

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

**Лабораторная работа №4 по курсу
«Операционные системы»**

**Тема работы
Обмена данными между процессами посредством
технологии «File mapping»**

Студент: Полонский Кирилл Андреевич
Группа: М8О-208Б-20
Вариант: 15
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/kirillpolonskii/OS/tree/master/os_lab4

Постановка задачи

Родительский процесс получает на вход имя файла, использующегося для записи, создаёт два отображаемых файла, два семафора и получает строки произвольной длины, посимвольно отображая их в общую память. Дочерний процесс проверяет строки на соответствие правилу "Строка должна начинаться с маленькой буквы"; правильные строки выводятся в стандартный поток вывода дочернего процесса, неправильные через второй отображаемый файл посылаются обратно в дочерний процесс и выводятся в стандартный поток вывода родительского процесса.

Общие сведения о программе

Файл lab4.cpp содержит весь исходный код двух процессов, сборка осуществляется с помощью утилиты make. В Makefile описаны флаги -fsanitize=address и -g для отслеживания ошибок и строк, в которых они возникают, а также флаг -pthread для использования семафоров.

Общий метод и алгоритм решения

Программа принимает на вход имя файла для вывода, создаёт отображаемые файлы memDataCheck для передачи строк из родительского процесса в дочерний и memDataError для передачи строк обратно. Строки передаются по одному символу, дочерний процесс просматривает каждый символ и,

встречая перенос строки, проверяет первый её символ на принадлежность отрезку чисел в ASCII коде. Проверив все символы, дочерний процесс записывает строки, начинающиеся с маленькой буквы, в memDataError. Родительский процесс просматривает его, формирует строки и выводит их в поток вывода.

Два семафора sem_par и sem_child позволяют организовать синхронную работу процессов. Дочерний процесс ждёт, пока родительский закончит пересылать строки и увеличит значение семафора sem_par. Родительский процесс ждёт, пока дочерний закончит пересылать строки и увеличит sem_child.

Исходный код

lab4.cpp

```
#include <iostream>
#include "stdio.h"
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <string>
#include "string.h"
#include "semaphore.h"
#include <fcntl.h>
#include "unistd.h"

#define END_CODE 4

int main() {
    std::cout << "Enter outFile name:" << std::endl;
    std::string outFile_name;
    std::cin >> outFile_name;
    std::cout << "Enter amount of strings:" << std::endl;
    int strAmount;
    std::cin >> strAmount;
    int outFile;
    if ((outFile = open(outFile_name.c_str(), O_WRONLY | O_TRUNC | O_CREAT, S_IWUSR | S_IRUSR)) == -1){
        return 1;
    }
}
```

```

    int fdCheck = shm_open("/fileCheck", O_RDWR | O_CREAT, S_IWUSR | S_IRUSR); // file for
passing to child
    int fdError = shm_open("/fileError", O_RDWR | O_CREAT, S_IWUSR | S_IRUSR); // file for passing
to parent
    if (fdCheck == -1 || fdError == -1){
        return 1;
    }

    char* memDataCheck = (char*) mmap(NULL, 4096, PROT_READ | PROT_WRITE,
MAP_SHARED, fdCheck, 0);
    char* memDataError = (char*) mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_SHARED,
fdError, 0);
    if (memDataCheck == MAP_FAILED || memDataError == MAP_FAILED){
        return 1;
    }
    sem_unlink("sem_par");
    sem_unlink("sem_child");
    sem_t* sem_par = sem_open("sem_par", O_CREAT, S_IWUSR | S_IRUSR, 0);
    if (sem_par == SEM_FAILED){
        std::cout << "Error in sem_open par\n";
        perror("par");
    }
    sem_t* sem_child = sem_open("sem_child", O_CREAT, S_IWUSR | S_IRUSR, 0);
    if (sem_child == SEM_FAILED){
        std::cout << "Error in sem_open child\n";
    }

    int pid = fork();    // When fork () is called, two completely identical processes arise.
                        // All code after the fork () is executed twice, both in the child
and parent processes
    switch (pid) {
        case -1: {
            std::cout << "Fork has errors.\n";
            return -1;
        }
        case 0: { // It's child process. Now we need to decide on the direction of data transfer -
            // if we need to transfer data from parent to child, then the parent closes the descriptor
            // for reading, and the child closes the descriptor for writing

            //usleep(100000);
            int val;

            if (sem_wait(sem_par) == -1){
                std::cout << "CHILD: error in sem_par\n";
            }
            ftruncate(fdError, 1);
            dup2(outFile, STDOUT_FILENO);

            int i = 0, j = 0, indForErr = 0;
            while (i < strAmount){

