Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

# Курсовой проект по курсу
# «Операционные системы»

Студент: Лютоев Илья Александрович
Группа: М8О-207Б-21
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

**Содержание**

1. Репозиторий
1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Исходный код
5. Демонстрация работы программы
6. Выводы

## Репозиторий

https://github.com/Uzym/OS/tree/main/cp

## Постановка задачи

Необходимо написать 3 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.

## Общие сведения о программе:

программа состоит из четырёх файлов: A.cpp, B.cpp, C.cpp и main.cpp, который объединяет в себе три предыдущих файла.

## Общий метод и алгоритм решения:

В начале работы в A.cpp создаются два дочерних процесса для B и C с помощью execl, сначала A с помощью getline считывает строку, передаёт в B количество считанных символов, а в C — количество считанных символов и саму строку посимвольно, затем B выводит количество введённых символов, C выводит строку и передаёт B количество выведенных символов, после чего B выводит количество выведенных символов и цикл начинается заново. Межпроцессорное взаимодействие основано на семафорах и пайпах.

## Исходный код:

A.cpp

```
//
// Created by lyutoev on 28.12.22.
//
#include "iostream"
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include "../include/note.h"

int sem_get(sem_t *sem)
{
    int state;
    sem_getvalue(sem, &state);
    return state;
}

int main()
{
```

```
int fdAC[2];
if (pipe(fdAC) == -1) {
    std::cerr << "pipe error\n";
    return EXIT_FAILURE;
}

int fdAB[2];
if (pipe(fdAB) == -1) {
    std::cerr << "pipe error\n";
    return EXIT_FAILURE;
}

int fdCB[2];
if (pipe(fdCB) == -1) {
    std::cerr << "pipe error\n";
    return EXIT_FAILURE;
}

sem_unlink("semA");
sem_t* semA = sem_open("semA", O_CREAT, 0777, 1);
if (semA == SEM_FAILED) {
    std::cerr << "semA error\n";
    return EXIT_FAILURE;
}
while (sem_get(semA) > START) {
    sem_wait(semA);
}

sem_unlink("semB");
sem_t* semB = sem_open("semB", O_CREAT, 0777, 0);
if (semB == SEM_FAILED) {
    std::cerr << "semB error\n";
    return EXIT_FAILURE;
}
while (sem_get(semB) > START) {
    sem_wait(semB);
}

sem_unlink("semC");
sem_t* semC = sem_open("semC", O_CREAT, 0777, 0);
if (semC == SEM_FAILED) {
    std::cerr << "semC error\n";
    return EXIT_FAILURE;
}
while (sem_get(semC) > START) {
    sem_wait(semC);
}

pid_t B_pid = fork();
if (B_pid == -1) {
    std::cerr << "fork error\n";
    return EXIT_FAILURE;
} else if (B_pid == 0) {
    execlp(
        "./B",
        std::to_string(fdAB[FD_OUTPUT]).c_str(),
        std::to_string(fdAB[FD_INPUT]).c_str(),
        std::to_string(fdCB[FD_OUTPUT]).c_str(),
        std::to_string(fdCB[FD_INPUT]).c_str(),
```

```
        NULL
    );
}

pid_t C_pid = fork();
if (C_pid == -1) {
    std::cerr << "fork error\n";
    return EXIT_FAILURE;
} else if (C_pid == 0) {
    execlp(
        "./C",
        std::to_string(fdAC[FD_OUTPUT]).c_str(),
        std::to_string(fdAC[FD_INPUT]).c_str(),
        std::to_string(fdCB[FD_OUTPUT]).c_str(),
        std::to_string(fdCB[FD_INPUT]).c_str(),
        NULL
    );
}

std::string str;
size_t size;
while (getline(std::cin, str)) {
    size = str.size();
    write(
        fdAB[FD_INPUT],
        &size,
        sizeof(size)
    );
    write(
        fdAC[FD_INPUT],
        &size,
        sizeof(int)
    );
    for (int i = 0; i < size; i ++) {
        char c = str[i];
        write(
            fdAC[FD_INPUT],
            &c,
            sizeof(char)
        );
    }
    sem_post(semB);
    sem_wait(semA);
}

while (sem_get(semC) < END) {
    sem_post(semC);
}
while (sem_get(semB) < END) {
    sem_post(semB);
}

sem_close(semA);
sem_close(semB);
sem_close(semC);

sem_destroy(semA);
sem_destroy(semB);
sem_destroy(semC);
```

```
    close(fdAB[FD_OUTPUT]);
    close(fdAB[FD_INPUT]);
    close(fdAC[FD_OUTPUT]);
    close(fdAC[FD_INPUT]);
    close(fdCB[FD_OUTPUT]);
    close(fdCB[FD_INPUT]);

    return EXIT_SUCCESS;
}
```

