Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Соколов Арсений Игоревич

Группа: М8О-207Б-21

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

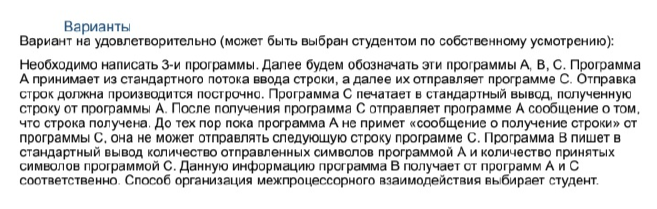
**Содержание**

1. Репозиторий
2. Постановка задачи
3. Демонстрация работы strace
4. Вывод

**Репозиторий**

https://github.com/valerasaray/os

**Постановка задачи**



**Методы и алгоритмы решения**

**a.c**

#include <fcntl.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#define BUF\_SIZE 255 // размер буфера для обмена данными между процессами

#define SHARED\_MEMORY "/shm\_file" // имя разделяемой памяти

#define S\_1 "/sem1" // имя первого семафора

#define S\_2 "/sem2" // имя второго семафора

#define S\_3 "/sem3" // имя четвертого семафора

int main()

{

int fd\_shm; // дескриптор разделяемой памяти

char\* shmem; // указатель на начало разделяемой памяти

char\* tmp = (char\*)malloc(sizeof(char) \* BUF\_SIZE); // временный буфер для чтения данных

char\* buf\_size = (char\*)malloc(sizeof(char) \* 10); // буфер для хранения размера буфера

// семафоры для синхронизации процессов

// sem\_open создает новый именованный семафор (параметр O\_CREAT).

// 0660 - задает права доступа

sem\_t\* sem1 = sem\_open(S\_1, O\_CREAT, 0660, 0);

sem\_t\* sem2 = sem\_open(S\_2, O\_CREAT, 0660, 0);

sem\_t\* sem3 = sem\_open(S\_3, O\_CREAT, 0660, 0);

if (sem1 == SEM\_FAILED || sem2 == SEM\_FAILED || sem3 == SEM\_FAILED) {

perror("ошибка sem\_open в программе A\n");

exit(1);

}

// shm\_open() создает новый именованный объект разделяемой памяти, будет открыта как чтение и запись (параметр O\_RDW)

if ((fd\_shm = shm\_open(SHARED\_MEMORY, O\_RDWR | O\_CREAT, 0660)) == -1) {

perror("ошибка shm\_open в программе A\n");

exit(1);

}

// ftruncate() устанавливает размер (BUF\_SIZE) объекта разделяемой памяти, связанного с файловым дескриптором.

if (ftruncate(fd\_shm, BUF\_SIZE) == -1) {

perror("ошибка ftruncate в программе A\n");

exit(-1);

}

/\* Создание разделяемой памяти (mmap)

NULL, указывает на то, что система сама выберет адрес для разделяемой памяти.

BUF\_SIZE определяет размер памяти

PROT\_WRITE и PROT\_READ определяют права доступа к памяти для записи и чтения соответственно.

MAP\_SHARED разделяет память между несколькими процессами

fd\_shm - это файловый дескриптор

0, указывает на то, что память начинается с начала объекта.\*/

shmem = (char\*)mmap(NULL, BUF\_SIZE, PROT\_WRITE | PROT\_READ, MAP\_SHARED, fd\_shm, 0);

// Формирование списка аргументов для запуска другого процесса

sprintf(buf\_size, "%d", BUF\_SIZE); // форматирует значение переменной BUF\_SIZE в строку и записывает ее в буфер buf\_size

char\* argv[] = {buf\_size, SHARED\_MEMORY, S\_2, S\_3, NULL}; // создает массив указателей на строки, который будет передан в функцию execvp() для запуска нового процесса

// цикл считывание входных данных со стандартного потока ввода

while (scanf("%s", tmp) != EOF) {

// создание дочернего процесса

pid\_t pid = fork();

// если процесс дочерний

if (pid == 0) {

// создание еще одного дочернего процесса pid\_1

pid\_t pid\_1 = fork();

if (pid\_1 == 0) {

sem\_wait(sem1);

printf("программа A отправила:\n");

// запускает новый процесс ./b.out

if (execve("./b.out", argv, NULL) == -1) {

perror("Не удалось выполнить в программе A\n");

