# Loading Operating System On RISC-V Simulator

Iman Rasouli Parto
Edris Nasihatkon

# Outline

- Introduction to the Linux Kernel
- Using Buildroot for Embedded Systems
- Simulating RISC-V with the Spike Simulator
- Results and Conclusions

# The Linux Kernel: Connecting Software to Hardware

- The Core Interface Between OS and Hardware
- Enabling Seamless Connection
- Core component of the OS
- Open-source, Customizable
- Key Functions

# Key Functions of the Linux Kernel

- ✓ Process Scheduling
- ✓ Multitasking
- ✓ Interrupt Handling
- ✓ Device Drivers
- ✓ Memory Management
- ✓ System Calls
- ✓ Power Management

# Process Scheduling

- Efficiently manages and allocates CPU time to running processes.
- Ensures fair distribution of CPU resources among multiple tasks.
- Implements various scheduling algorithms, such as the Completely Fair Scheduler (CFS) and Round Robin, to meet different workload requirements.

# Multitasking: Concurrent Process Execution

- Supports concurrent execution of multiple processes, allowing for seamless switching between tasks.
- Enables users to run multiple applications simultaneously, enhancing system usability and responsiveness.
- Provides process isolation to prevent one misbehaving application from affecting others.

# Interrupt Handling: Timely Event Response

- Manages hardware and software interrupts, ensuring timely response to events.
- Handles asynchronous events, such as hardware device signals and system calls, by interrupting the CPU's current task to service the interrupt.
- Guarantees that critical tasks can be addressed promptly, even in a multitasking environment.

# Device Drivers: Bridging Software and Hardware

- Provides interfaces and drivers for hardware components, enabling communication.
- Supports a wide range of hardware devices, including storage controllers, network interfaces, graphics cards, and input devices.
- Allows user-level applications to interact with hardware through standardized abstractions.

# Memory Management: Efficient Resource Allocation

- Allocates and manages system memory, including physical and virtual memory.
- Implements virtual memory to isolate processes and provide each with its own memory address space.
- Optimizes memory usage through techniques like demand paging, memory swapping, and memory protection

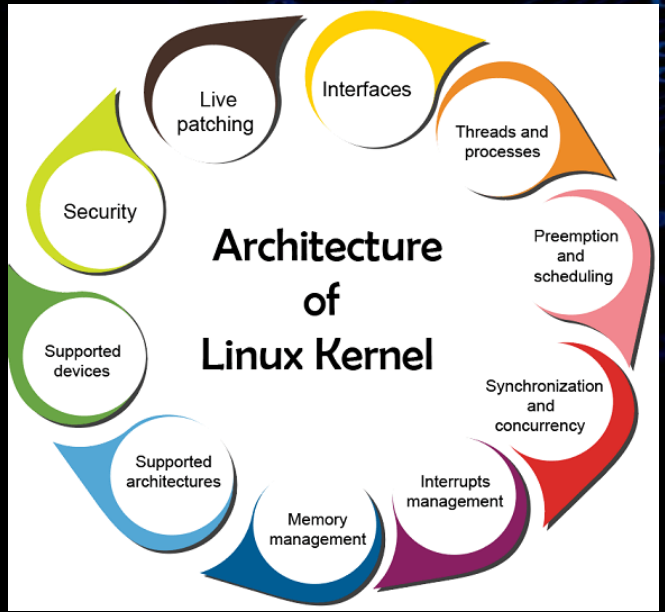
# System Calls: User-Kernel Interaction

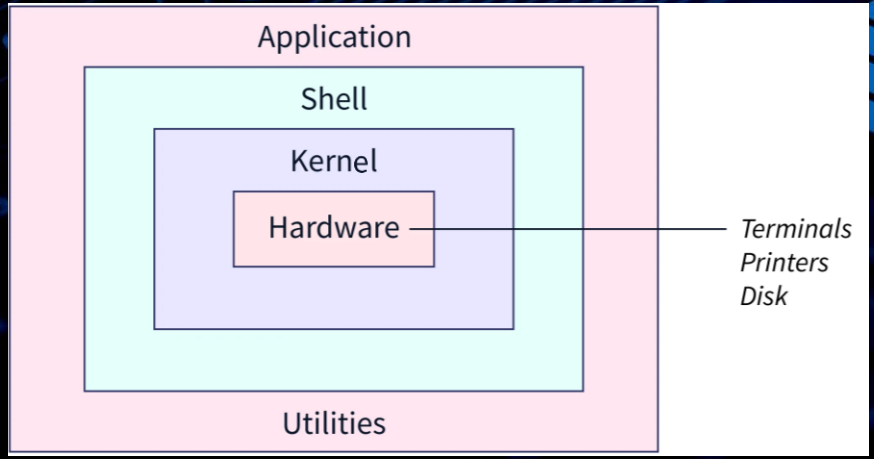
- Offers an interface for user-level applications to request services from the Kernel.
- Provides a set of predefined functions that allow user programs to perform privileged operations, such as file I/O, network communication, and process management.
- Acts as a bridge between user-level software and Kernel functions.

