Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**РАБОТА С ОБЩЕЙ ПАМЯТЬЮ**

Студент: Климов Иван Павлович

Группа: М8О–212Б–22

Вариант: 19

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Приобретение практических навыков в:

1. Освоение принципов работы с файловыми системами
2. Обеспечение обмена данных между процессами посредством технологии «File mapping»

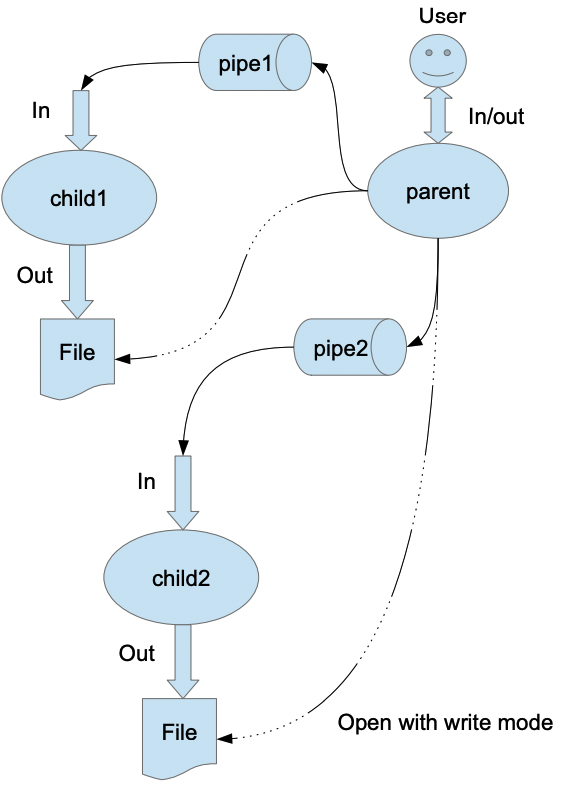
## Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы

Формулировка задания по варианту:

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Вариант 19) Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы удаляют все гласные из строк.



**Общие сведения о программе**

Программа компилируется из файла main.c, child1.c .Внутри main.c 3 функции:

1. int main(), создающая дочерние процессы, семафоры и т. д.
2. int create\_process() - создаёт процесс с помощью форк и возвращает айди процесса.

Внутри child 1 функция:

1. int generatir() – дя генерации чисел
2. int main(int argc, char\* argv[]) – принимает аргументы, передающиеся ей через execl и обрабатывает строку.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Более подробно изучить принципы работы семафоров. Разобраться в средствах работы с общей памятью (mmap, shm и т.д.)
2. Создать файлы через open(), как в первой лаб.оаб. №1.
3. Создать объект разделяемой памяти с помощью shm\_open, дабы именовать область виртуальной памяти, установить нужный нам размер этой памяти с помощью ftruncate и получить доступ к разделяемой памяти с помощью mmap.
4. Создать три семафора, которые будут ограничивать работу процессов main, child1
5. Создать дочерние процессы при помощи execl и передать в них нужные имена семафоров и имя объекта разделяемой памяти
6. Обработать внутри дочерних процессов строки и на ряду с этим увеличивать и ожидать сброса значения семафоров.
7. Записать полученные строки в нужные файлы.

main.c

#include "stdio.h"

#include "stdlib.h"

#include "unistd.h"

#include "sys/wait.h"

#include "string.h"

#include "sys/types.h"

#include "sys/stat.h"

#include "fcntl.h"

#include "sys/mman.h"

#include "semaphore.h"

#include <time.h>

const int STRING\_SIZE = 100;

int create\_process();

int generator();

int main(){

int i = 1;

srand(time(NULL));

sem\_unlink("/semafor\_pr");

sem\_unlink("/semafor\_ch1");

sem\_unlink("/semafor\_ch2");

write(1, "Создайте 2 файла\n", sizeof(char) \* strlen("Создайте 2 файла\n"));

char file\_name1[STRING\_SIZE], file\_name2[STRING\_SIZE];

read(STDIN\_FILENO, file\_name1, STRING\_SIZE);

read(STDIN\_FILENO, file\_name2, STRING\_SIZE);

int filenam1 = open(file\_name1, O\_RDWR | O\_CREAT | O\_TRUNC, S\_IRWXU);

int filenam2 = open(file\_name2, O\_RDWR | O\_CREAT | O\_TRUNC, S\_IRWXU);

///////////////////////////////////////////////////////////////////////////////

const char\* mmaped\_file\_name = "mmaped\_file";

int mmaped\_fd;

char\* mmaped\_fp;

mmaped\_fd = shm\_open(mmaped\_file\_name, O\_RDWR | O\_CREAT, 0777);

ftruncate(mmaped\_fd, STRING\_SIZE); //размер разделяемой памяти

mmaped\_fp = mmap(NULL, STRING\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, mmaped\_fd, 0);//возвращает указатель на начало отображенной области памяти

sem\_t\* semaphores[3];

semaphores[0] = sem\_open("/semafor\_pr", O\_CREAT, 0777, 1);

semaphores[1] = sem\_open("/semafor\_ch1", O\_CREAT, 0777, 0);

semaphores[2] = sem\_open("/semafor\_ch2", O\_CREAT, 0777, 0);

// int tmp;

