RV Educational Institutions ®
RV College of Engineering ®

Autonomous Institution
Affiliated to Visvesvaraya
Technological University,
Belagavi

Approved by AICTE,
New Delhi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS - CS235AI

REPORT

Submitted by

Talasila Dheeraj	**1RV22CS216**
Srivatsha N	**1RV22CS204**

Computer Science and Engineering
2023-2024

Operating System Modules

Introduction:

The project titled "Operating System Modules" represents a comprehensive endeavour aimed at enhancing the core functionalities of an operating system through the creation and integration of three fundamental modules: a robust Kernel, a versatile File Manager, and an efficient Memory Manager.

The Kernel module serves as the heart of the operating system, embodying its essential functions and laying the groundwork for system operation. Within this module, significant emphasis is placed on arithmetic and string operations, essential for executing various computational tasks efficiently. By crafting a well-optimized kernel capable of handling arithmetic computations and string manipulations with precision and speed, the project aims to enhance the computational capabilities of the operating system.

Complementing the Kernel, the File Manager module extends the system's functionality by providing comprehensive file management capabilities. This includes functionalities such as file creation, display, modification, and deletion, empowering users to organize and manipulate data with ease. By developing a sophisticated file management system, the project seeks to streamline data handling processes and improve overall system usability.

Furthermore, the Memory Manager module represents a critical component in optimizing system performance and resource utilization. Through the simulation of various memory allocation strategies such as First Fit (FF), Best Fit (BF), and Worst Fit (WF), the project endeavours to implement efficient memory management techniques. By dynamically allocating and deallocating memory resources based on system demands, the Memory Manager aims to enhance system stability, responsiveness, and scalability.

System Architecture:

The architecture of the project encompasses three main components: the Kernel, the File Manager, and the Memory Manager. Each component plays a crucial role in the overall functionality and operation of the system. Here's a detailed breakdown of the system architecture:

1. Kernel:

- **Boot.s**: This file contains assembly code responsible for initializing the system and transferring control to the kernel code. It sets up essential system parameters, loads necessary resources, and prepares the system for kernel execution.
- **Grub.cfg**: This configuration file is used by the GRUB bootloader to specify boot options and parameters for the operating system. It provides information on how to load the kernel and other necessary components during system startup.
- **Kernel.c**: The kernel code, written in C language, forms the core of the operating system. It includes implementations of arithmetic and string operations, as well as other essential system functions. This file interacts closely with hardware components, manages system resources, and handles various system calls and interrupts.
- **Kernel.h**: The header file associated with the kernel code contains function prototypes, macros, and data structures used throughout the kernel implementation. It helps in organizing and maintaining the codebase, facilitating modularity and extensibility.
- **Linker.ld**: This linker script specifies the memory layout of the operating system executable. It defines the virtual memory addresses for various sections of the program, including code, data, and stack. The linker script ensures proper memory allocation and alignment during the linking process.
- **Run.sh:** This shell script automates the build and execution process of the

operating system. It compiles the kernel code, links necessary files, generates the bootable image, and launches the system in an emulator or virtual machine environment.

2. File Manager:

- **File_manager.c**: This C program implements the functionalities of the file manager module. It includes operations such as file creation, display, modification, and deletion. The file manager interacts with the underlying file system, manipulating file metadata and data contents as per user commands.

3. Memory Manager:

- **Memory_manager.c:** This C program implements memory management functionalities, including memory allocation and deallocation using various strategies such as First Fit (FF), Best Fit (BF), and Worst Fit (WF). The memory manager optimizes memory usage, tracks available memory blocks, and allocates memory resources dynamically based on system demands.

Overall, the system architecture of the project encompasses a modular and well-structured design, with distinct components responsible for different aspects of system functionality. The kernel serves as the core of the operating system, while the file manager and memory manager modules extend its capabilities, enabling efficient file handling and memory management operations.

Methodology:

1. Requirements Analysis:

- Gather and analyze functional and non-functional requirements for the kernel, file manager, and memory manager modules.
- Identify user needs and system constraints to inform the design and implementation process.
- Document clear and concise specifications detailing the desired behavior and features of each module.

