Assignment-1 (Section-A)

SimpleLoader: An ELF Loader in C from Scratch

Due by 11:59pm on 2st September 2024
(Total **7%** weightage)
Instructor: Vivek Kumar

No extensions will be provided. Any submission after the deadline will not be evaluated. If you see an ambiguity or inconsistency in a question, please seek clarification from the teaching staff.

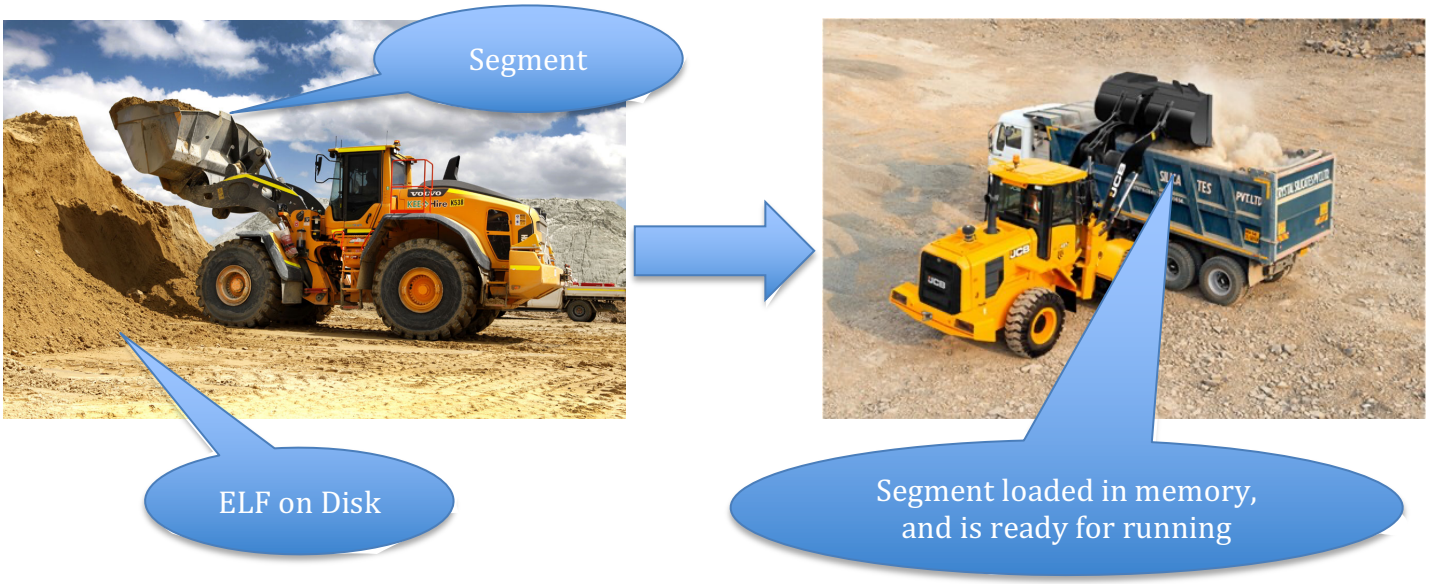
Plagiarism: **This is a pair programming-based assignment that you must do with the group member that you have already chosen. No change to the group is allowed. You are not allowed to discuss the approach/solution outside your group.** You should never misrepresent some other group's work as your own. In case any plagiarism case is detected, it will be dealt as per the new plagiarism policy of IIITD **and will be applied to each member in the group.**

Open-sourcing of this assignment solution is not allowed, even after the course gets over.

Even if you are a single member group, there will not be any relaxations in marking scheme/deadlines, and the same rubric will be followed for each group.

General Instructions

a) Hardware requirements:
   a) You will require a machine having any Operating System that supports Unix API. You cannot do this assignment on MacOS. You can either have a dual boot system having any Linux OS (e.g., Ubuntu), or install a WSL.
b) Software requirements are:
   a) C compiler and GNU make.
   b) You **must** do version controlling of all your code using github. You should only use a **PRIVATE** repository. If you are found to be using a **PUBLIC** access repository, then it will be considered plagiarism. NOTE that TAs will check your github repository during the demo.
c) Text highlighted in green color describes the deliverables eligible for a maximum of **1%** bonus marks.



