МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчёт по лабораторной работе № 22**

по дисциплине: ”Системное программирование”

на тему: ***”Многозадачность процессов***

***и этапы компиляции”***

Вариант -

Выполнил**:** студент группы 10701222 Зухта К.М.

Медведский Е.В.

Принял**:** пр. Давыденко Н.В.

# Лабораторная работа № 22.

Изучить встроенный инструментарий для разработки приложений под семейство ОС Linux и фундаментальные основы системного программирования с использованием компиляторов gcc/g++, отладчика gdb и других для проектирования, компиляции, отладки и запуска приложений на языке программирования С/C++. Изучить базовую концепцию операционной системы, «Процесс». Получить представление о многозадачности процессов.

## Задание 1.

Используя материалы (исходные файлы) упражнения №1 разработайте про-  
грамму, в которой дочерний процесс запускается системным вызовом fork в  
цикле, ограниченном счетчиком проходов по циклу. Через пять секунд роди-  
тельский процесс должен закончить работу, а дочерний процесс выдать ко-  
манду ps. Внимание. Список выполняющихся процессов можно получить с  
помощью команды $ps.  
Теперь нужно осознать, как осуществляется взаимодействие процессов. Стан-  
дартным образом для этой цели используется канал (pipe). Но сейчас мы осуще-  
ствим взаимодействие через текстовый файл. Создайте программу, в которой  
один процесс будет писать в файл какую-то строку, а второй будет считывать.  
Проведите компиляцию однофайлового проекта помощью скрипты bash с  
прохождением всех стадий компиляции.  
Сборка проекта должна содержать файлы с результатами препроцессинга.  
Исследуйте фалы препроцессора, найдите в них код своей программы.  
Определите размеры исходных, препроцессорных, ассемблерных, объектных  
и исполняемых файлов. С помощью соответствующей консольной команды  
или с помощь проводника определите форматы этих файлов. Результаты  
подтвердите скриншотами. Сделайте краткий вывод по выполненной работе.

# Листинг кода

(создание процессов с fork и взаимодействие через файл.)

main.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#define FILENAME "example.txt"

int main() {

pid\_t child\_pid = fork();

if (child\_pid < 0) {

perror("Fork failed");

return 1;

}

if (child\_pid == 0) {

// Дочерний процесс

// Запись в файл

FILE \*file = fopen(FILENAME, "w");

if (file == NULL) {

perror("Failed to open file");

return 1;

}

fprintf(file, "Hello from the child process!\n");

fclose(file);

// Выполнение команды ps через 5 секунд

sleep(5);

printf("Child process: Executing 'ps' command\n");

execlp("ps", "ps", NULL);

perror("execlp failed"); // Если execlp не сработал

return 1;

} else {

// Родительский процесс

for (int i = 0; i < 5; i++) {

printf("Parent process working! Iteration %d\n", i + 1);

sleep(1); // Ждем 1 секунду

}

// Чтение из файла

sleep(1); // Ждем, чтобы дочерний процесс успел записать

FILE \*file = fopen(FILENAME, "r");

if (file == NULL) {

perror("Failed to open file");

return 1;

}

char buffer[256];

fgets(buffer, sizeof(buffer), file);

printf("Parent process read: %s", buffer);

fclose(file);

printf("Parent process: Exiting\n");

return 0;

}

}

compile\_and\_analyze.sh (скрипт для компиляции и анализа)

#!/bin/bash

# Компиляция с выводом файлов препроцессора

gcc -E main.c -o main.i

gcc -S main.c -o main.s

gcc -c main.c -o main.o

gcc main.o -o main

# Определяем размеры файлов

echo "Size of source file:"

du -h main.c

echo "Size of preprocessed file:"

du -h main.i

echo "Size of assembly file:"

du -h main.s

echo "Size of object file:"

du -h main.o

echo "Size of executable file:"