}

} else if (pid\_1 > 0) {

sem\_wait(sem3); // блокирует семафор

if (execve("./c.out", argv, NULL) == -1) {

perror("Не удалось выполнить в программе A\n");

}

}

} else if (pid > 0) {

sprintf(shmem, "%s", tmp); // запись строки в разделяемую память

sem\_post(sem1); // разблокирует семафор

sem\_wait(sem2); // блокирует семафор

printf(" \n\n");

}

}

shm\_unlink(SHARED\_MEMORY); // удаляет именованный сегмент разделяемой памяти

// удаляет именованные семафоры

sem\_unlink(S\_1);

sem\_unlink(S\_2);

sem\_unlink(S\_3);

// закрывает все открытые семафоры

sem\_close(sem1);

sem\_close(sem2);

sem\_close(sem3);

// закрывает дескриптор

close(fd\_shm);

}

**b.c**

#include <fcntl.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main(int argc, char const\* argv[])

{

// Проверка количества переданных аргументов (минимум 2: размер буфера и название разделяемой памяти)

if (argc < 2) {

perror("args < 2 в программе B\n");

exit(1);

}

int buf\_size = atoi(argv[0]); // целое число, размер буфера

char const\* shared\_memory\_name = argv[1]; // название разделяемой памяти

char const\* sem3\_name = argv[3]; // название семафора для синхронизации процессов

int fd\_shm;

// открытие разделяемой памяти в режиме чтения и записи

if ((fd\_shm = shm\_open(shared\_memory\_name, O\_RDWR, 0660)) == -1) {

perror("ошибка shm\_open в программе B\n");

exit(1);

}

sem\_t\* sem3 = sem\_open(sem3\_name, 0, 0, 0); // Открытие семафора в режиме блокировки

if (sem3 == SEM\_FAILED) {

perror("ошибка sem3 в программе B\n");

exit(1);

}

char\* shmem = (char\*)mmap(NULL, buf\_size, PROT\_WRITE | PROT\_READ, MAP\_SHARED, fd\_shm, 0); // Получение доступа к разделяемой памяти

int size = strlen(shmem); // получение длины строки

printf("%d symbols\n", size);

sem\_post(sem3); // разблокирует семафор

}

**c.c**

#include <fcntl.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main(int argc, char\* const argv[])

{

// Проверка аргументов командной строки (минимум 2: размер буфера и название разделяемой памяти)

if (argc < 2) {

printf("args < 2 в программе C\n");

return 0;

}

int buf\_size = atoi(argv[0]); // размер буфера

char const\* shared\_memory\_name = argv[1]; // имя разделяемой памяти

// имена семафоров

char const\* sem2\_name = argv[2];

char const\* sem3\_name = argv[3];

int fd\_shm;

// Открытие разделяемой памяти в режиме чтения и записи

if ((fd\_shm = shm\_open(shared\_memory\_name, O\_RDWR, 0660)) == -1) {

perror("ошибка shm\_open в программе C\n");

exit(1);

}

sem\_t\* sem2 = sem\_open(sem2\_name, 0, 0, 0); // Открытие семафора 2 в режиме блокировки

sem\_t\* sem3 = sem\_open(sem3\_name, 0, 0, 0); // Открытие семафора 3 в режиме блокировки

if (sem2 == SEM\_FAILED || sem3 == SEM\_FAILED) {

perror("ошибка sem2 или sem3 в программе C\\n");

exit(1);

}

// Отображение разделяемой памяти в виртуальное адресное пространство (чтобы иметь доступ)

char\* shmem = (char\*)mmap(NULL, buf\_size, PROT\_WRITE | PROT\_READ, MAP\_SHARED, fd\_shm, 0);

// Создание нового процесса

pid\_t p = fork();

// если дочерний процесс

if (p == 0) {

printf("program C got:\n");

// запуск программы B

if (execve("b.out", argv, NULL) == -1) {

perror("ошибка execve в программе C\n");

exit(1);

}

} else if (p > 0) {

sem\_wait(sem3); // блокирует семафор

printf("%s\n", shmem);

}

sem\_post(sem2); // разблокирует семафор

}

**Вывод**

Благодаря данной лабораторной работе, я приобрел межпроцессорного взаимодействия программ.