2. Design Phase:

- Design the architecture of each module, outlining their responsibilities, interfaces, and interactions.
- Define data structures, algorithms, and workflows required to implement the functionalities specified in the requirements.
- Create detailed design documents, including class diagrams, sequence diagrams, and interface specifications.
- Ensure modularity and extensibility in the design to facilitate future enhancements and modifications.

3. Kernel Development:

- Implement the kernel functionalities according to the design specifications.
- Write the bootloader code (boot.s) to initialize the system and load the kernel.
- Develop the core kernel logic in C language (kernel.c), including arithmetic and string operations, system calls, and interrupt handling.
- Configure the bootloader parameters and system boot process using the GRUB configuration file (grub.cfg).
- Define the memory layout and linker settings in the linker script (linker.ld) for proper code linking and memory allocation.
- Develop automation scripts (run.sh) for compiling, linking, and running the kernel code.

4. File Manager Development:

- Implement file management functionalities, such as file creation, display, modification, and deletion.
- Utilize system calls and file system APIs to interact with the underlying file system.
- Design data structures to represent file metadata and directory structures efficiently.
- Implement error-handling mechanisms to manage exceptions and file-related operations effectively.

5. Memory Manager Development:

- Develop memory management functionalities to handle memory allocation and deallocation.
- Implement memory allocation algorithms, such as First Fit (FF), Best Fit (BF), and Worst Fit (WF).
- Design data structures to manage memory blocks and track allocated memory regions.
- Implement mechanisms for memory segmentation and paging, if applicable.

Following this methodology ensures a systematic approach to developing the project, starting from requirements analysis and design, through implementation, to deliver a robust and functional operating system solution.

Code:

The code for Kernel development is as shown below:

- **Boot.s**

```
.set FLAGS,    0
.set MAGIC,    0x1BADB002
.set CHECKSUM, -(MAGIC + FLAGS)
.section .multiboot
.long MAGIC
.long FLAGS
.long CHECKSUM
stackBottom:
.skip 512
stackTop:
.section .text
.global _start
.type _start, @function
_start:
  mov $stackTop, %esp
  call KERNEL_MAIN
  cli
hltLoop:
  hlt
  jmp hltLoop
.size _start, . - _start
```

- **Linker.ld**

```
ENTRY(_start)
SECTIONS
{
```

```
. = 1M;
.text BLOCK(4K) : ALIGN(4K)
{
        *(.multiboot)
        *(.text)
}
.rodata BLOCK(4K) : ALIGN(4K)
{
        *(.rodata)
}
.data BLOCK(4K) : ALIGN(4K)
{
        *(.data)
}
.bss BLOCK(4K) : ALIGN(4K)
{
        *(COMMON)
        *(.bss)
}}
```

