**操作系统及安全课程实践（网安）**

# 实验3 使用C语言开发一个基于RISC-V架构的操作系统

## 实验介绍

本实验以RISC-V架构上的操作系统[TATAKA OS](https://github.com/yztz/tatakOS)为基础，结合操作系统及安全的理论课，设置了环境配置、运行与调试、系统调用、进程管理、内存管理、文件管理、实验进阶等若干个小实验，使同学能够了解在异构平台上开发RISC-V操作系统的基本概念、步骤和方法，并将相关理论知识与具体代码实现相结合，掌握代码的修改和调试方法。

## 任务描述

实验有若干个小任务组成， 采用小组协作方式进行，每小组最多4人，自设一名组长组织小组的分工合作，第一次上机时由组长将小组成员名单（包括组长姓名、成员姓名及学号）交给老师 ，每个小任务由小组内指定其中一人在头歌平台提交任务评测，但要求小组内所有学生都掌握实验的原理、方法和结果，所有实验最终撰写一份实验报告，指导教师在实验指导过程中进行实验的抽样验收。具体实验任务发布在头歌平台<https://www.educoder.net/>。

## 实验目的

掌握RISC-V异构平台开发环境配置与调试方法

理解操作系统核心模块在RISC-V架构下的实现

具备通过代码修改实现系统功能扩展的能力

形成从理论到实践的完整开发链路认知

## 实验环境

1. RISC-V架构介绍

RISC-V（发音为“risk-five”）是一种基于精简指令集（RISC）原则的开源指令集架构（ISA），其名称中的“V”既代表它是加州大学伯克利分校设计的第五代RISC架构，也象征着其灵活多变的可扩展性（Variation）及对向量运算（Vectors）的支持。作为开放指令集的典范，RISC-V凭借其模块化设计和开源生态，正迅速成为嵌入式系统、物联网设备乃至高性能计算领域的重要选择。

### 起源与发展历程

RISC-V的诞生源于学术界对传统指令集的反思。2010年，加州大学伯克利分校的Krste Asanovic教授、Andrew Waterman和Yunsup Lee等研究人员发现，主流指令集（如x86、ARM）的复杂性和高昂授权费用严重阻碍了计算机体系结构的创新。为此，他们决定设计一种全新的开源指令集，以解决学术研究和工业应用中的“无米之炊”困境。2014年，RISC-V架构正式对外公开，并成立RISC-V基金会（现更名为RISC-V International），吸引了包括谷歌、华为、英特尔等全球325家企业和机构的参与，共同推动其标准化与生态建设。

2015年，首个稳定版本RISC-V v2.0发布，定义了32/64位基础指令集（RV32I/RV64I），其核心指令仅40余条，显著简化了硬件实现。此后，基金会陆续推出支持向量扩展（2017年）、安全增强（2019年）等功能的版本，逐步完善生态。2020年，RISC-V International将总部迁至瑞士，进一步强化其全球中立性，加速技术普及。

### 核心设计哲学

RISC-V的设计以“精简、模块化、可扩展”为核心理念，与传统架构形成鲜明对比：

* **极简基础指令集**：基础整数指令（RV32I/RV64I）仅40余条，硬件实现复杂度远低于x86（数千条指令）和ARM（数百条指令）。
* **模块化分层设计**：通过标准扩展（如M/A/C/F/D）灵活组合功能，用户可基于应用需求裁剪指令集。例如，物联网设备可仅保留整数指令，而高性能处理器可叠加向量（V）和浮点（F）扩展。
* **开放性与可定制性**：采用BSD开源协议，允许自由修改和商业化，彻底打破传统架构的授权壁垒。

### 技术特性与优势

1. **高效指令编码**  
   采用定长32位指令格式，支持16位压缩指令（C扩展），兼顾代码密度与解码效率。指令编码高度规整，避免x86和ARM中常见的复杂解码逻辑。
2. **安全特权模型**  
   定义三级特权级（用户态U、监督态S、机器态M），通过物理内存保护（PMP）和分页机制（SV39/SV48）实现硬件级隔离，为操作系统开发提供安全基础。
3. **跨平台兼容性**  
   支持从微控制器（RV32E）到服务器（RV64GC）的全场景覆盖。例如，西部数据将其用于存储控制器，而欧洲处理器计划（EPI）则基于RISC-V设计百亿亿次超算加速器。