du -h main

# Формат файлов

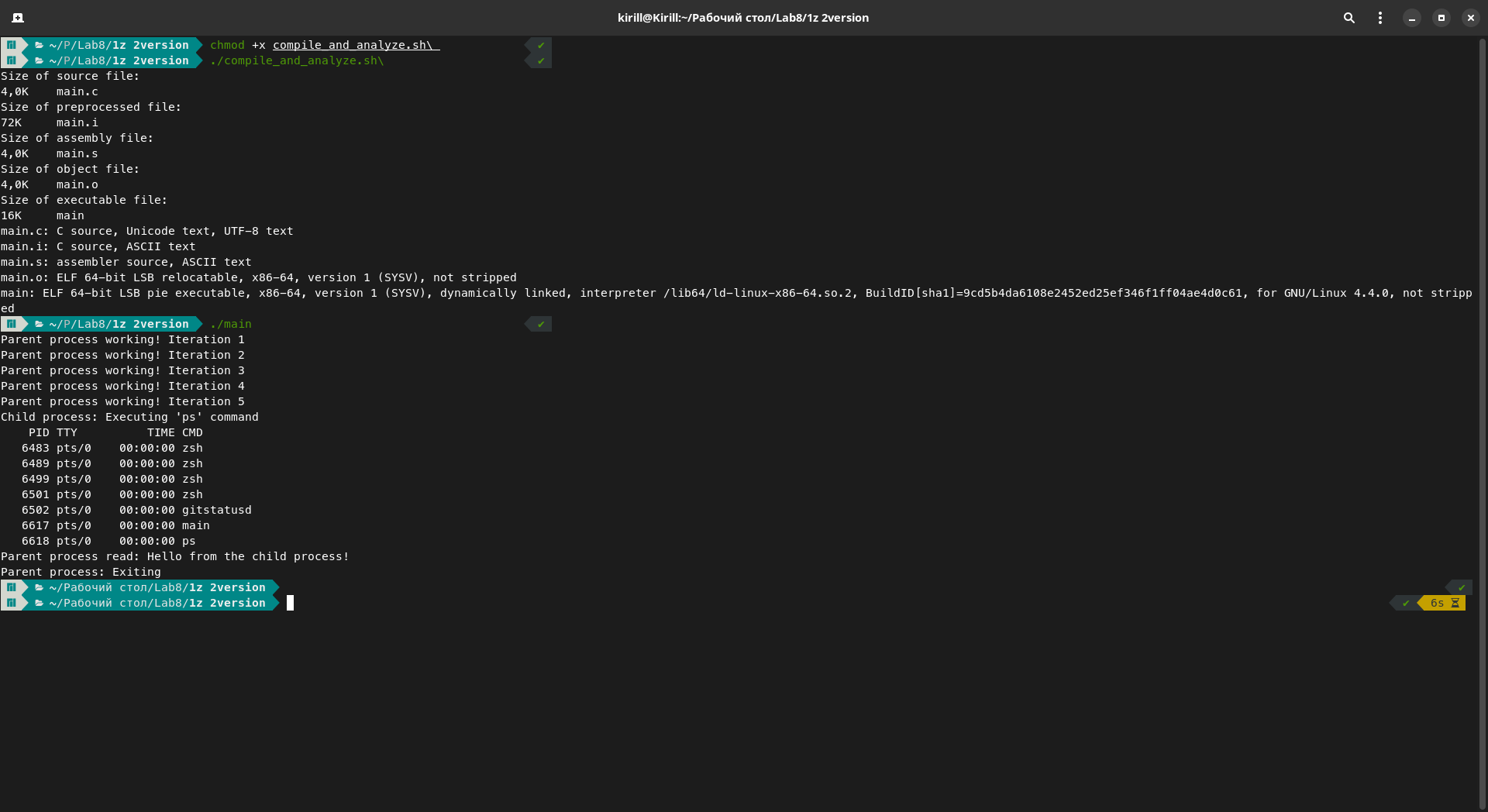
file main.c

file main.i

file main.s

file main.o

file main



После запуска скрипта мы получили информацию о размерах файлов и их форматах.

После запуска программы main.c мы получили

**Вывод родительского процесса**:

Parent process working! Iteration 1

Parent process working! Iteration 2

Parent process working! Iteration 3

Parent process working! Iteration 4

Parent process working! Iteration 5

**Вывод дочернего процесса(через 5 секунд после старта дочернего процесса**):

Child process: Executing 'ps' command

### Задание 2

Используя материалы (исходные файлы) упражнения №2 проведите компи-

ляцию проекта листингов 22.5 и 22.6 с помощью утилиты make с прохожде-

нием всех стадий компиляции обеих программ.

Сборка проекта должна содержать файлы с результатами препроцессинга.

Исследуйте фалы препроцессора, найдите в них код своей программы.

Определите размеры исходных, препроцессорных, ассемблерных, объектных

и исполняемых файлов. С помощью соответствующей консольной команды

или с помощь проводника определите форматы этих файлов. Результаты

подтвердите скриншотами. Сделайте краткий вывод по выполненной работе.

Программа листинга 22.8 должна запускать браузер, установленный в вашей

системе, или текстовый редактор, или проводник, или другую, написанную

вами программу, а также можно чтобы ваша программа запускала и то и дру-

гое и третье.

Проявите возможности многозадачности «во всей красе»!

**Программа 22.5** (файл uname\_program.c):

#include <stdio.h>

#include <unistd.h>

extern char \*\*environ;

int main(void) {

char \*uname\_args[] = {

"uname", "-a", NULL

};

execve("/bin/uname", uname\_args, environ);

fprintf(stderr, "Error\n");

return 0;

}

**Программа 22.6** (файл newprog\_program.c):

#include <stdio.h>

#include <unistd.h>

int main(void) {

char \*newprog\_args[] = { "Tee-hee!", "newprog\_arg1", "newprog\_arg2", NULL };

char \*newprog\_envp[] = { "USER=abrakadabra", "HOME=/home/abrakadabra", NULL };

printf("Old PID: %d\n", getpid());

execve("./newprog", newprog\_args, newprog\_envp);

fprintf(stderr, "Error\n");

return 0;

}

**Программа 22.8** (файл launch\_program.c):

#include <stdio.h>

#include <unistd.h>

int main(void) {

// Запуск браузера или текстового редактора

char \*browser\_args[] = { "firefox", NULL }; // Замените на ваш браузер

char \*editor\_args[] = { "gedit", NULL }; // Замените на ваш текстовый редактор

pid\_t browser\_pid = fork();

if (browser\_pid == 0) {

execve("/usr/bin/firefox", browser\_args, NULL);

fprintf(stderr, "Error launching browser\n");

return 1;

}

pid\_t editor\_pid = fork();

if (editor\_pid == 0) {

execve("/usr/bin/gedit", editor\_args, NULL);

fprintf(stderr, "Error launching editor\n");

return 1;

}

printf("Launched browser with PID=%d\n", browser\_pid);

printf("Launched editor with PID=%d\n", editor\_pid);

return 0;

}

Makefile

CC = gcc

CFLAGS = -Wall -Wextra

all: uname\_program newprog\_program launch\_program

uname\_program: uname\_program.c

$(CC) $(CFLAGS) -E uname\_program.c -o uname\_program.i

$(CC) $(CFLAGS) -S uname\_program.c -o uname\_program.s

$(CC) $(CFLAGS) -c uname\_program.c -o uname\_program.o

$(CC) uname\_program.o -o uname\_program

newprog\_program: newprog\_program.c

$(CC) $(CFLAGS) -E newprog\_program.c -o newprog\_program.i

$(CC) $(CFLAGS) -S newprog\_program.c -o newprog\_program.s

$(CC) $(CFLAGS) -c newprog\_program.c -o newprog\_program.o

$(CC) newprog\_program.o -o newprog\_program

launch\_program: launch\_program.c

$(CC) $(CFLAGS) -E launch\_program.c -o launch\_program.i

$(CC) $(CFLAGS) -S launch\_program.c -o launch\_program.s

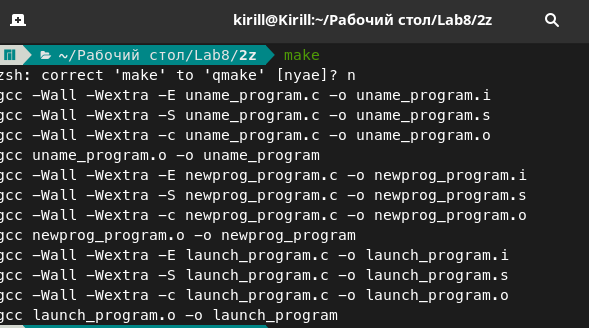
$(CC) $(CFLAGS) -c launch\_program.c -o launch\_program.o

$(CC) launch\_program.o -o launch\_program

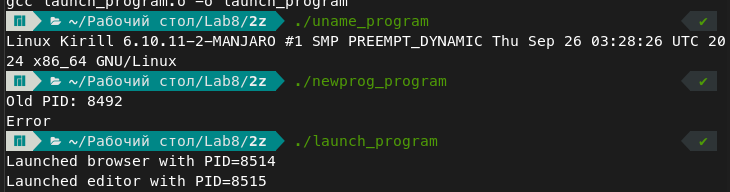
clean:

rm -f \*.o \*.i \*.s uname\_program newprog\_program launch\_program

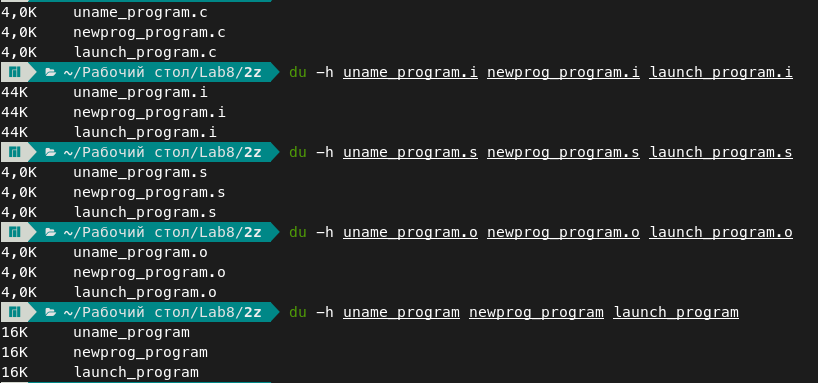
**Сборка проекта**:



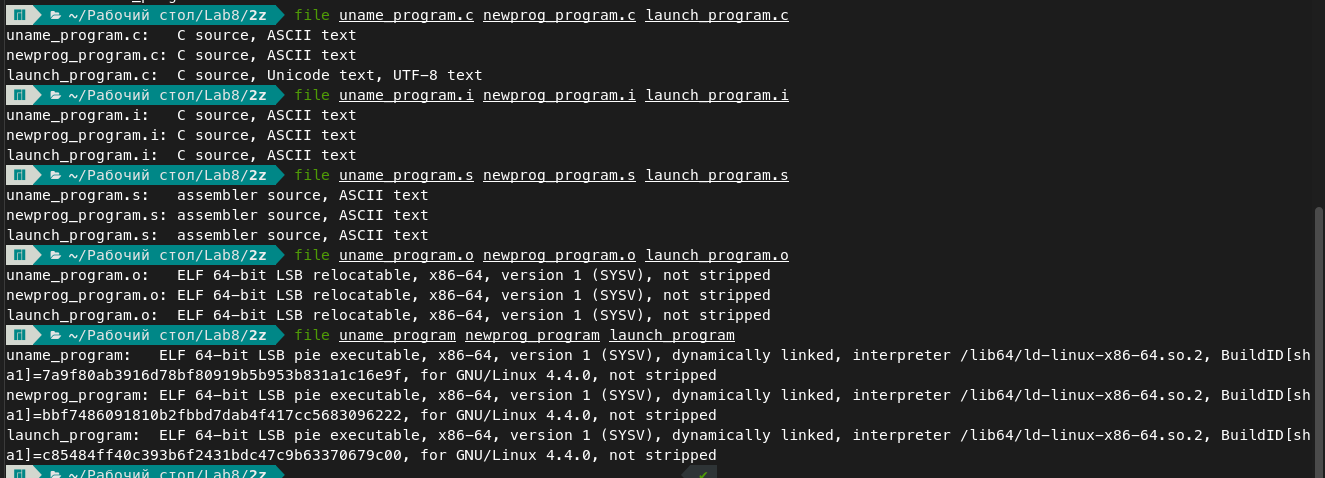
**Запуск программ**:



### Проверка размеров файлов



### Определение форматов файлов



Задание 3

Используя материалы, исходные файлы упражнения №3 (листинги 22.9 – 22.14)

создайте программу запускающую «микросервисы»: программу-календарь,

браузер, проводник, текстовый редактор, установленные в вашей системе.

С помощью очень полезной утилиты strace, запустите переданную ей вами раз-

работанную программу и выведите в стандартный поток ошибок отчет об ис-

пользованных системных вызовах. Полученный результат представьте в отчете

и дайте объяснения.

## Листинг кода

launch\_services.c

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

extern char \*\*environ; // Объявление переменной environ

int main(void) {

pid\_t result;

// Запуск календаря с execlp

result = fork();

if (result == 0) {

execlp("gnome-calendar", "gnome-calendar", NULL);

perror("Error launching gnome-calendar");

exit(1);

}

// Запуск браузера с execvp

result = fork();

if (result == 0) {

char \*browser\_args[] = { "firefox", NULL };

execvp("firefox", browser\_args);

perror("Error launching firefox");

exit(1);

}

// Запуск файлового менеджера с execl

result = fork();

if (result == 0) {

execl("/usr/bin/nautilus", "nautilus", NULL);

perror("Error launching nautilus");

exit(1);

}

// Запуск текстового редактора с execle

result = fork();

if (result == 0) {

execle("/usr/bin/gedit", "gedit", NULL, environ);

perror("Error launching gedit");

exit(1);

}

// Ожидание завершения всех процессов

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

wait(NULL);

}

return 0;

