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

Лабораторная работа №4 по курсу «Операционные системы»

Студент: Соколов Арсений Игоревич
Группа: М8О-207Б-21
Вариант: 17
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/valerasaray/os

Постановка задачи

Цель работы

Приобретение практических навыков в управлении процессами в ОС, обеспечение обмена данных между процессами посредством каналов.

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).

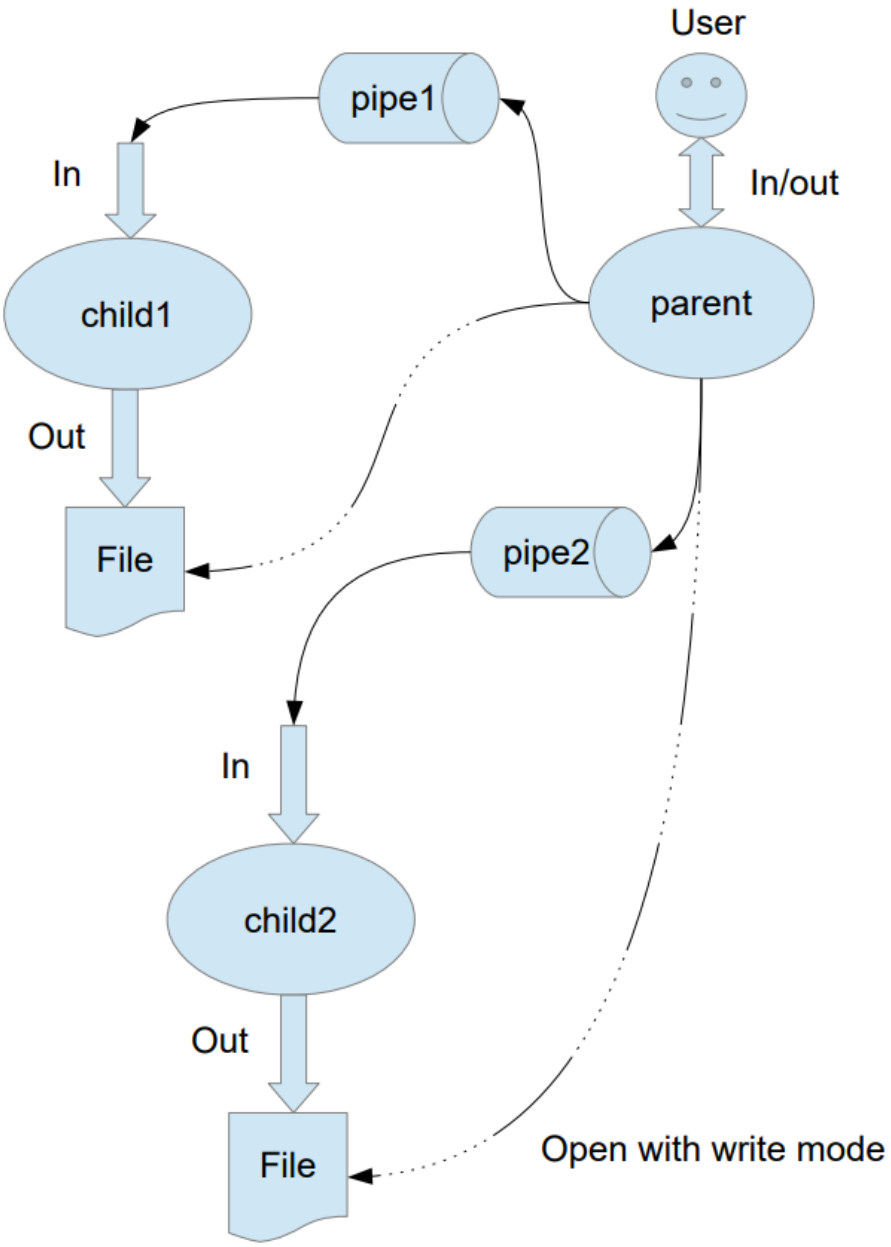
Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Группа вариантов 5

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 17

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы удаляют все гласные из строк.



Общие сведения о программе

Программа компилируется из файла main.cpp, child.cpp.

Общий метод и алгоритм решения

Пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на запись для дочернего процесса. Аналогично для второй строки и второго дочернего процесса. Родительский процесс принимает от пользователя строки произвольной длины и пересылает строки длины больше 10 символов в pipe2, иначе в pipe1. Дочерние процессы удаляют все гласные из строк и пишут результаты своей работы в стандартный вывод. Управление доступам к общим ресурсам между несколькими ресурсами реализуем через механизм синхронизации с использованием семафора, изменяя его значение.

