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-214БВ-24

Студент: Кириенко А. И.

Преподаватель Бахарев В. Д.

Оценка: _____

Дата: 12.11.25

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:

fork() - создание нового дочернего процесса

execl() - замена текущего образа процесса новым

wait() - ожидание завершения дочернего процесса

shm_open() - создание или открытие объекта разделяемой памяти (shared memory)

ftruncate() - изменение размера объекта разделяемой памяти

mmap() - отображение объекта памяти в адресное пространство процесса

munmap() - освобождение области разделяемой памяти

shm_unlink() - удаление именованного объекта разделяемой памяти

sem_open() - создание или открытие именованного семафора

sem_wait() - ожидание семафора (блокирует процесс при значении 0)

sem_post() - освобождение(увеличение) семафоры. разблокирует ожидающий процесс

sem_close() - закрытие семафора

sem_unlink() - удаление именованного семафора

perror() - вывод сообщения об ошибке системного вызова

exit() - завершение текущего процесса.

Для обмена данными между процессами создаются три области разделяемой памяти: (shm1 shm2 shm3) с помощью ситстемного вызова shm_open. shm1 - между родителем и первым ребенком; shm2 - между детьми; shm3 - между вторым ребенком и родителем.

Для каждой области памяти создается два семафора: sem*_empty - показывает, что буфер пуст и в него можно писать. sem*_full - показывает, что буфер заполнен и из него можно читать.

С помощью системного вызова fork() родительский процесс создает две копии себя, далее вызывается execl("./child1", …), который заменяет текущий код на программу child1. Аналогично с child2.

Основной цикл:

1. Родитель читает строку из стандартного ввода (stdin), записывает её в shm1 и разрешает чтение child1.
2. Child1 получает строку из shm1, переводит все буквы в нижний регистр и записывает результат в shm2.
3. Child2 получает данные из shm2, заменяет пробелмы и знаки табуляции символом "_", записывает результат в shm3 и сигнализирует родителю.
4. Родитель считывает из shm3 и выводит результат на экран.

Код программы

Parent.c

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

#define BUF_SIZE 1024

typedef struct {
    char data[BUF_SIZE];
    size_t len;
} shm_t;

int main(){
    pid_t pid = getpid(); //сохраняем идентификатор PID текущего процесса
    char shm1_name[64], shm2_name[64], shm3_name[64];
    char sem1_empty_name[64], sem1_full_name[64];
    char sem2_empty_name[64], sem2_full_name[64]; //создаем массивы символов, где будут храниться имена наших объектов POSIX
    char sem3_empty_name[64], sem3_full_name[64]; //три shared memory и шесть семафоров

    sprintf(shm1_name, "/shm1_%d", pid); //имена для объектов shared memory
    sprintf(shm2_name, "/shm2_%d", pid);
    sprintf(shm3_name, "/shm3_%d", pid);

    sprintf(sem1_empty_name, "/sem1_empty_%d", pid); //имена для семафоров
    sprintf(sem1_full_name,  "/sem1_full_%d", pid); // empty отвечатает за пустоту; =1 значит можно писать
    sprintf(sem2_empty_name, "/sem2_empty_%d", pid); // full отвечает за заполненность; =0 значит нечего читать
    sprintf(sem2_full_name,  "/sem2_full_%d", pid);
    sprintf(sem3_empty_name, "/sem3_empty_%d", pid);
    sprintf(sem3_full_name,  "/sem3_full_%d", pid);

    // создаем три объекта shared memory (возвращает дескриптор файла)
    int fd1 = shm_open(shm1_name, O_CREAT | O_RDWR, 0666);
    int fd2 = shm_open(shm2_name, O_CREAT | O_RDWR, 0666);
    int fd3 = shm_open(shm3_name, O_CREAT | O_RDWR, 0666);
```

