Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина «Конструирование программ»

**ОТЧЕТ**

к лабораторной работе №7

на тему:

**«ИНТЕГРАЦИЯ АССЕМБЛЕРНЫХ ПРЕРЫВАНИЙ В ПРОЕКТЫ НА C++»**

БГУИР 6-05-0612-02 73

|  |
| --- |
| Выполнила студентка группы 353504  ЛЕБЕДЕВА Милана Валерьевна |
|  |
| (дата, подпись студента) |
| Проверил ассистент каф. Информатики  РОМАНЮК Максим Валерьевич |
|  |
| (дата, подпись преподавателя) |

Минск 2024

# 1 Индивидуальное задание

**Задание 1. Вариант 13.** Создайте инструмент, который будет "пинговать" ваш диск, записывая и затем считывая небольшие блоки данных для измерения задержек. Идея состоит в том, чтобы определить задержку (латентность) записи и чтения диска. Запросите у пользователя размер блока данных для теста (например, 1 КБ, 10 КБ, 100 КБ). Запросите количество итераций (например, 100 пингов).

Запись на диск: Зафиксируйте текущее время с помощью прерывания таймера. Запишите блок данных указанного размера на диск. Зафиксируйте время после записи. Рассчитайте разницу между временами для определения времени записи.

Чтение с диска: Зафиксируйте текущее время. Прочтите блок данных с диска. Зафиксируйте время после чтения. Рассчитайте разницу между временами для определения времени чтения.

Рассчитайте среднее время записи и чтения за все итерации.

Используйте прерывания для обработки ошибок (например, недостаточно места на диске). Многопоточное тестирование: При запуске теста, создайте несколько потоков (например, 4), каждый из которых будет "пинговать" диск. Это позволит симулировать реальную среду, где диск может обрабатывать несколько операций одновременно (реализация на C++).

# 2 Выполнение работы

Для написания данной лабораторной работы используется NASM и редактор VS Code в среде Linux. Ниже представлен листинг кода файла .asm.

section .data

disk\_name db "/home/milana/projects/test/file.txt", 0

buffer db 1024 dup('A')

section .bss

start\_time resq 1

end\_time resq 1

disk\_descriptor resq 1

section .text

global open\_disk

global write\_block

global read\_block

global close\_disk

global get\_time

open\_disk:

;ОТКРЫТИЕ ДИСКА

;дескриптор - rdi, имя файла - rsi, флаги - rdx, mode - r10

;возврат: дескприптор диска - rax

mov rax, 257 ;syscall for open

lea rsi, [rel disk\_name]

mov rdx, 2 ;O\_RDWR (read/write)

mov r10, 0 ;no special flags (mode)

syscall

mov [rel disk\_descriptor], rax

cmp rax, -1 ;проверка на ошибку

je open\_error

ret

open\_error:

mov rax, 60 ;syscall для выхода

mov rdi, 1 ;код ошибки

syscall

get\_time:

; ПОЛУЧЕНИЕ ТЕКУЩЕГО ВРЕМЕНИ

rdtsc

shl rdx, 32

or rax, rdx

ret

;mov rax, 228 ;clock\_gettime

;mov rdi, 0 ;CLOCK\_REALTIME

;lea rsi, [rel start\_time] ;(структура timespec) - start\_time

;syscall

;ret

write\_block:

;ЗАПИСЬ БЛОКА НА ДИСК

;дескриптор - rdi, буфер с данными - rsi, количество байт - rdx

;rdi - первый параметр функции (размер блока данных в байтах)

call get\_time

mov rbx, rax

mov rax, 1 ;sys\_write

mov rdx, rdi

mov rdi, [rel disk\_descriptor]

lea rsi, [rel buffer]

syscall

;получаем время после записи

call get\_time

mov [rel end\_time], rax

mov rax, [rel end\_time]

sub rax, rbx ;вычисляем время записи

cmp rax, -1 ;проверка на ошибку

je write\_error

ret

write\_error:

mov rax, 60 ;syscall для выхода

mov rdi, 2 ;код ошибки

syscall

read\_block:

;ЧТЕНИЕ ДАННЫХ С ДИСКА

;дескриптор - rdi, буфер - rsi, количество байт - rdx

call get\_time

mov rbx, rax

mov rax, 0 ;sys\_read

mov rdx, rdi

mov rdi, [rel disk\_descriptor]

lea rsi, [rel buffer]

syscall

;получаем время после чтения

call get\_time

mov [rel end\_time], rax

sub rax, rbx ;вычисляем время чтения

cmp rax, -1 ;проверка на ошибку

je read\_error

ret

read\_error:

mov rax, 60 ;syscall для выхода

mov rdi, 3 ;код ошибки

syscall

close\_disk:

