Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу

«Операционные системы»

FILE MAPPING

Студент: Лошманов Юрий Андреевич
Группа: М8О–206Б–20
Вариант: 7
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020.

Постановка задачи

Цель работы

Целью является 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

Родительский процесс создает дочерний процесс и ждёт его завершения.

Дочерний процесс открывает файл в разделяемой памяти, расширяет его до указанного значения и отображает его в своё адресное пространство. Открывает файл с тестами и помещает результат суммы каждой его строчки в разделяемую память. После окончания считывания, дочерний процесс удаляет отображение разделяемой памяти и закрывает файловые дескрипторы.

Дождавшись завершения дочернего процесса, родительский процесс так же открывает и отображает файл разделяемой памяти. Печатает её содержимое, удаляет отображение и сам файл в разделяемой памяти.

В большинстве случаях, программы обработают ошибки, обработают их, выведут сообщение в консоль и удалят за собой файл в разделяемой памяти.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы функций shm_open, ftruncate, mmap, munmap и shm_unlink
2. Подключить библиотеки sys/mman.h, unistd.h и fcntl.h, необходимые для работы с вышеперечисленными функциями и их аргументами.
3. Написать файл потомка child.c.
4. Написать основной файл main.c, который создаёт процесс, запускающий файл потомка.
5. Создать текстовые файлы, который потомок будет принимать на чтение.

Основные файлы программы

shared.h:

```
#ifndef LAB4_SHARED_H
#define LAB4_SHARED_H

#include <stdlib.h>
#include <stddef.h>

#define MEMORY_NAME "/lab4_shm"
#define MEMORY_SIZE 4096
#define DATA_SIZE 256

#if DATA_SIZE > MEMORY_SIZE
#warning Segfault may occur
#endif

typedef struct line_sums {
    size_t size;
    long double data[DATA_SIZE];
} line_sums_t;

void check(int result, line_sums_t *data, int const* fd, char *message) {
    if (result) {
        fprintf(stderr, "%s", message);

        if (data != NULL) {
            munmap(data, MEMORY_SIZE);
        }

        if (fd != NULL) {
            shm_unlink(MEMORY_NAME);
            close(*fd);
        }

        exit(1);
    }
}

void sigsegv_handler() {
```

```
    fprintf(stderr, "Segmentation fault occur, try decrease DATA_SIZE macro\n");
    shm_unlink(MEMORY_NAME);
    exit(1);
}

#endif //LAB4_SHARED_H
```

main.c:

```
#include <sys/mman.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>

#include "shared.h"

int main(int argc, char *argv[]) {
    pid_t child_pid = fork();
    check(child_pid == -1, NULL, NULL, "Fork failed\n");

    if (child_pid == 0) {
        execl("child", "child", argc == 2 ? argv[1] : (char *)NULL, (char *)NULL);
        perror("Can't execute child\n");
        exit(1);
    } else {
        int status;
        check(waitpid(child_pid, &status, 0) == -1, NULL, NULL, "Waitpid error\n");
        if (WIFSIGNALED(status)) {
            fprintf(stderr, "Child process terminated by signal: %d\n",
WTERMSIG(status));
            shm_unlink(MEMORY_NAME);
            exit(1);
        }
        if (WEXITSTATUS(status) != 0) {
            exit(1);
        }
        int fd = shm_open(MEMORY_NAME, O_RDONLY, S_IRUSR | S_IWUSR);
        check(fd == -1, NULL, &fd, "Can't open shared memory file\n");

        line_sums_t *addr = mmap(NULL, MEMORY_SIZE, PROT_READ, MAP_SHARED, fd, 0);
        check(addr == (void *)-1, addr, &fd, "Mmap error\n");

        for (int i = 0; i < addr->size; i++) {
            printf("%Lf\n", addr->data[i]);
        }

        munmap(addr, MEMORY_SIZE);
        shm_unlink(MEMORY_NAME);
        close(fd);
    }
    return 0;
}
```

