RV College of Engineering

*Go, change the world*

# IMPLEMENTING KERNEL DEVELOPMENT IN C - USING KEYBOARD ,PING PONG GAME

## INTRODUCTION

**KERNEL -** The kernel serves as the core component of an operating system (OS), acting as the central hub that manages system resources and facilitates communication between hardware and software components. It represents the heart of the operating system, responsible for essential tasks such as process management, memory management, device drivers, and system calls. Essentially, the kernel serves as the intermediary between user applications and the underlying hardware, providing a layer of abstraction that enables programmers to interact with system resources in a standardized and efficient manner. **BOOTLOADER -**A bootloader serves as a critical component of computer systems, acting as the initial program that loads the operating system into memory during the system startup process. Essentially, it bridges the gap between the hardware of the computer and the software, ensuring the smooth transition from the hardware initialization phase to the execution of the operating system. Upon powering on the computer, the bootloader is typically the first code executed, residing in a special location known as the boot sector of the storage device. Its primary function is to locate the operating system kernel on the storage device, load it into memory, and transfer control to the kernel to begin the operating system's execution. Bootloaders come in various forms, ranging from simple programs embedded in read-only memory (ROM) to more sophisticated boot managers capable of managing multiple operating systems or boot configurations. Overall, bootloaders play a crucial role in the bootstrapping process of computer systems, ensuring their proper initialization and enabling the seamless launch of the operating system.

K

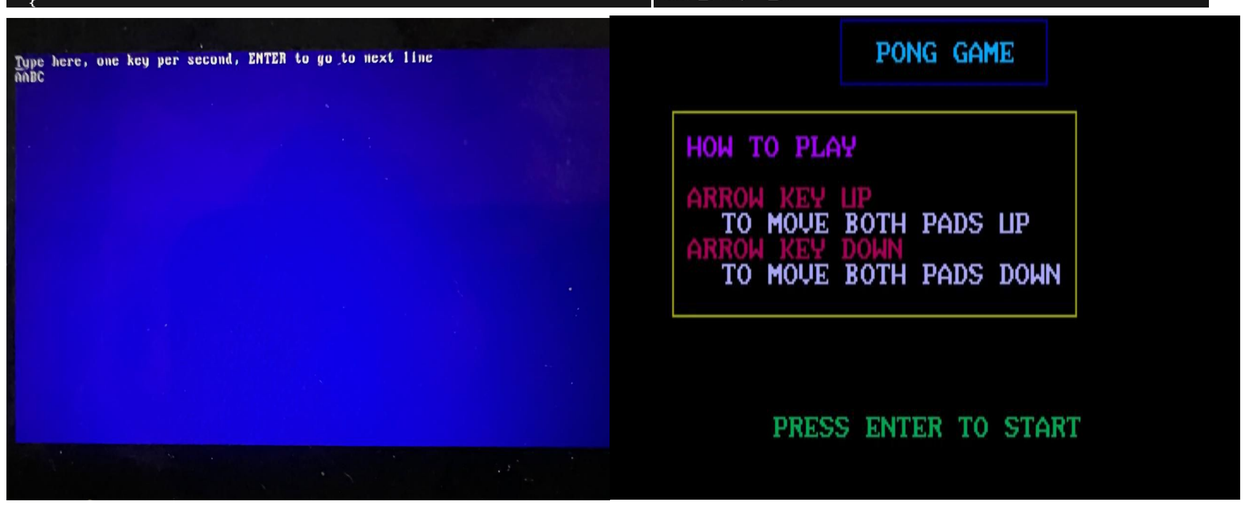
## PROBLEM STATEMENT:

Design and implement a kernel from scratch for an operating system course project, emphasizing core operating system concepts such as process management, memory management, and I/O operations. Integrate keyboard input/output functionality and utilize the VGA graphics array for graphical display. The objective is to develop a fully functional operating system kernel capable of managing processes, handling keyboard interactions, rendering graphics using VGA support, and facilitating real-time gameplay, showcasing proficiency in kernel development and system-level programming.

