# OS Assignment 1

Ishaan Agrawal and Aniket Gupta

August 24, 2023

## 1 Individual Contributions

- Aniket Gupta: loader.c
- Ishaan Agrawal: Main Makefile, sub-Makefiles

## 2 GitHub Repository

GitHub Repository Link.

## 3 Implementation

Implementation of loader.c is as follows:

1. when we give command ./launch ./fib, these executables are picked up by the main function. The value of argc is 2 and argv[0] is the executable of launch while argv[1] is the executable of fib.
2. Then load_and_run_elf function is called. We make a file descriptor for the fib exe, and open the file in read only mode.
3. 
```
ehdr = mmap(NULL, sizeof(Elf32_Ehdr), PROT_READ, MAP_PRIVATE, fd, 0);
```
   this command loads the entire binary content which is pointed by a pointer of type Elf32_Ehdr called ehdr. Similarly we have another pointer of type Elf32_Phdr
4. 
```
phdr = (Elf32_Phdr *)((char *)ehdr + ehdr->e_phoff);
```
   By casting ehdr to a char*, the memory region being pointed to by ehdr is treated like an array of bytes. Adding ehdr->e_phoff to this byte pointer would move the pointer forward by that many bytes.
5. 
```
for (i = 0; i < ehdr->e_phnum; i++) {
  if (phdr[i].p_type == PT_LOAD) {
    if (ehdr->e_entry >= phdr[i].p_vaddr && ehdr->e_entry < phdr[i].p_vaddr + phdr[i].p_memsz) { // checking if the entrypoint method is in the segment
      targetSegment = phdr[i]; // storing the segment containing the entrypoint method
      break; // since we have found the segment containing the entrypoint method, we break out of the loop
    }
  }
}
```
   This for loop is used to find a PT_LOAD section that would contain the entry point address.

```
if (phdr[i].p_type == PT_LOAD)
```

This statement just checks if we found the PT_LOAD segment.

```
if (ehdr->e_entry >= phdr[i].p_vaddr && ehdr->e_entry < phdr[i].p_vaddr + phdr[i].
    p_memsz)
```

This is the check we used to determine whether the "PT_LOAD" segment contains entrypoint address or not (since there can be multiple PT_LOADS). "p_vaddr" is the virtual address of that particular PT_LOAD segment, it points to the offset (start) of the chunk of memory held by the PT_LOAD segment. Similarly, "p_memsz" contains the total size of the whole segment in bytes.

6.
```
void *virtual_mem = mmap((void *)targetSegment.p_vaddr, targetSegment.p_memsz,
    PROT_READ | PROT_WRITE | PROT_EXEC, MAP_PRIVATE | MAP_FIXED, fd, targetSegment.
    p_offset); // allocating memory using mmap
```

This mmap statement creates a mapping into the memory starting from the virtual address till the size, i.e p_memsz

7.
```
int (*_start)() = (int (*)())(virtual_mem + (ehdr->e_entry - targetSegment.p_vaddr)
    );
```

This statement includes the navigation to the entrypoint address and typecasting the address to that of the function pointer matching _start in fib.c
"int (*_start)()" declares a function pointer named "_start" which points to a function returning a value of type int
"virtual_mem + (ehdr->e_entry - targetSegment.p_vaddr)" calculates the absolute address of _start

8. The master Makefile first sets the target to all, and then lists the dependencies 'bin/loader', 'bin/launcher' and 'bin/test'. Then separate make commands are run in the three respective directories.
9. A new bin folder is made within the directory, which contains the launch executable, and simpleloader.so
10. The new bin folder is used to run the testcase from the test directory.

## 4 How to Run

1. Download the directory from the GitHub repository.
2. Run the top-level Makefile using 'make'.
3. After successful execution, run 'make bonus'.
4. After getting output, run 'make clean'.