```
ftruncate(fd1, sizeof(shm_t));
ftruncate(fd2, sizeof(shm_t));
ftruncate(fd3, sizeof(shm_t));

shm_t *shm1 = mmap(NULL, sizeof(shm_t), PROT_READ | PROT_WRITE, MAP_SHARED, fd1, 0);
shm_t *shm2 = mmap(NULL, sizeof(shm_t), PROT_READ | PROT_WRITE, MAP_SHARED, fd2, 0);
shm_t *shm3 = mmap(NULL, sizeof(shm_t), PROT_READ | PROT_WRITE, MAP_SHARED, fd3, 0);

    //создаем семафоры для каждой пары связей (empty и full)
sem_t *sem1_empty = sem_open(sem1_empty_name, O_CREAT, 0666, 1);
sem_t *sem1_full = sem_open(sem1_full_name, O_CREAT, 0666, 0);
sem_t *sem2_empty = sem_open(sem2_empty_name, O_CREAT, 0666, 1);
sem_t *sem2_full = sem_open(sem2_full_name, O_CREAT, 0666, 0);
sem_t *sem3_empty = sem_open(sem3_empty_name, O_CREAT, 0666, 1);
sem_t *sem3_full = sem_open(sem3_full_name, O_CREAT, 0666, 0);


if (fork() == 0){
    execl ("./child1", "child1", shm1_name, shm2_name, sem1_empty_name,
        sem1_full_name, sem2_empty_name, sem2_full_name, (char*)NULL);
    perror("execl child1");
    exit(1);
}

if (fork() == 0){
    execl("./child2", "child2", shm2_name, shm3_name, sem2_empty_name,
        sem2_full_name, sem3_empty_name, sem3_full_name, (char*)NULL);
    perror("execl child2");
    exit(1);
}

char input[BUF_SIZE];

while(fgets(input, BUF_SIZE, stdin)){
    size_t len = strlen(input);
    sem_wait(sem1_empty);
    memcpy(shm1->data, input, len);
    shm1->len = len;

    //можно читать child1
    sem_post(sem1_full);

    sem_wait(sem3_full);
```

```
        write(STDOUT_FILENO, shm3->data, shm3->len);
        sem_post(sem3_empty);
    }

    wait(NULL);
    wait(NULL);

    //освобождаем ресурсы памяти
    munmap(shm1, sizeof(shm_t));
    munmap(shm2, sizeof(shm_t));
    munmap(shm3, sizeof(shm_t));

    //удаляем объекты shared memory
    shm_unlink(shm1_name);
    shm_unlink(shm2_name);
    shm_unlink(shm3_name);

    //закрываем семафоры
    sem_close(sem1_empty);
    sem_close(sem1_full);
    sem_close(sem2_empty);
    sem_close(sem2_full);
    sem_close(sem3_empty);
    sem_close(sem3_full);

    //удаляем именнованные семафоры и системы
    sem_unlink(sem1_empty_name);
    sem_unlink(sem1_full_name);
    sem_unlink(sem2_empty_name);
    sem_unlink(sem2_full_name);
    sem_unlink(sem3_empty_name);
    sem_unlink(sem3_full_name);

    return 0;
}
```

Child1.c

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

```c
#include <ctype.h>


#define BUF_SIZE 1024

typedef struct{
    char data[BUF_SIZE];
    size_t len;
}shm_t;

int main(int argc, char *argv[]){
    if (argc != 7){
        fprintf(stderr, "Error: usage child1 shm1 shm2 sem1_empty sem1_full sem2_empty
sem2_full\n");
        exit(1);
    }

    char *shm1_name = argv[1];
    char *shm2_name = argv[2];
    char *sem1_empty_name = argv[3];
    char *sem1_full_name = argv[4];
    char *sem2_empty_name = argv[5];
    char *sem2_full_name = argv[6];

    //открываем shared memory
    int fd1 = shm_open(shm1_name, O_RDWR, 0666);
    int fd2 = shm_open(shm2_name, O_RDWR, 0666);

    shm_t *shm1 = mmap(NULL, sizeof(shm_t), PROT_READ | PROT_WRITE, MAP_SHARED, fd1, 0);
    shm_t *shm2 = mmap(NULL, sizeof(shm_t), PROT_READ | PROT_WRITE, MAP_SHARED, fd2, 0);

