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-211Б-23

Студент: Королев И.А

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 21.12.24

Москва, 2024

Постановка задачи

Вариант 13.

Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «_».

Общий метод и алгоритм решения

Использованные системные вызовы:

- **pid_t fork(void);** – создает дочерний процесс, возвращая PID дочернего процесса родительскому процессу, и 0 – в дочернем. В родительском процессе используется для создания двух дочерних процессов, которые будут выполнять обработку данных;
- **int shm_open(const char *__name, int __oflag, mode_t __mode);** – открывает сегмент shm;
- **ssize_t read(int fd, void *buf, size_t count);** – читает данные из файлового дескриптора fd в буфер buf. В программе используется для чтения пользовательского ввода в родительском процессе и передачи данных через пайпы в дочерние процессы, а также для получения данных, обработанных дочерними процессами;
- **ssize_t write(int fd, const void *buf, size_t count);** – записывает данные из буфера buf в файловый дескриптор fd. Применяется для отправки данных между процессами через пайпы, а также для вывода на экран обработанных данных;
- **int execl(const char *path, const char *arg, ...);** – заменяет текущий образ процесса другим, выполняя указанную программу. В программе каждый дочерний процесс использует execl для вызова child1 или child2, чтобы выполнить их основную функцию;
- **void *mmap(void * addr, size_t len, int prot, int flags, int fd, off_t __offset) –** создает новый маппинг в виртуальном адресном пространстве;
- **sem_t *sem_open (const char * name, int oflag, …) -** открывает именованный семафор;
- **int sem_unlink (const char *name)** — удаляет именованный семафор;
- **int sem_wait(sem_t *sem) -** уменьшает (блокирует) семафор;
- **int sem_post(sem_t *sem) -** увеличивает (разблокирует) семафор;
- **int open(const char *pathname, int flags, mode_t mode) -** открытие\создание файла;
- **int close(int fd) -** закрыть файл;
- **void exit(int status)** – завершения выполнения процесса и возвращение статуса;
- **pid_t getpid(void)** – получение ID процесса;

Код программы

parent.c

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

#define SHM_NAME "shm_lab"
```

```c
#define SEM_READ1_NAME "/sem_read1"
#define SEM_WRITE1_NAME "/sem_write1"
#define SEM_READ2_NAME "/sem_read2"
#define SEM_WRITE2_NAME "/sem_write2"
#define BUF_SIZE 4096

int main() {
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Ошибка создания разделяемой памяти");
        exit(EXIT_FAILURE);
    }
    if (ftruncate(shm_fd, BUF_SIZE) == -1) {
        perror("Ошибка установки размера памяти");
        exit(EXIT_FAILURE);
    }

    char *shared_memory = mmap(NULL, BUF_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shared_memory == MAP_FAILED) {
        perror("Ошибка маппинга памяти");
        exit(EXIT_FAILURE);
    }

    sem_t *sem_read1 = sem_open(SEM_READ1_NAME, O_CREAT, 0666, 0);
    sem_t *sem_write1 = sem_open(SEM_WRITE1_NAME, O_CREAT, 0666, 1);
    sem_t *sem_read2 = sem_open(SEM_READ2_NAME, O_CREAT, 0666, 0);
    sem_t *sem_write2 = sem_open(SEM_WRITE2_NAME, O_CREAT, 0666, 0);

    if (!sem_read1 || !sem_write1 || !sem_read2 || !sem_write2) {
        perror("Ошибка создания семафоров");
        exit(EXIT_FAILURE);
    }

    pid_t child1 = fork();
    if (child1 == -1) {
        perror("Ошибка создания child1");
        exit(EXIT_FAILURE);
    }
    if (child1 == 0) {
        execl("./child1", "child1", NULL);
        perror("Ошибка запуска child1");
        exit(EXIT_FAILURE);
    }

    pid_t child2 = fork();
    if (child2 == -1) {
        perror("Ошибка создания child2");
        exit(EXIT_FAILURE);
    }
    if (child2 == 0) {
        execl("./child2", "child2", NULL);
        perror("Ошибка запуска child2");
        exit(EXIT_FAILURE);
    }

    printf("Введите строки (Enter для завершения):\n");
    char buffer[BUF_SIZE];
    while (1) {
        if (!fgets(buffer, BUF_SIZE, stdin)) {
            break;
        }

        if (buffer[0] == '\n') {
            sem_wait(sem_write1);
            strcpy(shared_memory, "END");
            sem_post(sem_read1);
```

