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: Матвеев Данил
Группа: М8О-207Б-21
Вариант: 22
Преподаватель: Черемисинов Максим
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/MrDenli/OsLabs

Постановка задачи

Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

8 вариант) В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

Общие сведения о программе

Программа компилируется из файла main.cpp с помощью cmake. Дочерний процесс представлен в child.cpp

Общий метод и алгоритм решения

shm_open

Использованные библиотеки: <sys/stat.h> + <fcntl.h>

Создает и открывает объект общей памяти POSIX, который эффективен для работы с несвязанными процессами, которые хотят использовать единый объект памяти. С флагом O_RDWR - открывает объект на чтение и запись. O_CREAT - создает объект, если он не существует. Аргумент mode означает права доступа, я их установил в переменной accessPerm, установив 644. В случае ошибки возвращает -1.

sem_open

Использованные библиотеки: <semaphore.h> + <fcntl.h>

Создает новый семафор POSIX, или открывает уже существующий. Семафор - число, не меньше 0. Семафоры можно уменьшать (sem_wait) и увеличивать (sem_post). При этом если применить операцию sem_wait к семафору, когда его значение 0, то sem_wait блокирует работу, пока значение не увеличится (для чего они и создавались). Именованные семафоры, также, как и объекты общей памяти, лежат на диске в директории /dev/shm. Если устанавлен аттрибут O_CREAT и семафор при этом существует, то аттрибуты значения и прав доступа игнорируются.

ftruncate

Использованные библиотеки: <unistd.h>

Устанавливает необходимую длину файла в байтах.

fstat

Использованные библиотеки: <sys/stat.h> + <sys/types.h>

Содержит информацию о файле, например, размер st_size, и заполняет буфер.

mmap

Использованные библиотеки: <sys/mman.h>

Создает отображение файла на память в пространстве процесса.

Алгоритм решения:

Считываем полностью текст из строк до EOF из потока ввода, затем открываем объект общей памяти, устанавливаем ему размер текста и отображаем на него текст.
Далее создаем семафор, увеличиваем / уменьшаем его значение до 1.
Вызываем fork(). Родительский процесс в цикле блокирует семафор, и ждет выполнения дочерних процессов.
Так как родительский и дочерний процессы представлены разными файлами, то придется заново закрывать и открывать объект общей памяти и семафор.
После вызова execl, дочерний процесс открывает объект общей памяти, отображает его на буфер, уменьшает значение семафора до 0(при этом родительский процесс заблокировался), выполняет свое преобразование затем ждет (так как одна итерация цикла while(1) может выполниться быстрее, чем дочерний процесс, и разблокирует родительский процесс, который выводит результат в консоль.

Исходный код

main.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>
using namespace std;
```

```
string backFile = "main1.back";
string semFile = "main1.semaphore";
int accessPerm = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;
// S_IWURS | S_IRUS | S_TRGRP | S_IROTH - пользователь имеет права на запись информации в файл | на чтение файла | группа имеет права на чтение файла | все остальные имеют права на чтение файла


int main(int argc, char const *argv[])
{
   string in;
   cin >> in;
   int file = open(in.c_str(), O_WRONLY);
   sem_t *semaphore = sem_open(semFile.c_str(), O_CREAT, accessPerm, 1);
   // Создаёт новый семафор с именем semFilde.c_str() ("main1.semaphore") (O_CREAT - создаётся если его ещё не существует), с правами доступа accesPerm, с начальным значением 1 нового семафора
   if (semaphore == SEM_FAILED){
       perror("sem_open");
      exit(EXIT_FAILURE);
   }
   int state = 0;
   int semErrCheck = sem_getvalue(semaphore, &state);
   if (semErrCheck == -1){
       perror("sem_getvalue");
      exit(EXIT_FAILURE);
   }
   while (state++ < 1) {
      sem_post(semaphore);
   }
   while (state-- > 2) {
```

