SimpleLoader - An ELF Loader in C from Scratch

Contributers :-

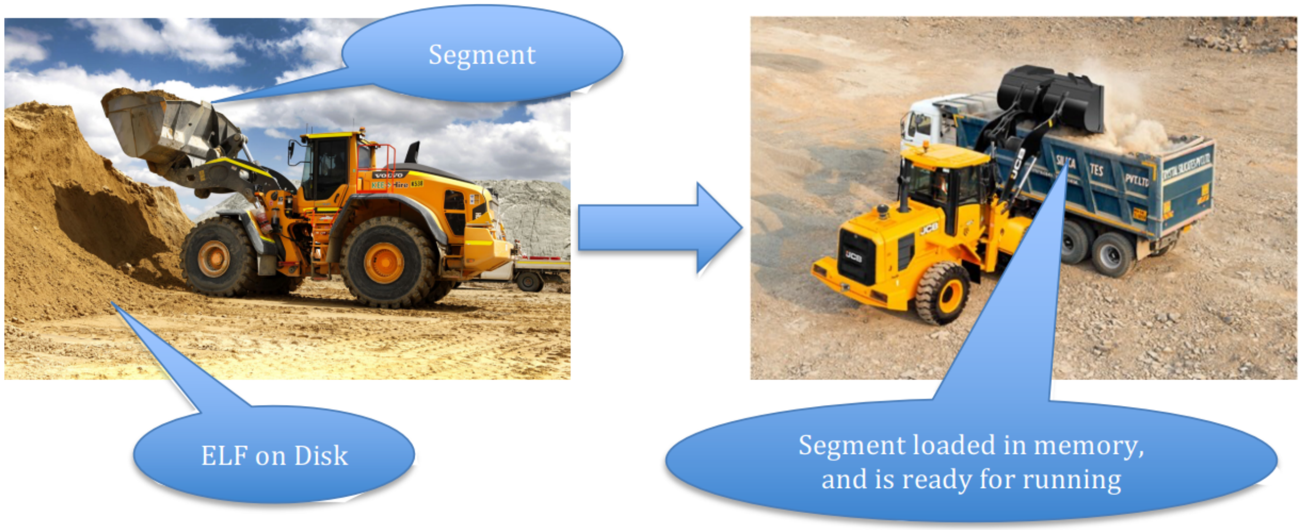
Vishal Kumar Maurya(2022580)
Subham Maurya (2022510)

Github Link for the Repository (Private) :- ***Click Here*** or https://github.com/vmaurya6622/OS-Project-S-3.git

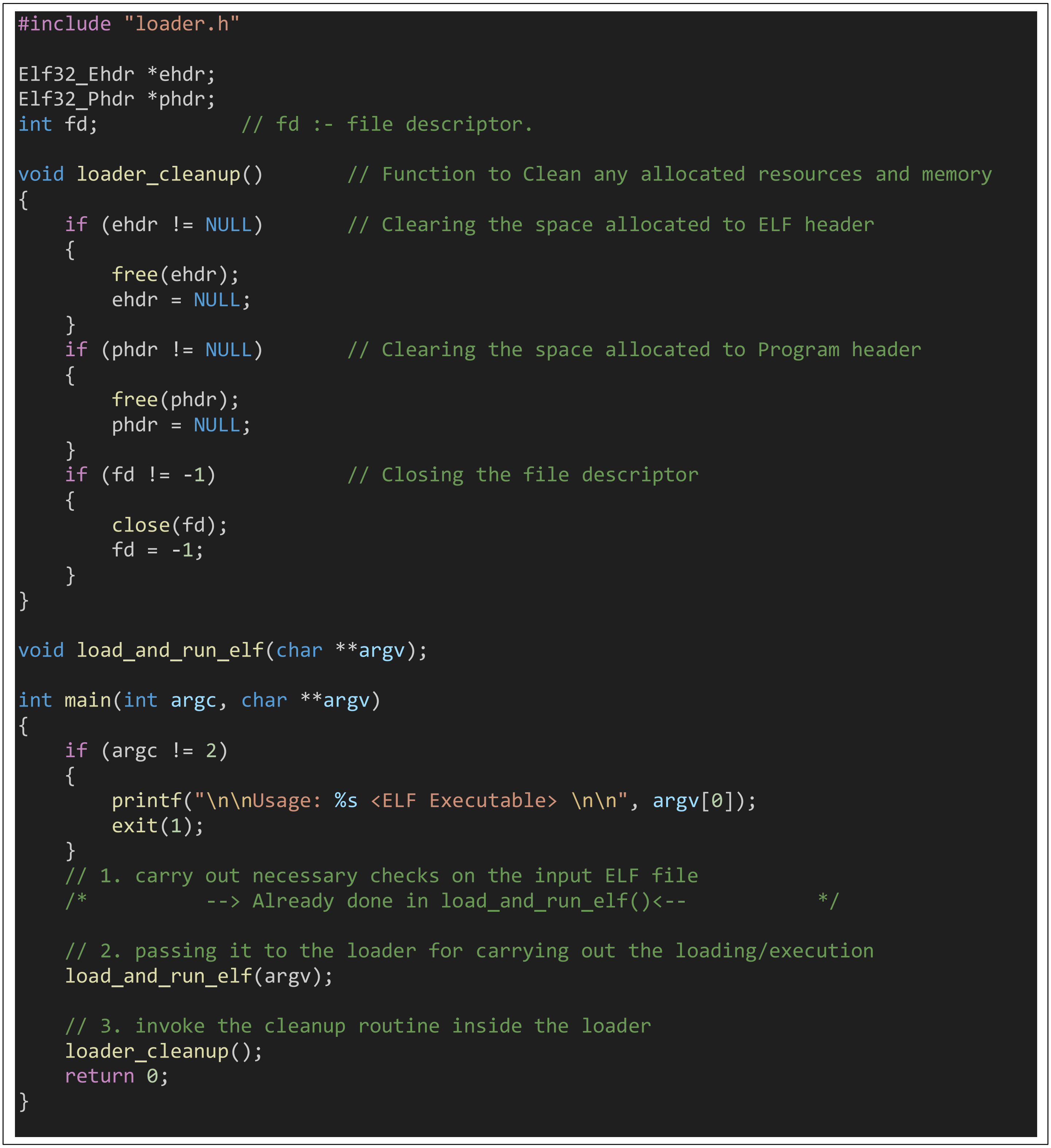
Files Contained :- Makefile, fib.c, loader.c, loader.h we have used deberian based (KALI linux) to complete our assignment and we have completed Without-Bonus part.

Contribution by Vishal Kumar Maurya (2022580) :- I have implemented the first three specified requirements by the loader and Run-elf function. we both have done work simultaneously for the implementation of for-loop to find the valid **entry point** address. i have done debugging and implemented the cleaner function.

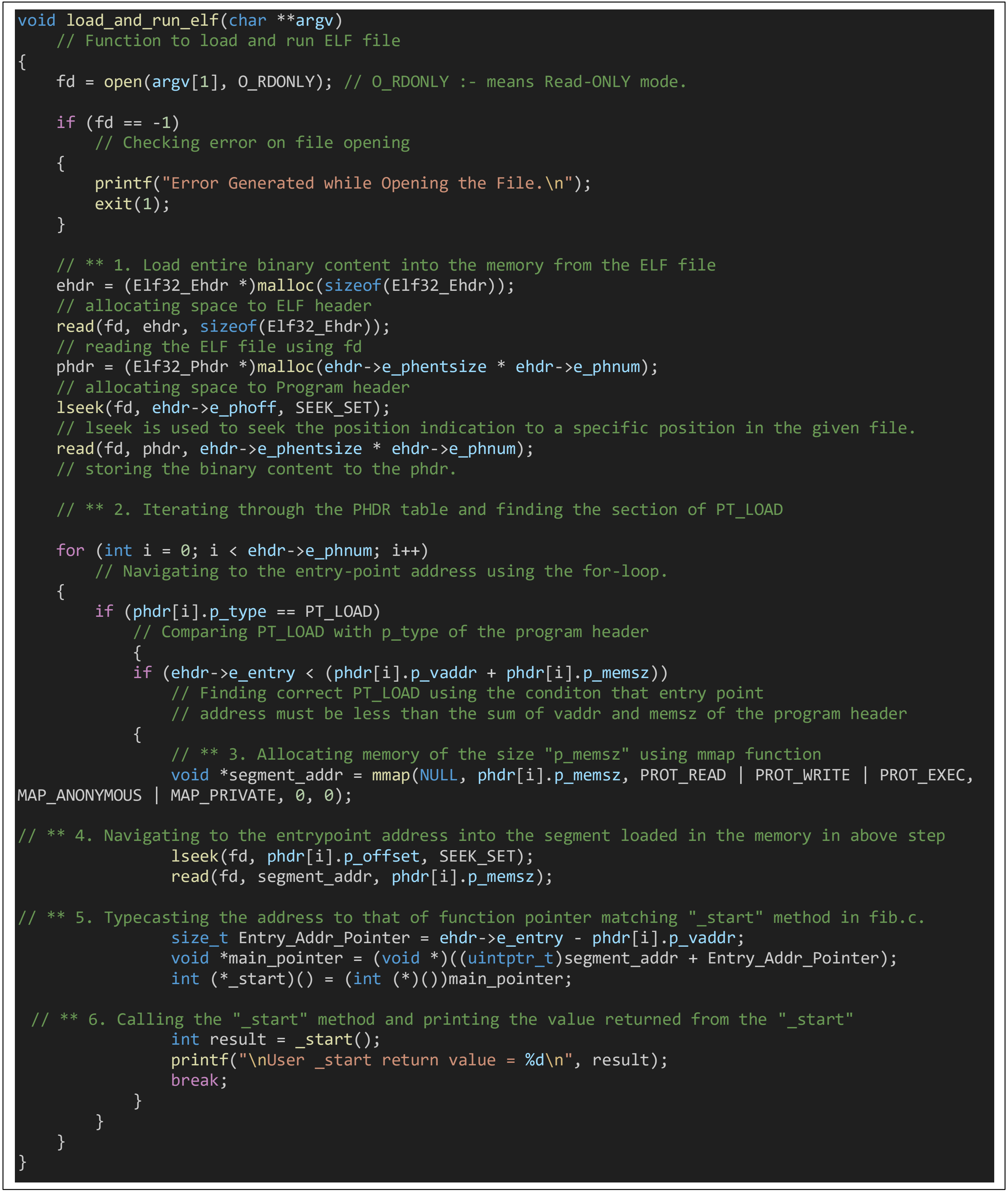
Contribution by Subham Maurya (2022580) :- I have implemented the Last three Specified requirements of the loader and Run-elf function. i have worked on comments and improved the visibility of the code. i have also contributed by doing debugging and searching the relevant sources to get help like from lecture slides and OS- Book.