Assignment Details

1. Summary

You have to implement a SimpleLoader for loading an ELF 32-bit executable in plain-C **without** using any library APIs available for manipulating ELF files. Your SimpleLoader should compile to a **shared library** (lib_simpleloader.so) that can be used to load and execute the executable using a helper program as shown below. Some starter code with desired directory structure and placeholder files are provided with this PDF.

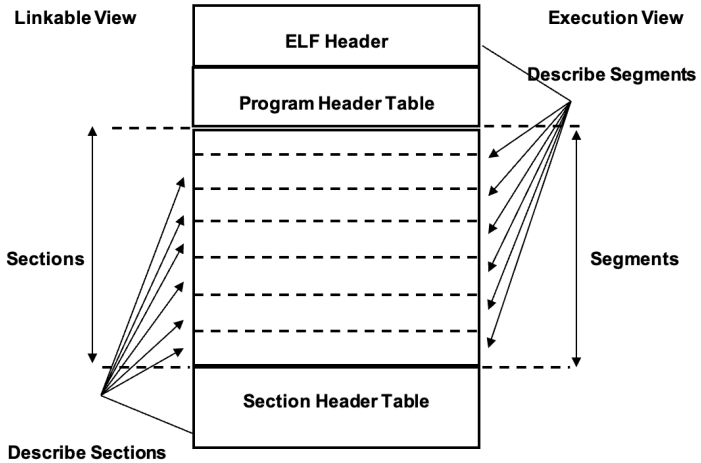
```
[vivek@possum]$ ./launch ../test/fib
User _start return value = 102334155
```

2. Directories and Files

```
[vivek@possum]$ ls -R
.:
Makefile  bin  launcher  loader  test

./bin:

./launcher:
Makefile  launch.c

./loader:
Makefile  loader.c  loader.h

./test:
Makefile  fib.c
```

a) Each folder will contain a Makefile to compile the C file present in that directory.
b) There will be a top-level Makefile which would compile all the C-files by internally invoking make command in each of the sub-directories shown above.
c) "test" folder contains the fib.c as the testcase to be run using SimpleLoader.
d) The executable "launch" and the library "lib_simpleloader.so" should be automatically saved inside the "bin" folder by the make command. The executable "fib" should remain inside "test" sub-directory.
e) You **must** pass the flag "-m32" to the gcc compiler for compiling each of the C files.
f) You **must** pass the flags "-m32", "-no-pie" and "-nostdlib" to the gcc compiler for compile the test case "fib.c". You **should not** do any modifications to "fib.c" to be able to use the SimpleLoader described over here.
g) You are **allowed** to pass any other relevant flags (e.g., for creating shared library, etc.) to the gcc compiler as needed.

3. SimpleLoader Implementation ("loader.c")



There are two views of ELF object files, the linkable view and execution view. The ELF header is always at offset zero of the object file. This can be considered as road map for the entire file organization. ELF header (EHDR) structure for 32-bit object file has members as following.

```
typedef struct
{
  unsigned char     e_ident[EI_NIDENT]; /* Marks file as obj file and provides
                                           machine-independent data */
  unsigned short    e_type;             /* Object file type eg. executable file,
                                           core file, shared obj file etc. */
  unsigned short    e_machine;          /* Tells the architecture of machine on which
                                           this ELF file was created */
  unsigned int      e_version;          /* Object file version */
  unsigned int      e_entry;            /* Entry point virtual address to which system
                                           first transfer control at process startup */
  unsigned int      e_phoff;            /* Program header table file offset */
  unsigned int      e_shoff;            /* Section header table file offset */
  unsigned int      e_flags;            /* Processor-specific flags */
  unsigned short    e_ehsize;           /* ELF header size in bytes */
  unsigned short    e_phentsize;        /* Size of each program header table structure*/
  unsigned short    e_phnum;            /* Count of total program header table entry */
  unsigned short    e_shentsize;        /* Size of each Section header table structure */
  unsigned short    e_shnum;            /* Count of total section header table entry */
  unsigned short    e_shstrndx;         /* Section header table index of the entry
                                           associated with string table secion */
} Elf32_Ehdr;
```

