимодействия.