REPORT

1. Global Variables:

- ➔ **Elf32_Ehdr *ehdr**`: Pointer to the ELF header.
- ➔ **Elf32_Phdr *phdr**`: Pointer to the program header.
- ➔ **int fd**`: File descriptor for the ELF binary.
- ➔ **size_t sizeStored**`: Size of the loaded segment
- ➔ **char *virtual_mem**`: Memory where the ELF segment is loaded.

2. System Calls:

- ➔ **`munmap():** A function similar to malloc().
- ➔ **close():** Used to close an opened file.
- ➔ **open():** Used to open a file (through file descryptor)
- ➔ **exit():** Used to terminate the program.
- ➔ **malloc():** <u>Used to allocate memory in the heap.</u>
- ➔ **lseek():** Used to reposition the read/write pointer within a file.
- ➔ **read():** Used to read data from a file descriptor in Unix- like operating systems.
- ➔ **mmap():** Used to map files or devices into memory, allowing direct manipulation of their contents as if they were array.

3. Main Loading and Execution Logic:

- ➔ **load_and_run_elf()**`: Orchestrates the entire process of loading and executing the ELF binary. It reads the ELF header, locates the PT_LOAD segment, loads it into memory, and then jumps to the entry point to execute the binary.

4. Cleanup:

- ➔ **loader_cleanup()**`: This funtion releases memory and performs cleanup operations after the execution is complete.

5. Main Function:

- ➔ **main()**`: This function validates the command-line argument, then calls `load_and_run_elf()` to load and execute the provided ELF binary. It also invokes the cleanup routine.

Code Execution:

1.File Access Initialization: Initiate the process by acquiring a file descriptor for the targeted ELF file. This descriptor serves as a conduit for accessing and reading the file's content.

2.Header Memory Allocation: Allocate dynamic memory to accommodate the ELF headers (ehdr) and program header tables (pht). These headers encapsulate critical details concern in the ELF file's internal structure, encompassing program segments. Utilize memory allocation mechanisms like malloc to ensure appropriate memory provisioning for these structures.

3.Header Information Retrieval: Post memory allocation, employ read and lseek functions to extract the ELF headers and program header tables from the file, storing them within the pre-allocated memory space. This retrieval procedure is pivotal for comprehending the intricate file arrangement.

4.Program Header Table Traversal: Navigate through the program header table (PHT), which encompasses comprehensive information pertaining to distinct program segments, including loadable segments. These segments embody components necessitating loading into memory to facilitate program execution.

5.Efficient Section Loading via mmap: Leverage the mmap function to create a mapping between sections of the ELF file and memory. Focusing on segments with a "load" classification, this approach streamlines the loading of program sections into memory without requiring manual data extraction. The resulting memory-mapped regions directly correspond to respective file sections.

6.In-Memory Loading and Entry Point Configuration: Employ a combination of lseek and read functions to load the executable content into memory effectively. Capture the entry point address indicated within the ELF header. The entry point designates initiation address.

7.Program Execution Commencement: Conclude the procedure by casting the entry point address into a function pointer. Subsequently, invoke this pointer, effectively triggering the initiation of the loaded program's execution.

Contributions:

1. Angadjeet Singh (2022071): Contribution – 50%
Details:

➔ Focused on the code implementation.
➔ Did all the logic framework and defined the required system calls and variables.

2. Apaar IIITD (2022089): Contribution – 50%
Details:

➔ Made basic structure of the code and worked intensively on error analysis.
➔ Made all the Makefiles for the bonus part of the assignment and

GITHUB : https://github.com/ANGADJEET/2022071-089_OS_Assignment_1