Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Управление процессами в ОС. Обеспечение обмена данных между процессами посредством технологии “File mapping”.**

Студент: Фролов М.А.

Группа: М80-206Б-20

Вариант: 7

Преподаватель: Соколов Андрей Алексеевич

Дата: 13.11.2021

Оценка: 5

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**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 memory-mapped files.

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

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.

7 вариант. В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

**Листинг программы**

**child.c**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <fcntl.h>

#include <string.h>

#include "unistd.h"

typedef enum{

reading\_suc,

reading\_eol,

reading\_wrong\_value,

reading\_eof,

} read\_rvl\_stat;

typedef enum{

reading\_before,

reading\_after,

} read\_status;

read\_rvl\_stat reading\_float(int fd, float\* cur){

bool dot\_fnd = false;

char c;

\*cur = 0;

double i = 0.1;

int k = read(fd, &c, sizeof(char));

while(k > 0){

if(c == '\n') return reading\_eol;

if(c == ' ') break;

if(((c < '0') || (c > '9')) && c != '.'){

return reading\_wrong\_value;

}

if(!dot\_fnd){

if(c == '.') dot\_fnd = true;

else{

\*cur = \*cur \* 10 + c - '0';

}

} else{

if(c == '.'){

return reading\_wrong\_value;

}

\*cur = \*cur + i \* (c - '0');

i /= 10;

}

k = read(fd, &c, sizeof(char));

}

if(k == 0) return reading\_eof;

return reading\_suc;

}

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

if(argc != 2){

perror("Error in execl\n");

return -1;

}

float sum = 0.0f, cur = 0.0f;

read\_rvl\_stat status = reading\_float(0, &cur);

while(status != reading\_wrong\_value){

sum += cur;

if(status == reading\_eof){

perror("Test should end with <endline>\n");

return -1;

} else if(status == reading\_eol){

break;

}

cur = 0.0f;

status = reading\_float(0, &cur);

}

if (status == reading\_wrong\_value){

perror("Wrong value in test\_file\n");

return -2;

}

int desc = open(argv[1], O\_RDWR);

if(desc < 0){

perror("Tmp file not created\n");

return -6;

}

float\* fd = mmap(0, sizeof(float),

PROT\_WRITE,

MAP\_SHARED, desc, 0);

if (fd == MAP\_FAILED){

perror("mmap error\n");

return -5;

}

fd[0] = sum;

if(msync(fd, sizeof(float), MS\_SYNC) < 0){

perror("Msync problem");

return -6;

}

if(munmap(fd, sizeof(float)) < 0){

perror("Munmap problem");

return -7;

}

close(desc);

return 0;

}

**String.c**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include "string.h"

void str\_create(string \*s){

s->buf = NULL;

s->size = 0;

s->capacity = 0;

}

void str\_destroy(string \*s){

s->size = 0;

free(s->buf);

s->capacity = 0;

s->buf = NULL;

}

int str\_capacity(string \*s){

return s->capacity;

}

int str\_size(string \*s){

return s->size;

}

bool str\_is\_empty(string \*s){

return s->size == 0;

}

bool str\_grow\_buf(string \*s){

int tmp = s->capacity \* 3 / 2;

if(!tmp){

tmp = 10;

}

char \*newd = realloc(s->buf, sizeof(char) \* tmp);

if(newd != NULL) {

s->buf = newd;

s->capacity = tmp;

return true;

}

return false;

}

bool str\_push(string \*s, char new\_char){

if(str\_size(s) == str\_capacity(s)){

if(!str\_grow\_buf(s))

return false;

}

s->buf[str\_size(s)] = new\_char;

s->size++;

return true;

}

bool str\_shrink\_buf(string \*s){

int tmp = s->capacity \* 4 / 9;

if(tmp < str\_size(s)){

return true;

}

char \*newd = realloc(s->buf, sizeof(char) \* tmp);

if(newd != NULL) {

s->buf = newd;

s->capacity = tmp;

return true;

