Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Управление процессами в ОС. Обеспечение обмена данных между процессами посредством технологии “File mapping”.**

Студент: Чистяков Д. М.

Группа: М80-206Б-20

Вариант: 20

Преподаватель: Соколов Андрей Алексеевич

Дата: 07.03.2022

Оценка:

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Постановка задачи**

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 memory-mapped files.

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

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.

20 вариант: Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

**Листинг программы**

**child1.c**

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <stdlib.h>

#include <stdbool.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <string.h>

#define MAX\_BUF 65534

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

{

    if (argc != 2)

    {

        perror("Wrong number of arguments\n");

        return 1;

    }

    char\* tmp\_name = argv[0];

    int tmp = open(tmp\_name, O\_RDWR|O\_APPEND, S\_IRWXU);

    if (tmp < 0)

    {

        perror("Problems with file\n");

        return 1;

    }

    struct stat buff;

    stat(tmp\_name, &buff);

    char\* buf = mmap(0, buff.st\_size, PROT\_READ|PROT\_WRITE, MAP\_SHARED, tmp, 0);

    close(tmp);

    int i = 0;

    while (i < buff.st\_size)

    {

        char\* str = malloc(MAX\_BUF);

        int ind = 0;

        while (buf[i] != '\n')

        {

            str[ind] = buf[i];

            ind++;

            i++;

        }

        str[ind] = '\n';

        i++;

        int l = strlen(str);

        str[l] = '\0';

        for (int j = 0; j <= (l - 2) / 2; j++)

        {

            char k = str[l - 2 - j];

            str[l - 2 - j] = str[j];

            str[j] = k;

        }

        write(1, str, sizeof(char) \* strlen(str));

    }

    close(tmp);

    remove(tmp\_name);

    return 0;

}

**main.c**

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <stdlib.h>

#include <stdbool.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <string.h>

#define MAX\_BUF 65534

int main()

{

    char\* fname1;

    char\* fname2;

    fname1 = malloc(MAX\_BUF);

    if (read(0, fname1, MAX\_BUF) < 0)

    {

        perror("ERROR\n");

        return 1;

    }

    int l1 = strlen(fname1);

    fname1[l1 - 1] = '\0';

    fflush(stdout);

    int f1 = open(fname1, O\_CREAT|O\_RDWR|O\_APPEND, S\_IRWXU);

    if (f1 < 0)

    {

        perror("Problems with file\n");

    return 1;

    }

    fname2 = malloc(MAX\_BUF);

    if (read(0, fname2, MAX\_BUF) < 0)

    {

        close(f1);

        perror("ERROR\n");

        return 1;

    }

    int l2 = strlen(fname2);

    fname2[l2 - 1] = '\0';

    fflush(stdout);

    if (strcmp(fname1, fname2) == 0)

    {

        close(f1);

        perror("The files have the same names\n");

    return 1;

    }

    int f2 = open(fname2, O\_CREAT|O\_RDWR|O\_APPEND, S\_IRWXU);

    if (f2 < 0)

    {

        close(f1);

        perror("Problems with file\n");

        return 1;

    }

    int tmp1 = open("tmp\_name1", O\_CREAT|O\_RDWR|O\_APPEND, S\_IRWXU);

    if (tmp1 < 0)

    {

        close(f1);

        close(f2);

        perror("Problems with file\n");

        return 1;

    }

    int tmp2 = open("tmp\_name2", O\_CREAT|O\_RDWR|O\_APPEND, S\_IRWXU);

    if (tmp2 < 0)

    {

        close(f1);

        close(f2);

        remove("tmp\_name1");

        perror("Problems with file\n");

        return 1;

    }

    char c = ' ';

    while (c != EOF)

    {

        char\* str;

        str = malloc(MAX\_BUF);

        int i = 0;

        c = ' ';

        bool is\_eof = false;

        while (c != '\n')

        {

            if (read(0, &c, sizeof(char)) < 1)

            {

                is\_eof = true;

                break;

            }

            str[i] = c;

            i++;

        }

        int l = strlen(str);

        str[l] = '\0';

        if (l-1 > 10)

        {

            write(tmp2, str, sizeof(char) \* strlen(str));

        }

        else

        {

            write(tmp1, str, sizeof(char) \* strlen(str));

        }

        if (is\_eof)

        {

            break;

        }

    }

    int pid1 = fork();

    if (pid1 == -1)

    {

        close(f1);

        close(f2);

        remove("tmp\_name1");

        remove("tmp\_name2");

        perror("Fork error\n");

        return 1;

    }

    if (pid1 == 0)

    {

        close(tmp1);

        if (dup2(f1, 1) < 0)

        {

            close(f1);

            close(f2);

            remove("tmp\_name1");

            remove("tmp\_name2");

            perror("Can't dup child.out to stdout");

            return 2;

        }

        if (execl("child.out", "tmp\_name1", "", NULL) == -1)

        {

            close(f1);

            close(f2);

            remove("tmp\_name1");

            remove("tmp\_name2");

            perror("Execl child problem");

            return 2;

        }

    }

    int pid2 = fork();

    if (pid2 == -1)

    {

        close(f1);

        close(f2);

        remove("tmp\_name1");

        remove("tmp\_name2");

        perror("Fork error\n");

        return 1;

    }

    if (pid2 == 0)

    {

        close(tmp2);

        if (dup2(f2, 1) < 0)

        {

            close(f1);

            close(f2);

            remove("tmp\_name1");

            remove("tmp\_name2");

            perror("Can't dup child.out to stdout");

            return 2;

        }

        if (execl("child.out", "tmp\_name2", "", NULL) == -1)

        {

            close(f1);

            close(f2);

            remove("tmp\_name1");

            remove("tmp\_name2");

            perror("Execl child problem");

            return 2;

        }

    }

    close(f1);

    close(f2);

    return 0;

}

**Примеры работы**

den@den-VirtualBox:~/Документы/lab4$ ./a.out

a.txt

b.txt

aaaaa

bbbbbbbbbbbbbbb

kkkkkkklllllll

yyyhhh

**a.txt:**

aaaaa

hhhyyy

**b.txt:**

bbbbbbbbbbbbbbb

lllllllkkkkkkk

**Вывод**

Выполнив данную лабораторную работу, я познакомился с механизмом отображения файлов на память и научился его использовать для реализации межпроцессорного взаимодействия. Используя данный метод можно добиться прироста производительности, так как вместо многократного выполнения небыстрых запросов на чтение и запись мы выполняем отображение файла на ОЗУ, после чего получаем произвольный доступ к данным.