The member e_phnum of ELF header structure gives the count of total program header present in the executable file. There will be a segment corresponding to each program header. Details on the type of segment can be obtained from the corresponding program header structure. The first program header starts at offset e_phoff bytes from the beginning of the object file. Let us now see the layout of a program header (PHRD) structure.

```
typedef struct
{
  unsigned int  p_type;         /* type of segment being described (e.g., PT_LOAD, etc.) */
  unsigned int  p_offset;       /* segment offset from the beginning of the ELF file */
  unsigned int  p_vaddr;        /* virtual address at which first byte of the segment resides in memory*/
  unsigned int  p_paddr;        /* ignore this for assignment */
  unsigned int  p_filesz;       /* ignore this for assignment */
  unsigned int  p_memsz;        /* total bytes this segment will occupy in memory */
  unsigned int  p_flags;        /* ignore this for assignment */
  unsigned int  p_align;        /* ignore this for assignment */

} Elf32_Phdr;
```

The working of SimpleLoader is exactly same as described in Lecture-03 slides. The bare minimum steps in loading and running an executable is as follows:

a) Use the “open” system call to get the file descriptor for the input binary and “read” system call to read the content of the binary into a heap allocated memory of appropriate size. Use malloc to allocate space for copying content of the binary.
b) Iterate through the PHDR table and find the section of **PT_LOAD** (**p_type**) type that contains the address of the entrypoint method in fib.c
c) Allocate memory of the size “**p_memsz**” using **mmap** function as shown below and then copy the segment content:

```
virtual_mem = mmap(NULL, ph.p_memsz, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_ANONYMOUS|MAP_PRIVATE, 0, 0);
```

d) Navigate to the entrypoint address (**e_entrypoint**) into the segment loaded in the memory in above step. The entrypoint address may not be the starting address in that segment indicated by "**p_vaddr**". You have to walk that segment to reach the virtual address referred by "**e_entrypoint**".
e) Once you reach that location, simply typecast the address to that of function pointer matching "_start" method in fib.c.
f) Call the "_start" method and print the value returned from the "_start".

As mentioned in the Lecture-03 slides, the above steps will work only with statically linked executable file that has to be loaded and executed. The lib_simpleloader.so is the shared library implementation of loader.c file.

4. Implementation of Launcher ("launch.c)

The job of the launch.c is to a) ask the user to provide the ELF file as a command line argument, b) carry out necessary checks on the input ELF file, c) passing it to the loader for carrying out the loading/execution, and d) invoke the cleanup routine inside the loader. The launcher.c has to include "loader.h" containing the declarations of the two APIs used in step-c and step-d mentioned above. Note that to compile launch.c you must include the path to the lib_simpleloader.so ("bin" folder) and also supply the library name as a parameter to the gcc. Also, note that you don't need to recompile launch.c and loader.c every time you want to load a new executable. However, the test case must be compiled if its executable doesn't exist, or if there are any new changes inside the test case since the last compilation.

5. Requirements

a. You should strictly follow the instructions provided above.
b. You should strictly follow the directory structure for your coding as mentioned above.
c. Proper error checking must be done at all places. Its up to you to decide what are those necessary checks.
d. Proper documentation should be done in your coding.
e. Your assignment submission should consist of two parts:
    a. A zip file containing your source files as mentioned above. Name the zip file as "group-ID.zip", where "ID" is your group ID specified in the spreadsheet shared by the TF.
    b. A design document **inside the above "zip"** file detailing the contribution of each member in the group, detailing your SimpleLoader implementation, and the link to your **private** github repository where your assignment is saved.
f. There should be **ONLY ONE** submission per group.
g. In case your group member is not responding to your messages or is not contributing to the assignment then please get in touch with the teaching staff immediately.
h. Text highlighted in green color describes the deliverables eligible for a maximum of **1%** bonus marks. You must have the correct implementation of the loader.c for getting the bonus marks.
i. If you are attempting bonus marks, you should use the starter code in the directory "**with-bonus**", otherwise "**without-bonus**" for submitting an implementation without the bonus marks.

6. Hint

Approach your assignment implementation in the following sequence:

a) Implement all the functionality of the loader inside a single loader.c file. It should directly be able to accept the executable of fib.c as a command line parameter, and then should be able to load/run it.
b) Once you are done with the above implementation, implement error handlings at all required places.
c) Now you should implement the launch.c file as mentioned above in requirements.
d) Your final step should be to create Makefiles, directory structure, and shared library version of loader.

7. Reading / Reference Materials

a. Lecture 02 and Lecture 03 slides
b. https://man7.org/linux/man-pages/man5/elf.5.html
c. https://wiki.osdev.org/ELF_Tutorial