```
            sem_wait(sem_write1);
            sem_post(sem_read2);

            break;
        }

        sem_wait(sem_write1);
        strcpy(shared_memory, buffer);
        sem_post(sem_read1);

        sem_wait(sem_write1);
        sem_post(sem_read2);

        sem_wait(sem_write2);
        printf("> %s\n", shared_memory);
        sem_post(sem_write1);
    }

    sem_wait(sem_write1);
    strcpy(shared_memory, "END");
    sem_post(sem_read1);

    sem_wait(sem_write1);
    sem_post(sem_read2);

    waitpid(child1, NULL, 0);
    waitpid(child2, NULL, 0);

    munmap(shared_memory, BUF_SIZE);
    shm_unlink(SHM_NAME);

    sem_close(sem_read1);
    sem_close(sem_write1);
    sem_close(sem_read2);
    sem_close(sem_write2);

    sem_unlink(SEM_READ1_NAME);
    sem_unlink(SEM_WRITE1_NAME);
    sem_unlink(SEM_READ2_NAME);
    sem_unlink(SEM_WRITE2_NAME);

    return 0;
}
```

child1.c

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

#define SHM_NAME "shm_lab"
#define SEM_READ1_NAME "/sem_read1"
#define SEM_WRITE1_NAME "/sem_write1"
#define BUF_SIZE 4096

int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Ошибка открытия памяти в child1");
        exit(EXIT_FAILURE);
```

```
    }

    char *shared_memory = mmap(NULL, BUF_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shared_memory == MAP_FAILED) {
        perror("Ошибка маппинга памяти в child1");
        exit(EXIT_FAILURE);
    }

    sem_t *sem_read1 = sem_open(SEM_READ1_NAME, 0);
    sem_t *sem_write1 = sem_open(SEM_WRITE1_NAME, 0);
    if (!sem_read1 || !sem_write1) {
        perror("Ошибка открытия семафоров в child1");
        exit(EXIT_FAILURE);
    }

    while (1) {
        sem_wait(sem_read1);

        if (strcmp(shared_memory, "END") == 0) {
            break;
        }

        for (size_t i = 0; shared_memory[i]; i++) {
            shared_memory[i] = tolower((unsigned char)shared_memory[i]);
        }

        sem_post(sem_write1);
    }

    munmap(shared_memory, BUF_SIZE);
    sem_close(sem_read1);
    sem_close(sem_write1);

    return 0;
}
```

child2.c

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

#define SHM_NAME "shm_lab"
#define SEM_READ2_NAME "/sem_read2"
#define SEM_WRITE2_NAME "/sem_write2"
#define BUF_SIZE 4096

int main() {
```

```
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Ошибка открытия памяти в child2");
        exit(EXIT_FAILURE);
    }

    char *shared_memory = mmap(NULL, BUF_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shared_memory == MAP_FAILED) {
        perror("Ошибка маппинга памяти в child2");
        exit(EXIT_FAILURE);
    }

    sem_t *sem_read2 = sem_open(SEM_READ2_NAME, 0);
    sem_t *sem_write2 = sem_open(SEM_WRITE2_NAME, 0);
    if (!sem_read2 || !sem_write2) {
        perror("Ошибка открытия семафоров в child2");
        exit(EXIT_FAILURE);
    }

    while (1) {
        sem_wait(sem_read2);

        if (strcmp(shared_memory, "END") == 0) {
            break;
        }

        for (size_t i = 0; shared_memory[i]; i++) {
            if (isspace((unsigned char)shared_memory[i])) {
                shared_memory[i] = '_';
            }
        }

        sem_post(sem_write2);
    }

    munmap(shared_memory, BUF_SIZE);
    sem_close(sem_read2);
    sem_close(sem_write2);

