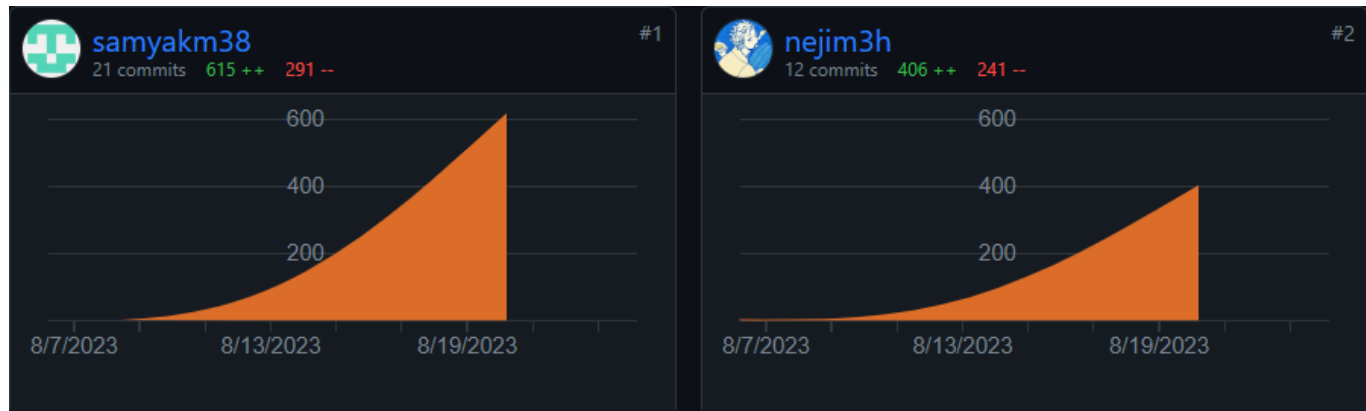


SimpleLoader: An ELF Loader in C from Scratch

Contributions



Samyak Mehta

- Iterated through the PHDR table, identifying the PT_LOAD section that contained the entrypoint address in "fib.c."
- directly typecasted the address to a function pointer, specifically for the "_start" method in "fib.c."
- executed the "_start" method using the function pointer and printed the returned value.
- Implemented launch.c
- Created Makefiles, directory structure, and shared library version of loader.
- Wrote the Code Documentation.
- Error Handling.
- Authored Design Document.

Nimit Panwar

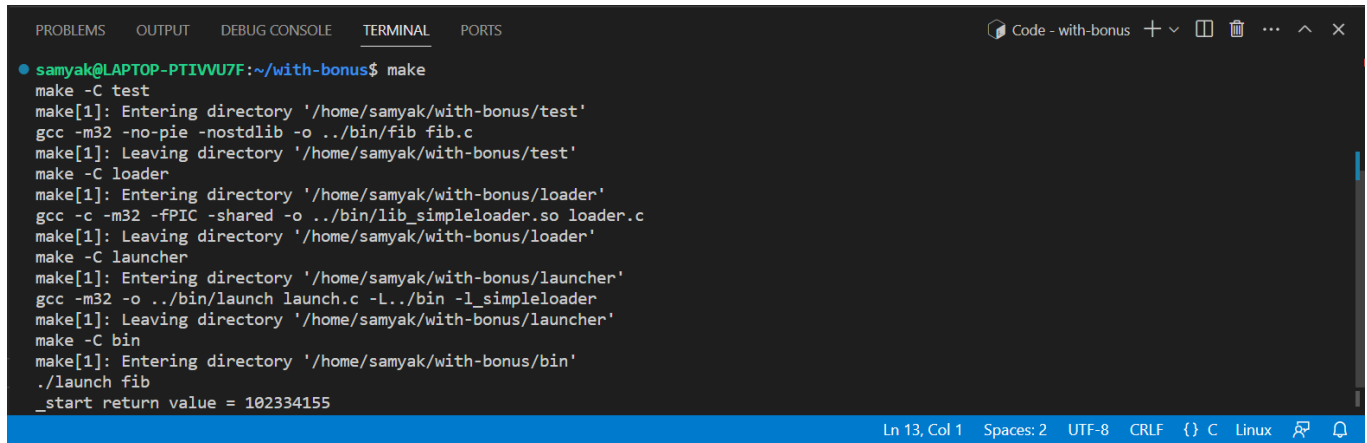
- I used the "open" system call to get the file descriptor for the binary file and employed "read" to load its content into a dynamically allocated heap memory.
- I navigated to the entrypoint address ("e_entrypoint") within the loaded memory segment, accounting for potential address differences from "p_vaddr."
- Using the "mmap" function, I allocated memory according to the "p_memsz" size specified in the segment, then copied the segment's content into this memory.
- Implemented the cleaner function.
- Wrote the Code Documentation
- Authored Design Document.

Simple Loader Implementation

1. First we open our elf file "fib" through the code in launch.c using a file descriptor.
2. We copy the content of the elf file into a variable "content".
3. Then we check if the elf header is valid and supported using functions in the launcher.c file.
4. Now the function "load_and_run_elf" is called which is provided to launch by the sharedlibrary "lib_simpleloader" compiled from loader.c stored in the loader folder.
5. we open the elf file once again and using the elf header struct, find the program header offset, no. of entries in the program header and the size of each entry in it.
6. We read and store the program header table in a memory location pointed by "phdr" .
7. We locate the executable segment by iterating through the program header table using PT_LOAD .
8. To find the actual address of the start of the executable segment we add the offset of that segment to the address of the start of the file(which is present at offset 0).
9. We allocate memory for the executable segment using the function "mmap" and then store the segment in it using memcpy.
10. Then we make a function pointer which takes no argument and outputs an integer.
11. Now we calculate the actual address of the "_start" method and store it in the "_start" variable
12. The address is calculated by the following pointer arithmetic calculation "virtual_mem + (ehdr->e_entry - reqPhdr->p_vaddr)".
13. virtual_mem is a pointer to the start of the executable segment and we add to it the difference between the virtual address of the actual "_start" method and the virtual address of the executable segment.
14. Then we call the "_start" function and save its result in the int type variable "result".
15. Finally we print the value stored in "result".
16. Then we call the function "loader_cleanup" to free the memory blocks allocated.

How to Run

After opening the "with-bonus" file in VS code, type "make" in the terminal to get the output.



```
samyak@LAPTOP-PTIWWU7F:~/with-bonus$ make
make -C test
make[1]: Entering directory '/home/samyak/with-bonus/test'
gcc -m32 -no-pie -nostdlib -o ../bin/fib fib.c
make[1]: Leaving directory '/home/samyak/with-bonus/test'
make -C loader
make[1]: Entering directory '/home/samyak/with-bonus/loader'
gcc -c -m32 -fPIC -shared -o ../bin/lib_simpleloader.so loader.c
make[1]: Leaving directory '/home/samyak/with-bonus/loader'
make -C launcher
make[1]: Entering directory '/home/samyak/with-bonus/launcher'
gcc -m32 -o ../bin/launch launch.c -L../bin -l_simpleloader
make[1]: Leaving directory '/home/samyak/with-bonus/launcher'
make -C bin
make[1]: Entering directory '/home/samyak/with-bonus/bin'
./launch fib
_start return value = 102334155
```

Link to Private github repository :

<https://github.com/nejim3h/OS-assignments>