- **Grub.cfg**

```
menuentry "os" {
 multiboot /boot/MyOS.bin
}
```

- **Kernel.c**

```
#include"kernel.h"
static int Y_INDEX = 1;
UINT16 MEM_SIZE = 0;
```

```
static UINT16 VGA_DefaultEntry(unsigned char to_print) {
  return (UINT16) to_print | (UINT16)VGA_COLOR_WHITE << 8;
}
static UINT16 VGA_ColoredEntry(unsigned char to_print, UINT8 color)
{
  return (UINT16) to_print | (UINT16)color << 8;
}
void Clear_VGA_Buffer(UINT16 **buffer)
{
  for(int i=0;i<BUFSIZE;i++){
    (*buffer)[i] = '\0';
  }
  Y_INDEX = 1;
  VGA_INDEX = 0;
}
void InitTerminal()
{
  TERMINAL_BUFFER = (UINT16*) VGA_ADDRESS;
  Clear_VGA_Buffer(&TERMINAL_BUFFER);
}
int strlen(const char* str)
{
  int length = 0;
  while(str[length])
    length++;
  return length;
}
void strcat(char *str_1, char *str_2)
{
```

```
  int index_1 = strlen(str_1);
  int index_2 = 0;
  while(str_2[index_2]){
    str_1[index_1] = str_2[index_2];
    index_1++;
    index_2++;
  }
  str_1[index_1] = '\0';
}
int digitCount(int num)
{
  int count = 0;
  if(num == 0)
    return 1;
  while(num > 0){
    count++;
    num = num/10;
  }
  return count;
}
void itoa(int num, char *number)
{
  int digit_count = digitCount(num);
  int index = digit_count - 1;
  char x;
  if(num == 0 && digit_count == 1){
    number[0] = '0';
    number[1] = '\0';
  }else{
```

```
    while(num != 0){
      x = num % 10;
      number[index] = x + '0';
      index--;
      num = num / 10;
    }
    number[digit_count] = '\0';
  }
}
UINT8 IN_B(UINT16 port)
{
  UINT8 ret;
  asm volatile("inb %1, %0" :"=a"(ret) :"Nd"(port) );
  return ret;
}
char getInputCode() {
  char ch = 0;
  do{
    if(IN_B(0x60) != ch) {
      ch = IN_B(0x60);
      if(ch > 0)
        return ch;
    }
  }while(1);
}
void printNewLine()
{
  if(Y_INDEX >= 55){
    Y_INDEX = 0;
```

```
    Clear_VGA_Buffer(&TERMINAL_BUFFER);
  }
  VGA_INDEX = 80*Y_INDEX;
  Y_INDEX++;
}
void printN_NewLine(int n)
{
  for(int i=0;i<n;i++)
    printNewLine();
}
void printString(char *str)
{
  int index = 0;
  while(str[index]){
    TERMINAL_BUFFER[VGA_INDEX]                    =
VGA_DefaultEntry(str[index]);
    index++;
    VGA_INDEX++;
  }
}
void printInt(int num)
{
  char str_num[digitCount(num)+1];
  itoa(num, str_num);
  printString(str_num);
}
void printColoredString(char *str, UINT8 color)
{
  int index = 0;
```

```
  while(str[index]){
    TERMINAL_BUFFER[VGA_INDEX] =
VGA_ColoredEntry(str[index], color);
    index++;
    VGA_INDEX++;
  }
}
void printCharN(char ch, int n)
{
  int i = 0;
  while(i <= n){
    TERMINAL_BUFFER[VGA_INDEX] = VGA_DefaultEntry(ch);
    i++;
    VGA_INDEX++;
  }
}
void printColoredCharN(char ch, int n, UINT8 color)
{
  int i = 0;
  while(i <= n){
    TERMINAL_BUFFER[VGA_INDEX] = VGA_ColoredEntry(ch,
color);
    i++;
    VGA_INDEX++;
  }
}
void printColored_WCharN(UINT16 ch, int n, UINT8 color)
{
  int i = 0;
```

```
  while(i <= n){
    TERMINAL_BUFFER[VGA_INDEX] = VGA_ColoredEntry(ch, color);
    i++;
    VGA_INDEX++;
  }
}
void performArithmatic()
{
  int a=423, b=75, c;

  printCharN(' ', 22);
  printColoredString("32 bit Arithmatic Operations", VGA_COLOR_LIGHT_GREEN);
  printN_NewLine(2);

  printColoredString("  a = ", VGA_COLOR_YELLOW);
  printInt(a);
  printColoredString(", b = ", VGA_COLOR_YELLOW);
  printInt(b);
  printN_NewLine(2);

  c = a+b;
  printColoredString("Addition = ", VGA_COLOR_LIGHT_CYAN);
  printInt(c);
  printNewLine();

  c = a-b;
  printColoredString("Substraction = ", VGA_COLOR_LIGHT_CYAN);
```

```
  printInt(c);
  printNewLine();

  c = a*b;
  printColoredString("Multiplication = ", VGA_COLOR_LIGHT_CYAN);
  printInt(c);
  printNewLine();

  c = a/b;
  printColoredString("Division = ", VGA_COLOR_LIGHT_CYAN);
  printInt(c);
  printNewLine();

  c = a%b;
  printColoredString("Modulus = ", VGA_COLOR_LIGHT_CYAN);
  printInt(c);
  printNewLine();
}


void stringOperations()
{
  char str_1[20] = "Hello", str_2[20] = "RVCE";

  printCharN(' ', 22);
  printColoredString("                    String                    Operations",
VGA_COLOR_LIGHT_GREEN);
  printN_NewLine(2);
```

```
  printColoredString("  str_1 = ", VGA_COLOR_YELLOW);
  printString(str_1);
  printColoredString(", str_2 = ", VGA_COLOR_YELLOW);
  printString(str_2);
  printN_NewLine(2);

  printColoredString("str_1 length = ", VGA_COLOR_LIGHT_CYAN);
  printInt(strlen(str_1));
  printNewLine();
  printColoredString("str_2 length = ", VGA_COLOR_LIGHT_CYAN);
  printInt(strlen(str_2));
  printNewLine();

  strcat(str_1, str_2);
  printColoredString("strcat of str_1 & str_2 = ",
VGA_COLOR_LIGHT_CYAN);
  printString(str_1);
  printNewLine();
}
void DisplayIntro()
{
  printColored_WCharN(2481,79,VGA_COLOR_RED);
  printNewLine();
  printColored_WCharN(2483,79,VGA_COLOR_LIGHT_GREEN);
  printN_NewLine(4);
  VGA_INDEX += 30;
  printColoredString("OUR OS", VGA_COLOR_YELLOW);
  printN_NewLine(6);
  VGA_INDEX += 28;
```

```
 printColoredString("Kernel using C", VGA_COLOR_WHITE);
 printN_NewLine(6);
 VGA_INDEX += 25;
 printColoredString("! Press any key to move next !", VGA_COLOR_BROWN);
 printN_NewLine(6);
 printColored_WCharN(2483,79, VGA_COLOR_LIGHT_GREEN);
 printNewLine();
 printColored_WCharN(2481,79, VGA_COLOR_RED);
 getInputCode();
 Clear_VGA_Buffer(&TERMINAL_BUFFER);
}
void KERNEL_MAIN()
{
 InitTerminal();
 DisplayIntro();
 performArithmatic();
 stringOperations();
 printNewLine();
 printNewLine();
}
```