    return 0;
}
```

Протокол работы программы

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs# cd lab3/src

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab3/src# gcc -o child1 -lm child1.c

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab3/src# gcc -o child2 -lm child2.c

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab3/src# gcc -o parent -lm parent.c

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab3/src# ./parent

Введите строки (Enter для завершения):

PriVeet KAk deLa

> priveet_kak_dela_

U menaA HorosHo

> u_menaa__horosho_

Ponyal

> ponyal_

A is kakoy ti grouuss &

> a_is_kakoy_ti__grouuss_&_

ZYa is 211

> zya_is_211_

Ponyatno

> ponyatno_

Vot orishol sdavat Osi

> vot_orishol_sdavat_osi_

NORMALNO

> normalno_

INTERESno   UTOGO

> interesno___utogo__

DOSTATOCHO dlya Testa

> dostatocho_dlya_testa__

Okaye pora prekrashat

> okaye_pora_prekrashat_

VSEWM POKA

> vsewm_poka_

Strace:

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab3/src# strace ./parent

execve("./parent", ["./parent"], 0x7ffd3d078930 /* 27 vars */) = 0

brk(NULL)   = 0x559fdf7e6000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd94281120) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe1198ea000

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=18883, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 18883, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe1198e5000

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) = 0x7fe1196bc000

mprotect(0x7fe1196e4000, 2023424, PROT_NONE) = 0

mmap(0x7fe1196e4000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fe1196e4000

mmap(0x7fe119879000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7fe119879000

mmap(0x7fe1198d2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7fe1198d2000

mmap(0x7fe1198d8000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fe1198d8000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe1196b9000

arch_prctl(ARCH_SET_FS, 0x7fe1196b9740) = 0

set_tid_address(0x7fe1196b9a10) = 1752

set_robust_list(0x7fe1196b9a20, 24) = 0

rseq(0x7fe1196ba0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fe1198d2000, 16384, PROT_READ) = 0

mprotect(0x559fd9d37000, 4096, PROT_READ) = 0

mprotect(0x7fe119924000, 8192, PROT_READ) = 0

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

munmap(0x7fe1198e5000, 18883) = 0

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

ftruncate(3, 4096) = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fe119923000

openat(AT_FDCWD, "/dev/shm/sem.sem_read1", O_RDWR|O_NOFOLLOW) = 4

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

getrandom("\xf7\xbf\xe7\xde\x4e\xee\x2a\x5b", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x559fdf7e6000

brk(0x559fdf807000) = 0x559fdf807000

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fe1198e9000

close(4) = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_write1", O_RDWR|O_NOFOLLOW) = 4

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

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fe1198e8000

close(4) = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_read2", O_RDWR|O_NOFOLLOW) = 4

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

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fe1198e7000

close(4) = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_write2", O_RDWR|O_NOFOLLOW) = 4

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

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fe1198e6000

close(4) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fe1196b9a10) = 1753

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fe1196b9a10) = 1754

newfstatat(1, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 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\320\270 (Ent"..., 65Введите строки (Enter для завершения):

) = 65

newfstatat(0, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0

read(0, privet

"privet\n", 1024) = 7

futex(0x7fe1198e9000, FUTEX_WAKE, 1) = 1

futex(0x7fe1198e7000, FUTEX_WAKE, 1) = 1

write(1, "> privet_\n", 10> privet_

) = 10

read(0, VOT takIE dela

"VOT takIE dela\n", 1024) = 15

futex(0x7fe1198e9000, FUTEX_WAKE, 1) = 1

futex(0x7fe1198e7000, FUTEX_WAKE, 1) = 1

write(1, "> vot_takie_dela_\n", 18> vot_takie_dela_

) = 18

read(0, OTLISCHO RABOTAEM

"OTLISCHO RABOTAEM\n", 1024) = 19

futex(0x7fe1198e9000, FUTEX_WAKE, 1) = 1

futex(0x7fe1198e7000, FUTEX_WAKE, 1) = 1

write(1, "> otlischo__rabotaem_\n", 22> otlischo__rabotaem_

) = 22

read(0,

"\n", 1024) = 1

futex(0x7fe1198e9000, FUTEX_WAKE, 1) = 1

futex(0x7fe1198e8000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY) = ? ERESTARTSYS (To be restarted if SA_RESTART is set)

wait4(-1, NULL, 0, NULL) = 1753

wait4(-1, NULL, 0, NULL) = 1754

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1753, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1754, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---

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

close(3) = 0 //parent

munmap(0x7f40c9f08000, 32) = 0

unlink("/dev/shm/sem_read1") = 0

munmap(0x7f40c9f07000, 32) = 0

unlink("/dev/shm/sem_read1") = 0

munmap(0x7f40c9f06000, 32) = 0

unlink("/dev/shm/sem_read2") = 0

munmap(0x7f40c9f41000, 512) = 0

close(4) **= 0 //child**

exit_group(0) **= ?**

+++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы была разработана программа, в которой родительский процесс принимает входные данные и отправляет их в общую память, после чего первый дочерний процесс обрабатывает эти данные и второй дочерний процесс, обрабатывает данные и возвращает их в родительский процесс, который выводит эти данные на экран. В ходе выполнения я столкнулся с трудностями использования одного семафора для общения между тремя процессами, все удалось решить добавив дополнительный семафор между двумя дочерними процессами.