RV College of Engineering

Go, change the world

Implementation Of Kernel Integrated VGA Array & I/O Controls Ping Pong Real Time Game

INTRODUCTION:

The Linux kernel is like the boss behind the scenes of your computer, making sure everything works together. It's usually pretty complicated, but this project is like building a fun mini-game inside the boss's office! We'll make a special version of the Linux kernel that lets you play a simple game of ping pong, all within the system itself. This will help us learn more about how the kernel works in a fun and interesting way.

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 displayThe 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.

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) is used to specify the layout of the kernel image in memory, Iso Image Generation and Execution

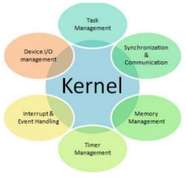
MECHANISM & RELEVANCE TO THE COURSE:

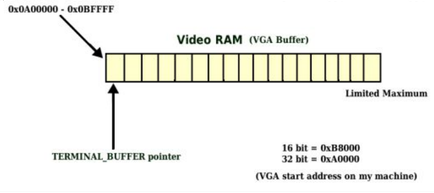
Graphics Rendering with VGA Buffer Concept: Utilizing the VGA graphics array buffer for rendering graphical interfaces within the kernel environment. Related API/System Calls: open(), read(), and write() ioctl(): Useful for configuring and controlling device-specific settings, such as manipulating the 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.

Game Logic and Event Handling Concept: Implementing the core game logic and event handling mechanisms within the kernel environment. Related API/System Calls: event_init(): Initializes the game event system, setting up event queues and data structures for managing game events. event_queue(): Adds a new event to the game event queue, such as paddle movement or ball collision.

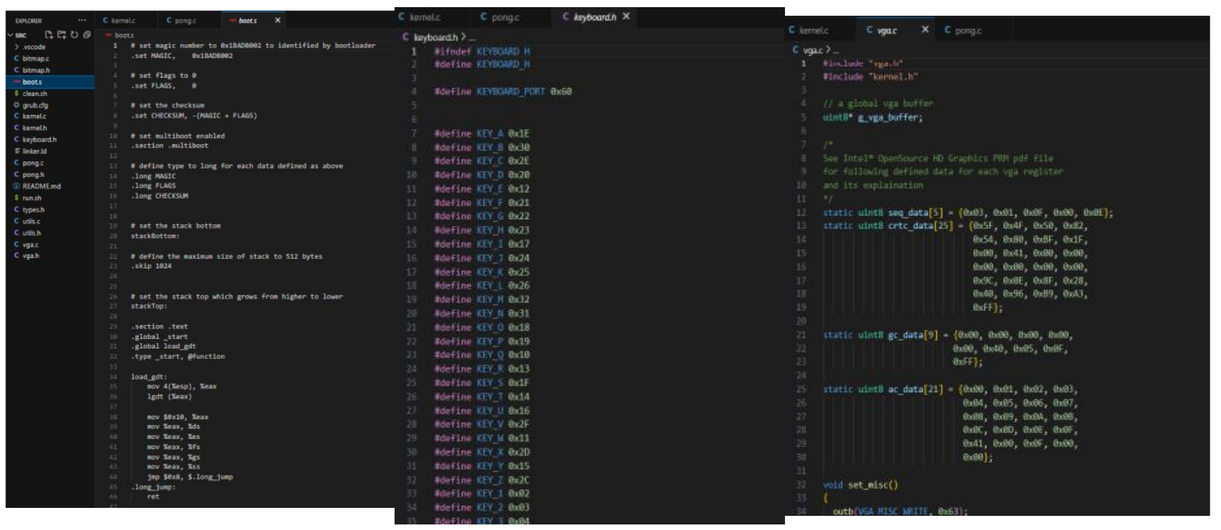
Kernel Architecture and Design Understanding the structure and organization of the kernel, including process management, memory management, and device drivers. Related API/System Calls: Process Management: fork(), exec(), exit() Memory Management: malloc(), free() Device Drivers: open(), read(), write()

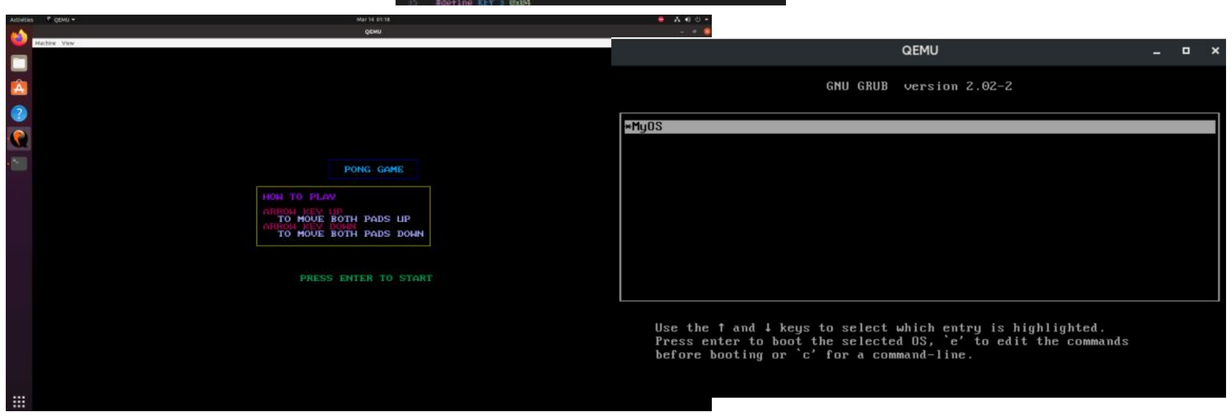




MAIN SOURCE CODE FILES:

We have around 15 files out of which the main files are: 1. boot.S 2. kernel.c 3. pong.c 4. linker.ld 5. vga.c 6. Keyboard.h





CONCLUSION

The kernel integrates the VGA graphics and keyboard input/output functionalities with the game logic module.communicates game state changes, such as paddle movement or ball collisions, to the VGA graphics for rendering on the screen. Similarly, user input captured from the keyboard is relayed to the game logic module to control game elements and progress the gameplay. The kernel structure serves as the foundation of the operating system, providing essential functionalities such as process management, memory management, and input/output handling. It acts as the intermediary between hardware and software, managing system resources and facilitating communication between user programs and hardware devices. Thestructure is modular, with distinct components responsible for specific tasks, ensuring efficient operation and scalability of the O.S.

Acknowledgements

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

Student Information: SNEHIL VUKKUSILA (1RV22CS241), PRANAV DARSHAN (1RV22CS143)