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: Махова А.Б.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 19.12.25

Москва, 2025

Постановка задачи

Вариант 6.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным

Реализовать с использованием shared memory и memory mapping

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void) – создание дочернего процесса.
- ssize_t write(int fd, void *buf, size_t count) – вывод данных в поток.
- ssize_t read(int fd, void *buf, size_t count) – ввод данных из потока.
- int execl(const char *path, const char *arg0, ..., NULL) – запуск исполняемого файла с заменой текущего процесса.
- int sem_wait(sem_t *sem) – блокировка семафора.
- sem_t* sem_open(const char *name, int oflag, mode_t mode, unsigned int value) – инициализация именованного семафора.
- int sem_post(sem_t *sem) – разблокировка семафора.
  pid_t waitpid(pid_t pid, int *status, int options) – ожидание завершения указанного процесса.
- int sem_unlink(const char *name) – удаление именованного семафора из системы.
- int sem_close(sem_t *sem) – закрытие дескриптора семафора.
- int shm_open(const char *name, int oflag, mode_t mode) – открытие области разделяемой памяти.
- int shm_unlink(const char *name) – удаление именованной области разделяемой памяти.

- void* mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset) – проекция объекта в память процесса.
- int munmap(void *addr, size_t length) – удаление проекции из памяти.
- void exit(int status) – корректное завершение процесса с кодом возврата.

Родитель создаёт область разделяемой памяти и семафор. После запроса имени файла у пользователя порождается дочерний процесс, в котором запускается серверная часть. Дочерний процесс открывает указанный файл, читает его построчно, извлекает числовые значения и выполняет их суммирование. При обнаружении невалидных данных формируется сообщение об ошибке. Каждый полученный результат или ошибка синхронно записываются в разделяемую память с использованием семафора. Родитель в цикле опрашивает память, извлекает подготовленные данные и отображает их на экране. После полной обработки входного файла дочерний процесс помещает в память маркер завершения (INT_MAX), что служит сигналом для родителя о прекращении работы. Получив этот сигнал, родитель завершает цикл чтения, освобождает все занятые ресурсы (семафор и область памяти) и завершает выполнение.

Код программы

client.c

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

#define SHM_SIZE 4096

const char SHM_NAME[] = "/sum_sh_memory";
const char SEM_NAME[] = "/sum_semaphore";

int main() {
    int shared_mem = shm_open(SHM_NAME, O_RDWR | O_CREAT | O_TRUNC, 0666);
    if (shared_mem == -1) {
        const char message[] = "ERR: cant create shared memory\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shared_mem, SHM_SIZE) != 0) {
        const char message[] = "ERR: cant resize shared memory\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
    }

    char* const shared_mem_buffer = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shared_mem, 0);
```

```
if (shared_mem_buffer == MAP_FAILED) {
    const char message[] = "ERR: cant map shared memory\n";
    write(STDERR_FILENO, message, sizeof(message) - 1);
    exit(EXIT_FAILURE);
}

int* length = (int*)shared_mem_buffer;
*length = 0;

sem_t* semaphore = sem_open(SEM_NAME, O_CREAT | O_RDWR, 0666, 1);
if (semaphore == SEM_FAILED) {
    const char message[] = "ERR: cant create semaphore\n";
    write(STDERR_FILENO, message, sizeof(message) - 1);
    exit(EXIT_FAILURE);
}

char file_path[128];
const char message[] = "Input file : ";
write(STDOUT_FILENO, message, sizeof(message) - 1);

int result = read(STDIN_FILENO, file_path, sizeof(file_path) - 1);
if (result <= 0) {
    const char error_message[] = "ERR: cant read filename\n";
    write(STDERR_FILENO, error_message, sizeof(error_message) - 1);
    exit(EXIT_FAILURE);
}
file_path[result - 1] = 0;

pid_t child = fork();

if (child == 0) {
    execl("./server", "server", file_path, NULL);
    const char message[] = "ERR: cant execute server\n";
```