```
    sem_wait(semaphore);
  }
  pid_t child = fork();
  if (child == -1){
        perror("fork");
        exit(EXIT_FAILURE);
  } else if (child == 0) {
     char *Child_argv[]={(char*)in.c_str(), NULL};
    execv("child", Child_argv);
        perror("execl");
        exit(EXIT_FAILURE);
  } else {
    int fd = shm_open(backFile.c_str(), O_RDWR | O_CREAT, accessPerm);
     // backFilde.c_str() ("main1.back") - определяет создаваемый объект разделяемой памяти для создания или открытия
     // O_RDWR | O_CREAT - Открывает объект для чтения и записи | Создаёт объект разделяемой памяти, если он ещё не существует
     // accesPerm - права доступа
    while (1) {
        semErrCheck = sem_getvalue(semaphore, &state);
        if (semErrCheck == -1){
            perror("sem_getvalue");
            exit(EXIT_FAILURE);
     }
     if (state == 0) {
            int semWaitErrCheck = sem_wait(semaphore);
            if (semWaitErrCheck == -1){
            perror("sem_wait");
            exit(EXIT_FAILURE);
            }
```

```
                struct stat statBuf;
                fstat(fd, &statBuf);
                // Возвращает информацию об опрашиваемом файле (заданного в виде файлового дескриптора fd) в буфер, на который указывает statbuf
                int mapSize = statBuf.st_size;
                char *mapped = (char *) mmap(NULL,
                                mapSize,
                                PROT_READ | PROT_WRITE,
                                MAP_SHARED,
                                fd,
                                0);
                // mapped отображает mapSize байт, начиная со смещения 0 файла, определённого файловым описателем fd, в память начиная с адреса NULL.
                // PROT_READ | PROT_WRITE описывают желаемый режим защиты памяти (Данные можно читать | В эту область можно записывать информацию)
                // MAP_SHARED опция отражения (Разделение использования этого отражения с другими процессами)
            cout << "Answer is:\n";
            for (int i = 0; i < mapSize; ++i) {
               cout << mapped [i];
            }
                return 0;
        }
         }
   }
   return 0;
}
```

child.cpp

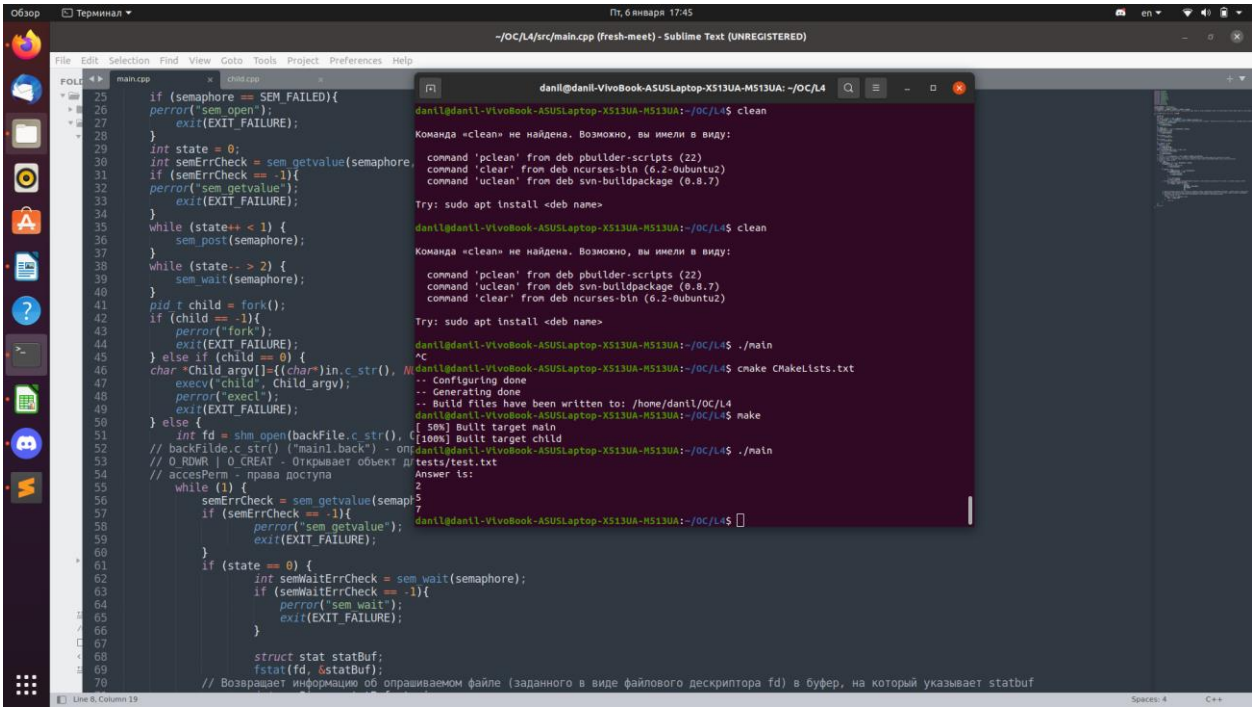
```
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <algorithm>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
using namespace std;

