Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу
«Операционные системы»

Файлы на памяти

Студент: Недосеков Иван Дмитриевич
Группа: М8О-206Б-19
Вариант: 9
Преподаватель: Соколов Андрей Алексеевич
Дата: _____
Оценка: _____
Подпись: _____

Москва, 2020

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.
вариант: 9. 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 float.

Общие сведения о программе

Программа родитель компилируется из файла parent.c, ребенок из child.c. Также используется заголовочные файлы: *unistd.h, string.h, semaphore.h, errno.h, sys/mman.h*. В программе используются следующие системные вызовы:

1. **mmap, munmap** — размещение файла на адресное пространство программы, и его снятия с адреса.
2. **shm_open, shm_unlink** — открытие (создание/удаление) именованного общего файла для нескольких процессов.
3. **sem_open, sem_close, sem_unlink** — создание семафоров для общения процессов.
4. **execv** — загружает код исполняемого файла в программу.
5. **exit** — завершение программы с указанным кодом.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

- Изучить принципы работы mmap, munmap.
- Изучить принципы работы shm_open, shm_unlink.
- Изучить принципы работы sem_open, sem_close, sem_unlink.
- Написать программу родителя и ребенка.
- Проверить программу на тестах.

Основные файлы программы

settings.h

Основные настройки, подключение библиотек и глобальные имена общих файлов и семафоров.

```
#pragma once
#define BUF_SIZE 400
#define POSSIBLE_FLOATS 85

const char SHM_File_Name[] = "/file_in_name";
const char SHM_Pipe[] = "/file_pipe";

const char SendFileNameSem[] = "/send_creating_SHM";
const char ReadFileNameSem[] = "/read_creating_SHM";

#include <unistd.h>
#include <sys/types.h>
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <sys/mman.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

#include <errno.h>
#include <limits.h>
#include <math.h>

#define SUCCESS 0
#define FAIL -1
```

parent.c

```c
#include "settings.h"

    sem_t *Send;
    sem_t *Received;
    void* addr;

void ParentWork(){
    int signals = 0;
    float len;
    float floats[ POSSIBLE_FLOATS ];
    while ( signals == 0 ){
        if  (sem_wait( Received ) == -1){
            perror("Parent: sem_wait");
            exit( errno );
        }
        memcpy( &len, addr, sizeof( float ) );
        if ( len <= 0){
            signals = len;
            break;
        }
        float* data = (float*)addr + 1;
        int strlen = len;
        memcpy( floats, data, sizeof( float ) * strlen );
        for ( int i = 0; i< strlen ; i++){
```

```
            if ( floats[i] == INFINITY ){
                printf("Deviation by zero\n");
                return;
            }
            printf("%f ", floats[i]);
        }
        printf("\n");
        sem_post( Send );
    }
    if ( signals == 0 ){
        sem_post( Send );
        return;
    }else if ( signals == -1 ){
        sem_post( Send );
        perror("Wrong format ( one float in string )\n");
        return;
    }else if ( signals == -2 ){
        sem_post( Send );
        printf( "File does not exist\n" );
        return;
    }
}

int main(){

    if ( (Send = sem_open(SendFileNameSem, O_CREAT, S_IWUSR | S_IRUSR, 0) ) ==
    ↪  SEM_FAILED ) {
        perror("Before fork sem does not create\n");
        exit( errno );
    }
    if ( (Received = sem_open(ReadFileNameSem, O_CREAT, S_IWUSR | S_IRUSR, 0) )
    ↪  == SEM_FAILED ) {
        perror("Before fork sem does not create\n");
        exit( errno );
    }
    int id = fork();
    if ( id < 0 ){
        perror("Fork error");
      exit( errno );

    }else if( id == 0){ // Программа ребенка

        if ( execv( "./os_lab4_child", NULL ) == -1 ){
            exit( errno );
        }
```

```
}else{ //Программа родителя

    char fileName[ NAME_MAX ];
    scanf( "%s", fileName );

    int transportName = shm_open( SHM_File_Name, O_RDWR | O_CREAT , S_IWUSR
    ↪  | S_IRUSR );
    if ( transportName == -1  ){
        perror("Cant create File_for_transport_Name\n");
        exit( errno );
    }

    if ( ftruncate(transportName, BUF_SIZE) == -1){
        perror("Ftruncate error");
        exit ( errno );
    }

    addr = mmap(NULL, NAME_MAX, PROT_WRITE | PROT_READ , MAP_SHARED,
    ↪  transportName, 0);
    if (addr == MAP_FAILED){
        perror("Mmap Parent error");
        exit(errno);
    }

    strcpy( (char*) addr, fileName );
    sem_post( Send );

    if  (sem_wait( Received ) == -1){
        perror("Parent: sem_wait");
        exit( errno );
    }

    sem_post( Send );

    ParentWork();
    if (munmap( addr, NAME_MAX ) == -1 ){
        perror("Unmap file_name Parent error");
        exit(errno);
    }
    if ( shm_unlink(SHM_File_Name) == -1 ){
        perror("Unlink file_name Parent error");
        exit(errno);
    }
}
```

```
        sem_close( Send );
        sem_unlink( SendFileNameSem );
        sem_close( Received );
        sem_unlink( ReadFileNameSem );
        exit( SUCCESS );

}
```