```
    write(STDERR_FILENO, message, sizeof(message) - 1);
    exit(EXIT_FAILURE);
}
else if (child == -1) {
    const char message[] = "ERR: cant fork\n";
    write(STDERR_FILENO, message, sizeof(message) - 1);
    exit(EXIT_FAILURE);
}

int running = 1;
while(running) {
    sem_wait(semaphore);

    int* current_length = (int*)shared_mem_buffer;
    char* data = shared_mem_buffer + sizeof(int);

    if (*current_length == INT_MAX) {
        running = 0;
    }
    else if (*current_length > 0) {
        write(STDOUT_FILENO, data, *current_length);
        *current_length = 0;
    }

    sem_post(semaphore);
    usleep(1000);
}

waitpid(child, NULL, 0);

sem_close(semaphore);
sem_unlink(SEM_NAME);
munmap(shared_mem_buffer, SHM_SIZE);
```

```
    shm_unlink(SHM_NAME);
    close(shared_mem);

    return 0;
}
```

server.c

```
#include <fcntl.h>
#include <errno.h>
#include <limits.h>
#include <semaphore.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define SHM_SIZE 4096

const char SHM_NAME[] = "/sum_sh_memory";
const char SEM_NAME[] = "/sum_semaphore";

int main(int argc, char** argv) {
    if (argc < 2) {
        const char message[] = "ERR: not enough arguments\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
    }

    const char* filename = argv[1];
```

```
    int shared_mem = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shared_mem == -1) {
        const char message[] = "ERR: cant open shared memory\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
    }

    char* const shared_mem_buffer = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shared_mem, 0);
    if (shared_mem_buffer == MAP_FAILED) {
        const char message[] = "ERR: cant map shared memory\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
    }

    sem_t* semaphore = sem_open(SEM_NAME, O_RDWR);
    if (semaphore == SEM_FAILED) {
        const char message[] = "ERR: cant open semaphore\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
    }

    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        const char message[] = "ERR: cant open file\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
    }

    char line[256];
    while (fgets(line, sizeof(line), file) != NULL) {
        line[strcspn(line, "\r\n")] = 0;

        if (strlen(line) == 0) {
```

```
    continue;
}

int numbers[100];
int count = 0;
char* token = strtok(line, " ");
int valid = 1;

while (token != NULL && count < 100) {
    char* p = token;

    if (*p == '-') p++;

    if (*p == '\0') {
        valid = 0;
        break;
    }

    int has_digits = 0;
    while (*p) {
        if (!isdigit(*p)) {
            valid = 0;
            break;
        }
        has_digits = 1;
        p++;
    }

    if (!valid || !has_digits) break;

    numbers[count] = atoi(token);
    count++;
    token = strtok(NULL, " ");
```

```
    }

    if (!valid || count == 0) {
        const char error_msg[] = "ERR: invalid input\n";
        sem_wait(semaphore);
        int* length = (int*)shared_mem_buffer;
        char* data = shared_mem_buffer + sizeof(int);
        *length = sizeof(error_msg) - 1;
        memcpy(data, error_msg, sizeof(error_msg) - 1);
        sem_post(semaphore);
    }
    else {
        int sum = 0;
        for (int i = 0; i < count; i++) {
            sum += numbers[i];
        }

        char result_str[64];
        int len = snprintf(result_str, sizeof(result_str), "%d\n", sum);

        sem_wait(semaphore);
        int* length = (int*)shared_mem_buffer;
        char* data = shared_mem_buffer + sizeof(int);
        *length = len;
        memcpy(data, result_str, len);
        sem_post(semaphore);
    }

    usleep(1000);
}

fclose(file);
```

```
    sem_wait(semaphore);
    int* length = (int*)shared_mem_buffer;
    *length = INT_MAX;
    sem_post(semaphore);

    sem_close(semaphore);
    munmap(shared_mem_buffer, SHM_SIZE);
    close(shared_mem);

    return 0;
}
```

Протокол работы программы

Тестирование:

```
vscode ➜ /workspaces/MAI_OS/lab3 (main) $ gcc -o client src/client.c -lrt -pthread
vscode ➜ /workspaces/MAI_OS/lab3 (main) $ gcc -o server src/server.c -lrt -pthread
vscode ➜ /workspaces/MAI_OS/lab3 (main) $ ./client
Input file : a.txt
6
ERR: invalid input
45
0
vscode ➜ /workspaces/MAI_OS/lab3 (main) $ ./client
Input file : b.txt
0
1
27
ERR: invalid input
```

Strace:

```
vscode ➜ /workspaces/MAI_OS/lab3 (main) $ strace ./client
execve("./client", ["./client"], 0xffffde9e7ed0 /* 35 vars */) = 0
brk(NULL)                               = 0xaaaade327000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xffff99e38000
faccessat(AT_FDCWD, "/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=15048, ...}, AT_EMPTY_PATH) = 0
```

mmap(NULL, 15048, PROT_READ, MAP_PRIVATE, 3, 0) = 0xffff99e34000

close(3) = 0

openat(AT_FDCWD, "/lib/aarch64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\340u\2\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1637400, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 1805928, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff99c4a000

mmap(0xffff99c50000, 1740392, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0) = 0xffff99c50000

munmap(0xffff99c4a000, 24576) = 0

munmap(0xffff99df9000, 40552) = 0

mprotect(0xffff99dd8000, 61440, PROT_NONE) = 0

mmap(0xffff99de7000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x187000) = 0xffff99de7000

mmap(0xffff99ded000, 48744, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff99ded000

close(3) = 0

set_tid_address(0xffff99e38f50) = 35722

set_robust_list(0xffff99e38f60, 24) = 0

rseq(0xffff99e39620, 0x20, 0, 0xd428bc00) = 0

mprotect(0xffff99de7000, 16384, PROT_READ) = 0

mprotect(0xaaaabeb41000, 4096, PROT_READ) = 0

mprotect(0xffff99e3d000, 8192, PROT_READ) = 0

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

munmap(0xffff99e34000, 15048) = 0

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

ftruncate(3, 4096) = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0xffff99e37000

**openat(AT_FDCWD, "/dev/shm/sem.sum_semaphore", O_RDWR|O_NOFOLLOW) = -1 ENOENT
(No such file or directory)**

getrandom("\x09\x15\x1a\x2a\x1a\x37\x3a\x06", 8, GRND_NONBLOCK) = 8

newfstatat(AT_FDCWD, "/dev/shm/sem.rq2AZl", 0xffffd2cc89b8, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/sem.rq2AZl", O_RDWR|O_CREAT|O_EXCL, 0666) = 4

write(4, "\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, 4, 0) = 0xffff99e36000

linkat(AT_FDCWD, "/dev/shm/sem.rq2AZl", AT_FDCWD, "/dev/shm/sem.sum_semaphore", 0) = 0

newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

getrandom("\x2d\x33\x81\xed\x5c\xa9\x69\xda", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0xaaaade327000

brk(0xaaaade348000) = 0xaaaade348000

unlinkat(AT_FDCWD, "/dev/shm/sem.rq2AZl", 0) = 0

close(4) = 0

write(1, "Input file : ", 13Input file :) = 13

read(0, a.txt

"a.txt\n", 127) = 6

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0xffff99e38f50) = 35816

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=1000000}, NULL) = 0

write(1, "6\n", 26

) = 2

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=1000000}, NULL) = 0

write(1, "45\n", 345

) = 3

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=1000000}, NULL) = 0

write(1, "0\n", 20

) = 2

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=1000000}, NULL) = 0

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=1000000}, NULL) = ?
ERESTART_RESTARTBLOCK (Interrupted by signal)

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=35816, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---

restart_syscall(<... resuming interrupted clock_nanosleep ...>) = 0

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=1000000}, NULL) = 0

wait4(35816, NULL, 0, NULL) = 35816

munmap(0xffff99e36000, 32) = 0

unlinkat(AT_FDCWD, "/dev/shm/sem.sum_semaphore", 0) = 0

munmap(0xffff99e37000, 4096) = 0

unlinkat(AT_FDCWD, "/dev/shm/sum_sh_memory", 0) = 0

close(3) = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В лабораторной работе использовались системные вызовы Linux для организации межпроцессного взаимодействия через разделяемую память (shared memory) и синхронизации с помощью семафоров. Была реализована клиент-серверная архитектура, где клиент создаёт разделяемую память и семафор, запрашивает имя файла у пользователя и запускает серверный процесс.