string backFile = "main1.back";
string semFile = "main1.semaphore";
int accessPerm = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;

int main(int argc, char const *argv[])
{
   int state = 0;
   sem_t *semaphore = sem_open(semFile.c_str(), O_CREAT, accessPerm, 1);
   if (semaphore == SEM_FAILED){
           perror("sem_open");
          exit(EXIT_FAILURE);
   }
   if (sem_getvalue(semaphore, &state) == -1){
           perror("sem_getvalue");
```

```
        exit(EXIT_FAILURE);
}
int semWaitErrCheck = sem_wait(semaphore);
if (semWaitErrCheck == -1){
        perror("sem_wait");
       exit(EXIT_FAILURE);
}
if (sem_getvalue(semaphore, &state) == -1){
        perror("sem_getvalue");
        exit(EXIT_FAILURE);
}
string str;
int res, a;
bool flag = true, zeroflag = true;
ifstream in_file;
in_file.open(argv[0]);
string line, answer = "";
while (getline(in_file, line) && zeroflag){
    if (!flag){
        answer = answer + to_string(res) + "\n";
        flag = true;
    }
    stringstream ss(line);
    while(ss >> a && zeroflag){
            if (flag){
                    flag = false;
                    res = a;
            } else {
                    if (a == 0){
```

```
                        zeroflag = false;
                        break;
                    }
                    res /=  a;
                }
            }
        }
        if (zeroflag)
            answer = answer + to_string(res) + "\n";
        if (!zeroflag)
            answer = answer + "division by 0\n";
        int mapSize = answer.size();
        int fd = shm_open(backFile.c_str(), O_RDWR, accessPerm);
        ftruncate(fd, mapSize);
        // Устанавливает длину файла с файловым дескриптором fd в mapSize байт
        if (fd == -1){
            perror("shm_open");
            exit(EXIT_FAILURE);
        }
        char *mapped = (char *)mmap(NULL, mapSize, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
        //MAP_SHARED - задаёт опции отражения (разделение использования этого отражения с другими процессами)
        if (mapped == MAP_FAILED){
            perror("mmap");
            exit(EXIT_FAILURE);
        }
        memset(mapped, '\0', mapSize);
        for (int i = 0; i < answer.size(); ++i) {
```

```
        mapped[i] = answer[i];
    }
    close(fd);
    usleep(00150000);
    sem_post(semaphore);
    sem_close(semaphore);
}
```

Демонстрация работы программы



Выводы

Мне понравилась данная лабораторная работа, я научился работать с процессами в языке c++, а именно разделять процессы, взаимодействовать между эти процессы. Я считаю эта лабораторная работа очень полезна для меня, потому что полученные навыки с большой вероятностью помогут мне в будущем.