Московский Авиационный Институт (Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика” Кафедра №806 “Вычислительная математика и программирование”

# Лабораторная работа №3 по курсу

**«Операционные системы»**

Группа: М8О-213Б-23

Студент: Мустафаев А.Р Преподаватель: Бахарев В.Д. Оценка:

Дата: 24.11.24

Москва, 2024

**Вариант 7.**

# Постановка задачи

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 процессы должны быть представлены разными программами.

Вместо каналов используется разделяемая память.

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.

# Общий метод и алгоритм решения

Использованные системные вызовы:

* write() – записываем число байт из буфера в указанный файловый дескриптор
* read() - чтение данных из файлового дескриптора
* shm\_open() - cоздание и открытие разделяемой памяти
* ftruncate() - установка размера разделяемой памяти
* mmap() - отображение разделяемой памяти в адресное пространство
* munmap() - закрытие отображения
* shm\_unlink() - удаление объекта разделяемой памяти
* open() - открытие файла или устройства и возвращение файлового дескриптора
* close() - закрытие файла или устройства, связанного с файловым дескриптором

Для начала создаем два файла parent.c и child.c для родительского и дочернего процессов соответственно. Далее в файле parent.c создаем shared memory и делаем fork данного процесса. Делаем условие: pid = 0 (то есть если ID процесса равно нулю, значит, это дочерний процесс), тогда выполняем блок кода для дочернего процесса, иначе блок кода родительского процесса.

В дочернем процессе запускаем программу для чтения файла (child.c) с помощью команды execlp. В программе child открываем файл, путь до которого ввели в родительском процессе, перенаправляем stdin в файл, читаем файл и обрабатываем вещественные числа, через указатель записываем в shared memory полученный результат.

В родительском процессе читаем результат из shared memory и выводим его на экран, ожидаем завершения дочернего процесса с помощью wait.

# Код программы

**parent.c**

#include <sys/mman.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <string.h>

#define BUF\_SIZE 1024

#define SHM\_SIZE 4096

int main() {

pid\_t pid;

int shm\_fd = shm\_open("/my\_shared\_memory", O\_CREAT | O\_RDWR, 0666);

if (shm\_fd == -1) {

const char msg[] = "error: failed to open shared memory\n";

write(STDERR\_FILENO, msg, strlen(msg));

exit(EXIT\_FAILURE);

}

if (ftruncate(shm\_fd, SHM\_SIZE) == -1) {

const char msg[] = "error: failed to truncate shared memory\n";

write(STDERR\_FILENO, msg, strlen(msg));

exit(EXIT\_FAILURE);

}

void \*ptr = mmap(0, SHM\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (ptr == MAP\_FAILED) {

const char msg[] = "error: failed to mmap\n";

write(STDERR\_FILENO, msg, strlen(msg));

exit(EXIT\_FAILURE);

}

char filename[BUF\_SIZE];

const char prompt[] = "Enter filename: ";

write(STDOUT\_FILENO, prompt, strlen(prompt));

int n = read(STDIN\_FILENO, filename, sizeof(filename) - 1);

if (n > 0) {

filename[n - 1] = '\0';

} else {

filename[0] = '\0';

}

pid = fork();

if (pid == -1) {

const char msg[] = "error: failed to fork\n";

write(STDERR\_FILENO, msg, strlen(msg));

exit(EXIT\_FAILURE);

}

if (pid == 0) {

execlp("./child", "child", filename, (char \*)NULL);

const char msg[] = "error: failed to execlp\n";

write(STDERR\_FILENO, msg, strlen(msg));

exit(EXIT\_FAILURE);

} else {

wait(NULL);

const char msg[] = "Read from shared memory:\n";

write(STDOUT\_FILENO, msg, strlen(msg));

write(STDOUT\_FILENO, (char \*)ptr, strlen((char \*)ptr));

munmap(ptr, SHM\_SIZE);

close(shm\_fd);

shm\_unlink("/my\_shared\_memory");

}

return 0;

}

**child.c**

#include <sys/mman.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define BUF\_SIZE 1024