;ЗАКРЫТИЕЕ ФАЙЛА

mov rax, 3 ;syscall для close

mov rdi, [rel disk\_descriptor]

syscall

ret

section .note.GNU-stack

Программа включает следующие сегменты: .data cодержит инициализированные данные disk\_name – строка с именем файла, который будет открыт, заканчивающаяся нулевым байтом, buffer – массив из 1024 байтов, заполненный символами 'A'. .bss используется для неинициализированных данных start\_time – резервирует место для хранения времени начала операции, end\_time – резервирует место для хранения времени завершения операции, disk\_descriptor – резервирует место для хранения дескриптора открытого файла. .text содержит исполняемый код. Метки open\_disk для открытия файла, get\_time получает текущее время, write\_block записывает данные из буфера на диск и вычисляет время записи, read\_block читает данные с диска в буфер и вычисляет время чтения, close\_disk закрывает открытый файл. Ниже представлен листинг кода файла .cpp.

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <limits>

#include <fstream>

#include <chrono>

#include <cstring>

#include <atomic>

extern "C" {

void open\_disk();

time\_t write\_block(size\_t size);

time\_t read\_block(size\_t size);

void close\_disk();

}

std::mutex console\_mutex;

std::atomic<time\_t> total\_write\_time{0};

std::atomic<time\_t> total\_read\_time{0};

void checkInput(size\_t& block\_size) {

std::cout << "Enter size of block (in Kb): ";

std::cin >> block\_size;

while (std::cin.fail() || block\_size <= 0) {

std::cout << "Invalid input. Try again: ";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cin >> block\_size;

}

block\_size \*= 1024; //в байты

}

void checkInputI(int& it) {

std::cout << "Enter count of iterations: ";

std::cin >> it;

while (std::cin.fail() || it <= 0) {

std::cout << "Invalid input. Try again: ";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cin >> it;

}

}

void disk\_ping(int thread\_id, size\_t block\_size, int iterations) {

time\_t thread\_write\_time = 0;

time\_t thread\_read\_time = 0;

for (int i = 0; i < iterations; ++i) {

time\_t write\_time = write\_block(block\_size);

thread\_write\_time += write\_time;

time\_t read\_time = read\_block(block\_size);

thread\_read\_time += read\_time;

{

std::lock\_guard<std::mutex> lock(console\_mutex);

std::cout << "Thread " << thread\_id << ", Iteration " << i + 1

<< ": Write time = " << write\_time << " ns, Read time = " << read\_time << " ns\n";

}

}

total\_write\_time += thread\_write\_time;

total\_read\_time += thread\_read\_time;

}

int main() {

size\_t block\_size;

int iterations;

checkInput(block\_size);

checkInputI(iterations);

open\_disk();

std::vector<std::thread> threads;

for (int i = 0; i < 4; ++i) { // Всегда 4 потока

threads.emplace\_back(disk\_ping, i + 1, block\_size, iterations);

}

for (auto& thread : threads) {

thread.join();

}

close\_disk();

time\_t avg\_write\_time = total\_write\_time / (iterations \* 4); // 4 потока

time\_t avg\_read\_time = total\_read\_time / (iterations \* 4);

std::cout << "\n--- Results ---\n";

std::cout << "Total Threads: 4\n";

std::cout << "Average Write Time: " << avg\_write\_time << " ns\n";

std::cout << "Average Read Time: " << avg\_read\_time << " ns\n";

return 0;

}

В файле .cpp определены внешние функции open\_disk(), write\_block(), read\_block() и close\_disk(), которые реализуют операции с диском. Функции checkInput() и checkInputI – функции проверки ввода, disk\_ping() выполняет операции записи и чтения в цикле, измеряя время каждой операции и выводя результаты, main() запрашивает у пользователя параметры (размер блока и количество итераций). Открывает диск, создает 4 потока, которые выполняют disk\_ping(), и ждет их завершения. Закрывает диск и вычисляет среднее время записи и чтения, выводя результаты.

# Вывод

В ходе лабораторной работы был изучен ассемблер NASM и интеграция его с языком C++ в среде VS Code на Linux, а также основы написания ассемблерного кода и методы его вызова из C++, что значительно расширяет функциональные возможности разработанной программы.