# Power Management: Optimizing Energy Usage

- Optimizes power usage through features like CPU frequency scaling and sleep states.
- Adjusts CPU clock frequencies and power states based on system load to conserve energy.
- Supports power-saving mechanisms to extend battery life in laptops and mobile devices.

# Linux Kernel



# Beyond Kernel; Now What?

- How System boots?
- How to login?
- How to access root?
- What commands does it support?
  - How to customize?

# Buildroot: Crafting Custom Embedded Linux Systems

- Simplifying Embedded OS Development
- Tailored Configurations at Your Fingertips
- Key Functions

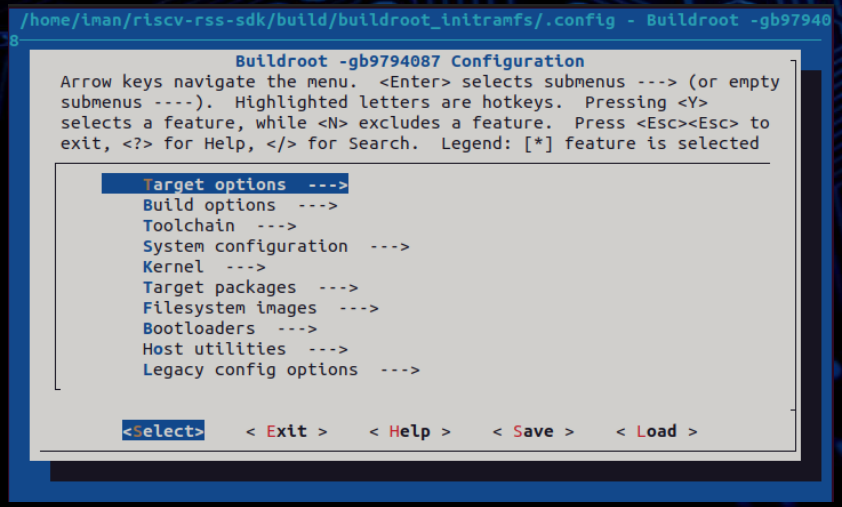
# Key Functions of Buildroot

- ✓ **Configuration Management:** Allows you to define and customize your embedded system's configuration, including target architecture, kernel version, and software packages.
- ✓ **Cross-Compilation:** Compiles software components for your target architecture, ensuring compatibility with your embedded hardware.
- ✓ **Package Management:** Provides a package management system for adding, removing, or updating software components in your embedded OS.
- ✓ **Root Filesystem Generation:** Generates a root filesystem for your embedded system, including directory structure, configuration files, and libraries.

# Key Functions of Buildroot

- ✓ **Bootloader Integration:** Supports integration with bootloaders like U-Boot, ensuring a smooth boot process on your embedded device.
- ✓ **Kernel Configuration:** Allows you to configure the Linux kernel to match your hardware and project requirements.
- ✓ **Optimization:** Streamlines the build process for size and performance, resulting in efficient and customized embedded Linux systems.

# Buildroot: Config Menu



# RISC-V Spike simulator: An Overview

- The Core Interface Between Software and Hardware
- Enabling Seamless Connection
- A Core Component of RISC-V Development
- Open-Source, Customizable
- Versatile Utility
- A Simulator (Not an Emulator)

# Test

- Simple C Code Compilation by GCC and Execution

```c
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
}
```

- Compile 32-bit Linux on simulator

# Results

test.s

```
    .file   "test.c"
    .option nopic
    .attribute arch, "rv32i2p0"
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .text
    .section    .rodata
    .align  2
.LC0:
    .string "Hello world!"
    .text
    .align  2
    .globl  main
    .type   main, @function
main:
    addi    sp,sp,-16
    sw  ra,12(sp)
    sw  s0,8(sp)
    addi    s0,sp,16
    lui a5,%hi(.LC0)
    addi    a0,a5,%lo(.LC0)
    call    puts
    li  a5,0
    mv  a0,a5
    lw  ra,12(sp)
    lw  s0,8(sp)
    addi    sp,sp,16
    jr  ra
    .size   main, .-main
    .ident  "GCC: (GNU) 11.1.0"
```

