RV College of Engineering

Go, change the world

IMPLEMENTING KERNEL DEVELOPMENT IN C

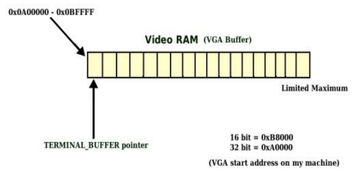
NTRODUCTION:

Operating system (OS) development is a complex yet foundational aspect of computer science. At its core lies the kernel, the engine driving hardware interaction and user experience. This report delves into the process of developing a basic kernel, starting with bootloader creation in 16-bit assembly and extending to keyboard input handling and display functionality.
Beginning with bootloader development, we explore the critical role of initializing the system before OS execution, spotlighting bootloaders like GNU GRUB. The project further examines keyboard input handling, showcasing port I/O operations to capture user keystrokes.
In summary, it provides a concise overview of kernel development, highlighting its pivotal role in system operation and user interaction.
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 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.

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:

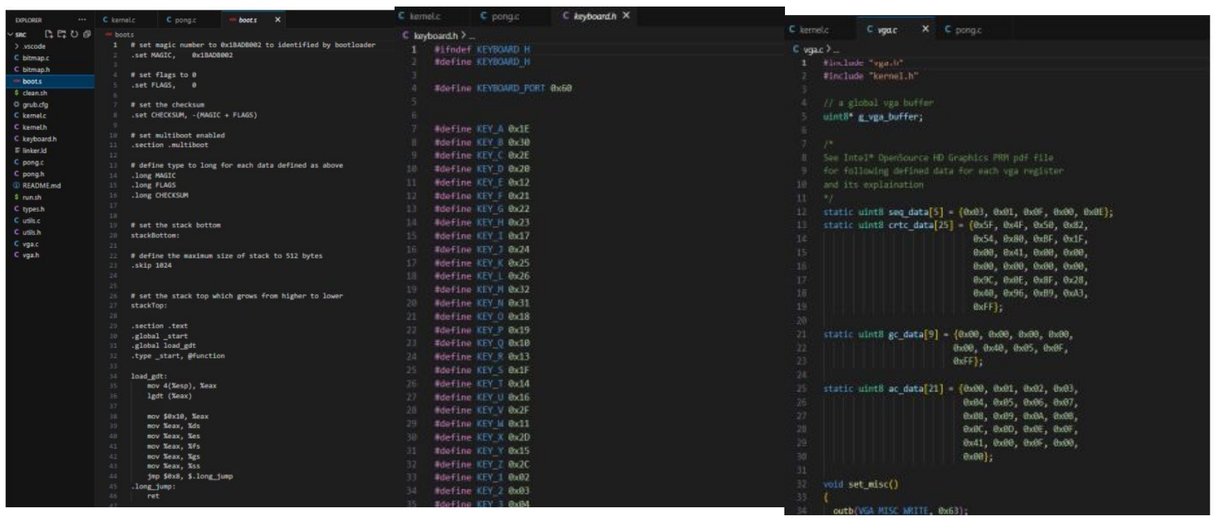
Managing keyboard input for user interactions and game control within the kernel environment. Related API/System Calls: kbd_init(): Initializes the keyboard device and sets up the necessary data structures and interrupt handlers for keyboard input. kbd_read(): Reads input from the keyboard device, translating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

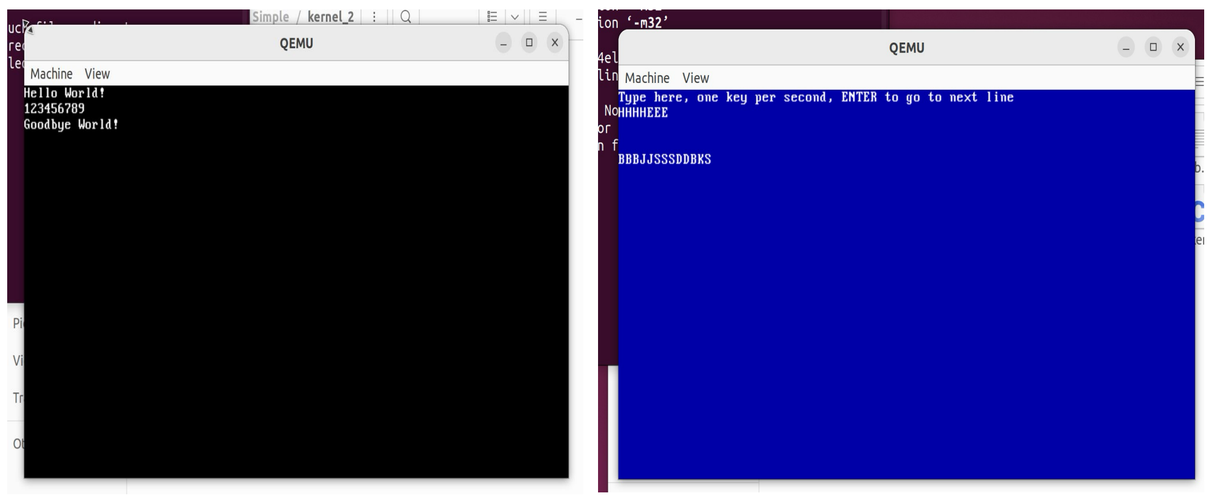
Kernel Architecture and Design:

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

Printing "Hello, World!" may seem trivial, but it symbolizes the initiation of communication between the kernel and the user. It establishes a foundational link between the underlying system and the outside world, showcasing the kernel's ability to convey information to the user space.

Integrating keyboard input handling elevates the kernel's functionality, enabling interaction beyond passive output. The ability to capture and process keyboard events lays the groundwork for building more complex user interfaces and command-line interfaces, essential for user interaction in any operating system.

Moreover, the process of creating a kernel fosters problem-solving skills, as developers encounter numerous challenges along the way. From memory allocation issues to synchronization problems, each hurdle presents an opportunity for learning and improvement.

In conclusion, creating a custom kernel and implementing basic functionalities like printing "Hello, World!" and handling keyboard input is not just an academic exercise; it's a journey of exploration and discovery. It equips developers with invaluable knowledge and skills, paving the way for deeper insights into operating system internals and laying the foundation for more ambitious projects in system software development."

Acknowledgements

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

TEAM MEMBERS: SRAVANI H (1RV22CS201),
SYED FARHAN ASHRAF (1RV22CS214)