}

return false;

}

char str\_pop(string \*s){

char tmp = s->buf[str\_size(s) - 1];

str\_shrink\_buf(s);

s->size--;

return tmp;

}

char str\_get(string \*s, int i){

return s->buf[i];

}

char\* str\_get\_all(string \*s){

return s->buf;

}

**String.h**

#ifndef STRING\_H\_

#define STRING\_H\_

#include <stdbool.h>

#include <stdlib.h>

typedef struct {

int size;

int capacity;

char \*buf;

} string;

void str\_create(string \*s);

void str\_destroy(string \*s);

int str\_capacity(string \*s);

int str\_size(string \*s);

bool str\_is\_empty(string \*s);

bool str\_grow\_buf(string \*s);

bool str\_push(string \*s, char new\_char);

bool str\_shrink\_buf(string \*s);

char str\_pop(string \*s);

char str\_get(string \*s, int i);

char\* str\_get\_all(string \*s);

#endif

**Main.c**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <fcntl.h>

#include "unistd.h"

#include "string.h"

void read\_filename(string\* str, int fd){

char cur = 'a';

while(read(fd, &cur, sizeof(char)) > 0){

if(cur == '\n'){

break;

}

str\_push(str, cur);

}

}

int main(){

char template[] = "/tmp/tmpXXXXXX";

int desc = mkstemp(template);

if(desc < 0){

perror("Tmp file can not be created\n");

return -6;

}

if(ftruncate(desc, sizeof(float)) < 0){

perror("Tmp file can not filled\n");

return -7;

}

string file\_name;

str\_create(&file\_name);

read\_filename(&file\_name, 0); // 0 = STDIN

str\_push(&file\_name, '\0');

int file = open(str\_get\_all(&file\_name), O\_RDONLY);

if(file < 0){

perror("File can't be opened");

str\_destroy(&file\_name);

return -5;

}

int pid = fork();

if(pid == -1){

perror("fork error");

return -1;

}

if(pid == 0){ //child

if(dup2(file, 0) < 0){

perror("Cannot dup fd[0] to stdin");

return -5;

}

if(execl("child", "child", template, NULL) == -1){

perror("Execl child problem");

return -7;

}

} else{ //parent

int status;

wait(&status);

if(WEXITSTATUS(status)){

return -1;

}

unlink(template);

float\* fd = mmap(0, sizeof(float),

PROT\_READ | PROT\_WRITE,

MAP\_SHARED, desc, 0);

if (fd == MAP\_FAILED){

perror("mmap error\n");

return -5;

}

printf("%f\n", fd[0]);

if(munmap(fd, sizeof(float)) < 0){

perror("Munmap problem");

return -6;

}

close(desc);

}

return 0;

}

**Примеры работы**

michael@michael-VirtualBox:~/Desktop/os/os\_laba\_4$ ls

a.out child child.c file.txt main.c

michael@michael-VirtualBox:~/Desktop/os/os\_laba\_4$ cat > file.txt

0.15564 0.9 0.2

michael@michael-VirtualBox:~/Desktop/os/os\_laba\_4$ ./a.out

file.txt

1.255640

**Вывод**

В данной лабораторной работе мной был изучен и применён на практике механизм межпроцессорного взаимодействия при помощи отображаемых файлов (технология «File Mapping»). Файл отображается на оперативную память, так что мы получаем доступ к его содержимому и можем обращаться с ним как с массивом.

Таким образом, вместо многократного выполнения небыстрых запросов на чтение и запись мы выполняем отображение файла на ОЗУ и получаем произвольный доступ за О(1). По этой причине при использовании технологии «File Mapping» можно добиться ускорения работы программы в несколько раз по сравнению с использованием, например, механизма межпроцессорного взаимодействия при помощи каналов.

В качестве недостатка можно выделить тот факт, что дочерние процессы обязательно должны знать имя отображаемого файла и выполнить их отображение перед началом работы. Также файлы занимают память, несравнимую с каналами.