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: _____

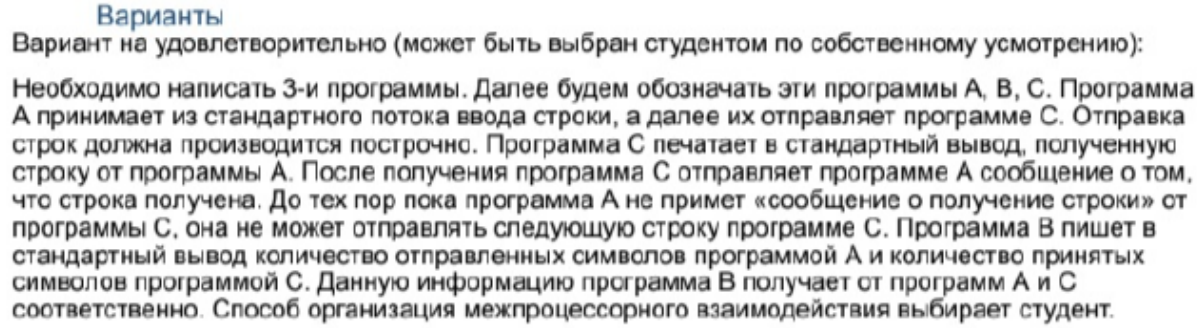
Москва, 2022

Содержание

Репозиторий

https://github.com/valerasaray/os

Постановка задачи



Варианты

Вариант на удовлетворительно (может быть выбран студентом по собственному усмотрению):

Необходимо написать 3-и программы. Далее будем обозначать эти программы A, B, C. Программа A принимает из стандартного потока ввода строки, а далее их отправляет программе C. Отправка строк должна производится построчно. Программа C печатает в стандартный вывод, полученную строку от программы A. После получения программа C отправляет программе A сообщение о том, что строка получена. До тех пор пока программа A не примет «сообщение о получение строки» от программы C, она не может отправлять следующую строку программе C. Программа B пишет в стандартный вывод количество отправленных символов программой A и количество принятых символов программой C. Данную информацию программа B получает от программ A и C соответственно. Способ организация межпроцессорного взаимодействия выбирает студент.

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.