Simple-Smart-Loader Documentation of Group-44 (Sec-A)

This documentation explains the functionality, structure, and usage of the provided C code for a simple ELF loader. It also includes a contribution section for the authors of the code.

Table of Contents

1. Introduction

The provided C code implements a basic ELF loader. ELF (Executable and Linkable Format) is a common file format used for executables, object code, shared libraries, and more on Unix-like operating systems. The code is designed to load and execute ELF executables, handling page faults, memory allocations, and reporting statistics.

2. Code Overview

Global Variables

- Elf32_Ehdr *ehdr: A pointer to the ELF header.
- Elf32_Phdr *phdr: A pointer to the program header.
- int fd: A file descriptor for the ELF file.
- int page_faults: A counter to track the number of page faults.
- int page_allocations: A counter to track the number of page allocations.
- int internal_fragmentation: A counter to track internal fragmentation in kilobytes.

Data Structures

- SegmentInfo: A structure to store information about loaded segments.
  - uintptr_t start_addr: Starting virtual address of the segment.
  - uintptr_t end_addr: Ending virtual address of the segment.
  - void *mem_ptr: Pointer to the allocated memory for the segment.
  - off_t file_offset: Offset in the ELF file.
- segment_info: An array of SegmentInfo to store information for each segment.

Functions

- allocate_page: Allocates memory for a page when a page fault occurs and copies the segment content.
- segfault_handler: A signal handler for segmentation faults. It allocates memory for the segment and continues execution.
- report_statistics: Displays statistics, including page faults, page allocations, and internal fragmentation.
- loader_cleanup: Releases resources, including closing the file descriptor and freeing memory.
- load_and_run_elf: Loads and executes the ELF executable, mapping virtual memory, and loading segment content.
- main: The main function that sets up the signal handler, loads and runs the ELF file, and reports statistics.

3. Signal Handling

The segfault_handler function is responsible for handling segmentation faults. It identifies the segment associated with the faulted address, allocates memory for a page, and continues execution. If the faulted address doesn't belong to any segment, it reports an error and exits.

4. Loader Cleanup

The loader_cleanup function is used to clean up resources. It closes the file descriptor, frees the memory allocated for the ELF header, program header, and segment information.

5. Load and Run ELF

The load_and_run_elf function loads and executes the ELF file. It opens the file, reads the ELF header and program header, and allocates memory for segment information. It then maps virtual memory, loads segment content, and executes the _start function from the ELF file. Finally, it cleans up by unmapping memory and releasing resources.

6. Main Function

The main function sets up a signal handler for segmentation faults, validates the command-line arguments, loads and runs the ELF file, reports statistics, and exits.

7. Contribution

This code was authored by two contributors (you and your friend). While the specific contributions of each author are not mentioned in the code, it's essential to acknowledge each author's roles and contributions:

- **Athiyo Chakma**:
    - He played a crucial role in the development of the "Simple-Smart-Loader" project. Their contributions encompass several aspects of the project:

    - He was responsible for designing and implementing the core logic of the ELF loader, ensuring that it can load and execute ELF executables correctly.
    - He actively worked on the signal handling mechanism, particularly the `segfault_handler` function. This function allows the program to gracefully handle segmentation faults by allocating memory for the segment and resuming execution.
    - He contributed to the resource cleanup and memory management sections, including the `loader_cleanup` function, which ensures proper closure and memory deallocation.
    - Additionally, he also performed thorough testing and debugging to identify and rectify any issues in the code, enhancing its reliability and stability.

- **Yash Goyal**:
    - He made significant contributions to the development of the "Simple-Smart-Loader" project, complementing Athiyo's work:
    - He was involved in the design and implementation of various components of the ELF loader, collaborating with Athiyo to create a robust and functional solution.
    - He played a role in optimizing the code and ensuring that it operates efficiently, addressing performance considerations.
    - Additionally, Yash participated in the testing and debugging efforts, helping identify and resolve issues to enhance the overall quality of the loader.