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

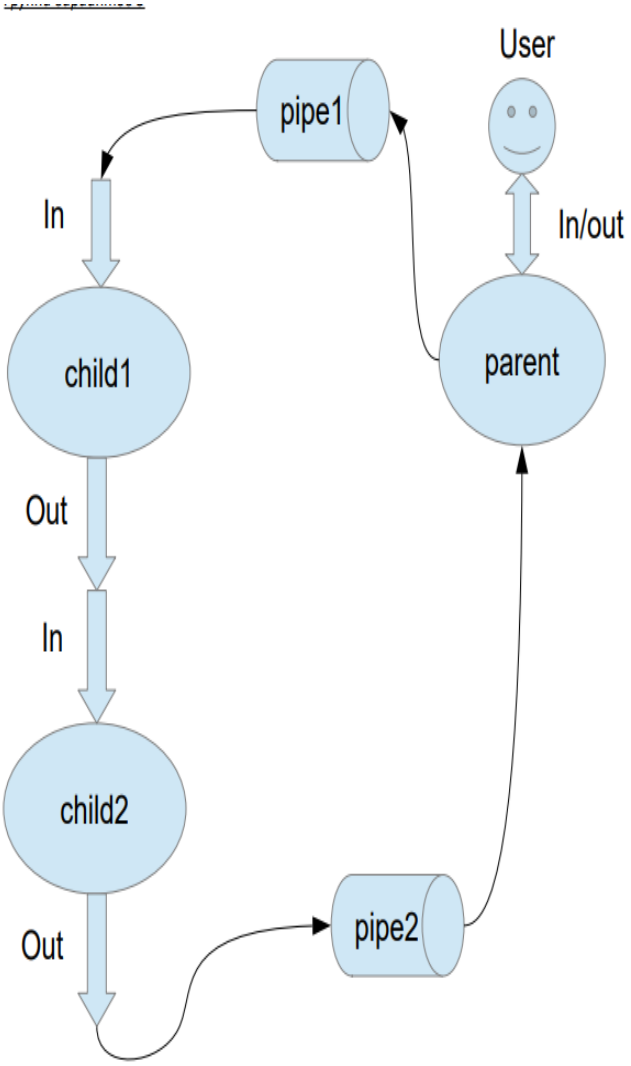
Преподаватель: Миронов Евгений Сергеевич

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: