1. Написал программный код на rust:

```
use anyhow::{Context, Result};
use clap::Parser;
use rayon::prelude::*;
use regex::Regex;
use std::collections::HashMap;
use std::io::{Read, Write};
use std::path::Path;
use std::process::Command;
use std::{fs, io};
use walkdir::WalkDir;
use lazy_static::lazy_static;

#[derive(Parser, Debug)]
#[command(author, version, about = "Reverse LDD wrapper (uses
objdump/readelf)")]
struct Args {
    /// Directory to scan (optional)
    #[arg(short, long, default_value = "./")]
    directory: String,

    /// Output file path (optional)
    #[arg(short, long)]
    output: Option<String>,
}

#[derive(Debug)]
struct ElfInfo {
    path: String,
    arch: String,
    libs: Vec<String>,
}

lazy_static! {
    static ref ARCH_REGEX: Regex = Regex::new(r"architecture:\s+
([^\s,]+)").unwrap();
    static ref NEEDED_REGEX: Regex = Regex::new(r"NEEDED\s+(\S+)").unwrap();
    static ref ELF_MAGIC: [u8; 4] = [0x7F, 0x45, 0x4C, 0x46]; // ELF magic
bytes
    static ref FILE_EXTENSIONS: [&'static str; 4] = [".so", "bin", ".elf",
""];
}
```

```rust
fn main() -> Result<()> {
    rayon::ThreadPoolBuilder::new()
        .num_threads(num_cpus::get())
        .build_global()?;

    let args = Args::parse();
    let scan_path = Path::new(&args.directory);

    if !scan_path.is_dir() {
        eprintln!("Error: Directory '{}' does not exist.", args.directory);
        return Ok(());
    }

    println!("[INFO] Scanning directory: {}", args.directory);

    // 1. Сканирование
    let results: Vec<ElfInfo> = WalkDir::new(scan_path)
        .into_iter()
        .par_bridge()
        .filter_map(|e| e.ok())
        .filter(|e| is_potential_elf_file(e.path()))
        .filter_map(|entry| {
            analyze_file(entry.path())
        })
        .collect();

    println!("[INFO] Scan complete. Found {} valid ELF executables.",
results.len());

    generate_report(&results, &args.output)?;

    Ok(())
}

fn is_potential_elf_file(path: &Path) -> bool {
    path.is_file() &&
        path.file_name()
            .and_then(|name| name.to_str())
            .map_or(false, |name|
                FILE_EXTENSIONS.iter().any(|ext| name.ends_with(ext))
            )
        && check_elf_magic_number(path).is_ok()
}

fn check_elf_magic_number(path: &Path) -> Result<(), String> {
```

```
    match fs::File::open(path) {
        Ok(mut file) => {
            let mut magic_number_buffer = [0; 4];
            let bytes_read = file.read(&mut magic_number_buffer).unwrap();

            if bytes_read != ELF_MAGIC.len() {
                return Err("Could not read exact 4 bytes for magic
number".to_owned())
            }

            match ELF_MAGIC.eq(&magic_number_buffer) {
                true => Ok(()),
                false => Err("That's not elf file".to_owned())
            }
        }
        Err(e) => Err("Could not read ELF file".to_owned()),
    }
}

fn analyze_file(path: &Path) -> Option<ElfInfo> {
    let path_str = path.to_string_lossy();

    let objdump_arch_result = Command::new("objdump")
        .args(["-f", "-p"])
        .arg(&*path_str)
        .output()
        .ok()?;

    let objdump_output = String::from_utf8_lossy(&objdump_arch_result.stdout);
    let arch =
ARCH_REGEX.captures(&objdump_output)?.get(1)?.as_str().trim().to_string();

    let mut libs = Vec::new();

    for capture in NEEDED_REGEX.captures_iter(&objdump_output) {
        if let Some(lib) = capture.get(1) {
            libs.push(lib.as_str().to_string())
        }
    }

    if libs.is_empty() {
        return None;
    }

    let elf_info = ElfInfo { arch, libs, path: path_str.to_string() };
    // println!("[INFO] Scan elf. Elf Info: {:?}.", elf_info);
```

```rust
    Some(elf_info)
}

fn generate_report(scan_results: &Vec<ElfInfo>, output_path: &Option<String>)
-> Result<()> {
    let mut report_data: HashMap<String, HashMap<String, Vec<String>>> =
HashMap::new();

    for result in scan_results {
        for lib in result.libs.iter() {
            report_data
                .entry(result.arch.clone())
                .or_default()
                .entry(lib.clone())
                .or_default()
                .push(result.path.clone());
        }
    }

    let mut buffer = Vec::new();

    writeln!(&mut buffer, "Report on dynamic used libraries")?;
    writeln!(&mut buffer, "Generated by bldd-rust-wrapper")?;
    writeln!(&mut buffer, "========================================")?;

    let mut sorted_archs: Vec<_> = report_data.keys().collect();
    sorted_archs.sort();

    for arch in sorted_archs {
        writeln!(&mut buffer, "---------- {} ----------", arch)?;

        let libs = report_data.get(arch).unwrap();
        let mut sorted_by_usage_libs: Vec<_> = libs.iter().collect();
        sorted_by_usage_libs.sort_by(|a, b| b.1.len().cmp(&a.1.len()));

        for (lib, paths) in sorted_by_usage_libs.iter() {
            let joined_paths = paths.join("\n\t\t-> ");
            writeln!(&mut buffer, "{} ({} execs) \n\t\t->{}", lib.to_string(),
paths.len(), joined_paths)?;
        }
        writeln!(&mut buffer)?;
    }

    if let Some(output_path) = output_path {
        fs::write(output_path, buffer)
            .with_context(|| format!("Failed to write report to {}",
```

```
output_path))?;
        println!("[INFO] Report saved to {}", output_path);
    }
    else{
        io::stdout().write_all(&buffer)?;
    }

    Ok(())
}
```