## MAIN SOURCE CODE FILES:

```
#include "pong.h"

// pads position y, will change on keys
uint16 pad_pos_y = 2;
// score count
uint32 score_count = 0;

// initialize game with into
static void init_game()
{
    uint8 b = 0;

    draw_string(120, 13, BRIGHT_CYAN, "PONG GAME");
    draw_rect(100, 4, 120, 25, BLUE);
    draw_string(10, 50, BRIGHT_MAGENTA, "HOW TO PLAY");
    draw_rect(2, 40, 235, 80, BROWN);
    draw_string(10, 70, BRIGHT_RED, "ARROW KEY UP");
    draw_string(30, 80, WHITE, "TO MOVE BOTH PADS UP");
    draw_string(10, 90, BRIGHT_RED, "ARROW KEY DOWN");
    draw_string(30, 100, WHITE, "TO MOVE BOTH PADS DOWN");
    draw_string(60, 160, BRIGHT_GREEN, "PRESS ENTER TO START");
#ifdef VIRTUALBOX
    sleep(10);
#endif
    while (1)
    {
        b = get_input_keycode();
        sleep(5);
        if (b == KEY_ENTER)
            break;
        b = 0;
    }
    clear_screen();
}

// update score count text
static void update_score_count()
{
    char str[32];

    itoa(score_count, str);
    draw_string(150, 2, WHITE, str);
}

// if lose then display final score & restart game
static void lose()
{
    uint8 b = 0;
    char str[32];

    itoa(score_count, str);
    clear_screen();
    draw_string(120, 15, BRIGHT_GREEN, "NICE PLAY");
    draw_string(125, 45, WHITE, "SCORE");
    draw_string(180, 45, WHITE, str);
    draw_string(45, 130, YELLOW, "PRESS ENTER TO PLAY AGAIN");

#ifdef VIRTUALBOX
    sleep(10);
#endif
    while (1)
    {
```

```
# set flags to 0
.set FLAGS,    0

# set magic number to 0x1BADB002 to identified by bootloader
.set MAGIC,    0x1BADB002

# set the checksum
.set CHECKSUM, -(MAGIC + FLAGS)

# set multiboot enabled
.section .multiboot

# define type to long for each data defined as above
.long MAGIC
.long FLAGS
.long CHECKSUM


# set the stack bottom
stackBottom:

# define the maximum size of stack to 512 bytes
.skip 4096


# set the stack top which grows from higher to lower
stackTop:

.section .text
.global _start
.type _start, @function


_start:


  # assign current stack pointer location to stackTop
        mov $stackTop, %esp

  # call the kernel main function
        call kernel_entry

        cli


# put system in infinite loop
hltLoop:

        hlt
        jmp hltLoop

.size _start, . - _st
```







## MECHANISM & RELEVANCE TO THE COURSE:

**Graphics Rendering with VGA Buffer Concept:**

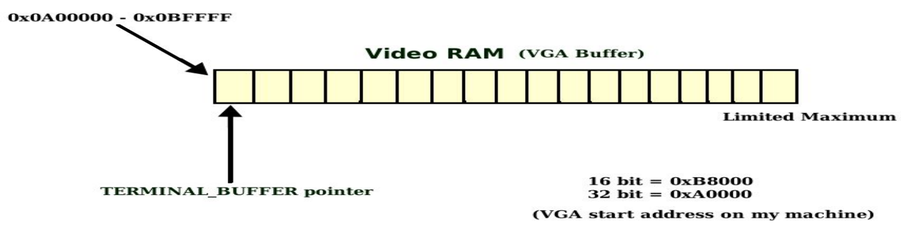
Utilizing the VGA graphics array buffer for rendering graphical interfaces within the kernel environment. Related API/System Calls: open(), read(), and write() ioctl(): Useful for configuring and controlling device- specific settings, such as manipulating the VGA graphics buffer for drawing game elements. signal(): Enables setting up signal handlers to handle asynchronous events, such as keyboard interrupts for processing user inputs during gameplay

**Keyboard Input Handling Concept:**

Managing keyboard input for user interactions and game control within the kernel environment. Related API/System Calls: kbd_init(): Initializes the keyboard device and sets up the necessary data structures and interrupt handlers for keyboard input. kbd_read(): Reads input from the keyboard device, translating scan codes into characters or key codes.

**Kernel Architecture and Design:**

Understanding the structure and organization of the kernel, including process management, memory management, and device drivers. Related API/System Calls: Process Management: fork(), exec(), exit() Memory Management: malloc(), free() Device Drivers: open(), read(), write()




## TOOLS & SYSTEM CALLS:

**GNU/Linux** :- Any distribution(Ubuntu/Debian/RedHat etc.).

**Assembler** :- GNU Assembler(gas) to assemble the assembly language file.

**GCC :-** GNU Compiler Collection, C compiler. Any version 4, 5, 6, 7, 8 etc.

**grub-mkrescue** :- Make a GRUB rescue image, this package internally calls the xorriso functionality to build an iso image.

**QEMU :- Quick EMUlator** to boot our kernel in virtual machine without rebooting the main system.

**Executable Kernel Image ISO:** The final output of the kernel build process is an executable kernel image Header Files: Header files (.h) contain declarations

**Linker Script: A linker script (.ld file) i**s used to specify the layout of the kernel image in memory, Iso Image Generation and Execution

## CONCLUSION

In conclusion, the process of creating a kernel that incorporates keyboard functionality and a ping pong game involves a meticulous and iterative approach. By understanding the intricacies of keyboard hardware and implementing efficient input handling mechanisms, we can enable users to interact with the system effectively. Additionally, integrating a fun and interactive game like ping pong showcases the versatility and entertainment potential of the kernel.Throughout the development process, attention to detail, rigorous testing, and optimization are essential to ensure smooth operation and responsiveness. Moreover, documenting the implementation details and maintaining the codebase facilitate future enhancements and troubleshooting.\Ultimately, the creation of such a kernel not only demonstrates technical proficiency but also fosters creativity and innovation in the realm of operating system development. It provides a solid foundation for further exploration and expansion of functionality, paving the way for the development of more sophisticated operating systems and interactive user experiences.


## Acknowledgements

**The authors thank Prof. Jyothi Shetty Dept. of CSE, RVCE, RVCE for the kind support received for completion of the project.**



**TEAM MEMBERS: Yash Lohia (1RV22CS237) , Siri A Bhat (1RV22CS197)**