child.c

```
#include "settings.h"

    void* addr;
    sem_t* Send;
    sem_t* Receive;

void writeInFile( long len, float* mas ){

    float count = len ;
    float* ad = (float*) addr + 1;
    if ( len < 0 ){
        memcpy( (float*) addr, &( count ), sizeof( float ) );
        sem_post( Receive );
        return;
    }
    if ( sem_wait( Send ) == -1 ){
        perror( "Wait Receive in Child Write\n" );
        exit( errno );
    }
    memcpy( (float*) addr, &( count ), sizeof( float ) );
    if ( len != 0 ){
        memcpy( ad, (void*) mas, sizeof( float ) * len );
    }
    sem_post( Receive );
}

int ChildWork(){
    float floats[ POSSIBLE_FLOATS ];
    char* string = NULL;
    size_t len;

    while( getline( &string, &len, stdin ) > 0 ){
```

```
        char* currTok = strtok( string, " ");
        if ( *currTok == '\n'  ){
            continue;
        }
        if ( currTok == NULL ){// exep
            printf("Child eof\n");
            writeInFile( 0, NULL );
            break;
        }
        float piv = strtof( currTok, NULL );
        int TokCounter = 0;
        while( currTok != NULL ){
            currTok = strtok( NULL, " ");
            if ( currTok == NULL ){
                break;
            }
            floats[ TokCounter ] = strtof( currTok, NULL );
            TokCounter++;
        }
        if ( TokCounter == 0 ){ // exep
            writeInFile( -1, NULL );
            free( string );
            return FAIL;
        }
        for ( int i = 0; i < TokCounter; i++ ){
            if ( floats[i] == 0 ){
                floats[i] = INFINITY;
                writeInFile( i + 1, floats );
                free( string );
                return( FAIL );
            }
            floats[i] = piv / floats[i];
        }
        writeInFile( TokCounter, floats );

    }
    writeInFile( 0, NULL );
    free( string );
    return SUCCESS;
}

int main(){


    if ( ( Send = sem_open(SendFileNameSem, 0)) == SEM_FAILED ) { // семафоры
```

```
        perror("Child: sem_open\n");
        exit (errno);
    }
    if ( ( Receive = sem_open(ReadFileNameSem, 0)) == SEM_FAILED ) {
        perror("Child: sem_open\n");
        exit (errno);
    }

    if ( sem_wait( Send ) == -1 ){ // ждем создания файла ждя имени
        perror("Child: sem_wait\n");
        exit( errno );
    }
    int transportName = shm_open( SHM_File_Name, O_RDWR, S_IWUSR | S_IRUSR );
    ↪  // коннектимся
    if ( transportName == -1  ){
        perror("Child: Cant open File_for_transport_Name\n");
        exit( errno );
    }

    addr = mmap(NULL, BUF_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
    ↪  transportName, 0);
    if (addr == MAP_FAILED){
        perror("Mmap Parent error\n");
        exit( errno );
    }

    char fileName[ NAME_MAX ];
    strcpy( fileName, (char*) addr );
    sem_post( Receive );
    int input = open( fileName, O_RDONLY );
    if ( input == -1 ){
        perror("Child: Cant open input file\n");
        writeInFile( -2, NULL );
        exit( errno );
    }

    dup2( input, STDIN_FILENO );

    if ( ChildWork() < 0 ){
        exit( FAIL );
    }
    exit( SUCCESS );
}
```

Пример работы

```
mx$ cat ../tests/03.src
../tests/03.t
mx$ cat ../test/tests/03.t
38.1497 -22.4932 -1.6701
65.1601 42.2433 50.4008
8.7974 135.2943 83.7411
134.6063 16.0079 39.8091
-12.4574 -82.5917 -1.2255
4.0717 46.4590 -77.8966
42.9741 40.3379 -80.8924
48.9088 -39.7069 -89.2353
150.8459 101.4028 -93.9988
-76.0763 -58.5932 13.4213
mx$ ./os_lab4_parent < ../tests/03.src
-1.696055 -22.842764
1.542495 1.292839
0.065024 0.105055
8.408741 3.381294
0.150831 10.165157
0.087641 -0.052271
1.065353 -0.531250
-1.231745 -0.548088
1.487591 -1.604764
1.298381 -5.668326
```

GitHub

Ссылка на репозиторий GitHub с проектом:
https://github.com/GrozniyToaster/os_lab_04

Вывод

Передавать через общие файл удобно много информации, но нужно аккуратно продумывать кто в данный момент записывает в файл, чтобы не появилось состояние гонки. Семафоры частично решают эту проблему, но семафор удобен, когда нужно только послать процессам, что можно приступить дальнейшему выполнению. Так как в лабораторной взаимодействуют 2 процесса в обе стороны, то возникали состояния dead_lock, когда один процесс ждал другого, а тот в сою очередь ждал первого.