RV College of Engineering®

Experiential Learning Phase - I :
CS235AI
Operating Systems

Shreya Chakote

Vanshika Khandelwal

1RV22CS189

1RV22CS224

Go, change the world®

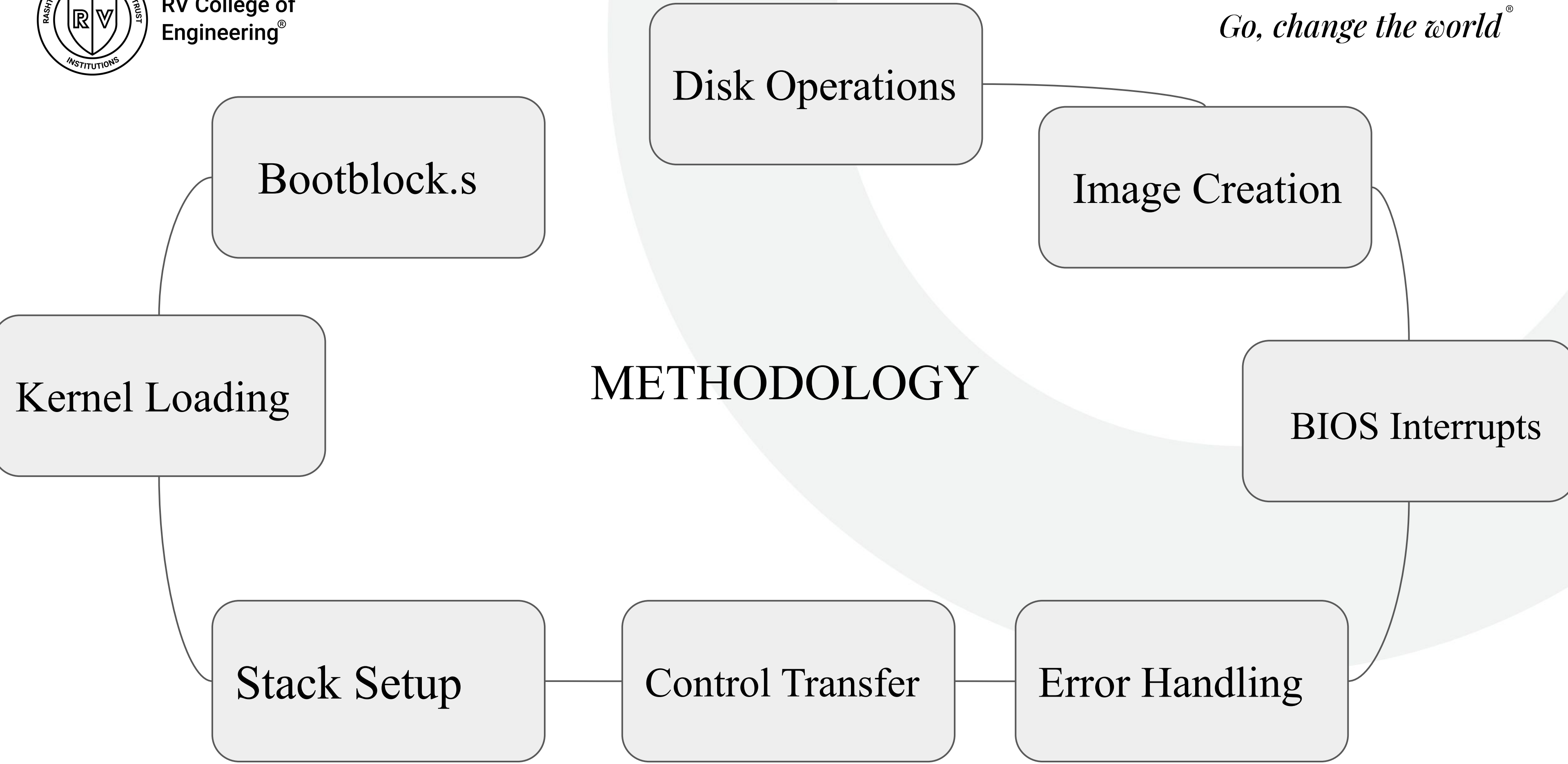
Problem Statement

Design and implement a custom bootloader for x86 architecture capable of efficiently loading an operating system kernel into memory and transferring control to it upon system startup. The bootloader should adhere to industry standards, support error handling, and provide extensibility for future enhancements.

Relevance of the project to the course

Additionally, the bootloader should be thoroughly tested for compatibility with various x86-based systems and demonstrate robustness in diverse boot scenarios. The project aims to deepen understanding of low-level system programming, boot processes, and hardware interaction while fostering practical skills in bootloader development.

METHODOLOGY



Bootblock.s

Disk Operations

Image Creation

BIOS Interrupts

Error Handling

Control Transfer

Stack Setup

Kernel Loading

STEPS FOR EXECUTION

To execute the code:

1. Save the bootloader assembly code in a file named bootloader.s.
2. Save the createimage C code in a file named createimage.c.
3. Open a terminal or command prompt.
4. Navigate to the directory containing the files.
5. Compile the bootloader assembly code using an assembler like NASM:

```
nasm -f bin bootloader.s -o bootloader.bin
```

6. Compile the createimage C code using a C compiler like GCC:

```
gcc createimage.c -o createimage
```

7. Run the createimage tool to generate the OS image file:

```
./createimage bootloader.bin kernel.bin os_image.img
```

   Replace kernel.bin with the filename of your kernel binary if it's different.

