OS Assignment 4

Group Details

Group - 103 **Members** : { Arjun Tandon <2022095>, Chahat Ahuja <2022138> }
https://github.com/arjun22095/os-assignment/tree/main/group103-smartloader-with-bonus

Implementation

1. The program first opens the `ELF executable` in `O_RDONLY` mode
2. It then initialises the global variables required
3. It loads the elf header in the global `ehdr` variable
4. It checks whether the ELF file is valid or not
5. Then we get the entry point of the executable from the ELF header and typecast it into a function pointer called `_start`
6. We set up a signal handler for `SIGSEGV` before we execute the `_start` function
7. The signal handler (`segfault_handler`) is the star of the show here
    1. First it increases the `PAGE_FAULT_COUNT` by one
    2. It extracts the memory address access that caused the SEGFAULT via the `siginfo_t *info`
    3. It uses that to calculate the page that must be allocated
    4. Once the page has been allocated via `mmap` using `MAP_FIXED`
    5. The page address which is of type `void *` is stored in a node in a specialised linked list so that we can use `munmap` to unmap all the pages post-execution for cleanup purposes.
    6. We iterate through the program headers in the hope that they will tell us the segment that should be written into this newly mmap-ed location via the `read()` function
    7. Once the header has been found, we check the corresponding segment and write it into this new page via the `read()` function
    8. If a header isn't found, it means that there is nothing to write on this page from the executable contents, the executable just needed that page to be allocated so that it can use it.
    9. The handler then returns to the program counter which must've caused the `SIGSEGV` and since it finds that the page has been allocated and any segments that were required to be read have been read, it faces no issue until another `SIGSEGV` (if any at all) happens, and then the handler handles that too.