SimpleSmartLoader Assignment

Group Members:
Asa(2022113),
Sargun(2022450)

GitHub Repository: https://github.com/sargun4/OS/tree/main

Project Overview:

SimpleSmartLoader in C. It loads and executes ELF binary files and tracks page faults, memory allocations, and internal fragmentation. It then calls the loader_cleanup function to release allocated memory and resources.

## Data structures:

1. **struct segment_data:**
    - type: An unsigned 32-bit integer representing the type of the ELF segment.
    - address: An Elf32_Addr type representing the starting virtual memory address where the segment should be loaded.
    - size: A size_t variable representing the size of the segment.
    - offset: An off_t type variable representing the offset of the segment in the ELF file.

This structure is used to store information about each segment in the ELF file, such as its type, memory address, size, and offset in the file.

2. **struct virtual_memory:**
    - address: A pointer to void (void*) representing the starting virtual memory address of an allocated memory region.
    - size: A size_t variable representing the total size of the allocated memory region.
    - loaded_size: A size_t variable representing the amount of loaded content within the allocated memory region.
    - fully_loaded: An integer (0 or 1) indicating whether the memory region is fully loaded.

This structure is used to manage the virtual memory regions allocated for different segments, track the loaded content, and determine if a region is fully loaded.

## Variables:

- `pagesize`: An integer variable representing the system's page size, set to 4096 by default.

- `fd`: A file descriptor used for opening and reading ELF files.

- `ehdr`: A pointer to the ELF header structure (`Elf32_Ehdr`) for storing information about the ELF file.

- `phdr`: A pointer to an array of program headers (`Elf32_Phdr`) that describe various segments in the ELF file.

- `segment_data`: A pointer to an array of custom structures (`struct segment_data`) that stores additional information about each segment.

- `allocated_memory`: A pointer to an array of custom structures (`struct virtual_memory`) that tracks allocated memory regions for each segment.

- `allocated_memory_size`: An integer representing the size of the `allocated_memory` array.

- `total_page_faults`: An integer to count the total number of page faults that occur during execution.

- `total_page_allocations`: An integer to count the total number of page allocations.

- `total_internal_fragmentation`: A size_t variable to store the total amount of internal fragmentation.

## Functions:

- `loader_cleanup()`: A function for releasing memory and performing cleanup. It unmaps allocated memory regions and frees dynamically allocated memory (e.g., `allocated_memory`, `ehdr`, `phdr`).

- `sigsegv_handler(int signum, siginfo_t *info, void *context)`: This function is a signal handler for SIGSEGV (segmentation fault) signals. It is called when a page fault occurs during program execution. The handler attempts to allocate memory for the missing page, load the missing segment content, and update relevant statistics.

- `load_and_run_elf(char **exe)`: This function is the main part of the loader. It opens an ELF file specified in `exe`, reads the ELF header and program headers, validates the ELF file, sets up the SIGSEGV signal handler, and attempts to execute the `_start` function from the ELF file. After execution, it prints various statistics, such as the return value of `_start`, total page faults, total page allocations, and internal fragmentation.

- `main(int argc, char const *argv[])`: The entry point of the program, but it doesn't contain any functionality in the provided code. It simply returns 0, indicating a successful program termination.

Contributions:
Asa:

Sargun:

To run the program:

1. Firstly go to loader directory by using- **cd loader**
2. Compile your loader by running the **make** command
3. Return to the "assignment-4" directory by using: **cd..**
4. THen, Navigate to the "test" directory: **cd test**
5. Finally, Run the "fib" program with your SimpleSmartLoader by using: **../bin/launch fib**
6. And similarly for “sum”:   **../bin/launch sum**

Conclusion
This code serves as an ELF loader that loads ELF files into memory, executes them, and monitors memory allocation and page faults while showing statistics-Total number of page faults,total number of page allocations,total amount of internal fragmentation.And finally printing the user returned value of the executable