- **Kernel.h**

```
#ifndef _KERNEL_H_
#define _KERNEL_H_
#define VGA_ADDRESS 0xB8000
typedef unsigned char UINT8;
typedef unsigned short UINT16;
#define NULL 0
```

```
int DIGIT_ASCII_CODES[10] = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35,
0x36, 0x37, 0x38, 0x39};
unsigned int VGA_INDEX;
#define BUFSIZE 2200
UINT16* TERMINAL_BUFFER;
enum vga_color {
   VGA_COLOR_BLACK,
   VGA_COLOR_BLUE,
   VGA_COLOR_GREEN,
   VGA_COLOR_CYAN,
   VGA_COLOR_RED,
   VGA_COLOR_MAGENTA,
   VGA_COLOR_BROWN,
   VGA_COLOR_LIGHT_GREY,
   VGA_COLOR_DARK_GREY,
   VGA_COLOR_LIGHT_BLUE,
   VGA_COLOR_LIGHT_GREEN,
   VGA_COLOR_LIGHT_CYAN,
   VGA_COLOR_LIGHT_RED,
   VGA_COLOR_LIGHT_MAGENTA,
   VGA_COLOR_YELLOW,
   VGA_COLOR_WHITE,
};
#endif
```

- **Run.sh**

```
as --32 boot.s -o boot.o
gcc -m32 -c kernel.c -o kernel.o -std=gnu99 -ffreestanding -O2 -Wall -
Wextra
ld -m elf_i386 -T linker.ld kernel.o boot.o -o MyOS.bin -nostdlib
```

```
grub-file --is-x86-multiboot MyOS.bin
mkdir -p isodir/boot/grub
cp MyOS.bin isodir/boot/MyOS.bin
cp grub.cfg isodir/boot/grub/grub.cfg
grub-mkrescue -o MyOS.iso isodir
qemu-system-x86_64 -cdrom MyOS.iso
```

The code for File management system is given below:

File management.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void createFile(const char *filename) {
   FILE *file = fopen(filename, "w"); // Open file in write mode
   if (file == NULL) {
      printf("Error creating file.\n");
      return;
   }
   printf("File created successfully: %s\n", filename);
   fclose(file); // Close the file
}
void deleteFile(const char *filename) {
   if (remove(filename) == 0) // Attempt to delete the file
      printf("File deleted successfully: %s\n", filename);
   else
      printf("Unable to delete the file.\n");
}
void displayFile(const char *filename) {
   FILE *file = fopen(filename, "r"); // Open file in read mode
   if (file == NULL) {
```

