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: Тулин Иван Денисович
Группа: М8О-201Б-21
Вариант: 17
Преподаватель Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/YusayuSharingan/Opsys-labs/tree/main/lab4

Постановка задачи

Цель работы

Приобретение практических навыков в:

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

Задание

10 вариант) В файле записаны команды вида: «число<endline>». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

Общие сведения о программе

Программа компилируется из файла parent.cpp. Также подключаются файлы child.cpp через execl в качестве отдельной программы.

Общий метод и алгоритм решения

Алгоритм решения повторяет вторую лабораторную, за исключением того, что теперь входной файл из внешней памяти помещается в shared memory и считывается из нее подобно массиву, а также выходные данные дочерней программы помещаются в отдельную страницу shared memory.

Исходный код

parent.cpp

```
#include <sys/mman.h>
#include <sys/wait.h>
#include <vector>
#include <fcntl.h>
#include <unistd.h>
```

```
#include <stdio.h>
#include "utils.h"


char name[] = "shared_memory";
char child[] = "./child";

std::vector <int> ParentRoutine(char child[], char filename[]){
   std::vector <int> output = {};

       int sfd;
       int *aptr = nullptr;
       SharingMemory(sfd, name);
       FtruncateShm(sfd);

       int pid = fork();
       if(pid == ERROR) {
             ForkError();
       } else if (pid == CHILD_ID) {
             int fd;
             int *fptr = nullptr;
             OpenFile(fd, filename);
             MakeMmap(&fptr, PROT_READ, MAP_SHARED, fd);
             execl(child, child, fptr, name, NULL);
       } else {
             wait(NULL);
             MakeMmap(&aptr, PROT_READ | PROT_WRITE, MAP_SHARED,
sfd);
             int cnt = 0;
             for(int i = 1; i < aptr[0]; i++) {
```

```
                output.push_back(aptr[i]);
            }

        }

        UnMapping();

        UnLinkingShm(name);

        return output;
}
```

utils.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include "utils.h"


void OpenFile(int &pfd, char *filename) {
   if((pfd = open(filename, O_RDWR)) == -1) {
      printf("Error: couldn't open file %s\n", filename);
      exit(EXIT_FAILURE);
   }
}
```

```
void SharingMemory(int &sfd, char name[]) {
    if((sfd = shm_open(name, O_CREAT | O_RDWR, S_IRWXU)) == -1) {
        printf("Error: couldn't share memory\n");
        exit(EXIT_FAILURE);
    }
}

void UnLinkingShm(char name[]) {
    if(shm_unlink(name) == -1) {
        printf("Error: couldn't unlink shared memory\n");
        exit(EXIT_FAILURE);
    }
}

void FtruncateShm(int &sfd) {
    if((ftruncate(sfd, getpagesize())) == -1) {
        printf("Error: couldn't truncate shared memory\n");
        exit(EXIT_FAILURE);
    }
}

void MakeMmap(int **ptr, int prot, int flags, int fd) {
    *ptr = (int*) mmap(nullptr, getpagesize(), PROT_READ | PROT_WRITE, flags,
fd, 0);
    if(ptr == MAP_FAILED) {
        printf("Error: mmap error\n");
        exit(EXIT_FAILURE);
    }
}
```

```
void UnMapping() {
    if(munmap(nullptr, getpagesize()) == -1) {
        printf("Error: munmap error\n");
        exit(EXIT_FAILURE);
    }
}

void ForkError() {
    printf("Error: could't create new process\n");
    exit(EXIT_FAILURE);
}

void ExecError(){
    printf("Error: couldn't execute child program");
    exit(EXIT_FAILURE);
}
```

child.cpp

```
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "utils.h"
#include "functions.h"


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

    int sfd;
    int *memory = nullptr;
```

```
    SharingMemory(sfd, argv[2]);
    MakeMmap(&memory, PROT_READ | PROT_WRITE, MAP_SHARED, sfd);

    int n, p = 0, ans = 1;
    char number[32];
    for (int i=0; argv[1][i] != EOS; i++) {
        number[p++] = argv[1][i];
        if(argv[1][i] == EOL) {
            p = 0;
            n = GetNumber(number);
            if (n < 0) {
                memory[0] = ans;
                exit(EXIT_FAILURE);
            } else if (IsPrime(n)) {
                memory[0] = ans;
                exit(EXIT_FAILURE);
            } else {
                memory[ans++] = n;
            }
        }
    }
    memory[0] = ans;

    return 0;
}
```

functions.cpp

```
#include "functions.h"

int GetNumber(char* s){
    int code = ZERO;
```

```
    int cntr = 0;
    if (s[cntr] - MINUS == 0){
        return -1;
    }
    int sum = 0;
    while (s[cntr] != EOS){
        code = s[cntr] - ZERO;
        if(code < 0 || code > 9){
            break;
        }
        sum = sum*10 + code;
        cntr++;
    }
    return sum;
}


int IsPrime(int num){
    if (num % 2 == 0){
        return num == 2;
    }
    int div = 3;
    while (div * div <= num && num % div != 0){
        div += 2;
    }
    return div * div > num;
}
```

Демонстрация работы программы

```
yorokobeshounen@YS:~/Рабочий стол/OpSys/lab4/build$ cat input1.txt
16
0
49
27
35
64
144
169
yorokobeshounen@YS:~/Рабочий стол/OpSys/lab4/build$ ./lab4
Input name of file
input1.txt
16 0 49 27 35 64 144 169
```

Выводы

Перекладывать файлы в оперативную память и читать их подобно массивам без использования дескрипторов очень удобно, а использование общей памяти у разных процессов намного удобнее использования pipe’ов.