Implemented source code is on the next page :-



```c
#include "loader.h"

Elf32_Ehdr *ehdr;
Elf32_Phdr *phdr;
int fd;            // fd :- file descriptor.

void loader_cleanup()       // Function to Clean any allocated resources and memory
{
    if (ehdr != NULL)       // Clearing the space allocated to ELF header
    {
        free(ehdr);
        ehdr = NULL;
    }
    if (phdr != NULL)       // Clearing the space allocated to Program header
    {
        free(phdr);
        phdr = NULL;
    }
    if (fd != -1)           // Closing the file descriptor
    {
        close(fd);
        fd = -1;
    }
}

void load_and_run_elf(char **argv);

int main(int argc, char **argv)
{
    if (argc != 2)
    {
        printf("\n\nUsage: %s <ELF Executable> \n\n", argv[0]);
        exit(1);
    }
    // 1. carry out necessary checks on the input ELF file
    /*          --> Already done in load_and_run_elf()<--           */

    // 2. passing it to the loader for carrying out the loading/execution
    load_and_run_elf(argv);

    // 3. invoke the cleanup routine inside the loader
    loader_cleanup();
    return 0;
}
```



```c
void load_and_run_elf(char **argv)
    // Function to load and run ELF file
{
    fd = open(argv[1], O_RDONLY); // O_RDONLY :- means Read-ONLY mode.

    if (fd == -1)
        // Checking error on file opening
    {
        printf("Error Generated while Opening the File.\n");
        exit(1);
    }

    // ** 1. Load entire binary content into the memory from the ELF file
    ehdr = (Elf32_Ehdr *)malloc(sizeof(Elf32_Ehdr));
    // allocating space to ELF header
    read(fd, ehdr, sizeof(Elf32_Ehdr));
    // reading the ELF file using fd
    phdr = (Elf32_Phdr *)malloc(ehdr->e_phentsize * ehdr->e_phnum);
    // allocating space to Program header
    lseek(fd, ehdr->e_phoff, SEEK_SET);
    // lseek is used to seek the position indication to a specific position in the given file.
    read(fd, phdr, ehdr->e_phentsize * ehdr->e_phnum);
    // storing the binary content to the phdr.

    // ** 2. Iterating through the PHDR table and finding the section of PT_LOAD

    for (int i = 0; i < ehdr->e_phnum; i++)
        // Navigating to the entry-point address using the for-loop.
    {
        if (phdr[i].p_type == PT_LOAD)
            // Comparing PT_LOAD with p_type of the program header
            {
            if (ehdr->e_entry < (phdr[i].p_vaddr + phdr[i].p_memsz))
                // Finding correct PT_LOAD using the conditon that entry point
                // address must be less than the sum of vaddr and memsz of the program header
            {
                // ** 3. Allocating memory of the size "p_memsz" using mmap function
                void *segment_addr = mmap(NULL, phdr[i].p_memsz, PROT_READ | PROT_WRITE | PROT_EXEC,
MAP_ANONYMOUS | MAP_PRIVATE, 0, 0);

// ** 4. Navigating to the entrypoint address into the segment loaded in the memory in above step
                lseek(fd, phdr[i].p_offset, SEEK_SET);
                read(fd, segment_addr, phdr[i].p_memsz);

// ** 5. Typecasting the address to that of function pointer matching "_start" method in fib.c.
                size_t Entry_Addr_Pointer = ehdr->e_entry - phdr[i].p_vaddr;
                void *main_pointer = (void *)((uintptr_t)segment_addr + Entry_Addr_Pointer);
                int (*_start)() = (int (*)())main_pointer;

 // ** 6. Calling the "_start" method and printing the value returned from the "_start"
                int result = _start();
                printf("\nUser _start return value = %d\n", result);
                break;
            }
        }
    }
}
```