}

makefile

CC = gcc

CFLAGS = -Wall -Wextra

all: launch\_services

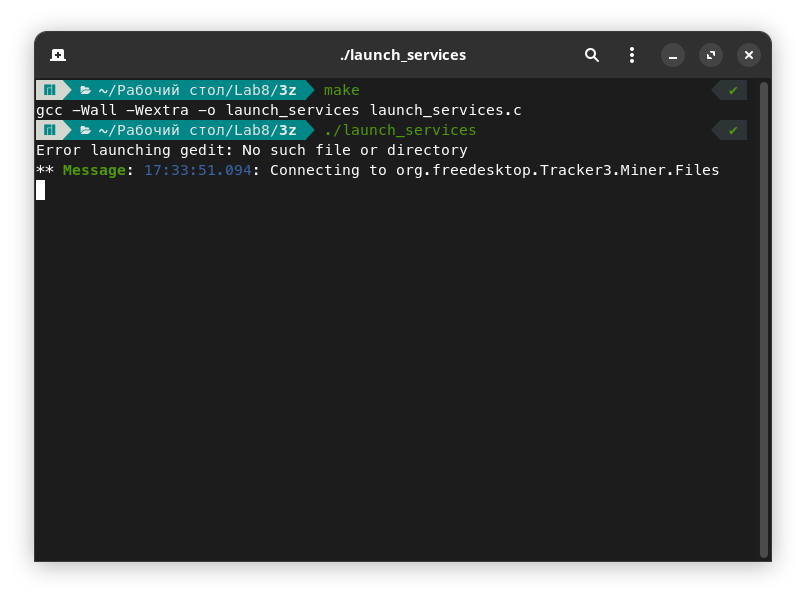
launch\_services: launch\_services.c

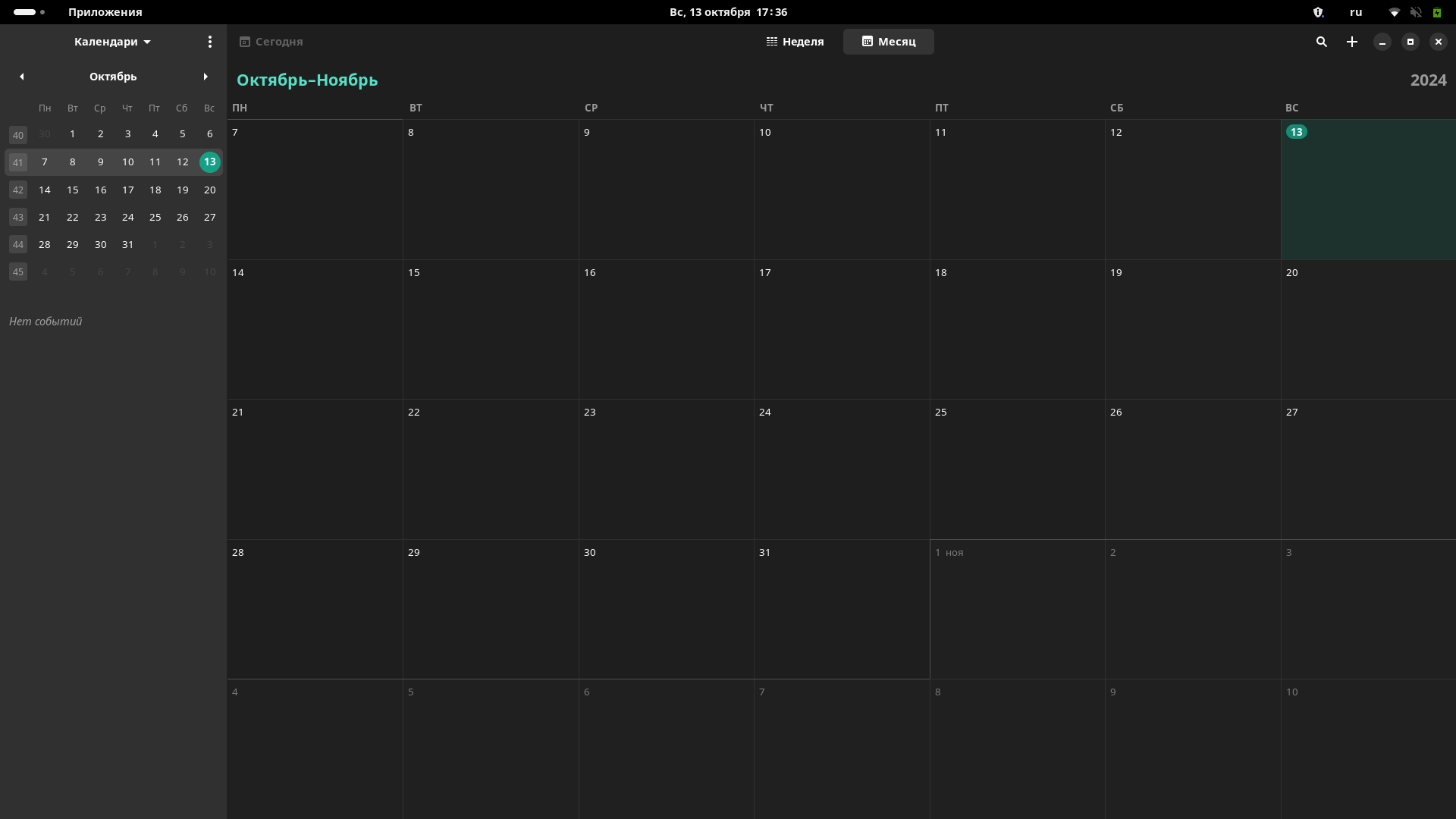
$(CC) $(CFLAGS) -o launch\_services launch\_services.c

