SIMPLE-LOADER

CONTRIBUTION IN THE PROJECT :

- **RONAK (2023446)** : Error Handling, Documentation, Deallocating Function
- **RIJUL(2023432)** : File Handling, Rest Of The Code, Compiling The Project

Link to GITHUB :

https://github.com/ronak23446/OS-PROJECT.git

DOCUMENTATION :

1. The written code implements a basic loader for Executable and Linkable Format (ELF) binaries, specifically designed to load and execute a 32-bit ELF file in a Unix-like environment. This process involves reading the binary file made with the `Makefile`, identifying and loading its relevant segments into memory, and then executing the program's entry point function.
2. The program begins by opening the ELF file using the `open()` system call. It then allocates memory for and reads the ELF header, which contains essential information about the binary, such as the location of the program headers.
3. These program headers, which describe the segments of the binary to be loaded into memory, are read next. The code then iterates through each program header to identify segments of type `PT_LOAD`, which are meant to be loaded into memory. For each such segment, the program uses `mmap()` to allocate memory with the appropriate permissions (read, write, execute). It then reads the segment data from the file into the allocated memory.
4. After all necessary segments are loaded into memory, the program retrieves the entry point address from the ELF header. This address points to the beginning of the program's execution within the loaded segments.
5. The program then casts this address to a function pointer and invokes it, effectively transferring control to the loaded ELF binary and beginning its

execution. The return value of the entry point function, named `_start`, is captured and printed to the console.

6. The program also incorporates error handling to ensure that any issues during file operations, memory allocation, or segment loading are properly managed. In the event of an error, the program displays an appropriate message and exits.
7. The above steps are all executed using the function named `load_and_run_elf`.
8. After execution, the `loader_cleanup` function performs cleanup to prevent resource leaks by freeing allocated memory and closing the file descriptor.
9. This implementation showcases fundamental concepts in low-level systems programming, including file I/O, memory management, and dynamic loading of executable code.