#define SHM\_SIZE 4096

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

if (argc != 2) {

const char msg[] = "Usage: ./child <filename>\n";

write(STDERR\_FILENO, msg, strlen(msg));

exit(EXIT\_FAILURE);

}

int filefd = open(argv[1], O\_RDONLY);

if (filefd == -1) {

const char msg[] = "error: failed to open file\n";

write(STDERR\_FILENO, msg, strlen(msg));

exit(EXIT\_FAILURE);

}

int shm\_fd = shm\_open("/my\_shared\_memory", O\_RDWR, 0666);

if (shm\_fd == -1) {

perror("shm\_open");

close(filefd);

exit(EXIT\_FAILURE);

}

void \*ptr = mmap(0, SHM\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (ptr == MAP\_FAILED) {

perror("mmap");

close(filefd);

close(shm\_fd);

exit(EXIT\_FAILURE);

}

dup2(filefd, STDIN\_FILENO);

close(filefd);

char buffer[BUF\_SIZE];

size\_t bytes\_read;

char num\_str[BUF\_SIZE];

int num\_index = 0;

int in\_number = 0;

double sum = 0;

while ((bytes\_read = read(STDIN\_FILENO, buffer, BUF\_SIZE)) > 0) {

for (size\_t i = 0; i < bytes\_read; i++) {

if (isdigit(buffer[i]) || buffer[i] == '.' || (buffer[i] == '-' && !in\_number)) {

if (num\_index < BUF\_SIZE - 1) {

num\_str[num\_index++] = buffer[i];

in\_number = 1;

} else {

const char msg[] = "error: number too long\n";

write(STDERR\_FILENO, msg, strlen(msg));

munmap(ptr, SHM\_SIZE);

close(shm\_fd);

exit(EXIT\_FAILURE);

}

} else if (in\_number) {

num\_str[num\_index] = '\0';

double number = atof(num\_str);

sum += number;

num\_index = 0;

in\_number = 0;

}

}

}

if (bytes\_read == -1) {

perror("read");

munmap(ptr, SHM\_SIZE);

close(shm\_fd);

exit(EXIT\_FAILURE);

}

if (in\_number) {

num\_str[num\_index] = '\0';

double number = atof(num\_str);

sum += number;

}

size\_t n = snprintf(ptr, SHM\_SIZE, "Sum: %f\n", sum);

if (n >= SHM\_SIZE) {

const char msg[] = "error: output too large for shared memory\n";

write(STDERR\_FILENO, msg, strlen(msg));

munmap(ptr, SHM\_SIZE);

close(shm\_fd);

exit(EXIT\_FAILURE);

}

munmap(ptr, SHM\_SIZE);

close(shm\_fd);

close(filefd);

return 0;

}

# Протокол работы программы

**Некорректный ввод:**

**traktor@traktor-MaiBook-X-series:~/OS/MAI\_OS/lab03/src$ ./parent**

**Enter filename:**

**error: failed to open file**

**Read from shared memory:**

**traktor@traktor-MaiBook-X-series:~/OS/MAI\_OS/lab03/src$ ./parent**

**Enter filename: test.txt**

**Read from shared memory:**

**Sum: 24.900000**

**Strace:**

traktor@traktor-MaiBook-X-series:~/OS/MAI\_OS/lab03/src$ strace -f ./parent

**execve("./parent", ["./parent"], 0x7ffdd39621a8 /\* 79 vars \*/) = 0**

brk(NULL) = 0x5eddfe79e000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7c950514c000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (Нет такого файла или каталога)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=73955, ...}) = 0

mmap(NULL, 73955, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7c9505139000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7c9504e00000

mmap(0x7c9504e28000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7c9504e28000

mmap(0x7c9504fb0000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7c9504fb0000

mmap(0x7c9504fff000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7c9504fff000

mmap(0x7c9505005000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7c9505005000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7c9505136000

arch\_prctl(ARCH\_SET\_FS, 0x7c9505136740) = 0

set\_tid\_address(0x7c9505136a10) = 61994

set\_robust\_list(0x7c9505136a20, 24) = 0

rseq(0x7c9505137060, 0x20, 0, 0x53053053) = 0

mprotect(0x7c9504fff000, 16384, PROT\_READ) = 0

mprotect(0x5eddfe3c5000, 4096, PROT\_READ) = 0

mprotect(0x7c9505184000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7c9505139000, 73955) = 0

