Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Тема работы**

**“Межпроцессорное взаимодействие через memory-mapped files”**

Студент: Ковриженков Дмитрий Олегович

Группа: М8О-203Б-23

Вариант: 13

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

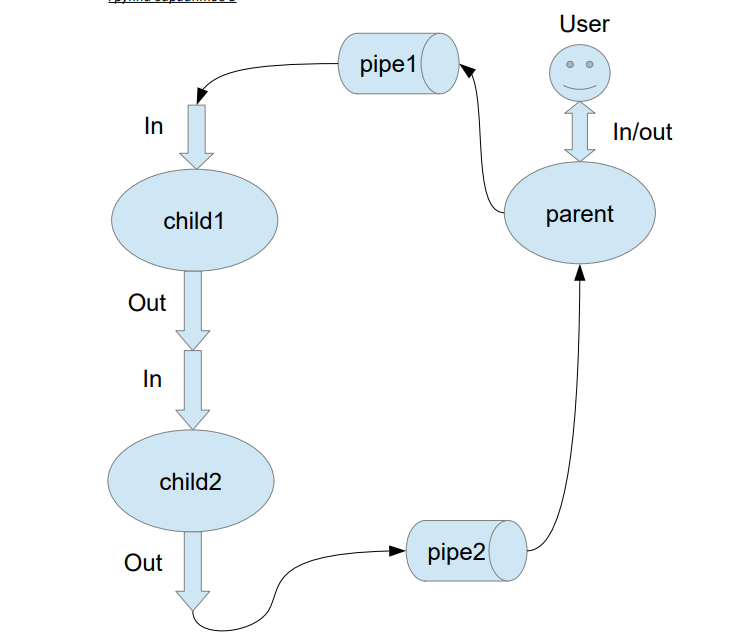
Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Лабораторная работа №1**

**Задача:** Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.



В 13 варианте Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «\_».

**Общие сведения**

В данной лабораторной работе реализуется межпроцессное взаимодействие с использованием разделяемой памяти (mmap) и создания дочерних процессов (fork). Родительский процесс (main.c) вызывает init\_child\_process(), который создает дочерний процесс. В дочернем процессе (processes.c) происходит ввод строки от пользователя, её обработка и вывод результата.

Обработка строки включает: Преобразование всех букв в нижний регистр (child1). Замена пробелов на подчеркивания (child2). Для обмена данными между процессами используется разделяемая память, создаваемая с помощью mmap() (memory\_map.c). В конце работы память освобождается через munmap(). Ошибки обрабатываются функцией handle\_error() (error\_handling.c), которая выводит сообщение и завершает выполнение программы. Программа использует: fork() для создания процессов. mmap() для организации разделяемой памяти между процессами. tolower() для изменения регистра символов. strncpy() для работы со строками с учетом размера буфера.

Вывод

В процессе выполнения лабораторной работы были изучены и применены механизмы межпроцессного взаимодействия, включая разделяемую память и системные вызовы работы с процессами. Основные выводы: mmap() позволяет эффективно передавать данные между процессами. fork() создает отдельный процесс для обработки строк, упрощая многопоточность. Разделяемая память ускоряет передачу данных по сравнению с pipe() или message queue. Корректная обработка ошибок (handle\_error()) делает код более устойчивым к сбоям. Данная лабораторная работа демонстрирует основные подходы к организации межпроцессного взаимодействия в Unix-подобных системах.

Приложение

src\error\_handling.c

#include <stdio.h>

#include <stdlib.h>

#include "error\_handling.h"

void handle\_error(const char \*message) {

perror(message);

exit(EXIT\_FAILURE);

}

src\ memory\_map.c

#include <sys/mman.h>

#include <stddef.h>

#include "memory\_map.h"

#include "error\_handling.h"

void\* create\_shared\_memory(size\_t size) {

void \*shared\_mem = mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

if (shared\_mem == MAP\_FAILED) {

handle\_error("mmap failed");

}

return shared\_mem;

}

void release\_shared\_memory(void \*shared\_mem, size\_t size) {

if (munmap(shared\_mem, size) == -1) {

handle\_error("munmap failed");

}

}

src\ processes.c

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include "error\_handling.h"

#include "memory\_map.h"

#include "processes.h"

void init\_child\_process(int BUFFER\_SIZE) {

pid\_t pid = fork();

if (pid < 0) {

handle\_error("fork error");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

char input[BUFFER\_SIZE];

char output[BUFFER\_SIZE];

while (1) {

printf("Введите строку (или нажмите Ctrl+D для завершения): ");

if (fgets(input, BUFFER\_SIZE, stdin) == NULL) {

break;

}

input[strcspn(input, "\n")] = '\0';

int result = process\_string(input, output, BUFFER\_SIZE);

if (result != 0) {

handle\_error("Не удалось обработать строку");

}

printf("Результат обработки: %s\n", output);

}

exit(0);

}

}

void child1(char \*shared\_mem) {

for (int i = 0; shared\_mem[i] != '\0'; i++) {

shared\_mem[i] = tolower(shared\_mem[i]);

}

}

void child2(char \*shared\_mem) {

for (int i = 0; shared\_mem[i] != '\0'; i++) {

if (shared\_mem[i] == ' ') {

shared\_mem[i] = '\_';

}

}

}

int process\_string(const char \*input, char \*output, size\_t size) {

if (size < strlen(input) + 1) {

return -1;

}

char \*shared\_mem = create\_shared\_memory(size);

if (!shared\_mem) {

return -1;

}

strncpy(shared\_mem, input, size - 1);

shared\_mem[size - 1] = '\0';

child1(shared\_mem);

child2(shared\_mem);

strncpy(output, shared\_mem, size - 1);

output[size - 1] = '\0';

release\_shared\_memory(shared\_mem, size);

return 0;

}

include\ error\_handling.h

#ifndef ERROR\_HANDLING\_H

#define ERROR\_HANDLING\_H

void handle\_error(const char \*message);

#endif

include\ memory\_map.h

#ifndef MEMORY\_MAP\_H

#define MEMORY\_MAP\_H

#include <sys/mman.h>

#include <stddef.h>

void\* create\_shared\_memory(size\_t size);

void release\_shared\_memory(void \*shared\_mem, size\_t size);

#endif

include\ processes.h

#ifndef PROCESSES\_H

#define PROCESSES\_H

#include <stddef.h>

void init\_child\_process(int BUFFER\_SIZE);

void child1(char \*shared\_mem);

void child2(char \*shared\_mem);

int process\_string(const char \*input, char \*output, size\_t size);

#endif

main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include "memory\_map.h"

#include "processes.h"

#include "error\_handling.h"

#define BUFFER\_SIZE 1024

int main() {

init\_child\_process(BUFFER\_SIZE);

wait(NULL);

return 0;

}

CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)

project(InterProcessCommunication C)

set(CMAKE\_C\_STANDARD 11)

set(CMAKE\_C\_STANDARD\_REQUIRED True)

include\_directories(include)

add\_executable(lab3

main.c

src/error\_handling.c

src/memory\_map.c

src/processes.c

)

Пример вывода:

