Simple Smart Loader

This code is an implementation of a smart simple loader for ELF files, designed to read, load, and execute an ELF binary while handling page faults that occur when accessing unloaded segments. It simulates demand paging by loading segments from the ELF file into memory only when needed, based on page faults. Below are its details:

Global Variables and Constants

- SizeofPage: Defines the size of a page in memory, typically 4096 bytes.
- faults, allocations, intFragmentation: Counters to track page faults, page allocations, and internal fragmentation (unused memory within allocated pages).
- elfhdr and phdr: Pointers to the ELF header and program headers.
- fd: File descriptor for the ELF file.
- EntryPoint: The program's entry point (the address where execution begins).

---

1. loader_cleanup()

- Frees any allocated memory for elfhdr and phdr.
- Closes the ELF file if it’s open.
- This is a cleanup function called at the end of execution or upon errors to release resources.

2. open_elf(const char *file_path)

- Opens the ELF file in read-only mode.
- Returns the file descriptor (fd) or -1 on failure.

3. load_and_read_elfhdr()

- Allocates memory for elfhdr and reads the ELF header from the file.
- If successful, elfhdr will hold metadata about the ELF binary (such as entry point and program header offset).
- Returns 0 on success or -1 on failure.

4. load_and_read_phdr()

- Allocates memory for phdr based on the number of program headers (e_phnum in elfhdr).
- Seeks to the program headers in the ELF file and reads them into phdr.

- Returns 0 on success or -1 on failure.

5. getEntryPoint()

- Returns the program's entry point address from the ELF header.

6. find_segment(void *addr)

- Searches for the program segment that contains a given virtual address (addr).
- Returns a pointer to the Program_Header structure for that segment if found, or NULL otherwise.

7. getPages(Program_Header *segment)

- Calculates the total number of pages required to map the memory region defined by the segment.
- Uses p_vaddr (start address) and p_memsz (memory size) to compute the number of pages required.

8. getFrag(Program_Header *segment, void *addr)

- Computes the amount of internal fragmentation for a given segment and page address.
- Determines how much of the last page or first page of the segment is unused due to alignment constraints.

9. SIGSEGV_handler(int sig, siginfo_t *info, void *context)

- Signal handler for segmentation faults (SIGSEGV).
- When a page fault occurs:
    - It identifies the page-aligned address of the fault and finds the segment containing this address.
    - Allocates and maps a new page at the faulting address.
    - Reads data from the ELF file to this page up to the segment's file size (p_filesz).
    - Fills any remaining bytes in the page with zeros if less than a full page was loaded.
    - Sets memory permissions based on segment flags (read, write, execute).
- Tracks page allocations, page faults, and calculates internal fragmentation.

10. setup_sigsegv_handler()

- Sets up the SIGSEGV (segmentation fault) handler using sigaction.
- Ensures that SIGSEGV_handler is called whenever a page fault occurs during program execution.

11. load_and_run_elf(const char *filename)

- This function is the main entry point for loading and running the ELF file:
    - Sets up the SIGSEGV handler to handle page faults.
    - Opens the ELF file, loads the ELF header, and loads the program headers.
    - Verifies that the file is in ELF format by comparing its magic number (e_ident).
    - Retrieves the entry point address from getEntryPoint().
    - Begins program execution at the entry point using a function pointer (start_func).
    - Prints statistics after execution, including page faults, allocations, internal fragmentation, and the program's return value.
    - Cleans up resources.

12. main(int argc, char **argv)

- Entry point for the loader program.
- Checks if the ELF file path is provided as a command-line argument.
- Calls load_and_run_elf with the provided ELF file path.
- Returns an error message if no ELF file is provided.

Code Flow

1. **Program Initialization**:
    - main verifies the command-line argument for the ELF file and calls load_and_run_elf.
2. **ELF File Handling**:
    - open_elf opens the ELF file in read-only mode.
    - load_and_read_elfhdr reads the ELF header into memory.
    - load_and_read_phdr reads the program headers into memory.
3. **Entry Point Setup**:
    - getEntryPoint retrieves the entry point address for the program.
    - setup_sigsegv_handler installs the SIGSEGV handler (SIGSEGV_handler) to handle page faults.
4. **Starting Execution**:
    - load_and_run_elf sets a function pointer to the entry point address and calls it to start execution.
    - Program execution is transferred to the loaded ELF file's entry point.
5. **Page Fault Handling**:
    - When an instruction accesses an unmapped segment, a SIGSEGV signal triggers the SIGSEGV_handler function.
    - The loader checks if the faulting address falls within a loadable segment.
    - It allocates and maps a new page, loads the relevant data from the ELF file, and sets the page’s permissions.

    - If a page does not need to load the full 4096 bytes, getFrag computes the internal fragmentation for that page.
6. **Program Termination**:
    - After execution completes, load_and_run_elf displays statistics on page faults, page allocations, internal fragmentation, and the program's return value.
    - loader_cleanup releases resources by freeing memory and closing the ELF file.

Contribution:

1. Abhinav Kashyap ,2023022: Implementation of function to load and read program headers, loader cleanup, mapping and unmapping and related errors, function to find the entry point, function to load and run the ELF file, and debugging, segmentation handler, getPages, find_segment.

2. Harsh Sharma , 2023233: Implementation of function to load and read ELF headers, file handling of ELF files and related errors, function to find the entry point, function to load and run the ELF file, and debugging, segmentation handler, getFrag, setup_sigsegv_handler.

GitHub Repository Link: https://github.com/Abhinav0821/SimpleSmartLoader