UNDERSTANDING STATIC AND DYNAMIC LIBRARIES IN C

Divyanshu Mewal
23ME30017

Static Libraries

Definition

Static libraries are collections of object files that are linked directly into the executable at compile time, making the program self-sufficient.

Advantages

They provide a standalone executable, ensuring that the program runs independently without any external dependencies.

Example

Creating a static library using the `ar` command and linking it with a C program.

Dynamic Libraries

Definition

Dynamic libraries are separate files outside of the executable, loaded at runtime, allowing for shared use among multiple programs.

Advantages

They reduce the overall size of the executable and allow for shared code among multiple processes.

Example

Creating a dynamic library using the `gcc` command and linking it with a C program.

Advantages

Static Libraries

- Independence from external files, faster execution, and easier debugging.
- Static linking simplifies the build process and reduces the complexity of managing library dependencies

Dynamic Libraries

- Reduced memory usage, easier updates, and better disk space utilization.
- Dynamic libraries are loaded on-demand, conserving system memory and improving overall efficiency.

Differences

- Static libraries are linked directly into the executable at compile time, resulting in larger executable files, while dynamic libraries are linked at runtime, allowing for shared code among multiple processes and easier updates.

- Static libraries provide simplicity, performance, and code protection, while dynamic libraries offer code sharing, versioning flexibility, and dynamic loading capabilities.

Linking Static Libraries

1 Process

Using the `gcc` command to link a static library with a C program, ensuring the correct path and filename of the library.

2 Compilation

Static libraries are linked during the compilation process, where the object code is combined with the executable.

3 Example

To compile a C program (program.c) with a static library (libexample.a):

```
gcc program.c -o program -lexample
```

Linking Dynamic Libraries

1 Process

Utilizing the `-l` flag with the `gcc` command to link a dynamic library, along with specifying the library path if necessary.

2 Runtime Linking

Dynamic libraries are loaded at runtime, when the executable is launched and needs to access the library functions.

3 Example

Dynamic library named libexample.so in a custom directory /path/to/library" :

```
gcc program.c -o program -L/path/to/library -lexample
```

Working of ld in Linux

Introduction

- It is a command-line utility used to combine object files generated by the compiler into an executable program or a shared library.
- ld is responsible for resolving symbols defined in one object file and referenced in another, ensuring that all dependencies are correctly linked.

Usage

- ld is typically invoked indirectly by the compiler driver (e.g., gcc) during the compilation and linking process.
- It can also be invoked directly from the command line to perform custom linking operations or manipulate object files and libraries.

Common command flags

1. -o <output>: Specifies the name of the output file produced by the linker.
2. -L <directory>: Adds a directory to the library search path.
3. -l<library>: Links the specified library to the program.

THE END

By:Divyanshu Mewal
23ME30017