OS Assignment - 4
Report

Submitted by Group ID Number - **62**

1. **Akshit Gupta 2022058**
2. **Manit Kaushik 2022277**

Contributions -

Manit Kaushik	Akshit Gupta
• Allocating the physical memory via the physical pages needed for the segment where Segmentation Fault occurred • Making the design document needed	• Making Segmentation Fault Handler function and identifying the segment where the Segmentation Fault occurred • Implementing error checks throughout the code wherever necessary

SimpleSmartLoader Implementation -

The SimpleSmartLoader is a program designed to load and execute ELF (Executable and Linkable Format) executable files. It manages the allocation of memory for program segments and handles segmentation faults, making it a crucial component in the execution of ELF binaries. In this assignment we tried to build-up on assignment 1 which was similar to this in the sense that we had to code for a Simple Loader however here we tried to change our method of memory allocation and tried reducing limitations to the loader coded in assignment 1.
This implementation report offers a detailed examination of the code, highlighting its core features and functionality.

- ### Global Variables:

  The SimpleSmartLoader code employs several global variables that play a cruicial role in its operation:

  - Elf32_Ehdr *ehdr and Elf32_Phdr *phdr: These pointers store references

to the ELF header and program header table, respectively. They are essential for accessing critical information about the loaded ELF file, such as the entry point and program segments which will be used further in the code and here are important to be declared as global variables.

- int fd: This integer represents the file descriptor of the opened ELF file. It facilitates reading from the file and would also be required further in the code.
- int page_faults: A counter used to track the number of segmentation faults encountered during program execution. Since an important part of the assignment is reporting on the number of faults, we would need to access this variable globally.
- int times_memory_allocated: Another counter that keeps track of the number of memory allocations made during program execution. This is again an important part of the assignment and hence we would again need to declare this variable globally to access it freely across the code.
- ssize_t overall_segment_size: This variable records the total size of allocated memory, helping calculate internal fragmentation within the program.

- **Loader Cleanup:**

The loader_cleanup() function is responsible for proper resource management. It releases memory allocated during program execution. It employs flags to check the validity of ehdr and phdr pointers before freeing them, ensuring that only valid memory is released. In case of errors or NULL pointers, the program reports an error and exits, ensuring proper error handling in this part of the code.

- **Page Allocation:**

The allocate_mem() function plays a critical role in handling segmentation faults. When a segmentation fault occurs, it calculates the required number of pages to allocate memory for the corresponding program segment. It uses the mmap() system call to allocate memory and reads the segment's content from the ELF file. The function also performs error checks during memory allocation, ensuring that the program can recover from allocation failures. The error checks also ensures that program exits if we receive an error ensuring proper error handling.

- **Handling Segmentation Faults:**

  The sigsegv_handler() function is a signal handler for SIGSEGV (segmentation fault). It responds to this signal by identifying the program segment responsible for the fault. It then calls allocate_mem() to allocate memory for that segment, ensuring that the memory gets allocated when a fault is captured and we can proceed with the functionality of the code. The function includes checks for NULL values in ehdr and phdr to ensure their validity. In the event of NULL values, an error is reported, and the program exits to maintain its stability.

- **Loading and Executing ELF:**

  The load_and_run_elf() function is the core of the SimpleSmartLoader. It begins by opening the specified ELF file and proceeds to read and allocate memory for the ELF header and program header table. It registers a SIGSEGV signal handler using sigaction to ensure that segmentation faults are managed and we can re direct to the segfault handling function in case a segmentation fault is registered. The function then attempts to execute the program by invoking the _start function, a standard entry point in ELF executables. It offers comprehensive error handling, providing the return value of the _start function to the user.

- **Main Function:**

  The main() function serves as the entry point of the program. It performs a series of critical checks, including verifying the existence of the input ELF file and ensuring read permissions. If any of these checks fail, the program provides informative error messages and exits the program. Once the initial checks are passed, the function calls load_and_run_elf() to load and execute the ELF file. After execution, it invokes the loader_cleanup() function to release allocated memory and closes the ELF file.
  The main function also provides valuable statistics to the user, including the number of page faults, memory allocations, and the total amount of internal fragmentation. These statistics offer insights into the program's performance and resource management obtaining which was a crucial part of the assignment.

Conclusion:

The SimpleSmartLoader is a well-structured program for loading and executing ELF executable files. It efficiently handles segmentation faults, allocates memory for program segments, and ensures error-free execution. The code includes rigorous error-checking mechanisms at various stages to maintain program reliability and stability.
This detailed implementation report has provided an in-depth examination of the code's key features and functionality, showcasing its effectiveness in loading and executing ELF

files while maintaining strict error handling and resource management.

Private Github Repository Link -
https://github.com/ManitK/CSE231-Assignments