Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету

"Операционные системы"

№3

Студент: Филиппов А. М.

Преподаватель: Миронов Е.С.

Группа: М8О-201Б-24

Дата:

Оценка:

Подпись:

Москва, 2025

Оглавление

Цель работы

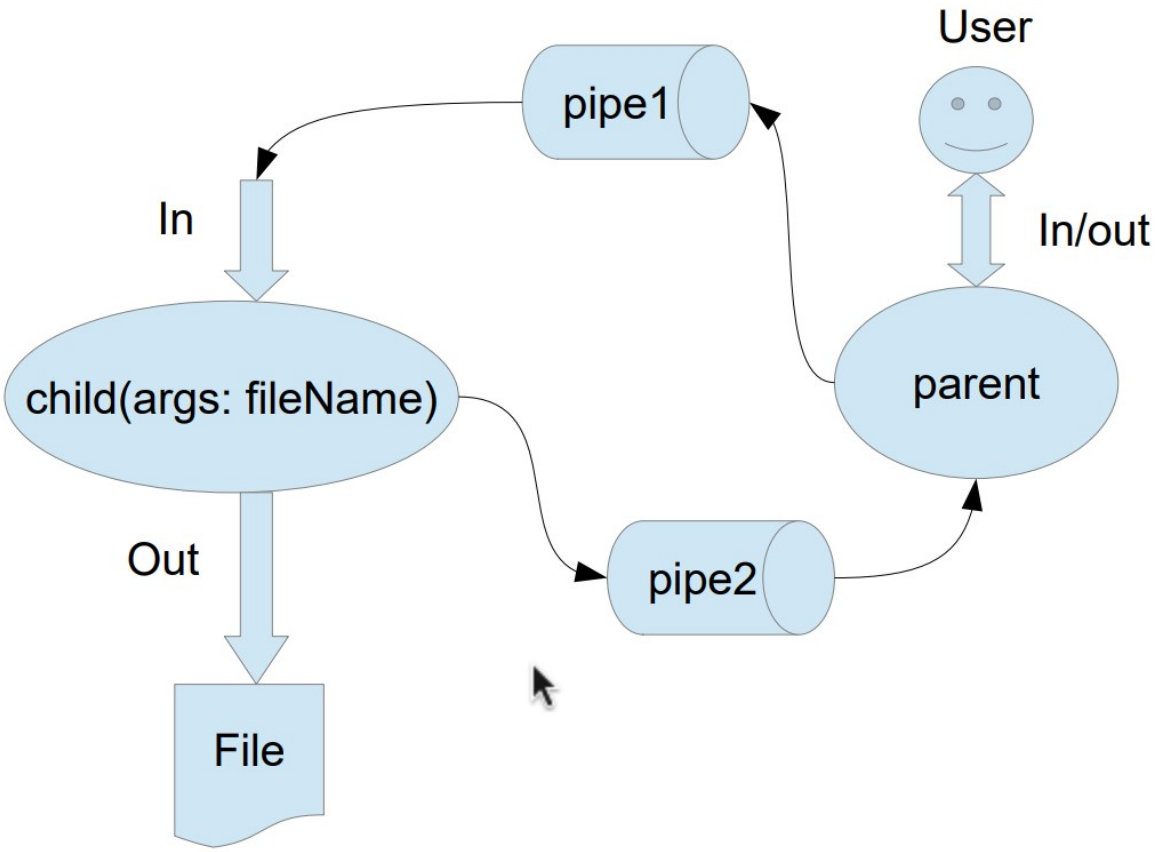
Приобретение практических навыков в:

- Освоение принципов работы с файловыми системами
- Обеспечение обмена данных между процессами посредством технологии «File mapping»

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.



Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

5 вариант) Пользователь вводит команды вида: «число

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: iostream, fcntl.h, cerrno, sys/mman.h, cstring, sys/wait.h. В программе используются следующие системные вызовы:

1. ftruncate - обрезает файл до заданного размера.
2. mmap, munmap - отображает файлы или устройства в памяти, или удаляет их отображение.
3. close - закрывает файловый дескриптор.
4. execl - запуск файла на исполнение.

Общий алгоритм решения

В данной программе создается отображаемый файл с помощью функций shm_open и ftruncate. Затем, используя функцию mmap, отображенный файл связывается с памятью процесса и получается указатель на эту память. Дочерние процессы изменяют данные в памяти, а родительский процесс выводит результат в стандартный поток вывода. По окончании работы, процесс освобождает ресурсы функциями munmap и shm_unlink.

Программа обрабатывает системные ошибки при работе с отображаемым

файлом, выводя сообщения об ошибках и завершая работу в случае их возникновения.

Реализация

main.c

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

#include "shared_block_t.h"

static int create_shared_block(shared_block_t **block, int *fd,
char shm_path[]) {
pid_t pid = getpid();
snprintf(shm_path, 64, "/lab03_shm_%d", pid);

*fd = shm_open(shm_path, O_CREAT | O_RDWR, 0666);
if (*fd == -1) {
perror("shm_open");
return -1;
}

if (ftruncate(*fd, sizeof(shared_block_t)) == -1) {
perror("ftruncate");
close(*fd);
shm_unlink(shm_path);
return -1;
}

*block = mmap(NULL, sizeof(shared_block_t), PROT_READ | PROT_WRITE,
MAP_SHARED, *fd, 0);
if (*block == MAP_FAILED) {
perror("mmap");
close(*fd);
shm_unlink(shm_path);
return -1;
}

return 0;
```

```
}

int main(void) {
shared_block_t *shared = NULL;
int shm_fd = -1;
char shm_path[64] = {0};

if (create_shared_block(&shared, &shm_fd, shm_path) == -1) {
return 1;
}

if (sem_init(&shared->sem_child, 1, 0) == -1) {
perror("sem_init child");
munmap(shared, sizeof(shared_block_t));
close(shm_fd);
return 1;
}

if (sem_init(&shared->sem_parent, 1, 0) == -1) {
perror("sem_init parent");
sem_destroy(&shared->sem_child);
munmap(shared, sizeof(shared_block_t));
close(shm_fd);
return 1;
}

char filename[256];
printf("Введите имя файла: ");
if (fgets(filename, sizeof(filename), stdin) == NULL) {
perror("Ошибка чтения имени файла");
sem_destroy(&shared->sem_child);
sem_destroy(&shared->sem_parent);
munmap(shared, sizeof(shared_block_t));
close(shm_fd);
return 1;
}
filename[strcspn(filename, "\n")] = 0;

pid_t child_pid = fork();
if (child_pid == -1) {
perror("fork");
sem_destroy(&shared->sem_child);
sem_destroy(&shared->sem_parent);
munmap(shared, sizeof(shared_block_t));
close(shm_fd);
shm_unlink(shm_path);
return 1;
}
```

```
if (child_pid == 0) {
munmap(shared, sizeof(shared_block_t));
close(shm_fd);

execlp("./child", "child", shm_path, filename, (char *)NULL);
perror("execlp");
return 1;
}

printf("Введите числа для проверки (число, затем Enter).\n");
printf("Ввод отрицательного или простого числа завершит программу.\n");

while (1) {
int number = 0;
printf("Введите число: ");
int ret = scanf("%d", &number);
if (ret != 1) {
if (feof(stdin)) {
fprintf(stderr, "EOF, завершаем работу.\n");
} else {
perror("scanf");
}
shared->number = -1;
sem_post(&shared->sem_child);
sem_wait(&shared->sem_parent);
break;
}

shared->number = number;
sem_post(&shared->sem_child);

if (sem_wait(&shared->sem_parent) == -1) {
if (errno == EINTR) {
continue;
}
perror("parent: sem_wait");
break;
}

if (shared->status) {
break;
}
}

int status = 0;
waitpid(child_pid, &status, 0);
```

```
sem_destroy(&shared->sem_child);
sem_destroy(&shared->sem_parent);
munmap(shared, sizeof(shared_block_t));
close(shm_fd);
shm_unlink(shm_path);

return 0;
}
```

child.c

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/mman.h>
#include <sys/shm.h>
#include <unistd.h>

#include "shared_block_t.h"

static int is_prime(int number) {
int divisors_count = 0;

if (number <= 1) {
return 0;
}

for (int i = 2; i * i <= number; i++) {
if (number % i == 0) {
divisors_count++;
}
}

return divisors_count == 0;
}

int main(int argc, char *argv[]) {
if (argc != 3) {
fprintf(stderr, "Usage: %s <shm_path> <output_filename>\n", argv[0]);
return 1;
}

const char *shm_path = argv[1];
const char *filename = argv[2];

int shm_fd = shm_open(shm_path, O_RDWR, 0666);
if (shm_fd == -1) {
```

```
perror("child: shm_open");
return 1;
}

shared_block_t *shared = mmap(NULL, sizeof(shared_block_t),
PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
if (shared == MAP_FAILED) {
perror("child: mmap");
close(shm_fd);
return 1;
}

close(shm_fd);

FILE *output_file = fopen(filename, "w");
if (output_file == NULL) {
perror("child: fopen");
munmap(shared, sizeof(shared_block_t));
return 1;
}

while (1) {
if (sem_wait(&shared->sem_child) == -1) {
if (errno == EINTR) {
continue;
}
perror("child: sem_wait");
fclose(output_file);
munmap(shared, sizeof(shared_block_t));
return 1;
}

int number = shared->number;

if (number < 0 || is_prime(number)) {
if (number < 0) {
printf("Число отрицательное, выход...\n");
} else {
printf("Число простое, выход...\n");
}
shared->status = 1;
sem_post(&shared->sem_parent);
break;
}

if (fprintf(output_file, "%d\n", number) < 0) {
perror("child: fprintf");
shared->status = 1;
```

```
sem_post(&shared->sem_parent);
break;
}

shared->status = 0;
sem_post(&shared->sem_parent);
}

fclose(output_file);
munmap(shared, sizeof(shared_block_t));
return 0;
}
```

Пример работы

```
…/lab_01/build master □ ? ❯ ./lab_01_osi_app
Введите имя файла: out.txt
Введите числа для проверки (число, затем Enter).
Ввод отрицательного или простого числа завершит программу.
Введите число: 14
Введите число: 66
Введите число: 55
Введите число: 13
Число простое, выход...
Дочерний процесс завершен.
Родительский процесс завершен.

…/lab_01/build master □ ? ❯ cat out.txt
14
66
55
```

Вывод

В Си помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например отображение файла в память, такой подход работает быстрее, за счет отсутствия постоянных вызовов read, write и тратит меньше памяти под кэш. После отображения возвращается void*, который можно привести к своему

указателю на тип и обрабатывать данные как массив, где возвращенный указатель – указатель на первый элемент.