```

```

        if (memDataCheck[j] == '\n'){
            ++i;

        }
        else if (memDataCheck[j] >= 65 && memDataCheck[j] <= 90) {
            std::string message("Correct string: ");
            while(memDataCheck[j] != '\n'){
                char curSymb = memDataCheck[j];
                message.push_back(curSymb);
                ++j;
            }
            ++i;
            ++j;
            message.push_back('\n');
            std::cout << message;
        } else {
            while(memDataCheck[j] != '\n'){
                char curSymb = memDataCheck[j];

                memDataError[indForErr] = curSymb;
                ++indForErr;
                ++j;
            }
            memDataError[indForErr] = '\n';
            ++i;
            ++indForErr;
            ++j;

        }
    }
    memDataError[indForErr] = (char) END_CODE;

    sem_post(sem_child);
    munmap(memDataCheck, 4096);
    munmap(memDataError, 4096);
    close(outFile);
    sem_close(sem_par);
    sem_close(sem_child);
    break;
}
default: { // It's a parent process.
    ftruncate(fdCheck, 1);
    std::cout << "Enter " << strAmount << " strings:" << "\n";
    int i = 0, j = 0;
    int prevStrLen = 0;
    while (i < strAmount){

        if (j == 0) getchar();
        char curSymb;
        scanf("%c", &curSymb);

```

```

        memDataCheck[j] = curSymb;

        if (curSymb == '\n'){
            ++i;

        }

        ++j;

    }
    memDataCheck[j] = (char) END_CODE;

    int val;
    sem_post(sem_par);
    if (sem_wait(sem_child) == -1){
        std::cout << "PARENT: error in sem_child\n";
    }

    i = 0;
    j = 0;
    while(memDataError[j] != END_CODE){

        std::string msgInFile("Incorrect string: ");
        std::string message = "In Parent: <";
        while(memDataError[j] != '\n'){
            char curSymb = memDataError[j];
            message.push_back(curSymb);
            msgInFile.push_back(curSymb);
            ++j;
        }
        message += "> doesn't start with capital\n";
        std::cout << message;
        msgInFile += "\n";
        write(outFile, msgInFile.c_str(), msgInFile.size());

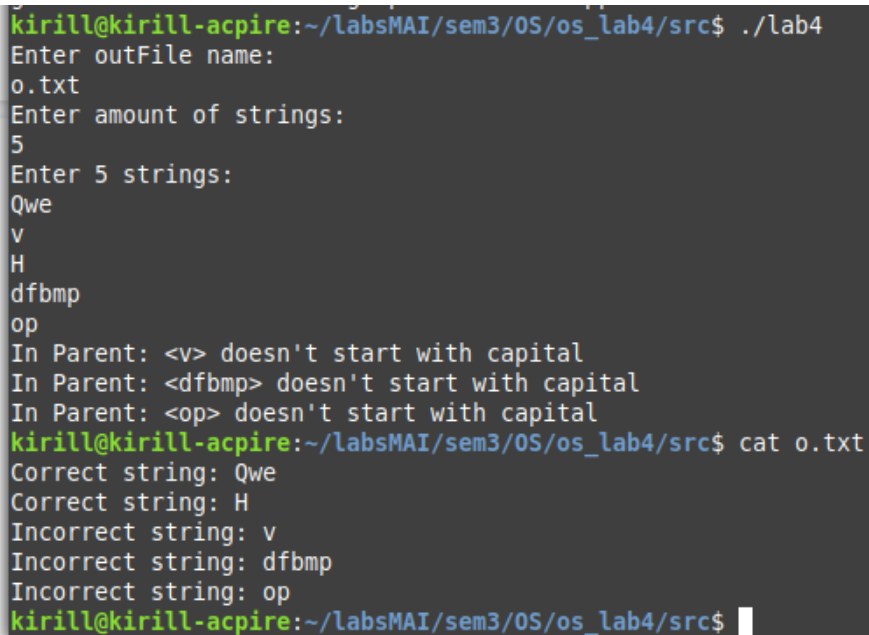
        ++j;

    }
    close(outFile);
    munmap(memDataCheck, 4096);
    munmap(memDataError, 4096);
    sem_close(sem_par);
    sem_close(sem_child);
}

}
return 0;
}

```

Демонстрация работы программы



```
kirill@kirill-acpire:~/labsMAI/sem3/OS/os_lab4/src$ ./lab4
Enter outFile name:
o.txt
Enter amount of strings:
5
Enter 5 strings:
Qwe
v
H
dfbmp
op
In Parent: <v> doesn't start with capital
In Parent: <dfbmp> doesn't start with capital
In Parent: <op> doesn't start with capital
kirill@kirill-acpire:~/labsMAI/sem3/OS/os_lab4/src$ cat o.txt
Correct string: Qwe
Correct string: H
Incorrect string: v
Incorrect string: dfbmp
Incorrect string: op
kirill@kirill-acpire:~/labsMAI/sem3/OS/os_lab4/src$
```

Выводы

В ходе выполнения лабораторной работы я улучшил свои знания о процессах в Linux и связанных с ними системными вызовами, а также научился межпроцессорному взаимодействию через отображаемые файлы.