### 生态工具链成熟

配套GNU工具链（riscv-gcc/riscv-gdb）、QEMU模拟器及LLVM编译器均已完善，并与Linux、FreeRTOS等主流操作系统深度适配。

### 与传统架构对比



### 应用与未来展望

目前，RISC-V已从学术原型走向工业落地：SiFive推出高性能AI芯片，阿里平头哥发布RISC-V物联网处理器，NASA将其用于航天计算机。在学术界，MIT、ETH等顶尖高校将其纳入体系结构课程，每年的[全国大学生计算机系统能力大赛](https://os.educg.net/)，都有采用RISC-V进行操作系统开发的赛道，官网也提供了历年的优秀作品网站<https://github.com/oscomp/>供感兴趣的同学学习，而本实验采用的系统原型正是杭电2022年参加该比赛并获奖的优秀作品[TATAKA OS](https://github.com/yztz/tatakOS)。随着RISC-V International持续推动标准迭代，这一架构有望重塑全球芯片产业格局，成为开源硬件的基石。

1. QEMU介绍

### 基本定义

QEMU（Quick Emulator）是一个通用的开源的机器仿真和虚拟化工具，由传奇程序员法布里斯·贝拉（Fabrice Bellard）编写。QEMU 能够提供跨体系结构的硬件模拟，支持 x86、ARM、MIPS、RISC-V 等多种架构。

QEMU 拥有多种不同的使用方式，而在实验中我们所使用的主要是 QEMU 的系统仿真模式。在此模式中，QEMU 能够模拟处理器的执行过程以及各种硬件设备的行为，从而提供包括处理器、内存和外部设备在内的整机虚拟模型。在此模型之上，我们能够运行一个完整的操作系统，而不需要任何额外硬件的支持。

QEMU 提供了高度定制化的硬件模拟能力，使得搭建指定硬件平台的运行环境十分容易。并且 QEMU 也提供了使用 GDB 进行调试的原生支持，使程序的开发更加便捷。正因如此，QEMU 成为了底层开发领域十分重要的工具。

### QEMU核心运行模式

| **模式** | **应用场景** | **性能特点** |
| --- | --- | --- |
| **全系统仿真** | 运行完整操作系统 | 依赖动态翻译，速度较慢 |
| **用户模式仿真** | 跨架构运行单个程序（如ARM程序调试） | 轻量级，效率较高 |
| **硬件加速虚拟化** | 结合KVM/Xen运行虚拟机 | 接近原生性能 |

### QEMU工作原理

在学习操作系统虚拟化相关内容时，理解 QEMU 的工作原理有助于加深对计算机体系结构和虚拟化技术的认识。QEMU 是一个功能强大的开源仿真器和虚拟机管理程序（VMM），广泛用于系统开发、调试和虚拟化应用。

**1. 虚拟化技术概述**

虚拟化（Virtualization）是一种通过软件模拟硬件环境，使得多个操作系统能够在同一台物理机器上运行的技术。其核心思想是通过抽象物理资源，使操作系统和应用程序可以在与物理硬件解耦的环境中运行。

根据实现方式的不同，虚拟机管理程序（Hypervisor）可以分为两类：

* **第一类虚拟机管理程序（Type 1 Hypervisor）**：直接运行在硬件之上，充当操作系统的角色，管理多个虚拟机，例如 KVM、VMware ESXi。
* **第二类虚拟机管理程序（Type 2 Hypervisor）**：运行在已有的操作系统之上，作为一个普通应用程序运行，例如 VMware Workstation、Oracle VirtualBox 和 QEMU。