8. The os_image.img file will be generated, containing the bootloader and kernel binaries.
9. You can then test the image file using an emulator like QEMU or by writing it to a bootable device such as a USB flash drive.

bootloader.asm

```
assembly
section .text
    global _start

_start:
    ; Set up disk segment
    mov ax, 0x0000      ; Load 0x0000 into AX register
    mov ds, ax          ; Set data segment (DS) to 0x0000 for disk access

    ; Set up register for reading kernel from disk
    mov ah, 0x02        ; BIOS function to read sectors
    mov al, 0x01        ; Number of sectors to read
    mov ch, 0x00        ; Cylinder number
mov dh, 0x00        ; Head number
mov cl, 0x02        ; Sector number (start at 2, assuming kernel starts at sector 2)
mov bx, buffer      ; Buffer address to load kernel into
int 0x13            ; BIOS interrupt call to read disk sectors

; Check for error
jc disk_error       ; Jump to error handling if carry flag set

; Set up stack for kernel
mov ax, 0x1000      ; Kernel start address
mov ss, ax          ; Set stack segment
mov sp, 0xFFFF      ; Set stack pointer
```

```
 ; Jump to kernel start address
     jmp 0x1000:0000

disk_error:
    ; Handle disk read error
    ; can implement error handling here, like displaying an error
       message and halting the system
    ; For simplicity, let's just loop indefinitely
    cli                  ; Disable interrupts
    hlt                  ; Halt processor

section .bss
    ; Define buffer for loading kernel
    buffer: resb 512    ; Reserve 512 bytes for buffer
```

createimage.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <elf.h>

#define SECTOR_SIZE 512

// Function to read ELF headers and extract bootloader and kernel info
void read_elf_headers(FILE *bootfile, FILE *kernelfile, Elf32_Ehdr
*boot_header, Elf32_Ehdr *kernel_header) {
    // Read ELF headers
    if (fread(boot_header, sizeof(Elf32_Ehdr), 1, bootfile) != 1) {
        fprintf(stderr, "Error: Unable to read bootloader ELF header\n");
exit(EXIT_FAILURE);
    }

    if (fread(kernel_header, sizeof(Elf32_Ehdr), 1, kernelfile) != 1) {
        fprintf(stderr, "Error: Unable to read kernel ELF header\n");
        exit(EXIT_FAILURE);
    }
}

// Function to write bootloader and kernel to image file
void write_bootloader_kernel(FILE *image, FILE *bootfile, FILE *kernelfile,
Elf32_Ehdr *boot_header, Elf32_Ehdr *kernel_header) {
    // Write bootloader to image
```

```
fseek(image, 0, SEEK_SET);
    if (fread(image, SECTOR_SIZE, 1, bootfile) != 1) {
        fprintf(stderr, "Error: Unable to write bootloader to image\n");
        exit(EXIT_FAILURE);
    }

    // Write kernel to image
    fseek(image, SECTOR_SIZE, SEEK_SET);
    if (fread(image, SECTOR_SIZE, kernel_header->e_entry - SECTOR_SIZE,
kernelfile) != kernel_header->e_entry - SECTOR_SIZE) {
        fprintf(stderr, "Error: Unable to write kernel to image\n");
        exit(EXIT_FAILURE);
    }
}
```

```
// Main function
int main(int argc, char *argv[]) {
    FILE *bootfile, *kernelfile, *image;
    Elf32_Ehdr boot_header, kernel_header;

     if (argc != 4) {
        fprintf(stderr, "Usage: %s <bootloader> <kernel> <image>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    bootfile = fopen(argv[1], "rb");
    kernelfile = fopen(argv[2], "rb");
    image = fopen(argv[3], "wb");
```

```
  if (!bootfile || !kernelfile || !image) {
      fprintf(stderr, "Error: Unable to open files\n");
      exit(EXIT_FAILURE);
   }
   read_elf_headers(bootfile, kernelfile, &boot_header, &kernel_header);

   write_bootloader_kernel(image, bootfile, kernelfile, &boot_header,
&kernel_header)

   fclose(bootfile);
   fclose(kernelfile);
   fclose(image);
   return 0;
}
```

REFERENCES

1.Journal Paper: 1Alycia Sebastian and 2Dr. K. Siva Sankar, "Design of a Dynamic Boot Loader for Loading an Operating System ", Journal of Computer Science . **: 26-08-2018**

2.Jebarajan, T. and K.S. Sankar, 2011. A method for developing an operating system for plug and play bootstrap loader for USB drive. Int. J. Comput. Sci., 8: 295-301.

3.Kanahasabapathy, S.S., J. Thanganadar and P. Lekshmikanthan, 2015. A novel approach to develop dynamic portable instructional operating system for minimal utilization. Int. Arab J. Inform. Technol., 12: 780-784.

4.Karna, A.K. and Y. Chen, 2014. Multi-operative USB HD: An all-in-one solution to IT supports and forensic experts. J. Software, 9: 847-858. DOI: 10.4304/jsw.9.4.847-858

5.Karna, A.K., 2010. Multipurpose USB hard disk: Your mini laptop. Proceedings of the International Conference on Information Technology and Computer Science, pp: 357-360. DOI: 10.1109/ITCS.2010.93

6.Sebastian, A. and K.S. Sankar, 2015. Design of a boot loader for operating system. Austral. J. Basic Applied Sci., 9: 368-374.

7.Sankar, K.S., 2010. A method for developing bootstrap loader and configuring network for live USB flash drive. Int. J. Decis. Mak. Supply Chain Logist., 1: 221-232.