**openat(AT\_FDCWD, "/dev/shm/my\_shared\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 3**

**ftruncate(3, 4096) = 0**

**mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7c950514b000**

**write(1, "Enter filename: ", 16Enter filename: ) = 16**

**read(0, test.txt**

**"test.txt\n", 1023) = 9**

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 62064 attached**

**, child\_tidptr=0x7c9505136a10) = 62064**

[pid 62064] set\_robust\_list(0x7c9505136a20, 24 <unfinished ...>

[pid 61994] wait4(-1, <unfinished ...>

[pid 62064] <... set\_robust\_list resumed>) = 0

**[pid 62064] execve("./child", ["child", "test.txt"], 0x7ffdfde15f58 /\* 79 vars \*/) = 0**

[pid 62064] brk(NULL) = 0x590eddac2000

[pid 62064] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x74f22ceb5000

[pid 62064] access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (Нет такого файла или каталога)

[pid 62064] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

[pid 62064] fstat(3, {st\_mode=S\_IFREG|0644, st\_size=73955, ...}) = 0

[pid 62064] mmap(NULL, 73955, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x74f22cea2000

[pid 62064] close(3) = 0

[pid 62064] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

[pid 62064] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

[pid 62064] pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

[pid 62064] fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 0

[pid 62064] pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

[pid 62064] mmap(NULL, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x74f22cc00000

[pid 62064] mmap(0x74f22cc28000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x74f22cc28000

[pid 62064] mmap(0x74f22cdb0000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x74f22cdb0000

[pid 62064] mmap(0x74f22cdff000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x74f22cdff000

[pid 62064] mmap(0x74f22ce05000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x74f22ce05000

[pid 62064] close(3) = 0

[pid 62064] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x74f22ce9f000

[pid 62064] arch\_prctl(ARCH\_SET\_FS, 0x74f22ce9f740) = 0

[pid 62064] set\_tid\_address(0x74f22ce9fa10) = 62064

[pid 62064] set\_robust\_list(0x74f22ce9fa20, 24) = 0

[pid 62064] rseq(0x74f22cea0060, 0x20, 0, 0x53053053) = 0

[pid 62064] mprotect(0x74f22cdff000, 16384, PROT\_READ) = 0

[pid 62064] mprotect(0x590edcf55000, 4096, PROT\_READ) = 0

[pid 62064] mprotect(0x74f22ceed000, 8192, PROT\_READ) = 0

[pid 62064] prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 62064] munmap(0x74f22cea2000, 73955) = 0

**[pid 62064] openat(AT\_FDCWD, "test.txt", O\_RDONLY) = 3**

**[pid 62064] openat(AT\_FDCWD, "/dev/shm/my\_shared\_memory", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = 4**

**[pid 62064] mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x74f22ceb4000**

**[pid 62064] dup2(3, 0) = 0**

[pid 62064] close(3) = 0

[pid 62064] read(0, "12.3 12.6\n", 1024) = 10

[pid 62064] read(0, "", 1024) = 0

[pid 62064] munmap(0x74f22ceb4000, 4096) = 0

[pid 62064] close(4) = 0

[pid 62064] close(3) = -1 EBADF (Неправильный дескриптор файла)

[pid 62064] exit\_group(0) = ?

[pid 62064] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL) = 62064

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=62064, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

**write(1, "Read from shared memory:\n", 25Read from shared memory:**

**) = 25**

**write(1, "Sum: 24.900000\n", 15Sum: 24.900000**

**) = 15**

**munmap(0x7c950514b000, 4096) = 0**

**close(3) = 0**

**unlink("/dev/shm/my\_shared\_memory") = 0**

exit\_group(0) = ?

+++ exited with 0 +++

# Вывод

Язык Си обеспечивает широкие возможности для организации синхронизации между различными приложениями. Одним из таких механизмов является разделяемая память (shared memory), которая позволяет нескольким приложениям совместно использовать общий ресурс и эффективно взаимодействовать через файл.