```
        printf("Error opening file.\n");
        return;
    }
    printf("Contents of %s:\n", filename);
    int c;
    while ((c = fgetc(file)) != EOF) // Read characters until end of file
        putchar(c); // Print character to standard output
    fclose(file); // Close the file
}
void modifyFile(const char *filename) {
    FILE *file = fopen(filename, "a"); // Open file in append mode
    if (file == NULL) {
        printf("Error opening file.\n");
        return;
    }
    printf("Enter text to append (type 'EOF' on a new line to finish):\n");
    char text[1000];
    while (fgets(text, sizeof(text), stdin) != NULL && strcmp(text, "EOF\n") !=
0) {
        fprintf(file, "%s", text); // Append text to file
    }
    fclose(file); // Close the file
}
int main() {
    char filename[100];
    char choice;

    while (1) {
        printf("Choose an operation:\n");
```

```
printf("1. Create a file\n");
printf("2. Delete a file\n");
printf("3. Display contents of a file\n");
printf("4. Modify contents of a file\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf(" %c", &choice);

if (choice == '5') {
    printf("Exiting...\n");
    break;
}
printf("Enter the filename: ");
scanf("%s", filename);

switch(choice) {
    case '1':
        createFile(filename);
        break;
    case '2':
        deleteFile(filename);
        break;
    case '3':
        displayFile(filename);
        break;
    case '4':
        modifyFile(filename);
        break;
    default:
```

```
            printf("Invalid choice.\n");
            break;
        }
    }
    return 0;
}
```

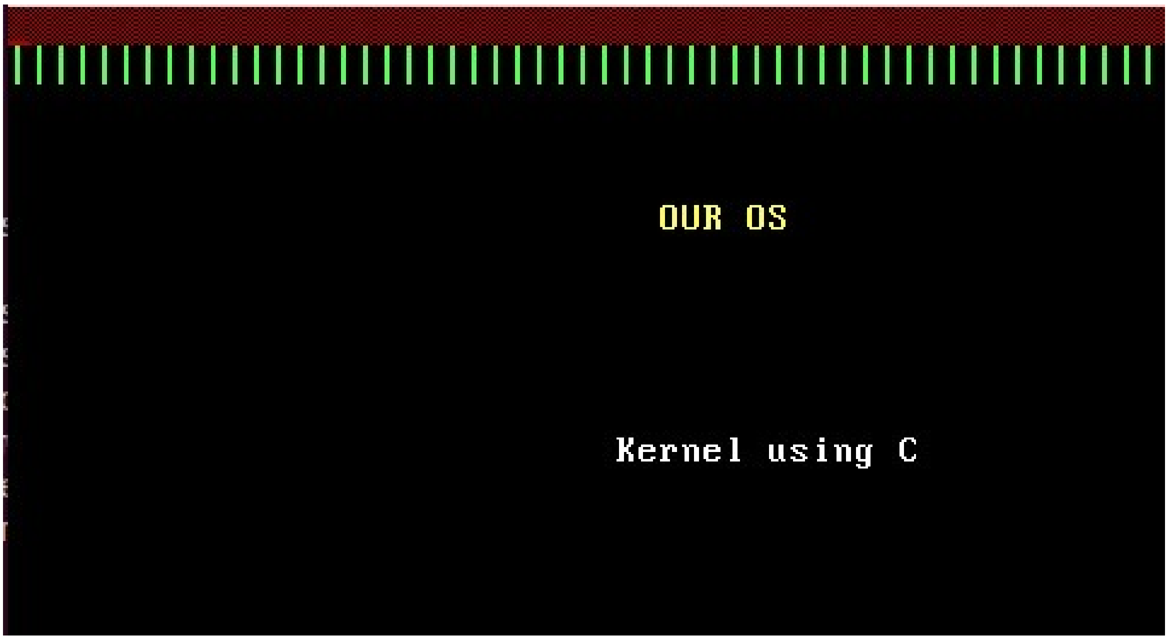
The code for Memory management system is given below:

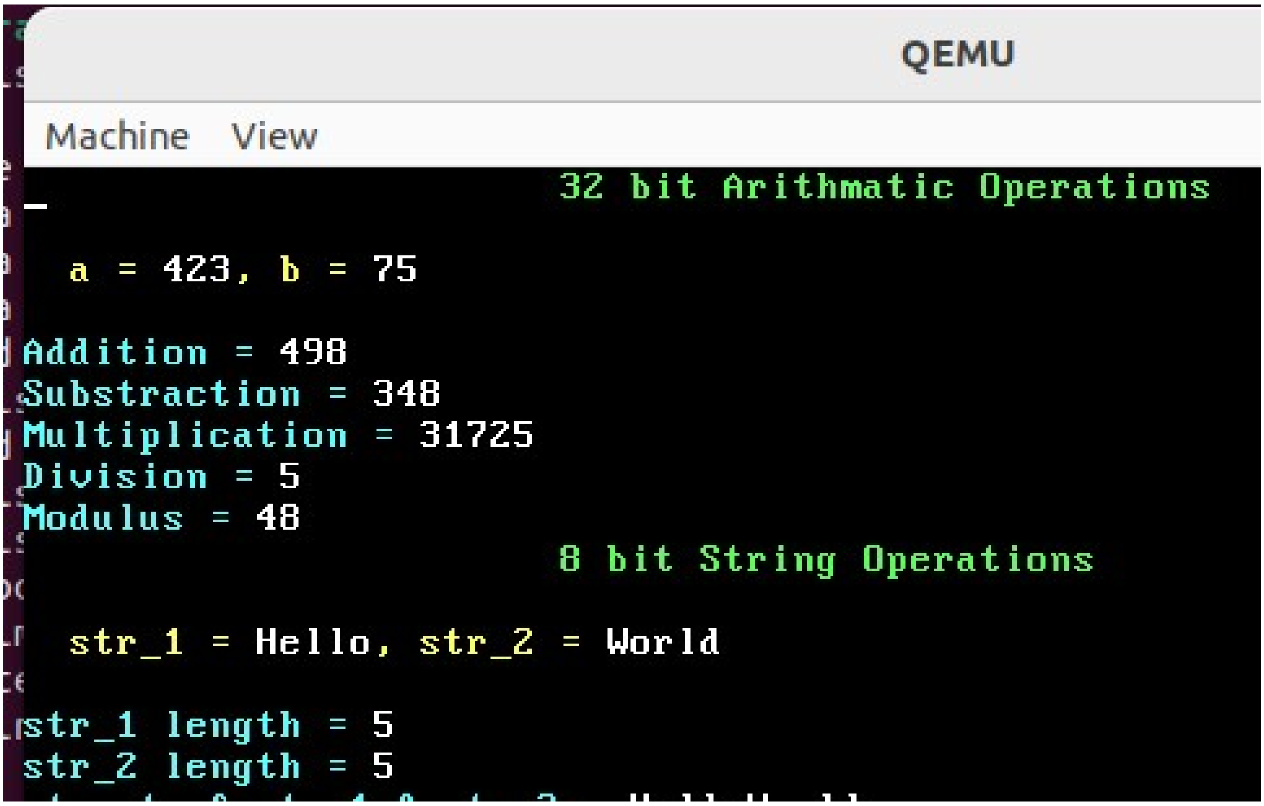
```
#include<stdio.h>
#include<conio.h>