**2. QEMU 的特点和工作原理**

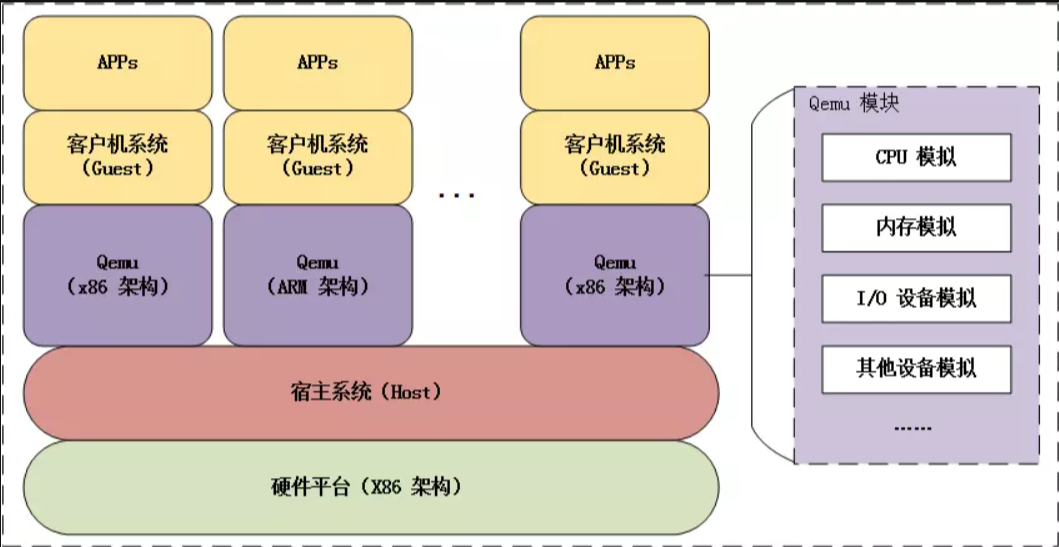
QEMU 是一个第二类虚拟机管理程序，但它的工作方式与一般的 Type 2 Hypervisor 不完全相同。

* **指令集架构级别的虚拟化**：
  + QEMU 采用软件仿真的方式，可以模拟不同架构的硬件环境。
  + 例如，可以在 x86 机器上运行 ARM、MIPS 或 RISC-V 平台的操作系统。
* **动态二进制翻译（Dynamic Binary Translation, DBT）**：
  + 直接解释执行机器码的方式效率较低，因此 QEMU 采用动态二进制翻译技术。
  + QEMU 使用 TCG（Tiny Code Generator）作为 JIT（Just-In-Time）翻译器，将客户机架构的机器码翻译为宿主机架构的机器码，并在运行时执行，以提高执行效率。

**3. QEMU 与 KVM 的结合**

QEMU 还可以与 KVM（Kernel-based Virtual Machine） 结合使用，以提升虚拟机性能。

* **KVM 的作用**：
  + KVM 是 Linux 内核中的一个模块，支持硬件辅助虚拟化（如 Intel VT-x、AMD-V）。
  + 通过 KVM，宿主机能够直接运行客户机的指令，而不必依赖 QEMU 的二进制翻译。
* **QEMU + KVM 的协作方式**：
  + QEMU 负责 I/O 虚拟化，例如磁盘、网络设备的模拟。
  + KVM 负责 CPU 和内存的虚拟化，提供接近原生硬件的性能。
  + 这种组合方式既保持了 QEMU 的灵活性（支持跨架构虚拟化），又提升了性能（利用 KVM 的硬件加速）。



### QEMU关键技术解析

1. **动态二进制翻译**

* 实时将目标架构指令（如RISC-V）翻译为主机架构指令（如x86）
* 代码缓存优化：通过Tiny Code Generator（TCG）减少重复翻译开销

1. **设备模拟能力**

* 完整模拟CPU、内存、磁盘、网络设备（如RISC-V VirtIO设备）
* 支持自定义设备开发（通过QEMU Device Model API）

1. **加速模式**

* KVM集成：利用CPU硬件虚拟化扩展（Intel VT-x/AMD-V），接管CPU和内存管理
* 启用方式：命令行添加-enable-kvm参数（需主机支持）

1. 交叉编译环境

### 为何需要交叉编译工具链？

| **编译类型** | **编译环境** | **执行环境** | **典型应用场景** |
| --- | --- | --- | --- |
| **本地编译** | x86主机编译 | x86主机运行 | 普通PC程序开发（如C++桌面应用） |
| **交叉编译** | x86主机编译 | RISC-V设备运行 | 嵌入式开发、操作系统内核构建 |

1. **架构差异壁垒**
   * **x86与RISC-V指令集不兼容**：x86采用复杂指令集（CISC），而RISC-V为精简指令集（RISC），二者机器码格式完全不同。
   * **示例冲突**：x86的mov指令在RISC-V中不存在对应编码，直接编译将导致非法指令错误。
2. **实验开发场景限制**
   * **目标平台资源受限**：RISC-V开发板通常内存有限（如8GB），无法承载GCC等大型编译工具运行。
   * **开发效率需求**：x86主机多核性能强劲（如12核CPU），编译速度远超嵌入式设备。

### 交叉编译工具链组成

1、[手工编译交叉编译工具链](https://github.com/riscv-collab/riscv-gnu-toolchain)**riscv-gnu-toolchain**。

1. 安装依赖

在安装交叉编译工具链之前，我们需要安装一些依赖：

**$ sudo apt install autoconf automake autotools-dev curl python3 python3-pip**

**$ sudo apt install libmpc-dev libmpfr-dev libgmp-dev gawk**

**$ sudo apt install build-essential bison flex texinfo gperf libtool patchutils**

**$ sudo apt install bc zlib1g-dev libexpat-dev ninja-build git cmake libglib2.0-dev**

1. 下载源码，通过以下命令来下载 RISC-V 交叉编译工具链的源码：

**$ git clone --recursive https://github.com/riscv/riscv-gnu-toolchain**

1. 编译安装，

进入源码目录，创建 build 文件夹并进入：

**$ cd riscv-gnu-toolchain**

**$ mkdir build**

**$ cd build**

riscv-gnu-toolchain 支持通过不同的编译命令和参数，生成不同版本的 toolchain。 分别支持： - 32bit 和 64bit - Newlib、glibc 和 musl 版本

* **编译 Newlib 版本：使用 make 命令**

**$ ./configure --prefix=${PWD}/riscv-newlib**

**$ make -j8**

**编译完成后，在 ${PWD}/riscv-newlib/bin 目录下生成 riscv64-unknown-elf-gcc**

* **编译 glibc 版本：使用 make linx 命令**

**$ ./configure --prefix=${PWD}/riscv64-linux**

**$ make linux -j8**

**默认编译的版本为 RV64GC (64-bit) ，更多的配置信息可以通过 ./configure --help 查看。**

1. 运行

**测试 toolchain 是否安装成功**

**$ riscv64-unknown-elf-gcc -v**

2、手工编译QEMU

**接下来，需要安装 QEMU。可以从**[**QEMU 官网**](https://www.qemu.org/)**下载源码并编译。以下是安装步骤：**

1. 下载 QEMU 源码

**wget https://download.qemu.org/qemu-7.1.0.tar.xz tar xvJf qemu-7.1.0.tar.xz cd qemu-7.1.0**

1. 安装依赖：

**sudo apt-get install libpixman-1-dev pip3 install sphinx==5.3.0**

1. 编译并安装 QEMU

**./configure --prefix=/opt/qemu**

**make -j4**

**make install**

### 使用配置好的开发环境的容器

1. **本实验已经提供了配置好开发环境的容器镜像，（链接： https://pan.quark.cn/s/1a16c11f00b8 提取码：Lbcz ）， 下载后，使用docker或isula读取镜像（注意，该镜像为x86环境下的容器）,下面以华为云上的OpenEuler系统（x86架构）上使用isula来完成环境搭建**

