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: Ивченко Анна Владимировна
Группа: М8О-208Б-20
Вариант: 4
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

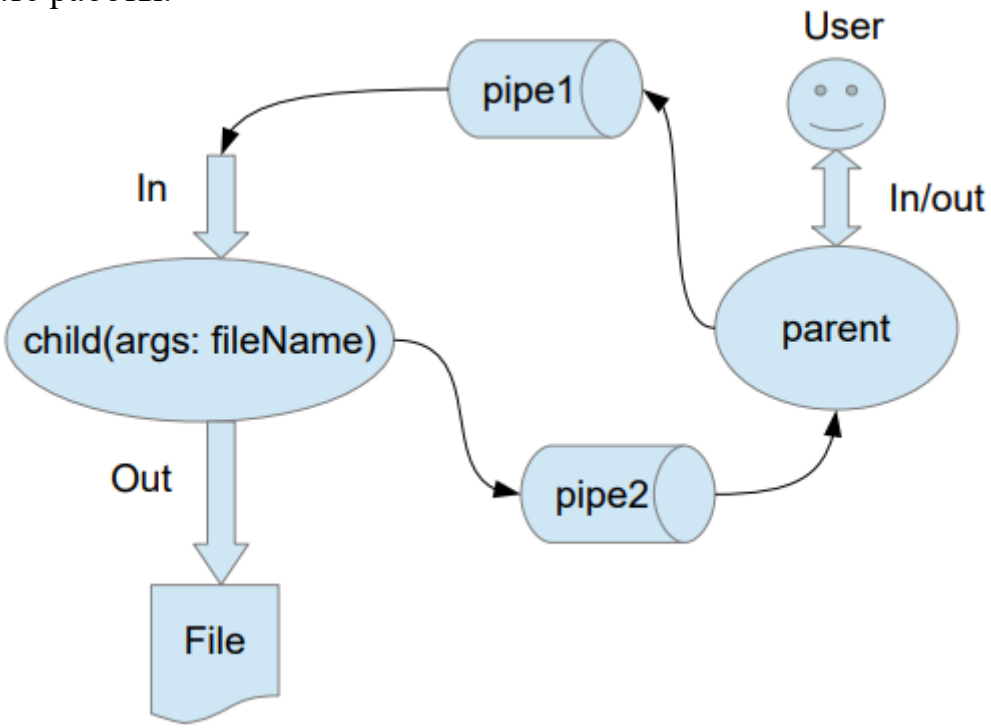
https://github.com/Anetta123/OS/tree/main/os_lab4/scr

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.



Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, fcntl.h, errno.h, sys/mman.h, sys/stat.h, string.h, stdbool.h, ctype.h, sys/wait.h, semaphore.h. В программе используются следующие системные вызовы:

1. shm_open - создаёт/открывает объекты общей памяти POSIX.
2. sem_open - инициализирует и открывает именованный семафор.
3. ftruncate - обрезает файл до заданного размера.
4. mmap, munmap - отображает файлы или устройства в памяти, или удаляет их отображение.
5. memset - заполнение памяти значением определённого байта.
6. sem_getvalue - возвращает значение семафора.
7. close - закрывает файловый дескриптор.
8. sem_close - закрывает именованный семафор.
9. execl - запуск файла на исполнение.
10. sem_getvalue - возвращает значение семафора.
11. sem_wait - блокирует семафор.
12. sem_post - разблокирует семафор.