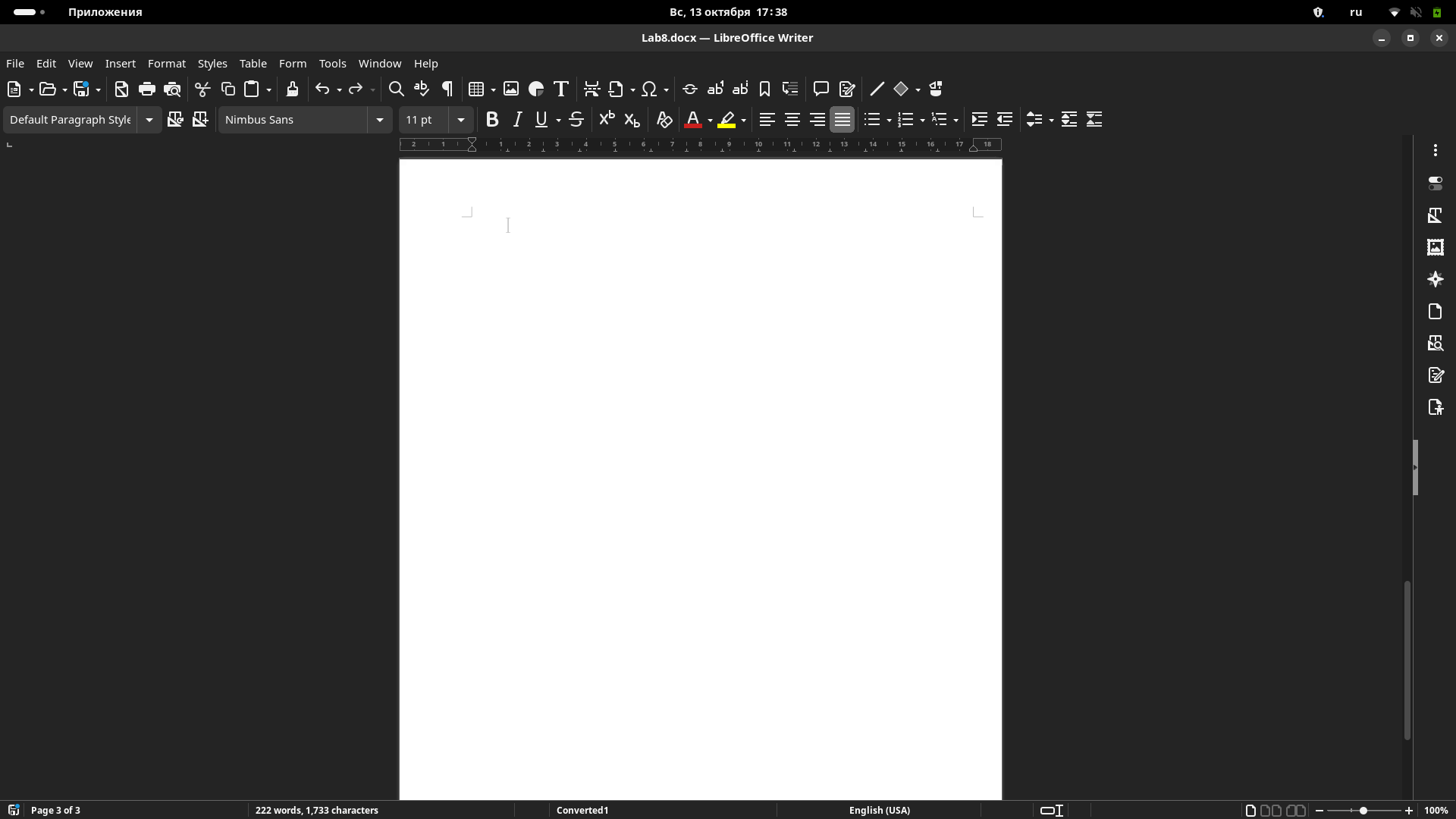
clean:

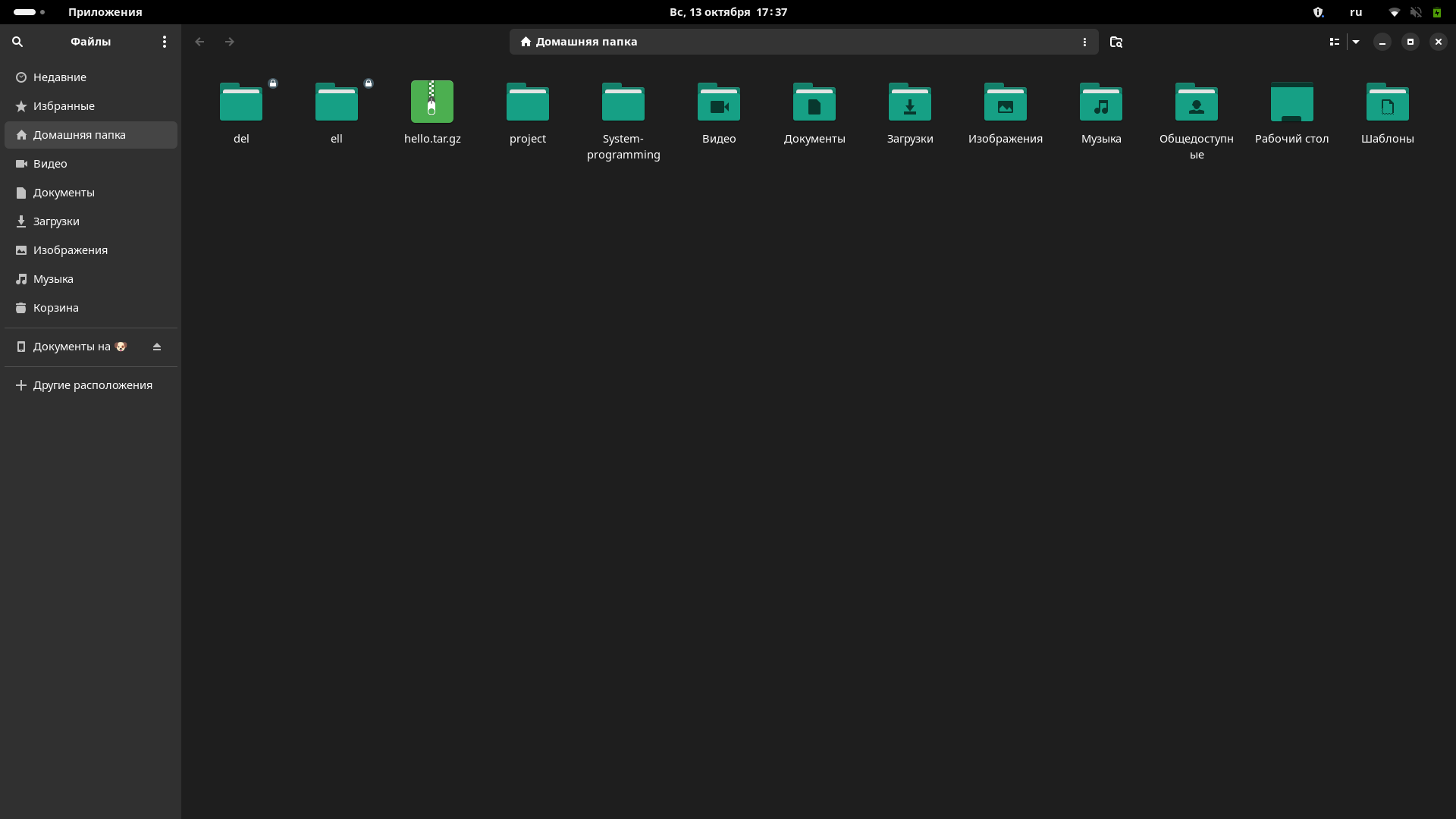
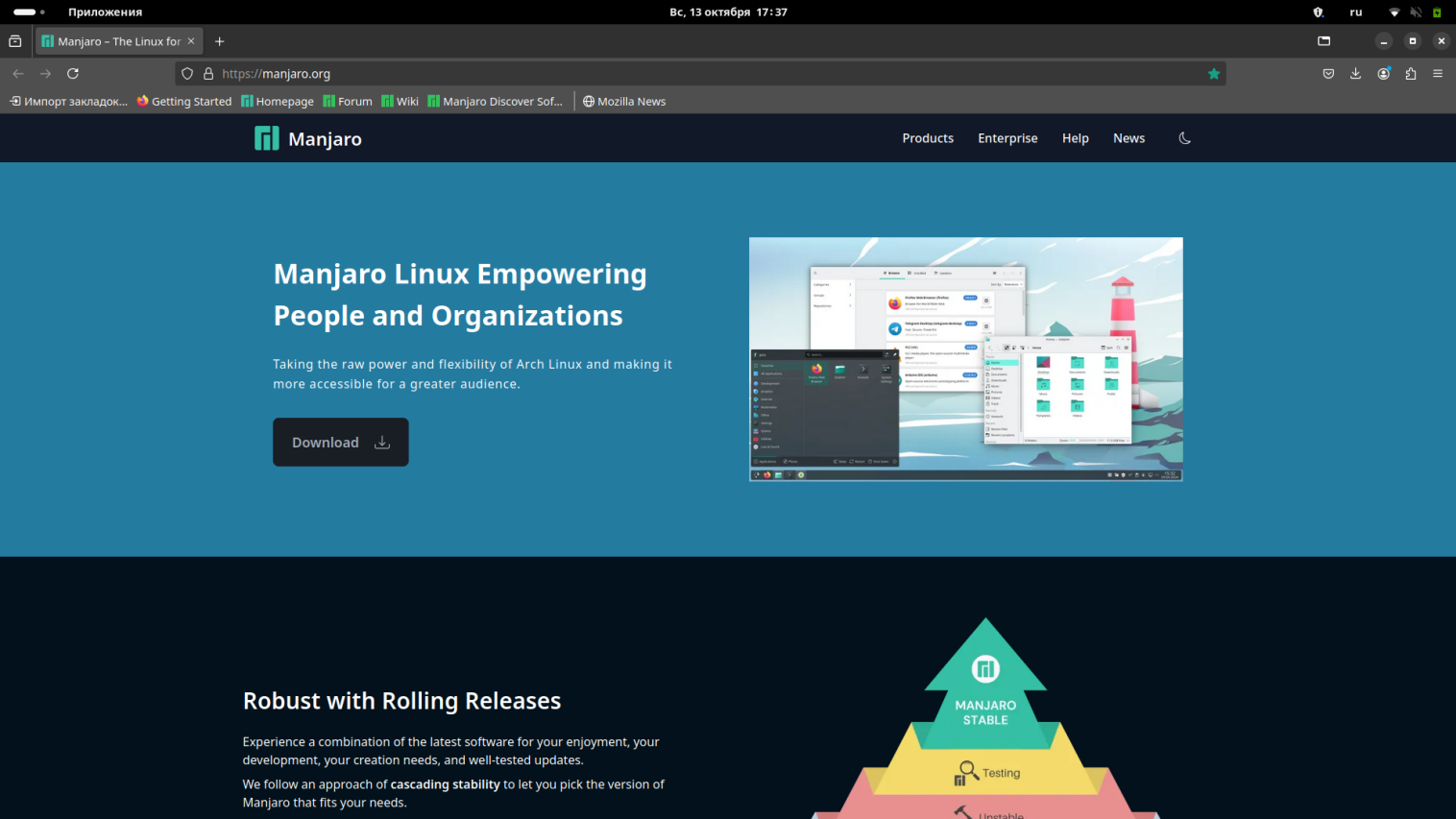
rm -f launch\_services