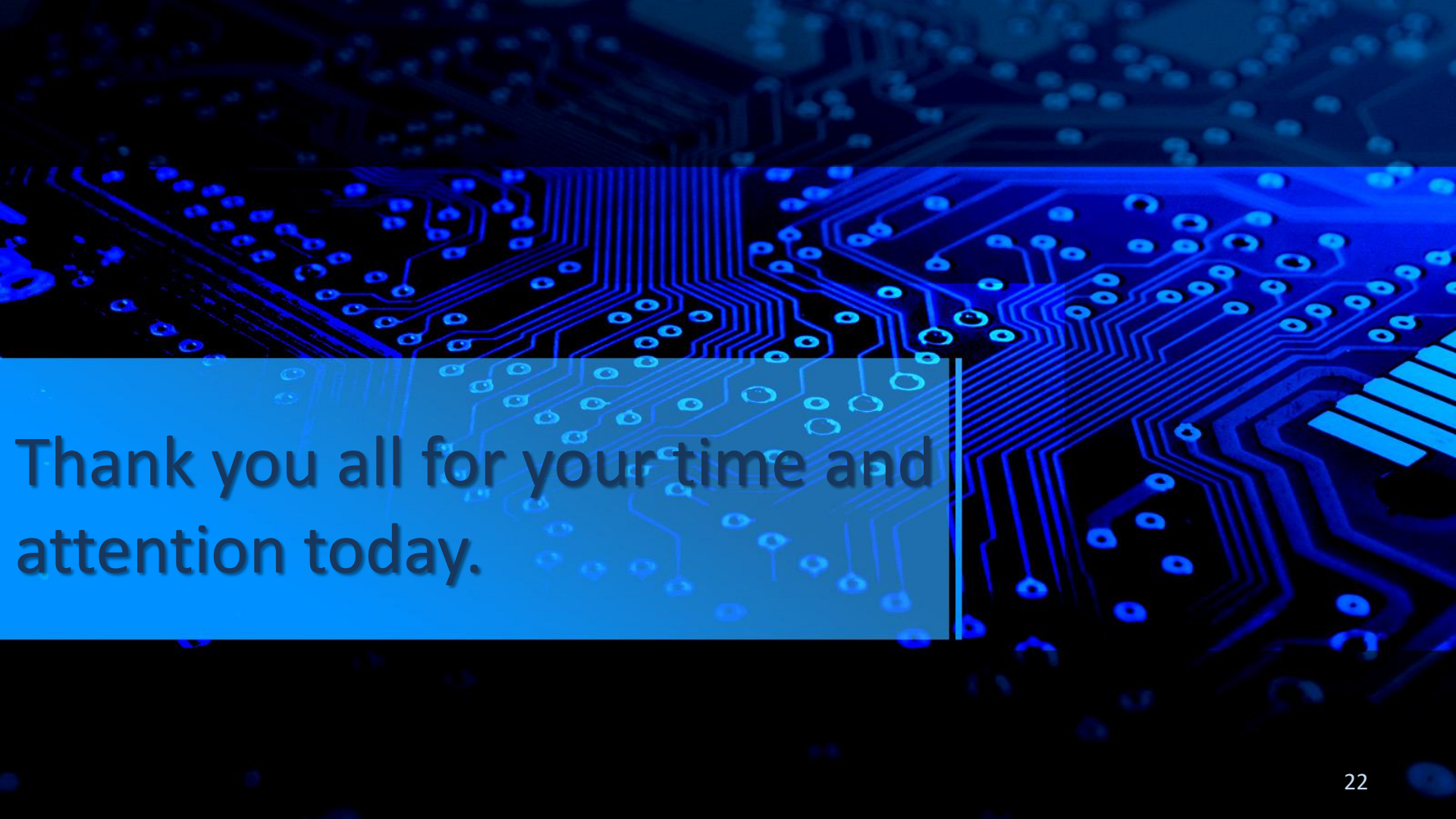
```
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
Starting network: OK

Welcome to Buildroot IMAN
buildroot login: root
root
Password: 123456

# mkdir test
mkdir test
# ls
ls
test
# mkdir test2
mkdir test2
# ls
ls
test    test2
#
```

# References


- www.redhat.com/topics/linux/linux-kernel
- https://en.wikipedia.org/wiki/Buildroot
- https://chipyard.readthedocs.io/en/latest/Software/Spike.html


Thank you all for your time and attention today.