Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Зинин Владислав Владимирович

Группа: М8О-208Б-20

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**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/frankeloff>

**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 трех файлов: A.cpp, B.cpp и C.cpp.

**Общий метод и алгоритм решения:** При запуске файла А создается еще два процесса: В и С. Общение между файлами происходит с помощью pipe’ов и семафоров. Файл А считывает строку, отправляет её размер файлу В. Файл В выводит размер строки, после чего строка передается из файла А в файл С, файл С вновь выводит строку, после чего передает размер выведенной строки файлу В после чего файл А снова готов принимать новую строку.

**Исходный код:**

**A.cpp**

#include <iostream>

#include <unistd.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <signal.h>

#include <stdarg.h>

int32\_t sem\_get(sem\_t \*sem)

{

int s;

sem\_getvalue(sem, &s);

return s;

}

void sem\_set(sem\_t \*sem, int n)

{

while (sem\_get(sem) < n)

{

sem\_post(sem);

}

while (sem\_get(sem) > n)

{

sem\_wait(sem);

}

}

int main()

{

int32\_t fdAC[2]; //pipe между А и С

int32\_t fdAB[2]; //pipe между А и В

int32\_t fdBC[2]; //pipe между В и С

pipe(fdAC);

pipe(fdAB);

pipe(fdBC);

if(sem\_unlink("SemA") == -1 || sem\_unlink("SemB") == -1 || sem\_unlink("SemC") == -1)

{

std::cout << "Unlink failed" << std::endl;

}

sem\_t \*semA = sem\_open("SemA", O\_CREAT, S\_IRWXU, 1); //получаем адрес семафора,чтение, выполнение и запись

sem\_t \*semB = sem\_open("SemB", O\_CREAT, S\_IRWXU, 2);

sem\_t \*semC = sem\_open("SemC", O\_CREAT, S\_IRWXU, 2);

if ((semA == SEM\_FAILED)||(semB == SEM\_FAILED)||(semC == SEM\_FAILED))

{

std::cout << "SEM\_FAILED" << std::endl;

return 0;

}

std::cout << "Enter some strings:\n";

pid\_t A;

switch(A = fork())

{

case -1:

std::cout << "Error fork" << std::endl;

return -1;

break;

case 0:

pid\_t C;

switch(C = fork())

{

case -1:

std::cout << "Error fork" << std::endl;

return -1;

break;

case 0:

execl("B", std::to\_string(fdAB[0]).c\_str(), std::to\_string(fdBC[0]).c\_str(), NULL);

break;

default:

execl("C", std::to\_string(fdAC[0]).c\_str(), std::to\_string(fdBC[1]).c\_str(), NULL);

break;

}

break;

default:

while(1)

{

while (sem\_get(semA) == 2)

{

continue;

}

std::cout << "A process" << std::endl;

std::string str;

std::cin >> str;

if (str == "exit")

{

sem\_set(semA, 0);

sem\_set(semB, 0);

sem\_set(semC, 0);

sem\_close(semA);

sem\_close(semB);

sem\_close(semC);

sem\_destroy(semA);

sem\_destroy(semB);

sem\_destroy(semC);

close(fdAC[0]);

close(fdAC[1]);

close(fdAB[0]);

close(fdAB[1]);

close(fdBC[0]);

close(fdBC[1]);

return 0;

}

int size = str.length();

write(fdAC[1], &size, sizeof(int));

write(fdAB[1], &size, sizeof(int));

for (int i = 0; i < size; ++i)

{

write(fdAC[1], &str[i], sizeof(char));

}

sem\_set(semB, 1);

sem\_set(semA, 2);

}

break;

}

}

**B.cpp**

#include <iostream>

#include <unistd.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <stdarg.h>

#include <signal.h>

int32\_t sem\_get(sem\_t \*sem)

{

int s;

sem\_getvalue(sem, &s);

return s;

}

void sem\_set(sem\_t \*sem, int n)

{

while (sem\_get(sem) < n)

{

sem\_post(sem);

}

while (sem\_get(sem) > n)

{

sem\_wait(sem);

}

}

int main(int args, char\* argv[])