# Скриншоты









**Ответы на вопросы**

**Зухта**

Приведите известные вам примеры компилируемых и интерпритируемых языков программирования?

### Компилируемые языки

1. C - язык общего назначения, часто используемый для системного программирования.

2. C++ - расширение языка C с поддержкой объектно-ориентированного программирования.

3. Rust - современный язык, ориентированный на безопасность и производительность.

4. Go - язык, разработанный Google, известный своей простотой и эффективностью.

5. Swift - язык, используемый для разработки приложений под iOS и macOS.

### Интерпретируемые языки

1. Python - популярный язык для веб-разработки, научных вычислений и автоматизации.

2. JavaScript - язык, используемый в основном для создания интерактивных веб-страниц.

3. Ruby - язык, известный своей элегантностью и используемый в веб-разработке.

4. PHP - язык, часто применяемый для серверной части веб-приложений.

5. Perl - язык, известный своими возможностями обработки текста и скриптами.

Некоторые языки, такие как Java и C#, используют комбинацию компиляции и интерпретации, так как сначала компилируются в байт-код, который затем выполняется виртуальной машиной.

**Медведский**

В чем разница использования двух форм директивы: **#include <filename>** или **#include “filename”**?

В директивах #include есть важные нюансы:

#include <filename>:

Используется для системных заголовочных файлов. Препроцессор ищет файлы в системных каталогах, определённых компилятором.

#include "filename":

Используется для пользовательских заголовочных файлов. Сначала препроцессор ищет файл в директории текущего проекта, а затем в системных каталогах.

Короче говоря, угловые скобки < > больше подходят для стандартных библиотек, а кавычки " "—для файлов, которые создал ты или другие разработчики на проекте. Это может помочь избежать конфликтов и путаницы при компиляции! Как тебе это объяснение?