BHARATIYA VIDYA BHAVAN'S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

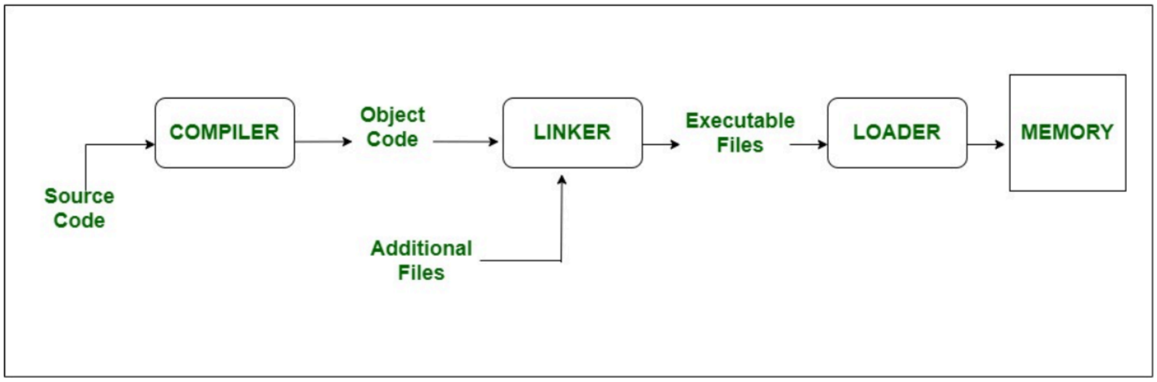
(Empowered Autonomous Institute Affiliated to University of Mumbai)

[Knowledge is Nectar]

Department of Computer Engineering

Course - System Programming and Compiler Construction (SPCC)

UID	2021300101
Name	Adwait Purao
Class and Batch	TE Computer Engineering - Batch B
Date	01/05/24
Lab #	10
Aim	Design linker/loader
Objective	Demonstrate a user-driven linker/loader simulation to resolve symbols and load an executable based on input object file data.
Theory	**Linkers and Loaders** **Introduction** Linkers and loaders are essential components of a computer system that facilitate the execution of programs. They play a crucial role in the process of transforming separate object files into an executable program and loading it into memory for execution. [1, p. 641] **Linkers** A linker is a program that combines multiple object files generated by the compiler or assembler, along with necessary library routines, into a single executable file. [2, p. 233] The main tasks performed by a linker are: 1. Symbol Resolution: The linker resolves external references by associating each symbol reference with its corresponding definition in one of the object files or libraries. [1, p. 642] 2. Relocation: The linker assigns final memory addresses to the instructions and data in the object files, adjusting any references accordingly. [1, p. 643] 3. Library Integration: The linker includes only the required library routines in the final executable file, ensuring efficient memory usage. [2, p. 234] The output of the linker is an executable file that contains the combined and relocated code and data from the input object files and libraries. [1, p. 645]



Loaders

A loader is a program responsible for loading the executable file into memory and preparing it for execution. [3, p. 147] The main tasks performed by a loader are:

1. Memory Allocation: The loader allocates memory segments for the code, data, and stack sections of the program. [1, p. 646]
2. Relocation: For relocatable executables, the loader adjusts the memory addresses of instructions and data according to the assigned memory locations. [2, p. 236]
3. Symbol Resolution: In some cases, the loader resolves external references that could not be resolved by the linker, such as dynamic linking of shared libraries. [3, p. 149]
4. Execution Preparation: The loader sets up the necessary data structures and registers for program execution, transferring control to the program's entry point. [2, p. 237]

Loaders can be classified into different types based on when and how they load the program into memory:

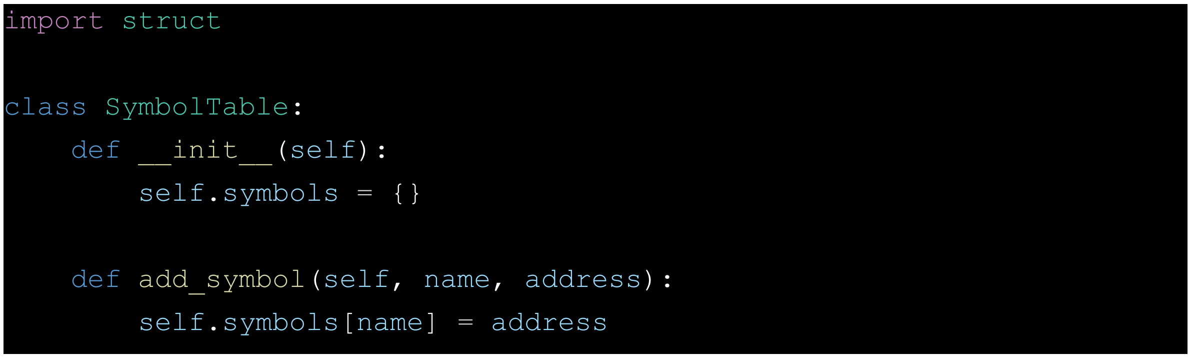
1. Bootstrap Loader: This loader is responsible for loading the operating system kernel into memory during system startup. [3, p. 150]
2. Relocating Loader: This loader handles the relocation of instructions and data references in the executable file. [1, p. 647]
3. Dynamic Loader: This loader loads shared libraries or dynamically linked code into memory as needed during program execution. [2, p. 238]

Differences between Linker and Loader are as follows: [4]

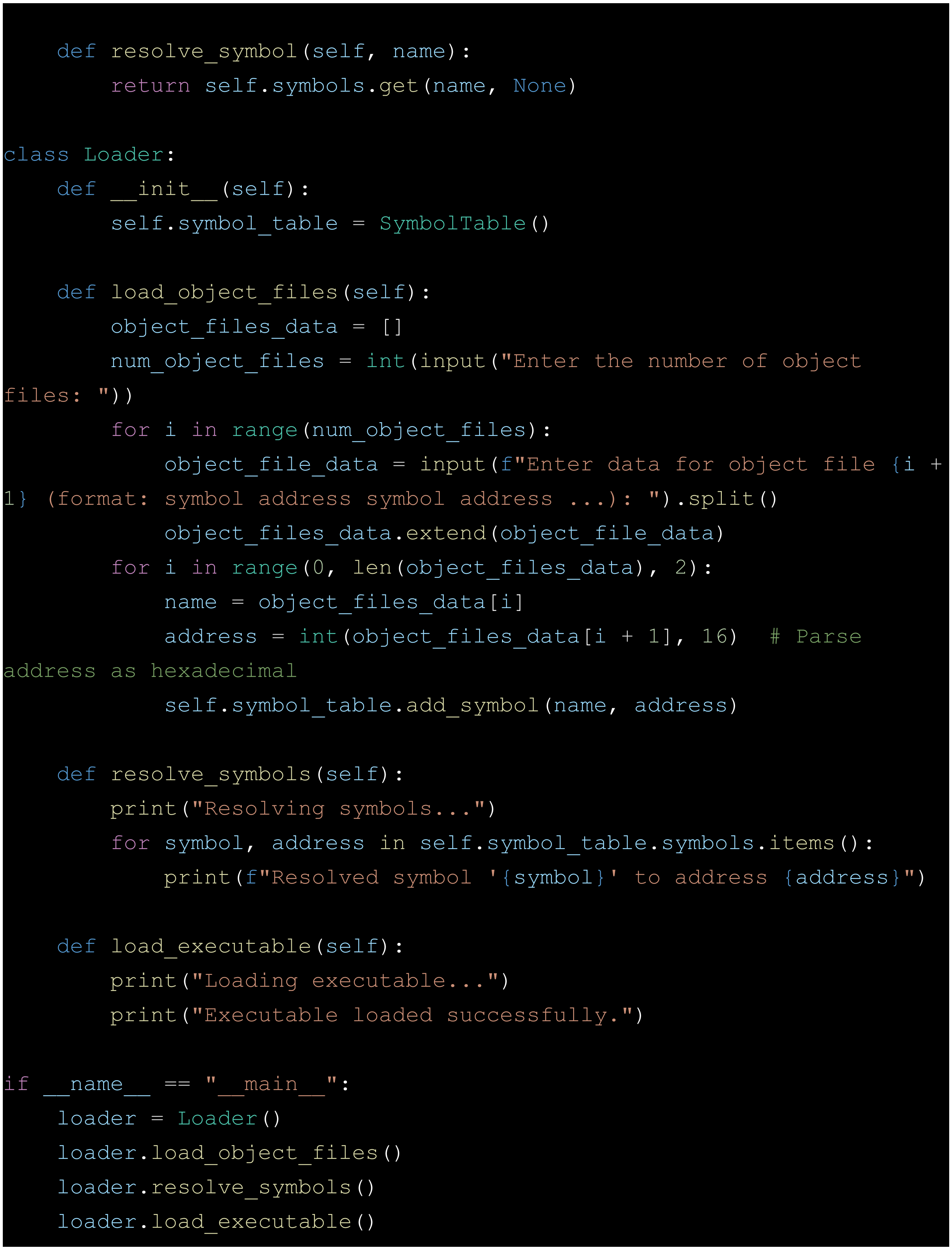
LINKER	LOADER

The main function of Linker is to generate executable files.	Whereas main objective of Loader is to load executable files to main memory.
The linker takes input of object code generated by compiler/assembler.	And the loader takes input of executable files generated by linker.
Linking can be defined as process of combining various pieces of codes and source code to obtain executable code.	Loading can be defined as process of loading executable codes to main memory for further execution.
Linkers are of 2 types: Linkage Editor and Dynamic Linker.	Loaders are of 4 types: Absolute, Relocating, Direct Linking, Bootstrap.
Another use of linker is to combine all object modules.	It helps in allocating the address to executable codes/files.
Linker is also responsible for arranging objects in program's address space.	Loader is also responsible for adjusting references which are used within the program.

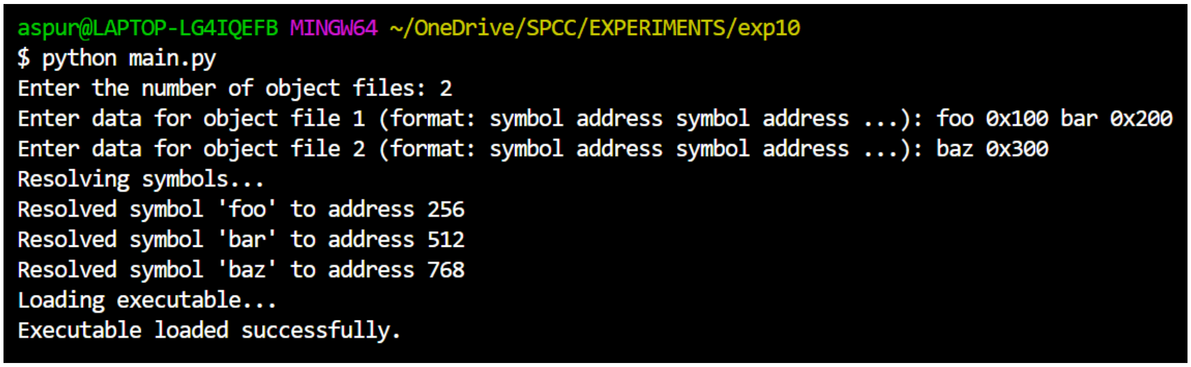
Implementation / Code



```python
import struct


class SymbolTable:
    def __init__(self):
        self.symbols = {}

    def add_symbol(self, name, address):
        self.symbols[name] = address
```



```
    def resolve_symbol(self, name):
        return self.symbols.get(name, None)

class Loader:
    def __init__(self):
        self.symbol_table = SymbolTable()

    def load_object_files(self):
        object_files_data = []
        num_object_files = int(input("Enter the number of object
files: "))
        for i in range(num_object_files):
            object_file_data = input(f"Enter data for object file {i +
1} (format: symbol address symbol address ...): ").split()
            object_files_data.extend(object_file_data)
        for i in range(0, len(object_files_data), 2):
            name = object_files_data[i]
            address = int(object_files_data[i + 1], 16)  # Parse
address as hexadecimal
            self.symbol_table.add_symbol(name, address)

    def resolve_symbols(self):
        print("Resolving symbols...")
        for symbol, address in self.symbol_table.symbols.items():
            print(f"Resolved symbol '{symbol}' to address {address}")

    def load_executable(self):
        print("Loading executable...")
        print("Executable loaded successfully.")

if __name__ == "__main__":
    loader = Loader()
    loader.load_object_files()
    loader.resolve_symbols()
    loader.load_executable()
```

<table>
<tr><td></td><td>[redacted]</td></tr>
<tr><td><b>Output</b></td><td>


```
aspur@LAPTOP-LG4IQEFB MINGW64 ~/OneDrive/SPCC/EXPERIMENTS/exp10
$ python main.py
Enter the number of object files: 2
Enter data for object file 1 (format: symbol address symbol address ...): foo 0x100 bar 0x200
Enter data for object file 2 (format: symbol address symbol address ...): baz 0x300
Resolving symbols...
Resolved symbol 'foo' to address 256
Resolved symbol 'bar' to address 512
Resolved symbol 'baz' to address 768
Loading executable...
Executable loaded successfully.
```

</td></tr>
<tr><td><b>Conclusion</b></td><td>The program prompts the user to input data for object files, then proceeds to resolve symbols and simulate loading an executable into memory. This illustrates a simplified workflow akin to that of a linker/loader.</td></tr>
<tr><td><b>References</b></td><td>[1] Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools (2nd ed.). Addison-Wesley.

[2] Muchnick, S. S. (1997). Advanced Compiler Design and Implementation. Morgan Kaufmann.

[3] Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems (4th ed.). Prentice Hall.

[4] <i>Difference between Linker and Loader</i>. (2020, August 11). GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-linker-and-loader/</td></tr>
</table>