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: Старцев Иван Романович

Группа: М8О-201Б-21

Вариант: 9

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/IvanTvardovsky/OS-labs

**Постановка задачи**

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.

**Группа вариантов 2:**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

**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. Количество чисел может быть произвольным.

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используются дополнителье файлы с родительской и дочерней программами parent.c и child.c, общие утилиты utils.c

**Общий метод и алгоритм решения**

Опишу новые для себя системные вызовы:

shm\_open

Использованные библиотеки: <sys/stat.h> + <fcntl.h>

Создает и открывает объект общей памяти POSIX, который эффективен для работы с несвязанными процессами, которые хотят использовать единый объект памяти. С флагом O\_RDWR - открывает объект на чтение и запись. O\_CREAT - создает объект, если он не существует. Аргумент mode означает права доступа, я их установил в переменной accessPerm, установив 644. В случае ошибки возвращает -1.

ftruncate

Использованные библиотеки: <unistd.h>

Устанавливает необходимую длину файла в байтах.

mmap/munmap

Использованные библиотеки: <sys/mman.h>

Создает отображение файла на память в пространстве процесса.

Алгоритм решения:

Алгоритм очень схож с алгоритмом из лабораторной работы №2, только вместо pipe используется общая память. А также различные вспомогательные функции для работы с памятью и строками — memcpy, gcvt, strcat и т. д.

**Исходный код**

**main.c**

#include "parent.h"

#include <stdio.h>

#include <stdlib.h>

int main() {

ParentRoutine(stdin);

return 0;

}

**child.c**

#include "utils.h"

#include <sys/mman.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <fcntl.h>

int main(const int argc, const char\* argv[]) {

if (argc != 2) {

printf("Necessary arguments were not provided\n");

exit(EXIT\_FAILURE);

}

FILE\* out = fopen(argv[1], "r");

if (!out) {

printf("Failed to open file\n");

exit(EXIT\_FAILURE);

}

int fd = shm\_open(MEMORY\_NAME, O\_CREAT | O\_RDWR, 0777);

if (fd == -1) {

printf("Can't open shared memory file1\n");

exit(EXIT\_FAILURE);

}

if (ftruncate(fd, MEMORY\_SIZE) == -1) {

printf("Can't extent shared memory file\n");

shm\_unlink(MEMORY\_NAME);

exit(EXIT\_FAILURE);

}

char\* addr;

addr = mmap(NULL, MEMORY\_SIZE, PROT\_WRITE | PROT\_READ, MAP\_SHARED, fd, 0);

if (addr == (char\*)-1 ) {

printf("Mmap error\n");

shm\_unlink(MEMORY\_NAME);

exit(EXIT\_FAILURE);

}

char\* input;

char\* buf;

char\* temporary;

const int chunkSize = 256;

char\* outputString = (char\*) malloc(chunkSize);

int bufferSize = chunkSize;

int currentBufferSize = 0;

while ((input = ReadString(stdin)) != NULL) {

int index = 0, inputLen = strlen(input), flag = 0, bufFlag = 1;

buf = ReadNumber(input, index);

index += strlen(buf) + 1;

float result = atof(buf), output;

free(buf);

if (index == inputLen) bufFlag = 0;

while (index < inputLen) {

if (flag) {

if (atof(buf) == 0) {

exit(EXIT\_FAILURE);

}

result /= atof(buf);

free(buf);

} else {

++flag;

}

buf = ReadNumber(input, index);

index += strlen(buf) + 1;

if (buf == NULL) {

break;

}

}

if (bufFlag) {

result /= atof(buf);

free(buf);

}

free(input);

output = (float)((int)(result \* 100)) / 100;

temporary = (char\*) malloc(chunkSize \* sizeof(char));

gcvt(output, 5, temporary);

currentBufferSize += strlen(temporary) + 1;

while (currentBufferSize >= bufferSize) {

outputString = realloc(outputString, bufferSize + chunkSize);

bufferSize += chunkSize;

}

strcat(outputString, temporary);

strcat(outputString, " ");

free(temporary);

close(fd);

}

printf("%s\n", outputString);

memcpy(addr, outputString, (strlen(outputString) + 1) \* sizeof(char));

free(input);

free(outputString);

fclose(out);

return 0;

}

**parent.c**

#include "parent.h"

#include "utils.h"