Исходный код

main.cpp

```
#include <iostream>
#include <string>
#include <mutex>
#include <algorithm>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <fcntl.h>
#include <errno.h>
#include <fstream>
#include <string.h>

using namespace std;

int main()
{
   string current_str;
   int child_tag;
   fstream res_file;
   string child1, child2;
   cout << "Введите имя для первого дочернего файла: ";
   cin >> child1;
   cout << "Введите имя для второго дочернего файла: ";
   cin >> child2;

   shm_unlink("1.back");
   shm_unlink("2.back");
   sem_unlink("_sem1");
   sem_unlink("_sem2");
   sem_t *sem1 = sem_open("_sem1",  O_CREAT, S_IWUSR | S_IRUSR |
S_IRGRP | S_IROTH, 0);
   sem_t *sem2 = sem_open("_sem2",  O_CREAT, S_IWUSR | S_IRUSR |
S_IRGRP | S_IROTH, 0);
   int state = 0;
   std::mutex mx;
```

```
    mx.lock();
    while (++state < 2) {
        sem_post(sem1);
    }
    while (--state > 1) {
        sem_wait(sem1);
    }
    mx.unlock();
    state = 0;
    mx.lock();
    while (++state < 2) {
        sem_post(sem2);
    }
    while (--state > 1) {
        sem_wait(sem2);
    }
    mx.unlock();
    pid_t f_id1 = fork();
    if (f_id1 == -1)
    {
        cout << "Ошибка fork с кодом -1, возвращенным в
родительском процессе, child1 не создан" << endl;
        exit(EXIT_FAILURE);
    }
    else if (f_id1 == 0)
    {
        sem_close(sem1);
        string child = child1;
        execlp("./child", child.c_str(), "1.back", "_sem1",
NULL);
        perror("Ошибка execlp");
    }

    pid_t f_id2 = fork();
    if (f_id2 == -1)
    {
        cout << "Ошибка fork с кодом -1, возвращенным в
родительском процессе, child2 не создан" << endl;
```

```
        exit(EXIT_FAILURE);
    }
    else if (f_id2 == 0)
    {
        sem_close(sem2);
        string child = child2;
        execlp("./child", child.c_str(), "2.back", "_sem2",
NULL);
        perror("Ошибка execlp");
    }

    else
    {
        while (getline(std::cin, current_str))
        {
            int s_size = current_str.size() + 1;
            char *buffer = (char *) current_str.c_str();
            if (current_str.size() <= 10)
            {
                int fd = shm_open("1.back", O_RDWR | O_CREAT,
S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH);
                ftruncate(fd, s_size);
                char *mapped = (char *) mmap(NULL, s_size,
PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
                memset(mapped, '\0', s_size);
                sprintf(mapped, "%s", buffer);
                munmap(mapped, s_size);
                close(fd);
                sem_wait(sem1);
            }
            else
            {
                int fd = shm_open("2.back", O_RDWR | O_CREAT,
S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH);
                ftruncate(fd, s_size);
                char *mapped = (char *) mmap(NULL, s_size,
PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
                memset(mapped, '\0', s_size);
```

```
                sprintf(mapped, "%s", buffer);
                munmap(mapped, s_size);
                close(fd);
                sem_wait(sem2);
            }
        }
    }
    return 0;
}
```

child.cpp

```
#include <iostream>
#include <string>
#include <algorithm>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <fstream>
#include <set>

using namespace std;

int main(int argc, char const *argv[])
{
    char *semFile = (char *) argv[2];
    sem_t *sem = sem_open(semFile,  O_CREAT, S_IWUSR | S_IRUSR |
S_IRGRP | S_IROTH, 0);
    std::string vovels = "aoueiy";
    std::set<char> volSet(vovels.begin(), vovels.end());
    string filename = argv[0];
    fstream cur_file;
    cur_file.open(filename, fstream::in | fstream::out |
fstream::app);
    char *backfile = (char *) argv[1];
```

```
    int state;
    while (1)
    {
        sem_getvalue(sem, &state);
        if (state == 0) {
            int fd = shm_open(backfile, O_RDWR | O_CREAT, S_IWUSR
| S_IRUSR | S_IRGRP | S_IROTH);
            struct stat statBuf;
            fstat(fd, &statBuf);
            int size_of_str = statBuf.st_size;
            ftruncate(fd, size_of_str);
            char *mapped = (char *) mmap(NULL, size_of_str,
PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
            std::string allocated = mapped;
            string result_str;
            for (int i = 0; i < size_of_str; i++) {
                if (volSet.find(std::tolower(allocated[i])) ==
volSet.cend()) {
                    result_str.push_back(allocated[i]);
                }
            }
            cur_file << result_str << endl;
            close(fd);
            munmap(mapped, sizeof(int));
            sem_post(sem);
        }
    }

}
```

Демонстрация работы программы

```
[valerasaray@valerasaray build]$ ./main
Enter the name for first child file: child1
Enter the name for second child file: child2
svavsevvsvsveva
ppfodpooooodpos
dfopsoeee
evbe
epeeppprpppddfffde
[valerasaray@valerasaray build]$ cat child1

dfps
vb
[valerasaray@valerasaray build]$ cat child2
svvsvvsvsvv
ppfdpdps
pppprpppddfffd
```

Выводы

В ходе выполнения лабораторной работы №4 я приобрел практические навыки в управлении процессами в ОС, и в обеспечении обмена данных между процессами посредством каналов с использованием семафоров.