child.c:

```
#include <sys/mman.h>
#include <stdlib.h>
#include <unistd.h>
#include <stddef.h>
#include <stdio.h>
#include <fcntl.h>
#include <signal.h>

#include "shared.h"

void sum_numbers(FILE *input, line_sums_t *addr) {
    long double sum = 0;
    int flag = 0;
    addr->size = 0;
    while (1) {
        wchar_t c;
        do {
            c = fgetc(input);
        } while (c == ' ' || c == '\t');

        if (c == '\n' || c == EOF) {
            if (flag) {
                if (addr->size == DATA_SIZE) {
                    fprintf(stderr, "Buffer overflow\n");
                    return;
                }

                addr->data[addr->size++] = sum;
                sum = 0;
            }
            flag = 0;
            if (c == EOF) {
                break;
            }
        } else if (c == '-' || (c >= '0' && c <= '9') || c == '.') {
            flag = 1;
            char buff[100];
            ungetc(c, input);
            fscanf(input, "%s", buff);
            sum += strtod(buff, NULL);
        } else {
            fprintf(stderr, "Unexpected symbol while parsing file\n");
            return;
        }
    }
}

int main(int argc, char *argv[]) {
    signal(SIGSEGV, sigsegv_handler);

    int fd = shm_open(MEMORY_NAME, O_EXCL | O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    check(fd == -1, NULL, &fd, "Can't open shared memory file\n");
    check(ftruncate(fd, MEMORY_SIZE) == -1, NULL, &fd, "Can't extent shared memory
file\n");

    line_sums_t *addr = mmap(NULL, MEMORY_SIZE, PROT_WRITE, MAP_SHARED, fd, 0);
    check(addr == (void *)-1, addr, &fd, "Mmap error\n");
    FILE *input = NULL;
    if (argc == 2) {
        input = fopen(argv[1], "r");
    } else {
        input = fopen("input.txt", "r");
    }
    check(input == NULL, addr, &fd, "Can't open file\n");
    sum_numbers(input, addr);
    fclose(input);
    munmap(addr, MEMORY_SIZE);
    close(fd);
    return 0;
}
```

Пример работы

```
yuryloshmanov@air-urij Desktop % git clone https://github.com/yuryloshmanov/os_lab_4
Клонирование в «os_lab_4»…
remote: Enumerating objects: 31, done.
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 31 (delta 5), reused 28 (delta 5), pack-reused 0
Распаковка объектов: 100% (31/31), 5.60 KiB | 318.00 KiB/s, готово.
yuryloshmanov@air-urij Desktop % cd os_lab_4/test
yuryloshmanov@air-urij test % ./run.sh
-- The C compiler identification is AppleClang 12.0.0.12000032
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc -
skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/yuryloshmanov/Desktop/os_lab_4/src
Scanning dependencies of target child
[ 25%] Building C object CMakeFiles/child.dir/child.c.o
[ 50%] Linking C executable child
[ 50%] Built target child
Scanning dependencies of target main
[ 75%] Building C object CMakeFiles/main.dir/main.c.o
[100%] Linking C executable main
[100%] Built target main

test1.txt
1.3 1.2 33.5 -1.0
1.3 1.2 33.5 1
1.3 1.00 33.5
1.3 1.2 33.5
1.3 1.2 4

Answer:
35.000000
37.000000
35.800000
36.000000
6.500000


test2.txt
0.11232 44.4324 22.0324
-1.0 -2.0 -3

Answer:
66.577120
-6.000000


test3.txt


Answer:


test4.txt
55.212 344.1234 0 2 4 1
-2 -1 3

Answer:
406.335400
0.000000

yuryloshmanov@air-urij test % ./run.sh clean
yuryloshmanov@air-urij test %
```

Вывод

Эта лабораторная работа помогла мне понять как работает file-mapping в теории и на практике. Я понял как можно создавать отображения физических файлов, делать анонимные отображения. Так как я делал каждый процесс отдельным файлом, мне потребовалось дополнительно изучить как работать с разделяемой памятью, а именно с функцией shm_open. Данная лабораторная работа была на основе второй, поэтому я так же исправил некоторые недочёты и добавил обработку ошибок.