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: Мусаелян Ярослав
Группа: М8О-207Б-21
Вариант: 10
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/YMusaelyan/os

Постановка задачи

Цель работы

Приобретение практических навыков в: освоение принципов работы с файловыми системами, 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 через отображаемые файлы (memory-mapped files).
Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

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 файлов main.cpp, child.cpp. В программе используются следующие системные вызовы:

1. fork() - создает процесс
2. execlp() - передает процесс другой программе
3. sem_unlink() - удаляет именованный семафор
4. sem_open() - инициализирует и открывает именованный семафор
5. sem_post() - разблокировать семафор
6. sem_wait() - заблокировать семафор
7. memcpy() - копирует область памяти
8. mnmap() и munmap() - отражает и снимает отражение файлов или устройств в памяти
9. sem_close() - закрывает именованный семафор
10. dup2() - создает копию файлового дескриптора
11. close() - закрывает файловый дескриптор

Общий метод и алгоритм решения

Пользователь пишет название файла. Открываем заданные файл на чтение. Создаем семафор, увеличиваем или уменьшаем его значение до 1. Создаем процесс. Родительский процесс в цикле блокирует семафор и ждет, пока выполниться дочерний процесс. В нем создадим функцию на

проверку на числа. Если число составное, то функция выдает значение 1, если отрицательное или простое (для однозначности добавим к этим числам 0 и 1) — значение -1. Если число составное, то записываем результат, разделяя построчно, в остальных случаях передаем знак конца файла и завершаем дочерний процесс.

Исходный код

main.cpp

```
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <sstream>
#include <iostream>
#include <fcntl.h>
#include <string>
#include <cstring>

using namespace std;

int main(int argc, char const *argv[])
{
    string file_name;
    cin >> file_name;

    int file = open(file_name.c_str(), O_RDONLY);

    if (file == -1) {
        cerr << "error file\n";
        return 0;
    }

    sem_unlink("_sem");
    sem_t *sem = sem_open("_sem",  O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH,
0);

    int state = 0;
    while (++state < 2) {
        sem_post(sem);
    }

    while (--state > 1) {
        sem_wait(sem);
    }

    sem_getvalue(sem, &state);

    pid_t id = fork();
    if (id == -1) {
        cerr << "error fork";
        return 0;
    } else if (id == 0) {
        sem_close(sem);
        execlp("./child", to_string(file).c_str(), NULL);
    } else {
```

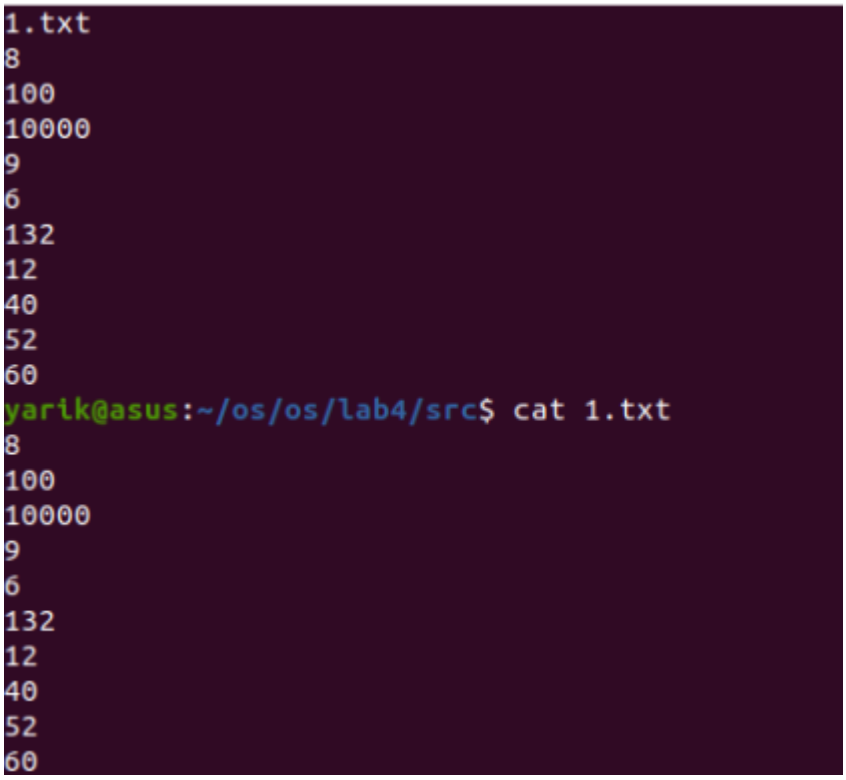
```
        while (true) {
            sem_getvalue(sem, &state);
            if (state == 0) {
                int fd = shm_open("_back", O_RDWR | O_CREAT, S_IWUSR | S_IRUSR |
S_IRGRP | S_IROTH);
                char *mapped = (char *) mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
                char *allocated = (char *) malloc(sizeof(mapped));
                memcpy(allocated, mapped, sizeof(mapped));
                if (*allocated == '\0') {
                    free(allocated);
                    close(fd);
                    munmap(mapped, sizeof(int));
                    close(file);
                    sem_close(sem);
                    return 0;
                } else {
                    cout << allocated << endl;
                }

                free(allocated);
                close(fd);
                munmap(mapped, sizeof(int));
                sem_post(sem);
                sem_post(sem);
            }
        }
    }

    return 0;
}
```

child.cpp

```
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <sstream>
#include <iostream>
#include <fcntl.h>
#include <string>
#include <cstring>

using namespace std;

int func(int number) {
    int composite = 0;

    if (number < 2) {
        composite = -1;
    } else {
        for (int i = 2; i * i <= number; i++) {
            if (number % i == 0) {
                composite = 1;
                break;
            }
        }
        if (composite != 1) {
            composite = -1;
```

```
        }
    }
    return composite;
}

int main(int argc, char const *argv[])
{
    sem_t *sem = sem_open("_sem",  O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH,
0);
    int file = atoi(argv[0]);
    if (dup2(file, 0) == -1) {
        cerr << "error dub\n";
        return 0;
    }

    int n;

    while (cin >> n) {
        int fd = shm_open("_back", O_RDWR | O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP |
S_IROTH);
        if (func(n) == 1) {
            ftruncate(fd, sizeof(int));
            char *mapped = (char *) mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
            sprintf(mapped, "%d", n);
            munmap(mapped, sizeof(int));
            close(fd);
        } else {
            ftruncate(fd, sizeof(char));
            char *mapped = (char *) mmap(NULL, sizeof(char), PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
            sprintf(mapped, "%c", '\0');
            munmap(mapped, sizeof(char));
            close(fd);
        }
        sem_wait(sem);
        sem_wait(sem);
    }

    int fd = shm_open("_back", O_RDWR | O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP |
S_IROTH);
    ftruncate(fd, sizeof(char));
    char *mapped = (char *) mmap(NULL, sizeof(char), PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    sprintf(mapped, "%c", '\0');
    munmap(mapped, sizeof(char));
    close(fd);
    sem_wait(sem);
    sem_wait(sem);

    sem_close(sem);

    return 0;
}
```

Демонстрация работы программы



Выводы

Данная лабораторная работа была очень полезной. Я познакомился с процессами и с управлением процессов в ОС, освоил принципы работы с файловыми системами, обеспечение обмена данных между процессами посредством технологии «File mapping».