#include <sys/mman.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <fcntl.h>

void ParentRoutine(FILE\* stream) {

char inputFileName[256];

fscanf(stream, "%s", inputFileName);

char outputFileName[256];

fscanf(stream, "%s", outputFileName);

int inputFile;

inputFile = open(inputFileName, O\_RDONLY);

if (inputFile < 0) {

char message[] = "Can't open input file\n";

write(STDERR\_FILENO, &message, sizeof(message) - 1);

exit(EXIT\_FAILURE);

}

int id = fork();

if (id == 0) {

dup2(inputFile, STDIN\_FILENO);

char\* argv[3];

argv[0] = getenv("child");

argv[1] = inputFileName;

argv[2] = NULL;

if (execv(getenv("child"), argv) == -1) {

printf("Failed to exec\n");

exit(EXIT\_FAILURE);

}

} else if (id > 0) {

int status;

waitpid(id, &status, 0);

// Дочерний завершился из-за необработанного сигнала

if (WIFSIGNALED(status)) {

fprintf(stderr, "Child process terminated by signal: %d\n", WTERMSIG(status));

shm\_unlink(MEMORY\_NAME);

exit(EXIT\_FAILURE);

}

if (WEXITSTATUS(status) != 0) {

exit(EXIT\_FAILURE);

}

FILE \*outputFile;

outputFile = fopen(outputFileName, "w");

int fd = shm\_open(MEMORY\_NAME, O\_CREAT | O\_RDWR, 0777);

if (fd == -1) {

printf("Can't open shared memory file\n");

exit(EXIT\_FAILURE);

}

char \*addr = mmap(NULL, MEMORY\_SIZE, PROT\_READ, MAP\_SHARED, fd, 0);

if (addr == (char\*)-1 ) {

printf("Mmap error\n");

exit(EXIT\_FAILURE);

}

fprintf(outputFile, "%s", addr);

munmap(addr, MEMORY\_SIZE);

shm\_unlink(MEMORY\_NAME);

close(fd);

fclose(outputFile);

} else {

printf("Failed to fork\n");

exit(EXIT\_FAILURE);

}

}

**utils.c**

#include "utils.h"

const char MEMORY\_NAME[] = "/shm";

const int MEMORY\_SIZE = 4096;

const int DATA\_SIZE = 256;

char\* ReadString(FILE\* stream) {

if (feof(stream)) {

return NULL;

}

const int chunkSize = 256;

char\* buffer = (char\*) malloc(chunkSize);

int bufferSize = chunkSize;

if (!buffer) {

printf("Couldn't allocate buffer");

exit(EXIT\_FAILURE);

}

int readChar;

int idx = 0;

while ((readChar = getc(stream)) != EOF) {

buffer[idx++] = readChar;

if (idx == bufferSize) {

buffer = realloc(buffer, bufferSize + chunkSize);

bufferSize += chunkSize;

}

if (readChar == '\n') {

break;

}

}

buffer[idx] = '\0';

return buffer;

}

char\* ReadNumber(char\* string, int idx) {

const int chunkSize = 256;

char\* buffer = (char\*) malloc(chunkSize);

int bufferSize = chunkSize;

if (!buffer) {

printf("Couldn't allocate buffer");

exit(EXIT\_FAILURE);

}

if (string[idx] == ' ' || string[idx] == '\n' || string[idx] == '\0' || string[idx] == EOF) {

free(buffer);

return NULL;

}

int bufInd = 0;

while (string[idx] != ' ' && string[idx] != '\n') {

buffer[bufInd] = string[idx];

++idx;

++bufInd;

if (bufInd == bufferSize) {

buffer = realloc(buffer, bufferSize + chunkSize);

bufferSize += chunkSize;

}

}

if (strlen(buffer) == 0) {

free(buffer);

return NULL;

}

buffer[bufInd] = '\0';

return buffer;

}

**Демонстрация работы программы**

Ввод в консоль:

tvard@tvard-HVY-WXX9:~/os/OS-labs/lab4$ cat test.txt

8.0 2.0 -4.0 -1.0

0.0 3.2 2.09

-10.0 -10.0 -10.0

1337.0 137.0

1 1 1 1 1 1 1

tvard@tvard-HVY-WXX9:~/os/OS-labs/lab4$ ./lab4

test.txt

output.txt

1 0 -0.1 9.75 1

**Выводы**

Проделав данную лабораторную работу, я приобрёл практический опыт и практические навыки, необходимые для работы с отображаемой памятью.