**[root@ecs-ddc9 ~]# unzip** [**kairos-lab-os-c-eh@v0.2.1.zip**](mailto:kairos-lab-os-c-eh@v0.2.1.zip)

**[root@ecs-ddc9 ~]# isula load -i kairos-lab-os-c-eh\@v0.2.1.tar**

**[root@ecs-ddc9 ~]# isula images**

**REPOSITORY TAG IMAGE ID CREATED SIZE**

**kairos-lab-os-c-eh latest d3fbb5a6b042 2025-03-22 23:23:11 2.614 GB**

1. **第一次运行容器镜像**

**[root@ecs-ddc9 ~]# mkdir oslab**

**[root@ecs-ddc9 ~]# cd oslab**

**[root@ecs-ddc9 ~]# isula run -it --privileged --net host -v /root/oslab:/root/oslab kairos-lab-os-c-eh**

**这里相当于将本地路径/root/oslab映射到容器路径/root/oslab**

**root@localhost:/#ls**

**bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var**

**root@localhost:/# riscv64-unknown-elf-gcc -v**

**Using built-in specs.**

**…**

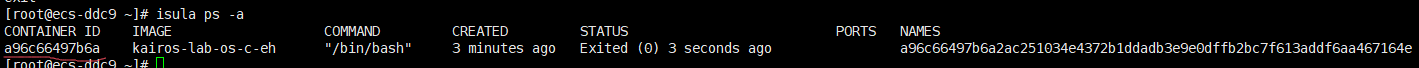
**gcc version 10.2.0 ()**

**退出容器**

**[root@ecs-ddc9 ~]# exit**

1. **以后运行容器镜像**

**查看容器id**



**启动容器，并进入容器的bash**

**[root@ecs-ddc9 ~]# isula start a96c66497b6a**

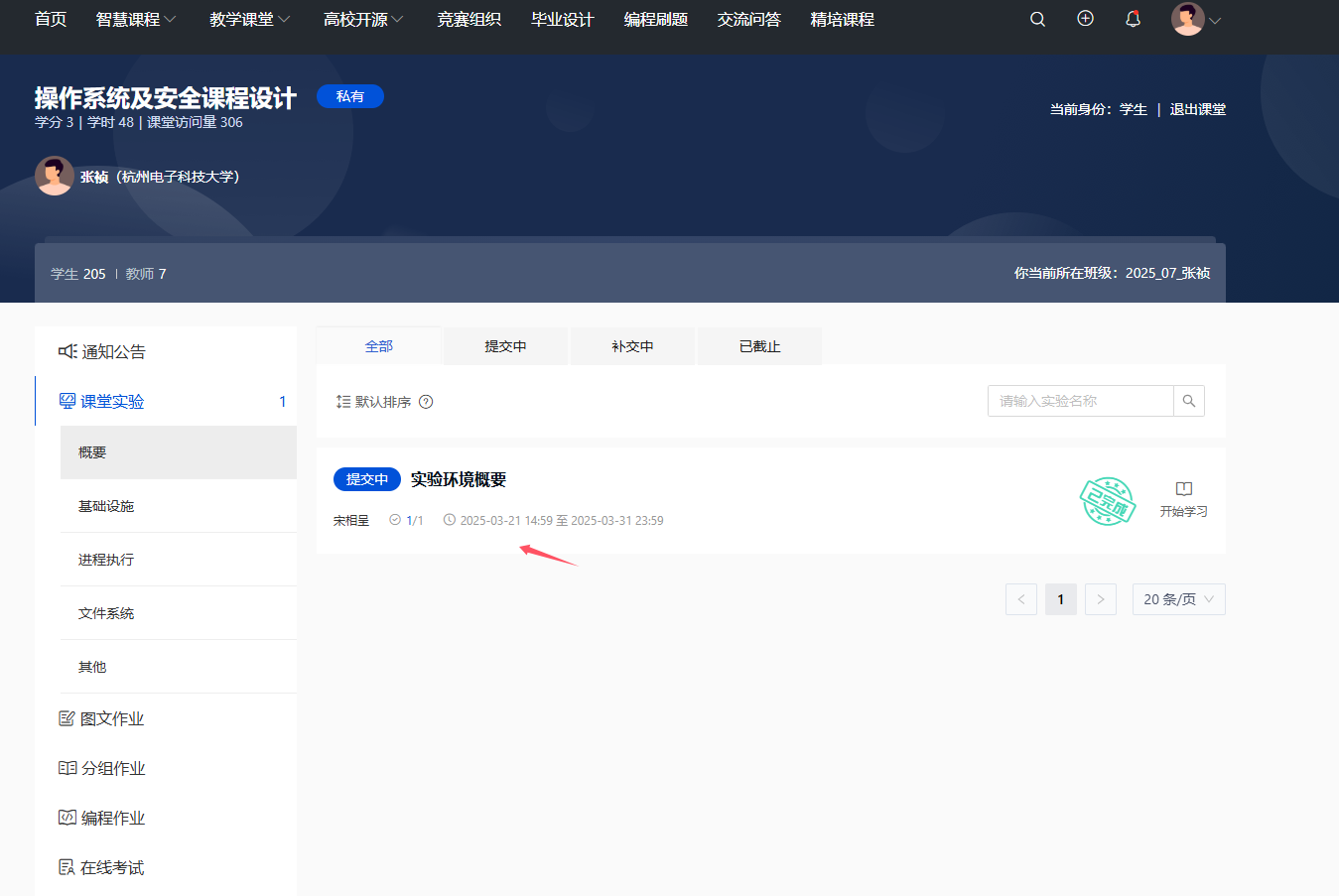
**[root@ecs-ddc9 ~]# isula exec -it a96c66497b6a /bin/bash**

