МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №4

по курсу операционные системы I семестр, 2021/22 уч. год

Студент *Сикорский Александр Александрович, группа М8О-208Б-20*
Преподаватель *Миронов Евгений Сергеевич*
Вариант *21*
Оценка
Дата
Подпись

Содержание

Репозиторий

`https://github.com/sikorskii/os/tree/master/lab4`

Постановка задачи

Задание: Вариант 21:

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

Исходный код лежит в 3 файлах:

1. main.cpp - файл, в котором происходит создание дочерних процессов, работа с пользователем и открытие семафоров.

2. child1.cpp- код дочерних процессов. Принимают строки от родителя, разворачивают их и пишут в указанный файл.

3. mem.h - Некоторые константы и значения, нужные для работы с mmap.

Программа собирается с помощью CMakeList. Нужно получитьдва исполняемых файла, как и во второй лабораторной работе.

Общий метод и алгоритм решения

Сначала пользователь должен ввести названия двух файлов, в которые дочерние процессы должны будут писать развернутые строки. Весь алгоритм программы идентичен второй лабораторной, за исключением способа межпроцессного взаимодействия. Здесь используется mmap, так что нужно создавать shared memory object-ы, мапить их и решать проблемы синхронизации. В программе используется по 2 семафора для каждого дочернего процесса. На одном семафоре блокируется ребенок, ожидающий, когда родитель напишет ему строку. На другом блокируется родитель, ожидающий, когда ребенок

обработает строку и будет готов принимать следующую. В конце работы программы закрываем все открытое и очищаем все выделенное.

Исходный код

mem.h

```
//
// Created by aldes on 30.10.2021.
//

#ifndef LAB1_MEM_H
#define LAB1_MEM_H

#include <fcntl.h>

const char *file1 = "shared1";
const char *file2 = "shared2";
const char *sem1 = "sem1";
const char *sem2 = "sem2";
const char *sem11 = "sem11";
const char *sem22 = "sem22";
const int MAX_LENGTH = 50;
const int NUMBER_OF_BYTES = MAX_LENGTH * sizeof(char);

unsigned accessPerms = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;

void handleError(const char* message) {
    perror(message);
    exit(1);
}


#endif //LAB1_MEM_H
```

main.h

```
#include "unistd.h"
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <semaphore.h>
```

```
#include <sys/mman.h>


#include "mem.h"

int main() {

    char buf[MAX_LENGTH];

    char *filename1;
    char *filename2;

    printf("Enter file1 name\n");
    //scanf("%s", buf);
    filename1 = fgets (buf, sizeof(buf) - 1, stdin);
    asprintf(&filename1, "%s", buf);

    printf("Enter file2 name\n");
    //scanf("%s", buf);
    filename2 = fgets (buf, sizeof(buf) - 1, stdin);
    asprintf(&filename2, "%s", buf);

    pid_t child1_pid, child2_pid;

    sem_t *firstSem = sem_open(sem1, O_CREAT, 0644, 0);
    sem_t *firstSemOut = sem_open(sem11, O_CREAT, 0644, 0);
    sem_t *secondSem = sem_open(sem2, O_CREAT, 0644, 0);
    sem_t *secondSemOut = sem_open(sem22, O_CREAT, 0644, 0);

    int fd1 = shm_open(file1,
                       O_RDWR | O_CREAT,
                       accessPerms);

    ftruncate(fd1, NUMBER_OF_BYTES);
    if (fd1 < 0)
        handleError("fd1 create error");

    caddr_t memptr1 = static_cast<caddr_t>(mmap(NULL,
                                                NUMBER_OF_BYTES,
                                                PROT_READ | PROT_WRITE,
                                                MAP_SHARED,
                                                fd1,
```

```
                                                0));

if (memptr1 == (caddr_t)-1)
    handleError("memptr1 mapping error");

int fd2 = shm_open(file2,
                   O_RDWR | O_CREAT,
                   accessPerms);

ftruncate(fd2, NUMBER_OF_BYTES);
if (fd2 < 0)
    handleError("fd2 create error");

caddr_t memptr2 = static_cast<caddr_t>(mmap(NULL,
                                            NUMBER_OF_BYTES,
                                            PROT_READ | PROT_WRITE,
                                            MAP_SHARED,
                                            fd2,
                                            0));
if (memptr2 == (caddr_t)-1)
    handleError("memptr2 mapping error");


child1_pid = fork();

if (child1_pid == -1)
    handleError("fork error");


else if (child1_pid == 0) { //child1

    printf("[%d] It's child1\n", getpid());
    fflush(stdout);
    execl("child1.out", "child1", filename1, sem1, sem11, file1, NULL); //execution

}

else { //parent

    printf("[%d] It's parent. Child id: %d\n", getpid(), child1_pid);
    fflush(stdout);
```

```
child2_pid = fork();

if (child2_pid == -1)
    handleError("fork 2 error");

else if (child2_pid == 0) { //child2

    printf("[%d] It's child2\n", getpid());
    fflush(stdout);
    execl("child1.out", "child2", filename2, sem2, sem22, file2, NULL); //execut

}

//parent code below

free(filename1);
free(filename2);

char *str;


while (true) {
    char c[50];
    str = fgets(c, sizeof(c), stdin);

    if (strlen(c) == 1)
        continue;

    if (str == nullptr)
        break;

    if ((strlen(c) - 1) % 2 == 0) {
        strcpy(memptr1, c);
        //sleep(2);
        printf("Parent opens first sem\n");
        sem_post(firstSem);
        sem_wait(firstSemOut);


    } else {
        strcpy(memptr2, c);
        //sleep(2);
```

```
            printf("Parent opens second sem\n");
            sem_post(secondSem);
            sem_wait(secondSemOut);
        }
    }
    char c[2] = "\0";

    strcpy(memptr1, c);
    sem_post(firstSem);
    sem_wait(firstSemOut);

    strcpy(memptr2, c);
    sem_post(secondSem);
    sem_wait(secondSemOut);

    munmap(memptr1, NUMBER_OF_BYTES);
    munmap(memptr2, NUMBER_OF_BYTES);

    close(fd1);
    close(fd2);

    sem_close(firstSem);
    sem_close(firstSemOut);
    sem_close(secondSem);
    sem_close(secondSemOut);

    shm_unlink(file1);
    shm_unlink(file2);
    return 0;
  }
}
```