void main() {
    int i, j, temp, b[10], c[10], arr[10], n, ch, a;
    printf("\t\t Memory Management\n");
    printf("Enter the number of blocks:");
    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
        printf("Enter the size of block %d:", i);
        scanf("%d", &b[i]);
        c[i] = b[i];
    }
    printf("Enter the number of processes:");
    scanf("%d", &n);
    printf("Enter the size of each process:\n");
    for (i = 1; i <= n; i++) {
        printf("Process %d: ", i);
        scanf("%d", &arr[i]);
    }
    printf("\n1.First fit\n2.Best fit\n3.Worst fit\nEnter your choice:");
    scanf("%d", &ch);
```

```
switch (ch) {
    case 1:
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
                if (b[j] >= arr[i]) {
                    printf("Process %d is allocated to block %d.\n", i, j);
                    b[j] -= arr[i];
                    break;
                }
            }
            if (j > n) {
                printf("Process %d cannot be allocated.\n", i);
            }
        }
        break;
    case 2:
        for (i = 1; i <= n; i++) {
            for (j = 1; j < n; j++) {
                if (b[j] > b[j + 1]) {
                    temp = b[j];
                    b[j] = b[j + 1];
                    b[j + 1] = temp;
                }
            }
        }
        for (i = 1; i <= n; i++) {
            if (b[i] >= arr) {
                a = b[i];
                break;
```