**root@localhost:/#**

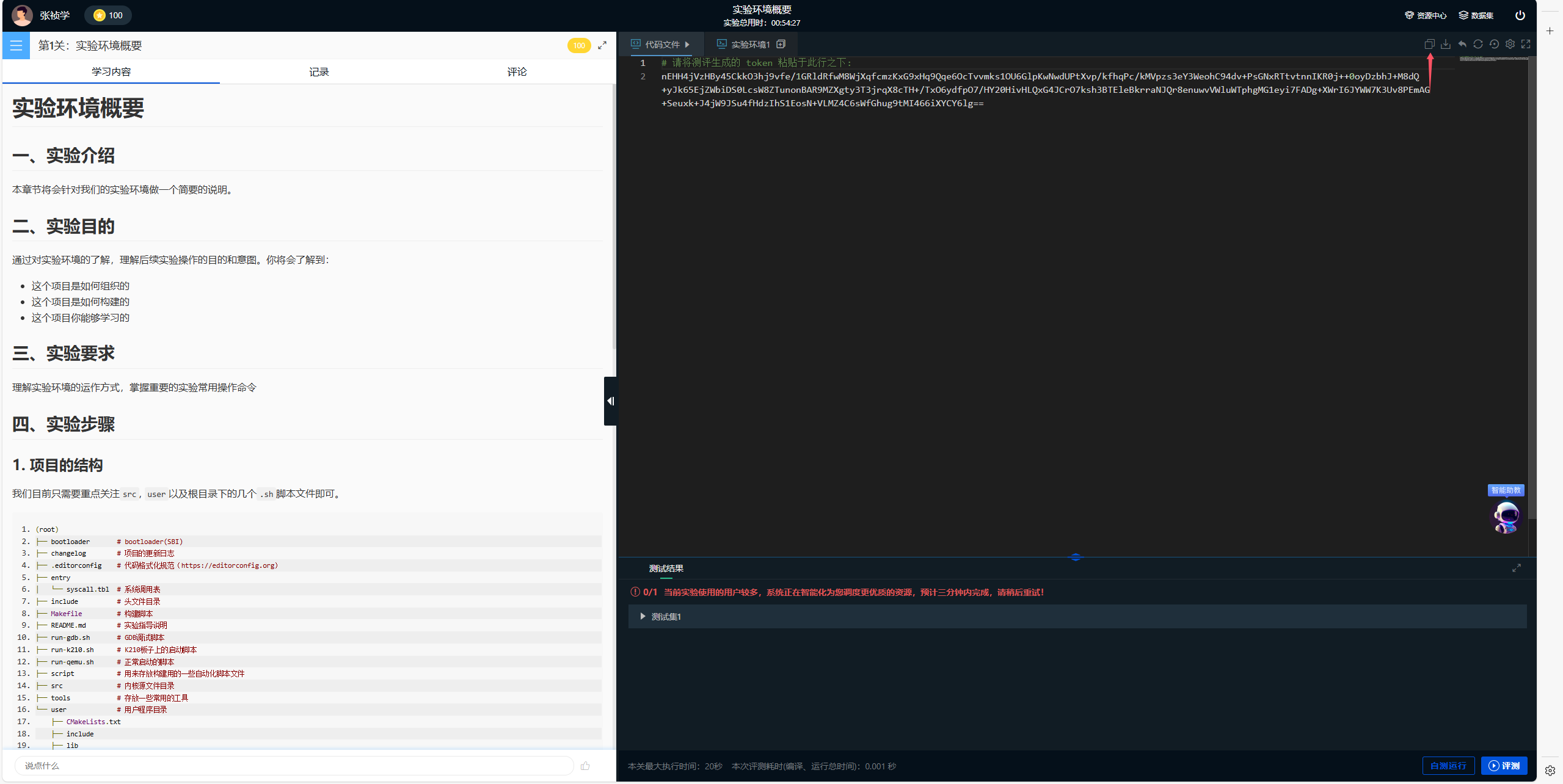
1. 头歌平台完成实验

### 登录头歌实验平台

1. 登录平台后，通过老师发布的邀请码进入课堂，查看发布的实验任务（注意完成截止时间）



1. 点击开始学习



该界面左侧显示的为实验要求，请仔细阅读，右侧为评估代码提交区域，点击红色箭标注的位置，复制代码下载链接，链接类似如下https://git.educoder.net/xxxxx/xxxxxx.git