child1.cpp

```
//
// Created by aldes on 19.09.2021.
//

#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <semaphore.h>
```

```
#include <sys/mman.h>
#include "mem.h"

// 0 - reading
// 1 - writing
void reverse(char *str) {
    int i = 0;
    int j = (int)strlen(str) - 1;

    while(i < j) {
        char temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
}

int main(int argc, char* argv[]) {
    printf("\ni am child and i will write in file %s\n", argv[1]);


    FILE *fp = fopen(argv[1], "w");
    fprintf(fp, "child been here\n");

    char buf[50];

    sem_t *sem = sem_open(argv[2], 0);
    sem_t *semOut = sem_open(argv[3], 0);

    int fd = shm_open(argv[4], O_RDWR, accessPerms);
    if (fd < 0)
        handleError("child fd error");

    char* memptr = static_cast<char*>(mmap(NULL,
                                           NUMBER_OF_BYTES,
                                           PROT_READ | PROT_WRITE,
                                           MAP_SHARED,
                                           fd,
                                           0));
    if (memptr == (char*)-1)
        handleError("child memptr error");
```

```
    while (true) {
        if(!sem_wait(sem)) {
            printf("%d sem opened!\n", getpid());

            if (strcmp(memptr, "\0") == 0)
                break;

            reverse(memptr);
            printf("reversed: %s\n", memptr);
            fprintf(fp, "Got and reversed from shared memory:");
            fprintf(fp, memptr);
            fprintf(fp, "\n");
            printf("%d inside\n", getpid());
            sem_post(semOut);
        }
    }
    printf("%d is about to end his work\n", getpid());
    sem_post(semOut);

    munmap(memptr, NUMBER_OF_BYTES);
    close(fd);
    sem_close(sem);
    sem_close(semOut);
    unlink(argv[4]);

    fclose(fp);
    return 0;
}
```

Демонстрация работы программы

Ниже приведен пример работы программы.

>"string" обозначает пользовательский ввод.

test.txt

```
>aboba
>cringe
>discan
>string
>test
```

```
>tex
>ctrl+D


child been here
Got and reversed from shared memory:
gnirts
Got and reversed from shared memory:
tset


child been here
Got and reversed from shared memory:
narcsid
Got and reversed from shared memory:
xet
```

Выводы

В ходе работы я познакомился с новым видом межпроцессного взаимодействия и перевел код второй лабораторной работы на mmap. Главной сложностью работы могу отметить обеспечение синхронизации трех процессов.