{

int32\_t fdAB[2];

int32\_t fdBC[2];

fdAB[0] = atoi(argv[0]);

fdBC[0] = atoi(argv[1]);

sem\_t \*semA = sem\_open("SemA", O\_CREAT, S\_IRWXU, 1);

sem\_t \*semB = sem\_open("SemB", O\_CREAT, S\_IRWXU, 2);

sem\_t \*semC = sem\_open("SemC", O\_CREAT, S\_IRWXU, 2);

if ((semA == SEM\_FAILED)||(semB == SEM\_FAILED)||(semC == SEM\_FAILED))

{

std::cout << "SEM\_FAILED" << std::endl;

return 0;

}

while(1)

{

while(sem\_get(semB) == 2)

{

continue;

}

if (sem\_get(semB) == 0)

{

sem\_close(semA);

sem\_close(semB);

sem\_close(semC);

sem\_destroy(semA);

sem\_destroy(semB);

sem\_destroy(semC);

close(fdAB[0]);

close(fdBC[0]);

exit(0);

}

int size;

read(fdAB[0], &size, sizeof(int));

std::cout << "B process: Number of input symbols is " << size << std::endl;

sem\_set(semC, 1);

sem\_set(semB, 2);

while (sem\_get(semB) == 2)

{

continue;

}

if (sem\_get(semB) == 0)

{

sem\_close(semA);

sem\_close(semB);

sem\_close(semC);

sem\_destroy(semA);

sem\_destroy(semB);

sem\_destroy(semC);

close(fdAB[0]);

close(fdBC[0]);

exit(0);

}

read(fdBC[0], &size, sizeof(int));

std::cout << "B process: Number of output symbols is " << size << std::endl;

sem\_set(semA, 1);

sem\_set(semB, 2);

}

}

**C.cpp**

#include <iostream>

#include <unistd.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <stdarg.h>

#include <signal.h>

int32\_t sem\_get(sem\_t \*sem)

{

int s;

sem\_getvalue(sem, &s);

return s;

}

void sem\_set(sem\_t \*sem, int n)

{

while (sem\_get(sem) < n)

{

sem\_post(sem);

}

while (sem\_get(sem) > n)

{

sem\_wait(sem);

}

}

int main(int args, char\* argv[])

{

int32\_t fdAC[2];

int32\_t fdBC[2];

fdAC[0] = atoi(argv[0]);

fdBC[1] = atoi(argv[1]);

sem\_t \*semA = sem\_open("SemA", O\_CREAT, S\_IRWXU, 1);

sem\_t \*semB = sem\_open("SemB", O\_CREAT, S\_IRWXU, 2);

sem\_t \*semC = sem\_open("SemC", O\_CREAT, S\_IRWXU, 2);

if ((semA == SEM\_FAILED)||(semB == SEM\_FAILED)||(semC == SEM\_FAILED))

{

std::cout << "SEM\_FAILED" << std::endl;

return 0;

}

while(1)

{

while(sem\_get(semC) == 2)

{

continue;

}

if (sem\_get(semC) == 0)

{

sem\_close(semA);

sem\_close(semB);

sem\_close(semC);

sem\_destroy(semA);

sem\_destroy(semB);

sem\_destroy(semC);

close(fdAC[0]);

close(fdBC[1]);

exit(0);

}

int size;

std::string str;

read(fdAC[0], &size, sizeof(int));

int count\_size = 0;

for (int i = 0; i < size; ++i)

{

char c;

read(fdAC[0], &c, sizeof(char));

str.push\_back(c);

count\_size++;

}

std::cout << "C process " << str << std::endl;

write(fdBC[1], &count\_size, sizeof(int));

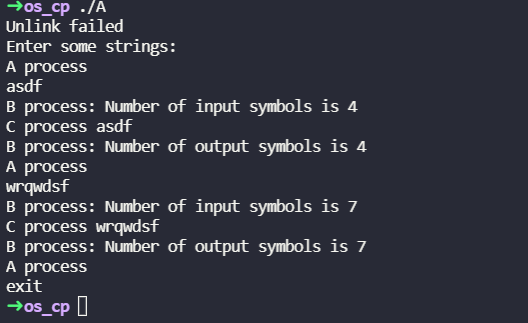
sem\_set(semB, 1);

sem\_set(semC, 2);

}

}

**Демонстрация работы программы**



**Вывод**

В результате выполнения курсового проекта я закрепил знания об использовании пайпов и семафоров.