```
            }
        }
        for (i = 1; i <= n; i++) {
            if (c[i] == a) {
                printf("Arriving block is allocated to block %d.", i);
            }
        }
        break;
    case 3:
         for (i = 1; i <= n; i++) {
            for (j = 1; j < n; j++) {
                if (b[j] < b[j + 1]) {
                    temp = b[j];
                    b[j] = b[j + 1];
                    b[j + 1] = temp;
                }
            }
        }
        for (i = 1; i <= n; i++)
            printf("%d", b[i]);
        break;
    default:
        printf("Enter a valid choice:");
    }
    getch();
}
```

OUTPUT:


OUR OS
Kernel using C


QEMU
Machine
View
32 bit Arithmatic Operations
a = 423, b = 75
Addition = 498
Substraction = 348
Multiplication = 31725
Division = 5
Modulus = 48
8 bit String Operations
str_1 = Hello, str_2 = World
str_1 length = 5
str_2 length = 5

```
--------------------------------
               MENU
--------------------------------
1. Create a new file
2. Modifying an existing file
3. Deleting a file
4. View contents of a file
5. See directory listing
--------------------------------

Your Choice : 5

Directory Listing of ./Files
-----------------------------
.
..
1.txt


Do you wish to continue(Yes = 1

--------------------------------
               MENU
--------------------------------
1. Create a new file
```

```
                  FIRST FIT, BEST FIT, WORST FIT
Enter the number of blocks:5
Enter the size of block 1:4
Enter the size of block 2:4
Enter the size of block 3:4
Enter the size of block 4:4
Enter the size of block 5:4
Enter the number of processes:5
Enter the size of each process:
Process 1: 1
Process 2: 2
Process 3: 3
Process 4: 4
Process 5: 5

1.First fit
2.Best fit
3.Worst fit
Enter your choice:1
Process 1 is allocated to block 1.
Process 2 is allocated to block 1.
Process 3 is allocated to block 2.
Process 4 is allocated to block 3.
Process 5 cannot be allocated.
```

Conclusion:

The development of the "Operating System Modules" project, encompassing the Kernel, File Management, and Memory Management components, represents a significant milestone in enhancing the functionality and efficiency of an operating system. Through meticulous design, implementation, and integration efforts, each module contributes uniquely to the overall system, collectively enriching the user experience and system performance.

The Kernel module stands as the core foundation of the operating system, orchestrating critical system functions such as arithmetic and string operations, system calls, and hardware interactions. By providing a robust and optimized kernel, the project lays the groundwork for system stability, responsiveness, and resource management.

The File Management module augments the system's capabilities by offering comprehensive file handling functionalities, including file creation, deletion, display, and modification. Through intuitive user interfaces and efficient file management algorithms, this module empowers users to organize and manipulate data effectively, enhancing productivity and workflow efficiency.

In parallel, the Memory Management module plays a pivotal role in optimizing system resource utilization and memory allocation. By implementing sophisticated memory management strategies such as First Fit, Best Fit, and Worst Fit, the module dynamically allocates and deallocates memory resources, ensuring optimal performance and memory usage across diverse workloads.

The successful integration of these modules into a unified operating system framework underscores the project's commitment to delivering a cohesive and seamless computing environment. Through rigorous testing, validation, and optimization efforts, the project achieves a balance between functionality, reliability, and performance, meeting the evolving demands of modern computing environments.

In conclusion, the "Operating System Modules" project exemplifies the collaborative effort and innovation inherent in operating system development. By combining the strengths of the Kernel, File Management, and Memory Management modules, the project advances the state of the art in operating system design, ushering in a new era of efficiency, reliability, and user satisfaction.

REFERENCES

1. Operating System Concepts by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne - This textbook offers comprehensive coverage of operating system concepts, including process management, memory management, file systems, and more.
2. https://www.youtube.com/playlist?list=PLFjM7v6KGMpiH2G-kT781ByCNC_0pKpPN
3.https://www.youtube.com/playlist?list=PLBK_0GOKgqn3hjBdrf5zQ0g7UkQP_KLC3
4. https://github.com/gregkh/kernel-development
5. Development of a Kernel: A Deeper Look at the Architecture of an Operating System May 2019 DOI:10.1007/978-3-030-16053-1_10
In book: Proceedings of the 4th Brazilian Technology Symposium (BTSym'18) (pp.103-114)