// sem\_getvalue(semaphores[0], &tmp);

// printf("tmp: %d", tmp);

pid\_t pid\_1 = create\_process();

if(pid\_1 == 0){//ch1

dup2(filenam1, 1);

close(filenam1);

execl("child","", mmaped\_file\_name, "/semafor\_ch1", "/semafor\_pr", NULL);

perror("exec");

exit(-1);

} else {

pid\_t pid\_2 = create\_process();

if(pid\_2 == 0){//ch2

dup2(filenam2, 1);

close(filenam2);

execl("child", "", mmaped\_file\_name, "/semafor\_ch2", "/semafor\_pr", NULL);

perror("exec");

exit(-1);

} else {//prnt

char new\_string[STRING\_SIZE];

write(1, "Введите строки\n", sizeof(char) \* strlen("Введите строки\n"));

while(fgets(new\_string, STRING\_SIZE, stdin)){

//if(i == 4){ break;}

int r = generator();

if(r < 81){

sem\_wait(semaphores[0]);

strcpy(mmaped\_fp, new\_string);

mmaped\_fp[strlen(new\_string)] = 0;

sem\_post(semaphores[1]);

}else{

sem\_wait(semaphores[0]);//frezz ch2

strcpy(mmaped\_fp, new\_string);

mmaped\_fp[strlen(new\_string)] = 0;

sem\_post(semaphores[2]);

}

//printf("%d, %d\n", i, r);

i++;

}

printf("10\n");

sem\_wait(semaphores[0]);

mmaped\_fp[0] = 0;

sem\_post(semaphores[1]); sem\_post(semaphores[2]);

printf("10\n");

wait(NULL);

printf("END\n");

close(filenam1);

close(filenam2);

if(munmap(mmaped\_fp, STRING\_SIZE) == -1){//закрываем разделяемую память

perror("munmap");

}

if(close(mmaped\_fd) == -1){//закрываем объеект раз. памяти

perror("close");

}

if(shm\_unlink(mmaped\_file\_name) == -1){ //удаляем объект РП из Файловой системы.

perror("shm\_unlink");

}

}

}

return 0;

}

int create\_process() {

pid\_t pid = fork();

if (-1 == pid)

{

perror("Error while fork");

}

return pid;

}

int generator(){

int rng = rand() % 100;

return rng;

}

child.c

#include "stdio.h"

#include "stdlib.h"

#include "unistd.h"

#include "sys/wait.h"

#include "string.h"

#include "sys/types.h"

#include "sys/stat.h"

#include "fcntl.h"

#include "sys/mman.h"

#include "semaphore.h"

const int MAX\_SIZE = 100;

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

char mmap\_file\_name[MAX\_SIZE];

strcpy(mmap\_file\_name, argv[1]);

char semaphor\_name[MAX\_SIZE];

char semaphor\_pr\_name[MAX\_SIZE];

strcpy(semaphor\_name, argv[2]);

strcpy(semaphor\_pr\_name, argv[3]);

int mmap\_f\_d = shm\_open(mmap\_file\_name, O\_RDWR | O\_CREAT, 0777);

ftruncate(mmap\_f\_d, MAX\_SIZE);

char\* mmap\_f\_pointer = mmap(NULL, MAX\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, mmap\_f\_d, 0);

sem\_t\* semaphor = sem\_open(semaphor\_name, 0);

sem\_t\* semaphor\_pr = sem\_open(semaphor\_pr\_name, 0);

write(1, "``Начало работы``\n", sizeof(char) \* strlen("``Начало работы``\n"));

char vowels[] = {'a', 'e', 'i', 'o', 'u', 'y'};

char string[MAX\_SIZE];

while(1){

// printf("Жду\n");

sem\_wait(semaphor);

// printf("Дождался\n");

for (int i = 0; i < strlen(mmap\_f\_pointer); ++i) {

string[i] = mmap\_f\_pointer[i];

}

string[ strlen(mmap\_f\_pointer) ] = 0;

sem\_post(semaphor\_pr);

if(strlen(string) == 0){ break; }

for (int index = 0; index < strlen(string); ++index) {

if (memchr(vowels, string[index], 6) == NULL) {

write(1, &string[index], 1);

}

}

}

munmap(mmap\_f\_pointer, 0);

sem\_close(semaphor);

sem\_close(semaphor\_pr);

return 0;

}

Пример работы:

ivanklimov@MacBook-Air-Ivan-2 src % gcc child.c -o child

ivanklimov@MacBook-Air-Ivan-2 src % gcc main.c -o main

ivanklimov@MacBook-Air-Ivan-2 src % ./main

Создайте 2 файла

1

2

Введите строки

qwerty

asd

sdf

fdgdfgd

fdggdfdf^D

ivanklimov@MacBook-Air-Ivan-2 src %

**Вывод**

В течение выполнения данной лабораторной работы я научился работать с семафорами на практике, я очень хорошо понял, как работают механизмы разделяемой памяти. Такой системный вызов как mmap позволяет выделять часть виртуальной памяти под свои нужды. Используя этот вызов в связке с функцией shm\_open можно выделить область памяти и дать ей имя, благодаря чему можно будет обращаться к этой области из разных процессов. Благодаря /dev/shm папке, например, можно узнать, что не все объекты были удалены после выполнения программы с помощью unlink