    //открываем семафоры
    sem_t *sem1_empty = sem_open(sem1_empty_name, 0);
    sem_t *sem1_full = sem_open(sem1_full_name, 0);
    sem_t *sem2_empty = sem_open(sem2_empty_name, 0);
    sem_t *sem2_full = sem_open(sem2_full_name, 0);

    while(1){

        //ждем пока родитель запишет
        sem_wait(sem1_full);

        if (shm1->len == 0) break;
```

```
        for (size_t i = 0; i < shm1->len; i++) shm2->data[i] = tolower((unsigned
char)shm1->data[i]);
        shm2->len = shm1->len;

        //освобождаем shm1 (можно снова писать)
        sem_post(sem1_empty);
        sem_post(sem2_full); //сообщаем, что в shm2 появились новые данные

    }

    shm2->len = 0;
    sem_post(sem2_full);

    munmap(shm1, sizeof(shm_t));
    munmap(shm2, sizeof(shm_t));

    return 0;

}
```

Child2.c

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

#define BUF_SIZE 1024

typedef struct{
    char data[BUF_SIZE];
    size_t len;
}shm_t;

int main(int argc, char *argv[]){
    if (argc != 7){
        fprintf(stderr, "Error: usage child2 shm2 shm3 sem2_empty sem2_full sem3_empty
sem3_full\n");
        exit(1);
    }

    char *shm2_name = argv[1];
    char *shm3_name = argv[2];
```

```
    char *sem2_empty_name = argv[3];
    char *sem2_full_name = argv[4];
    char *sem3_empty_name = argv[5];
    char *sem3_full_name = argv[6];

    //открываем shared memory
    int fd2 = shm_open(shm2_name, O_RDWR, 0666);
    int fd3 = shm_open(shm3_name, O_RDWR, 0666);

    shm_t *shm2 = mmap(NULL, sizeof(shm_t), PROT_READ | PROT_WRITE, MAP_SHARED, fd2, 0);
    shm_t *shm3 = mmap(NULL, sizeof(shm_t), PROT_READ | PROT_WRITE, MAP_SHARED, fd3, 0);

    //открываем семафоры
    sem_t *sem2_empty = sem_open(sem2_empty_name, 0);
    sem_t *sem2_full = sem_open(sem2_full_name, 0);
    sem_t *sem3_empty = sem_open(sem3_empty_name, 0);
    sem_t *sem3_full = sem_open(sem3_full_name, 0);

    while(1){
        sem_wait(sem2_full);
        if (shm2->len == 0) break;
        for (size_t i = 0; i < shm2->len; i++){
            if (shm2->data[i] == ' ' || shm2->data[i] == '\t') shm3->data[i] = '_';
            else shm3->data[i] = shm2->data[i];
        }

        shm3->len = shm2->len;

        sem_post(sem2_empty);
        sem_post(sem3_full);
    }
    shm3->len = 0;
    sem_post(sem3_full);

    munmap(shm2, sizeof(shm_t));
    munmap(shm3, sizeof(shm_t));

    return 0;
}
```

Протокол работы программы

Тестирование:

```
nastik@HUAWEI:/mnt/c/nastya/laba3 OS$ gcc child2.c -o child2 -lrt -pthread
nastik@HUAWEI:/mnt/c/nastya/laba3 OS$ ./parent
HHHHHH   LLLLL    pppp  =
hhhhhh___lllll____pppp__=_
000999  o       p       ;;;
000999_o_p_;;;
HElOLoLoLoLoLoLoL OO PP lp;L
helololololololol_oo_pp_lp;l_
```

Вывод

В ходе выполнения лабораторной работы я закрепила новые знания и навыки в области работы процессами и системными вызовами. В результате работы была реализована программа, в которой три процесса обмениваются информацией через разделяемую память с применением семафоров для синхронизации.