**SimpleLoader Assignment**

Group Members:

**Asa(2022113),**

**Sargun(2022450)**

**GitHub Repository:** https://github.com/sargun4/OS/tree/main

**Project Overview:**

The SimpleLoader project aims to develop a custom ELF 32-bit executable loader in C, capable of loading and executing ELF files. The loader will be implemented as a shared library (lib\_simpleloader.so) and integrated with a helper program (launch.c) for loading and executing ELF executables.

**Contributions:**

Asa:

Implemented the memory mapping and copying of segments from the ELF file.

Implemented the ELF header parsing and program header processing logic.

Collaborated on testing, debugging the loader and Makefile creation.

Designed the Makefile structure for compiling the loader and helper program.

Contributed to the GitHub repository

Sargun:

Handled error checking and memory allocation in the loader code.

Assisted in designing the project structure and directory layout.

Documentation

Implemented the entry point retrieval and execution logic.

Contributed to the GitHub repository

**Implementation Details:**

**Loader (loader.c):**

1.Global variables are defined:

* + Elf32\_Ehdr \*ehdr; for storing the ELF header.
  + Elf32\_Phdr \*phdr; for storing the program header.
  + void \*virtual\_mem; for storing the virtual memory where the ELF segment will be loaded.
  + int fd; as the file descriptor for the opened ELF file.

2. The loader\_cleanup function is defined for memory deallocation and cleanup. It uses the munmap function to release the virtual memory and free to deallocate the allocated ehdr and phdr

3.The load\_and\_run\_elf function is defined to load and execute the ELF executable specified in the exe argument.

* + It opens the ELF file for reading.
  + Reads the ELF header into the allocated memory and checks if it has been read successfully.
  + Validates the ELF magic number to ensure it's a valid ELF file.
  + Iterates through the program headers to find a suitable segment of type PT\_LOAD that contains the entry point address.
  + Allocates virtual memory using mmap and copies the segment content to that memory.
  + Calculates the entry point address within the loaded segment.
  + Typecasts the entry point address to a function pointer and invokes the \_start function from the loaded segment.
  + Prints the return value from \_start.

**Launcher(launch.c):**

1. It first checks if the correct number of command line arguments has been provided. If not exactly two arguments are provided (including the program name itself), an error message is printed explaining the correct usage, and the program exits with an error code.
2. The program then retrieves the name of the ELF executable from the second command line argument (argv[1]) and assigns it to the elfFileName variable. It checks if the specified ELF file exists using the access function with the F\_OK flag. If the file does not exist, an error message is printed, and the program exits with an error code.
3. Next, the program checks if the specified ELF file is readable using the access function with the R\_OK flag. If the file is not readable, an error message is printed, and the program exits.
4. Once the file checks are complete, the program proceeds to the main tasks:

It calls a function load\_and\_run\_elf(argv) to load and execute the specified ELF executable. The argv array is passed as an argument, presumably for further processing within the load\_and\_run\_elf function.

1. After the execution of the ELF is complete, the program calls loader\_cleanup() to perform cleanup operations related to the loader.

Finally, the main function returns 0.

**Output of Fib(fib.c):**

User \_start return value = 102334155

**The executable “launch” and the library “lib\_simpleloader.so” are saved inside the “bin” folder by the make command. The executable “fib” is inside “test” sub-directory.**