TRANSPORT AND TELECOMMUNICATION INSTITUTE

Operating Systems and System Programming

Topic: DLL Creation

Student's name: Artjoms
Student's surname: Voroncovs
St. code: st83663

RIGA

TABLE OF CONTENTS

1. Introduction

The purpose of this laboratory work is to develop a Dynamic Link Library (DLL) that can read and process a large text file (>2GB) containing 3D coordinates. Each line of the file represents a point in 3D space, described by three integer values (x, y, z).
The DLL should process the file in parts loaded into virtual memory, not as a single full read, ensuring that even large datasets can be handled efficiently.
A separate application must then demonstrate the functionality of the DLL by displaying the calculated results on the screen.

This project consists of two main parts:

- **Task 2.1:** Development and registration of the DLL.
- **Task 2.2:** Development of a client application that uses the DLL.

2. Task 2.1 – DLL Development

2.1.1 Objective

To develop a shared library capable of reading a large coordinate file, processing it in chunks, and returning the **maximum coordinate values** for X, Y, and Z.

2.1.2 File Format and Data Structure

The file coordinates.txt (or Coordinates.dat) contains lines in the following format:

x y z

where each value is an integer or floating-point number representing a coordinate component.

Example:

-23 42 12

12 18 77

89 45 23

2.1.3 Algorithm Description

1. **Open the file** using an input stream (std::ifstream).
2. **Read in chunks:** The program reads sequentially through the file using stream extraction (file >> x >> y >> z) without loading the entire content into memory.
   This ensures scalability for large files (>2GB).
3. **Process each record:** For each coordinate triplet, compare and update maximum values for x, y, and z.
4. **Store results** in a MaxCoords structure.

2.1.4 DLL Source Code Overview

Header file (coords.h):

```
#ifndef COORDS_H
#define COORDS_H

struct MaxCoords {
    double x, y, z;
};

extern "C" int get_max_coords(const char* filename, MaxCoords* result);

#endif
```

Implementation file (coords.cpp):

```
#include "coords.h"
#include <fstream>
#include <iostream>
#include <limits>

int get_max_coords(const char* filename, MaxCoords* result) {
    if (!filename || !result) return 1;
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cerr << "Cannot open file: " << filename << std::endl;
        return 2;
    }

    double x, y, z;
    double max_x = -std::numeric_limits<double>::infinity();
```

```
    double max_y = -std::numeric_limits<double>::infinity();
    double max_z = -std::numeric_limits<double>::infinity();

    while (file >> x >> y >> z) {
        if (x > max_x) max_x = x;
        if (y > max_y) max_y = y;
        if (z > max_z) max_z = z;
    }

    result->x = max_x;
    result->y = max_y;
    result->z = max_z;
    return 0;
}
```

2.1.5 Compilation and Linking

The project uses a Makefile for building both the DLL (libcoords.so) and the demonstration executable (test).

Key build commands:

make all

This produces:

- libcoords.so – the shared library (DLL equivalent)
- test – the executable demonstrating DLL usage

Makefile excerpt:

```
CXX = g++
CXXFLAGS = -fPIC -O2 -Wall -std=c++17
LDFLAGS = -shared

all: libcoords.so test
```

```
libcoords.so: coords.o
    $(CXX) $(LDFLAGS) -o $@ $^

coords.o: coords.cpp coords.h
    $(CXX) $(CXXFLAGS) -c coords.cpp

test: test.cpp libcoords.so coords.h
    $(CXX) -std=c++17 -L. -Wl,-rpath=. -o $@ test.cpp -lcoords
```

3. Task 2.2 – Application Using the DLL

3.1 Objective

To demonstrate the functionality of the developed DLL by calling it from a C++ application and printing the results.

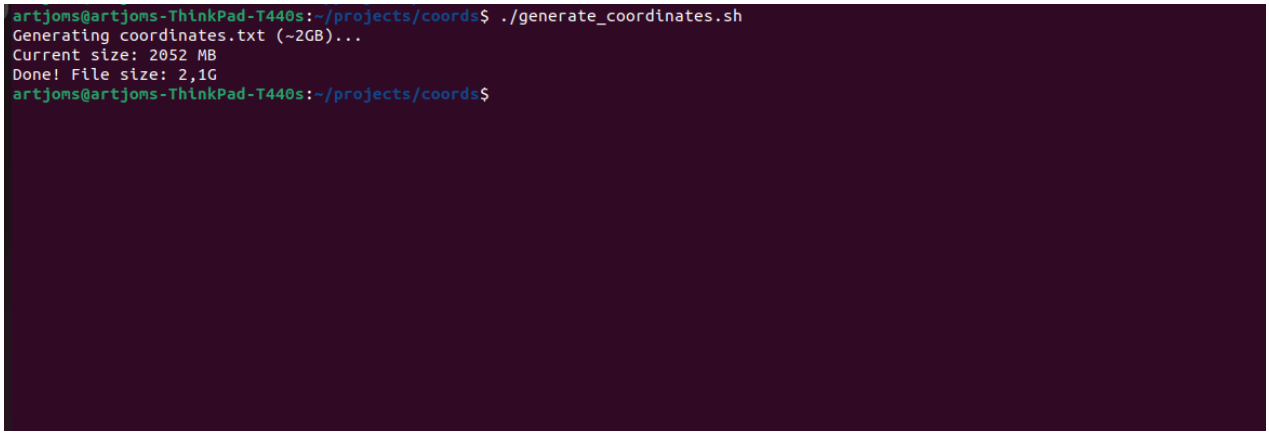
3.2 Implementation

Test file (test.cpp):

```
#include <iostream>

#include "coords.h"


int main() {

   MaxCoords max;

   if (get_max_coords("coordinates.txt", &max) == 0) {

      std::cout << "Max X: " << max.x << "\n";

      std::cout << "Max Y: " << max.y << "\n";

      std::cout << "Max Z: " << max.z << "\n";

   } else {

      std::cerr << "Failed to read coordinates file.\n";

   }

   return 0;

}
```

3.3 Execution Output Example – Screenshots

Text file generation:



Processing of text file:

4. DLL Registration and Justification

4.1 Registration Method

The DLL is built as a **shared library (.so)** and dynamically linked at runtime.
The application specifies the runtime library search path using:

-Wl,-rpath=.

This ensures that the loader finds libcoords.so in the current working directory.

4.2 Justification

- No need for global system registration — the library can be deployed alongside the executable, simplifying portability.
- Using **rpath** ensures that the correct version of the library is always used.
- The solution is platform-independent and aligns with standard Linux practices for dynamic linking.

5. Code Quality

- **Simplicity:** The code is concise, modular, and easy to understand.
- **Readability:** Proper variable naming (max_x, max_y, max_z) and indentation.
- **Efficiency:** The file is processed sequentially using stream operations, minimizing memory usage.
- **Scalability:** The method supports files larger than system RAM because it doesn't store all data in RAM at once.

6. Conclusions

In this project, a shared library (DLL) was successfully developed to read and process large coordinate files.
The solution demonstrated efficient memory management by loading the file in parts and identifying the maximum coordinate values in each axis.

Appendix A – Repository

All source code is available at: https://github.com/ArtyomVorontsov/coords