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
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. 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 С. Отправка  
строк должна производится построчно. Программа C печатает в стандартный вывод, полученную  
строку от программы A. После получения программа C отправляет программе А сообщение о том,  
что строка получена. До тех пор, пока программа А не примет «сообщение о получение строки» от  
программы С, она не может отправлять следующую строку программе С. Программа B пишет в  
стандартный вывод количество отправленных символов программой А и количество принятых  
символов программой С. 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  ~/workshop/os/OS/cp/build   main ±  ./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

а

A: 2

а

C: 2

аб

A: 4

аб

C: 4

аb

A: 3

аb

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.