## B.cpp

```
//
// Created by lyutoev on 28.12.22.
//

#include "iostream"
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include "../include/note.h"

int sem_get(sem_t *sem)
{
    int state;
    sem_getvalue(sem, &state);
    return state;
}

int main(int argc, char const *argv[])
{
    int fdAB[2], fdCB[2];

    fdAB[FD_OUTPUT] = std::stoi(argv[0]);
    fdAB[FD_INPUT] = std::stoi(argv[1]);

    fdCB[FD_OUTPUT] = std::stoi(argv[2]);
    fdCB[FD_INPUT] = std::stoi(argv[3]);

    sem_t* semA = sem_open("semA", O_CREAT, 0777, 1);
    sem_t* semB = sem_open("semB", O_CREAT, 0777, 0);
    sem_t* semC = sem_open("semC", O_CREAT, 0777, 0);

    size_t sizeA, sizeC;
    while (sem_get(semB) != END) {
        sem_wait(semB);
        if (sem_get(semB) == END) {
            break;
        }
        read(fdAB[FD_OUTPUT], &sizeA, sizeof(sizeA));
        std::cout << "A: " << sizeA << "\n";

        sem_post(semC);
        sem_wait(semB);
        if (sem_get(semB) == END) {
            break;
        }
```

```
        read(fdCB[FD_OUTPUT], &sizeC, sizeof(sizeC));
        std::cout << "C: " << sizeC << "\n";

        sem_post(semA);
        if (sem_get(semB) == END) {
            break;
        }
    }

    sem_close(semA);
    sem_close(semB);
    sem_close(semC);

    close(fdAB[FD_OUTPUT]);
    close(fdAB[FD_INPUT]);
    close(fdCB[FD_OUTPUT]);
    close(fdCB[FD_INPUT]);

    return EXIT_SUCCESS;
}
```

## C.cpp

```
//
// Created by lyutoev on 28.12.22.
//
#include "iostream"
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include "../include/note.h"
int sem_get(sem_t *sem)
{
    int state;
    sem_getvalue(sem, &state);
    return state;
}
int main(int argc, char const *argv[])
{
    int fdAC[2], fdCA[2], fdCB[2];

    fdAC[FD_OUTPUT] = std::stoi(argv[0]);
    fdAC[FD_INPUT] = std::stoi(argv[1]);
    fdCB[FD_OUTPUT] = std::stoi(argv[2]);
    fdCB[FD_INPUT] = std::stoi(argv[3]);
    sem_t* semA = sem_open("semA", O_CREAT, 0777, 1);
    sem_t* semB = sem_open("semB", O_CREAT, 0777, 0);
    sem_t* semC = sem_open("semC", O_CREAT, 0777, 0);
    char c;
    int size;
    while (sem_get(semC) != END) {
        sem_wait(semC);
        if (sem_get(semC) == END) {
```

```
            break;
        }
        read(fdAC[FD_OUTPUT], &size, sizeof(int));
        std::string str;
        for (int i = 0; i < size; i ++) {
            read(fdAC[FD_OUTPUT], &c, sizeof(char));
            str.push_back(c);
        }
        std::cout << str << '\n';

        std::size_t sizeB = str.size();
        write(fdCB[FD_INPUT], &sizeB, sizeof(std::size_t));
        sem_post(semB);
    }

    sem_close(semA);
    sem_close(semB);
    sem_close(semC);
    close(fdAC[FD_OUTPUT]);
    close(fdAC[FD_INPUT]);
    close(fdCB[FD_OUTPUT]);
    close(fdCB[FD_INPUT]);
    return EXIT_SUCCESS;
}
```

### note.h

```
#ifndef _NOTE_H_
#define _NOTE_H_

#define FD_INPUT 1
#define FD_OUTPUT 0
#define STDIN 0
#define STDOUT 1

#define END 2
#define START 0

#endif
```

### CMakeLists.txt

```
cmake_minimum_required(VERSION 3.23)
project(cp)

set(CMAKE_CXX_STANDARD 23)

add_executable(A src/A.cpp)
add_executable(B src/B.cpp)
add_executable(C src/C.cpp)
```

## Демонстрация работы программы

```
lyutoev@lyutoev Ξ ~/workshop/os/OS/cp/build Ξ  main ± Ξ ./A
lrg lkrl gkrlg
```

```
A: 14
lrg lkrl gkrlg
C: 14
g ldsk gfkdgo kfdo kofk h
A: 25
g ldsk gfkdgo kfdo kofk h
C: 25
ds fldsk gkdg
A: 13
ds fldsk gkdg
C: 13
a
A: 2
a
C: 2
аб
A: 4
аб
C: 4
ab
A: 3
ab
C: 3
keokeo
A: 7
keokeo
C: 7
prlgprko
A: 8
prlgprko
C: 8
+0e
A: 3
+0e
C: 3
```

## Выводы

При создании курсового проекта я улучшил свои знания и навыки в области операционных систем.