Исходный код

Main.cpp

```
#include <fcntl.h>
#include <semaphore.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>
#include <iostream>

#include "shrmmen.h"

using namespace std;

int main() {

        char filename [30];
        cout << "Enter a filename: " << endl;
        cin >> filename;
        //cin.ignore();

  size_t map_size = 0;
  char *in = (char *)calloc(1, sizeof(char)); //поток ввода
  char c;
  while ((c = getchar()) != EOF) {
                in[map_size] = c;
                in = (char *)realloc(in, (++map_size + 1) * sizeof(char));
  }
  in[map_size++] = '\0';
  //read string stream
  int fd = shm_open(BackingFile, O_RDWR | O_CREAT, AccessPerms);
  if (fd == -1) {
```

```
                perror("OPEN");
                exit(EXIT_FAILURE);
}
sem_t *semptr = sem_open(SemaphoreName, O_CREAT, AccessPerms, 2);
if (semptr == SEM_FAILED) {
                perror("SEM_OPEN");
                exit(EXIT_FAILURE);
}
int val;

ftruncate(fd, (off_t)map_size);
caddr_t memptr = static_cast<char*>(mmap(
        NULL,
        map_size,
        PROT_READ | PROT_WRITE,
        MAP_SHARED,
        fd,
        0));
if (memptr == MAP_FAILED) {
                perror("MMAP");
                exit(EXIT_FAILURE);
}
memset(memptr, '\0', map_size);
sprintf(memptr, "%s", in);
free(in);
if (sem_getvalue(semptr, &val) != 0) {
                perror("SEM_GETVALUE");
                exit(EXIT_FAILURE);
}
while (val++ < 1) {
                sem_post(semptr);
}
while (val-- > 2){
                sem_wait(semptr);
```

```
}
int pid_0 = 0;
if ((pid_0 = fork()) == 0) {
                munmap(memptr, map_size);
                close(fd);
                sem_close(semptr);
                execl("child", "child", filename, NULL);
                perror("EXECL");
                exit(EXIT_FAILURE);
} else if (pid_0 < 0) {
                        perror("FORK");
                        exit(EXIT_FAILURE);
}
while (true) {
                if (sem_getvalue(semptr, &val) != 0) {
                perror("SEM_GETVALUE");
                exit(EXIT_FAILURE);
                }
                if (val == 0) {
                if (sem_wait(semptr) == -1) {
                                perror("SEM_WAIT");
                                exit(EXIT_FAILURE);
                }
                printf("%s", memptr);
                return EXIT_SUCCESS;
                }
}
}
```

Child.cpp

```
#include <fcntl.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
```

```
#include <sys/stat.h>
#include <unistd.h>
#include <string>
#include <iostream>

#include "shrmmen.h"

using namespace std;

int main(int argc, char **argv) {
  int map_fd = shm_open(BackingFile, O_RDWR, AccessPerms);
  if (map_fd < 0) {
          perror("SHM_OPEN");
          exit(EXIT_FAILURE);
  }
  struct stat statbuf;
  fstat(map_fd, &statbuf);
  const size_t map_size = statbuf.st_size;
  caddr_t memptr = static_cast<char*>(mmap(
            NULL,
            map_size,
            PROT_READ | PROT_WRITE,
            MAP_SHARED,
            map_fd,
            0));
  if (memptr == MAP_FAILED) {
          perror("MMAP");
          exit(EXIT_FAILURE);
  }
  sem_t *semptr = sem_open(SemaphoreName, O_CREAT, AccessPerms, 2);
  if (semptr == SEM_FAILED) {
          perror("SEM_OPEN");
          exit(EXIT_FAILURE);
  }
```

```
if (sem_wait(semptr) != 0) {
	perror("SEM_WAIT");
	exit(EXIT_FAILURE);
}
char *out = (char *)calloc(1, sizeof(char));
size_t m_size = 0;
int flag = 0;
string first;
string second;

FILE *filename = fopen(argv[1], "w");
if (filename == NULL) cout << "fopen error";

for (int i = 0; i + 1 < map_size; ++i) {// преобразование
	if (flag == 0) {
		first.push_back(memptr[i]);
	}
	else if (flag == 1) {
		second.push_back(memptr[i]);
	}
	if (memptr[i] == ' ' && flag == 0) {
		flag = 1;
	}
	else if ((memptr[i] == ' ' || memptr[i] == '\n') && flag == 1) {
		if (atof(second.c_str()) == 0) {
			perror("division by zero error\n");
			break;
		}
		first = to_string(atof(first.c_str())/atof(second.c_str()));
		second = "";
		if (memptr[i] == '\n') {
			fprintf(filename, "%s\n", first.c_str());
			flag = 0;
			//cout << first << endl;
```

```
                    first = "";
                    second = "";
                }
            }
        }
        fclose(filename);
        out[m_size++] = '\0';
        ftruncate(map_fd, (off_t)m_size);
        memset(memptr, '\0', m_size);
        sprintf(memptr, "%s", out);
        free(out);
        close(map_fd);
        usleep(00150000);
        sem_post(semptr);
        sem_close(semptr);
        return EXIT_SUCCESS;
}
```

Shrmem.h

```
#ifndef SRC__SHRMEM_H_
#define SRC__SHRMEM_H_
#include <fcntl.h>
const char *BackingFile = "os_lab4.back";
const char *SemaphoreName = "os_lab4.semaphore";
unsigned AccessPerms = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;
#endif//SRC__SHRMEM_H_
```

Выводы

В Си помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например отображение файла в память, такой подход работает быстрее, за счет отсутствия постоянных вызовов read, write и тратит меньше памяти под кэш. После отображения возвращается void*, который можно привести к своему указателю на тип и обрабатывать данные как массив, где возвращенный указатель – указатель на первый элемент.