Assignment-4 (Section-A)

SimpleSmartLoader: An Upgraded SimpleLoader in C

Due by 11:59pm on 3rd November 2023
(Total **6%** 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.** 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.

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

General Instructions

a) Hardware requirements:
   a) You will require a machine having any Operating System that supports Unix APIs. You should not use MacOS for OS assignments. 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.

Assignment Details

Recall your SimpleLoader implementation from Assignment-1. It had few limitations which did not made it smart. In this assignment, you have to upgrade your SimpleLoader implementation from assignment-1 and implement a SimpleSmartLoader. This loader is smart just because it does not load any program segments upfront. It loads a segment only when needed **while** the program is under execution (i.e., lazily – as it happens in case of Linux as well). Its working is really simple and is explained below. As you have already implemented SimpleLoader, we are not going to give minute details on the implementation of SimpleSmartLoader. SimpleSmartLoader will also work only for ELF 32-bit executable without having any references to glibc APIs.

1. Implementation Details

a) You don't have to mmap the memory upfront for any segment, including the PT_LOAD type segment containing the entrypoint address.
b) The SimpleSmartLoader will directly attempt running the _start method by typecasting the entrypoint address. However, doing so will generate a segmentation fault as that memory page does not exist.
c) Segmentation faults are generated for invalid memory accesses, but in this assignment, segmentation faults are happening due to accessing unallocated segments. As the address generating the segmentation fault is valid, this segmentation fault can be treated as a page fault because the physical page allocation hasn't happened for that segment address.
d) SimpleSmartLoader should handle such segmentation faults by allocating memory using mmap and then load the appropriate segments (i.e., loading and copying is carried out lazily). Total memory allocated by mmap **must** be multiple of page size (4KB).

e) SimpleSmartLoader should be able to dynamically mmap memory and copy the complete content of a segment during the program execution (in one-shot) without terminating the program. The program should simply resume after the segmentation fault is handled appropriately.
f) You are free to implement either of the static or shared library-based implementation of SimpleSmartLoader.
g) You should use the same Makefile from assignment-1 to compile your loader as well as the testcases.
h) You can use the fib.c provided in assignment-1 as a test case even here. Another testcase is also attached herewith (sum.c).
i) After the execution of the executable, SimpleSmartLoader must **report: (i)** the total number of page faults and **(ii)** total number of page allocations carried out for that executable's execution, (iii) total amount of internal fragmentation in KB.

3. Bonus (+1 marks)

You should carry out page-by-page allocation for a segment **instead** of one-shot allocation. In this case, the virtual memory for intra-segment space will be contiguous, but the physical memory may/may_not be contiguous. There will be multiple page-faults even for loading a single segment that you would have to handle. For example, if the .text segment is of 5KB, if the page fault address is within the first 4KB boundary, then you would only allocate one page (4KB) instead of 5KB in one shot. When the address access would happen between 4KB-5KB, there will be another page fault upon which you will allocate the second page and load the remainder of the .text segment.

4. Requirements

a. You should strictly follow the instructions provided above.
b. Proper error checking must be done at all places. Its up to you to decide what are those necessary checks.
c. Proper documentation should be done in your coding.
d. 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 SimpleSmartLoader implementation, and the link to your **private** github repository where your assignment is saved.
e. There should be **ONLY ONE** submission per group.
f. 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.