# **Contribution:** Both contributed equally

Github Repository:
[Sanchay117/OS-Smart-Loader: Assignment 4 Of Operating Systems Course]

Components

1. ELF Header Reading

- **Function**: `void read_elf_header()`
- Reads the ELF header from the executable and verifies its validity by checking the magic numbers (`0x7f`, `E`, `L`, `F`).
- Exits with an error if the header is invalid.

2. Program Header Reading

- **Function**: `void read_programm_header(const int i, const unsigned short int programm_header_size)`
- Reads the program headers sequentially to identify segment details.

3. Page Fault Handler

- **Function**: `void handle_page_fault(int signum, siginfo_t *sig, void* context)`
- Catches a segmentation fault and identifies the address causing the fault.
- Allocates a 4KB page at the fault address using `mmap`.
- Reads data from the file into the allocated memory page.
- Tracks internal fragmentation if the segment size does not align with 4KB boundaries.

4. Loader Cleanup

- **Function**: `void loader_cleanup()`
- Frees allocated memory and unmaps segments using `munmap`.

5. Main Execution Logic

- **Entry point**: `int main(int argc, char** argv)`
- Sets up the SIGSEGV signal handler.
- Reads the ELF and program headers.
- Begins execution by typecasting the entry point address and invoking it as a function.

Signal Handling

Setup

- **Function**: `void setup_signal_handler()`
- Configures a signal handler for SIGSEGV using `sigaction`.
- Calls `handle_page_fault()` when a segmentation fault occurs.

Page Fault Logic

- The handler checks which program segment covers the faulting address.
- Aligns the faulting address to a 4KB boundary and allocates memory with `mmap`.
- Reads the relevant data from the executable into the allocated memory.
- Reports page allocations and internal fragmentation.

Memory Management

Allocation

- Pages are allocated using `mmap` with the `MAP_FIXED | MAP_PRIVATE | MAP_ANONYMOUS` flags.
- Ensures allocation is done only when needed, minimizing memory usage.

Cleanup

- Unmaps memory after program execution using `munmap`.
- Frees dynamically allocated data structures (e.g., ELF header).

Error Handling

- Validates ELF file format and program header reading.
- Checks the success of `mmap` and `lseek` operations.
- Exits gracefully if any error occurs during reading or memory mapping.