2. Проделал кросс-компиляцию:

```
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1$ cd ./for_tests/
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1/for_tests$ ls
lab1.test_arm  lab1.test_arm64  lab1.test.c  lab1.test_x86  lab1.test_x86_64  script.sh  test.sh
```

3. Сам скрипт:

```
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1/for_tests$ cat script.sh
#!/bin/bash

# x86_64 (64-bit Intel/AMD)
x86_64-linux-gnu-gcc -o lab1.test_x86_64 lab1.test.c -ldl

# ARM64 (AArch64)
aarch64-linux-gnu-gcc -o lab1.test_arm64 lab1.test.c -ldl

# ARM (32-bit)
arm-linux-gnueabihf-gcc -o lab1.test_arm lab1.test.c -ldl

# x86 (32-bit)
gcc -m32 -o lab1.test_x86 lab1.test.c -ldl
```

4. Тестовый файл на C:

```
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1/for_tests$ cat lab1.test.c
#include <stdio.h>
#include <dlfcn.h>
#include <gnu/lib-names.h>

// Функция для демонстрации использования библиотек
void print_system_info() {
    #ifdef __x86_64__
        printf("Architecture: x86_64\n");
    #elif defined(__aarch64__)
        printf("Architecture: ARM64\n");
    #elif defined(__arm__)
        printf("Architecture: ARM\n");
    #elif defined(__i386__)
        printf("Architecture: x86\n");
    #else
        printf("Architecture: Unknown\n");
    #endif

    // Демонстрация использования системных библиотек
    void *handle;

    // Загрузка библиотек для разных архитектур
    handle = dlopen(LIBC_SO, RTLD_LAZY);
    if (handle) {
        printf("Loaded standard C library\n");
        dlclose(handle);
    }

    handle = dlopen("libm.so.6", RTLD_LAZY);
    if (handle) {
        printf("Loaded math library\n");
        dlclose(handle);
    }
}

int main() {
    print_system_info();
    return 0;
}
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1/for_tests$
```

5. Вызов --help:

```
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1$ ./lab1 --help
Reverse LDD wrapper (uses objdump/readelf)

Usage: lab1 [OPTIONS]

Options:
  -d, --directory <DIRECTORY>  Directory to scan (optional) [default: ./]
  -o, --output <OUTPUT>        Output file path (optional)
  -h, --help                   Print help
  -V, --version                Print version
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1$
```

6. Работа приложения:

```
  -V, --version               Print version
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1$ ./lab1 -d ./for_tests/ -o ./archs_tests.txt
[INFO] Scanning directory: ./for_tests/
[INFO] Scan complete. Found 4 valid ELF executables.
[INFO] Report saved to ./archs_tests.txt
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1$
```

```
ildar-islamov@ildar-islamov-VMware-Virtual-Platform:~/labs/lab1$ cat archs_tests.txt
Report on dynamic used libraries
Generated by bldd-rust-wrapper
========================================
---------- aarch64 ----------
libc.so.6 (1 execs)
                ->./for_tests/lab1.test_arm64

---------- armv7 ----------
libc.so.6 (1 execs)
                ->./for_tests/lab1.test_arm

---------- i386 ----------
libc.so.6 (1 execs)
                ->./for_tests/lab1.test_x86

---------- i386:x86-64 ----------
libc.so.6 (1 execs)
                ->./for_tests/lab1.test_x86_64
```