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: Смирнов А.В.
Группа: М8О-207Б-21
Вариант: 16
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Liguha/OS

Постановка задачи

Цель работы

Приобретение практических навыков в:

1. Освоение принципов работы с файловыми системами
2. 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 родительского процесса компилируется из file_mapping.c, использует заголовочные файлы stdio.h, stdlib.h, unistd.h, sys/mman.h, fcntl.h, semaphore.h, string.h, errno.h. В программе используются следующие системные вызовы:

1. unlink() – удаление имени из файловой системы
2. fork() – создание дочернего процесса
3. open() – открытие файла
4. close() – закрытие файла
5. write() – запись последовательности байт
6. lseek() - установка смещения в файловом дескрипторе
7. mmap() - создание отражения файла в памяти
8. munmap() - удаление отражения файла в памяти

Общий метод и алгоритм решения

Родительский процесс крутиться в бесконечном цикле, пока не получит на вход пустую строку — знак завершения работы. Аналогично в цикле находится и дочерний процесс — обработчик строк. Синхронизация процессов достигается по средствам 2 семафоров, так после прочтения строки и записи её в образ файла родительский процесс открывает семафор 1 и начинает ждать открытия семафора 2. Открытие семафора 1 позволяет дочернему процессу обработать строку, записать результат в образ второго файла, открыть семафор 2 и закрыть семафор 1. Тем самым продолжается работа родительского процесса, который считывает результат из образа второго файла и выводит ошибку, если она была.

Исходный код

file_mapping.c

```
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "fcntl.h"
#include "sys/mman.h"
#include "string.h"
#include "errno.h"
#include "semaphore.h"

#define CHECK_ERROR(expr, message) \
	do \
	{ \
		int res = (expr); \
		if (res == -1) \
		{ \
			perror(message); \
			return -1; \
		} \
	} while (0)

#define UNLINK_ERROR(expr, message) \
	do \
	{ \
		int res = (expr); \
		if (res == -1 && errno == EACCES) \
		{ \
			perror(message); \
			return -1; \
		} \
	} while (0) \

const int MAX_LENGTH = 10000;
const int SIZE = MAX_LENGTH + sizeof(int);
const int zero = 0;

int main()
{
	UNLINK_ERROR(unlink("file1"), "unlink error");
	UNLINK_ERROR(unlink("file2"), "unlink error");
	int file1 = open("file1", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
	int file2 = open("file2", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
	if (file1 == -1 || file2 == -1)
	{
		perror("open error");
		return -1;
	}
	CHECK_ERROR(lseek(file1, SIZE - 1, SEEK_SET), "lseek error");
```

```
CHECK_ERROR(lseek(file2, SIZE - 1, SEEK_SET), "lseek error");
CHECK_ERROR(write(file1, &zero, 1), "write error");
CHECK_ERROR(write(file2, &zero, 1), "write error");
sem_t* sem1 = sem_open("!semaphore1", O_CREAT, S_IRUSR | S_IWUSR, 0);
sem_t* sem2 = sem_open("!semaphore2", O_CREAT, S_IRUSR | S_IWUSR, 0);
if (sem1 == SEM_FAILED || sem2 == SEM_FAILED)
{
        perror("sem_open error");
        return -1;
}
int id = -1;
CHECK_ERROR(id = fork(), "fork error");

// child
if (id == 0)
{
        void* in = mmap(NULL, SIZE, PROT_READ, MAP_SHARED, file1, 0);
        void* ans = mmap(NULL, SIZE, PROT_WRITE, MAP_SHARED, file2, 0);
        if (in == MAP_FAILED || ans == MAP_FAILED)
        {
                perror("mmap error");
                return -1;
        }
        UNLINK_ERROR(unlink("result.txt"), "unlink error");
        int fout = open("result.txt", O_CREAT | O_WRONLY, S_IRUSR);
        if (fout == -1)
        {
                perror("open error");
                return -1;
        }
        char* str = calloc(MAX_LENGTH, sizeof(char));
        if (str == NULL)
        {
                perror("calloc error");
                return -1;
        }
        while (1)
        {
                CHECK_ERROR(sem_wait(sem1), "sem_wait error");
                int n = 0;
                memcpy(&n, in, sizeof(int));
                if (n == 0)
                        break;
                memcpy(str, in + sizeof(int), n);
                int p = (n - 2 > 0) ? n - 2 : 0;
                if (str[p] != '.' && str[p] != ';')
                {
                        char err[] = "Last symbol is \'.\'\n";
                        int k = strlen(err);
                        err[k - 3] = str[p];
```

```
                        memcpy(ans, &k, sizeof(int));
                        memcpy(ans + sizeof(int), err, k);
                }
                else
                {
                        memcpy(ans, &zero, sizeof(int));
                        CHECK_ERROR(write(fout, in + sizeof(int), n), "write error");
                }
                CHECK_ERROR(sem_post(sem2), "sem_post error");
        }
        CHECK_ERROR(munmap(in, SIZE), "munmap error");
        CHECK_ERROR(munmap(ans, SIZE), "munmap error");
        CHECK_ERROR(close(fout), "close error");
        free(str);
}

// parent
else
{
        void* out = mmap(NULL, sizeof(int), PROT_WRITE, MAP_SHARED, file1, 0);
        void* ans = mmap(NULL, sizeof(int), PROT_READ, MAP_SHARED, file2, 0);
        if (out == MAP_FAILED || ans == MAP_FAILED)
        {
                perror("mmap error");
                return -1;
        }
        char* err = calloc(MAX_LENGTH, sizeof(char));
        if (err == NULL)
        {
                perror("calloc error");
                return -1;
        }
        char* str;
        size_t s = 0;
        int n = getline(&str, &s, stdin);
        while (n > 0)
        {
                memcpy(out, &n, sizeof(int));
                memcpy(out + sizeof(int), str, n);
                CHECK_ERROR(sem_post(sem1), "sem_post error");
                CHECK_ERROR(sem_wait(sem2), "sem_wait error");
                int k;
                memcpy(&k, ans, sizeof(int));
                if (k != 0)
                {
                        memcpy(err, ans + sizeof(int), k);
                        printf("%s", err);
                }
                n = getline(&str, &s, stdin);
        }
```

```
                memcpy(out, &n, sizeof(int));
                CHECK_ERROR(sem_post(sem1), "sem_post error");
                CHECK_ERROR(munmap(out, SIZE), "munmap error");
                CHECK_ERROR(munmap(ans, SIZE), "munmap error");
                free(err);
        }

        CHECK_ERROR(sem_close(sem1), "sem_close error");
        CHECK_ERROR(sem_close(sem2), "sem_close error");
        CHECK_ERROR(close(file1), "close error");
        CHECK_ERROR(close(file2), "close error");
        CHECK_ERROR(unlink("file1"), "unlink error");
        CHECK_ERROR(unlink("file2"), "unlink error");
}
```

Демонстрация работы программы

```
liguha@Laptop:~/OS/LR4/build$ cat result.txt
cat: result.txt: Нет такого файла или каталога
liguha@Laptop:~/OS/LR4/build$ ./a.out
abcde
Last symbol is 'e'
acb def g
Last symbol is 'g'
abcd.
123 456;
.
;

Last symbol is ' '

Last symbol is '
'
liguha@Laptop:~/OS/LR4/build$ cat result.txt
abcd.
123 456;
.
;
```

Выводы

Составлена и отлажена программа на языке Си, осуществляющая работу и взаимодействие между процессами с использованием отображаемых файлов. Так, получены навыки в обеспечении обмена данных между процессами посредством технологии «File mapping».