1. 转到实验3搭建的代码实验环境，下载代码，有两种方式：
   * 在容器内通过git clone <https://git.educoder.net/xxxxx/xxxxxx.git>下载代码，完成实验（可能会由于docker或isula版本问题，导致git clone运行失败）
   * 在容器外通过容器文件夹映射的方式，在映射的容器外文件夹内容git clone下载代码，然后再容器内相应映射的文件夹内完成实验（如果在Windows的docker下，可能会因为Windows下的命名规则问题，导致后续实验时，容器内编译失败）
2. 通过文本编辑器编辑代码，可以采用vim或vscode（安装插件“Docker”与“Dev Containers”以及Remote SSH插件）
3. 进入容器，测试

**[root@ecs-ddc9 ~]# isula start 5546a2a31ac9**

**[root@ecs-ddc9 ~]# isula exec -it 5546a2a31ac9 /bin/bash**

**root@ecs-ddc9:/#**

**root@ecs-ddc9:~/oslab/s9lrng67po20250321164454# chmod +x run-qemu.sh**

**root@ecs-ddc9:~/oslab/s9lrng67po20250321164454#**

**mmio-bus.0**

**[rustsbi] RustSBI version 0.2.2, adapting to RISC-V SBI v1.0.0**

**.\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_.\_\_\_\_\_\_\_\_\_\_\_. \_\_\_\_\_\_\_..\_\_\_\_\_\_ \_\_**

**| \_ \ | | | | / | | / || \_ \ | |**

**| |\_) | | | | | | (----`---| |----`| (----`| |\_) || |**

**| / | | | | \ \ | | \ \ | \_ < | |**

**| |\ \----.| `--' |.----) | | | .----) | | |\_) || |**

**| \_| `.\_\_\_\_\_| \\_\_\_\_\_\_/ |\_\_\_\_\_\_\_/ |\_\_| |\_\_\_\_\_\_\_/ |\_\_\_\_\_\_/ |\_\_|**

**[rustsbi] Implementation: RustSBI-QEMU Version 0.1.1**

**[rustsbi-dtb] Hart count: cluster0 with 1 cores**

**[rustsbi] misa: RV64ACDFIMSU**

**[rustsbi] mideleg: ssoft, stimer, sext (0x222)**

**[rustsbi] medeleg: ima, ia, bkpt, la, sa, uecall, ipage, lpage, spage (0xb1ab)**

**[rustsbi] pmp0: 0x10000000 ..= 0x10001fff (rw-)**

**[rustsbi] pmp6: 0x2000000 ..= 0x200ffff (rw-)**

**[rustsbi] pmp12: 0xc000000 ..= 0xc3fffff (rw-)**

**[rustsbi] enter supervisor 0x80200000**

**.bss 0x8022c980-0x802f0e98 (0xc4518) cleared**

**tataka os**

**hart 0 start**

**Lab 1 - Overview**

**Kernel starts successfully!**

**Edit user/src/init0.c to run your program.**

**Enjoy it!**

**child welcome exited with 0**

**root@ecs-ddc9:~/oslab/s9lrng67po20250321164454#**

1. 评估实验

**root@ecs-ddc9:~/oslab/s9lrng67po20250321164454# eh --lab-id 1 -t**

**All test cases passed!**

**下面是用于登记评分的 token，请复制到头歌平台(Web端)中的 `result.txt` 文件中，并执行测评:**

**VCCl0V0CO0QW135NAEmKEkZdo5E1xO7PB2Fwe1YQAv83AjFfusOTMzKdEZO2L39ieWYDU219a6f1mZ1lLeNOLD+i1kqEWCmBWBl2xzBxkOpVmXDuu5G93XSeb4F+HNCfrcfb6pTNHGTqg7/xXnxHD8q4PhkdSy/pLK6xBnutC4pYGr51udoP2oV7GdAZqNVBet0SKwG16MphvTQneAaJE0wr5ETFh6vTwL+6L0a7VUDOqYzUMpxqDJq7Zn6gIVYY+rbLkHpPV86Q7b0CzOJM6nzMER5ibyWFWdjabPlkinaI6SGH0xAzhx61morIfUj8yjFiZA7DMaZfG223OvU2RA==**

**root@ecs-ddc9:~/oslab/s9lrng67po20250321164454#**

1. 复制评估的token到头歌平台，ctrl+s保存后，点击红色箭头位置的评估按钮，查看评估结果

